

Simulink[®] Test[™]

Reference



MATLAB[®]&SIMULINK[®]

R2022a



How to Contact MathWorks



Latest news: www.mathworks.com
Sales and services: www.mathworks.com/sales_and_services
User community: www.mathworks.com/matlabcentral
Technical support: www.mathworks.com/support/contact_us



Phone: 508-647-7000



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

Simulink® Test™ Reference

© COPYRIGHT 2015–2022 by The MathWorks, Inc.

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

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

Trademarks

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

Patents

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

Revision History

March 2015	Online Only	New for Version 1.0 (Release 2015a)
September 2015	Online Only	Revised for Version 1.1 (Release 2015b)
October 2015	Online only	Rereleased for Version 1.0.1 (Release 2015aSP1)
March 2016	Online Only	Revised for Version 2.0 (Release 2016a)
September 2016	Online Only	Revised for Version 2.1 (Release 2016b)
March 2017	Online Only	Revised for Version 2.2 (Release 2017a)
September 2017	Online Only	Revised for Version 2.3 (Release 2017b)
March 2018	Online Only	Revised for Version 2.4 (Release 2018a)
September 2018	Online Only	Revised for Version 2.5 (Release 2018b)
March 2019	Online Only	Revised for Version 3.0 (Release 2019a)
September 2019	Online Only	Revised for Version 3.1 (Release 2019b)
March 2020	Online Only	Revised for Version 3.2 (Release 2020a)
September 2020	Online Only	Revised for Version 3.3 (Release 2020b)
March 2021	Online Only	Revised for Version 3.4 (Release 2021a)
September 2021	Online Only	Revised for Version 3.5 (Release 2021b)
March 2022	Online Only	Revised for Version 3.6 (Release 2022a)

1	Functions
2	Classes
3	Methods
4	Blocks
5	Language Operators

Functions

disp

Display results of `sltest.AssessmentSet` or `sltest.Assessment`

Syntax

`disp(as)`

Description

`disp(as)` displays the results of the assessment object `as`.

Examples

Display results of an assessment

Display the results of the assessment `as`, where `as` is an `sltest.Assessment`.

`disp(as)`

```
sltest.Assessment
Package: sltest

Properties:
    Name: 'Simulink:verify_high'
    BlockPath: [1x1 Simulink.SimulationData.BlockPath]
    Values: [1x1 timeseries]
    Result: Fail
```

Input Arguments

as — Assessment object

`sltest.Assessment` | `sltest.AssessmentSet`

Assessment object for which to display results.

Example: `as`

See Also

`sltest.getAssessments` | `sltest.Assessment` | `sltest.AssessmentSet`

Introduced in R2016b

Test Manager

Model and code testing in different execution environments, manage test suites, and analyze and report results

Description

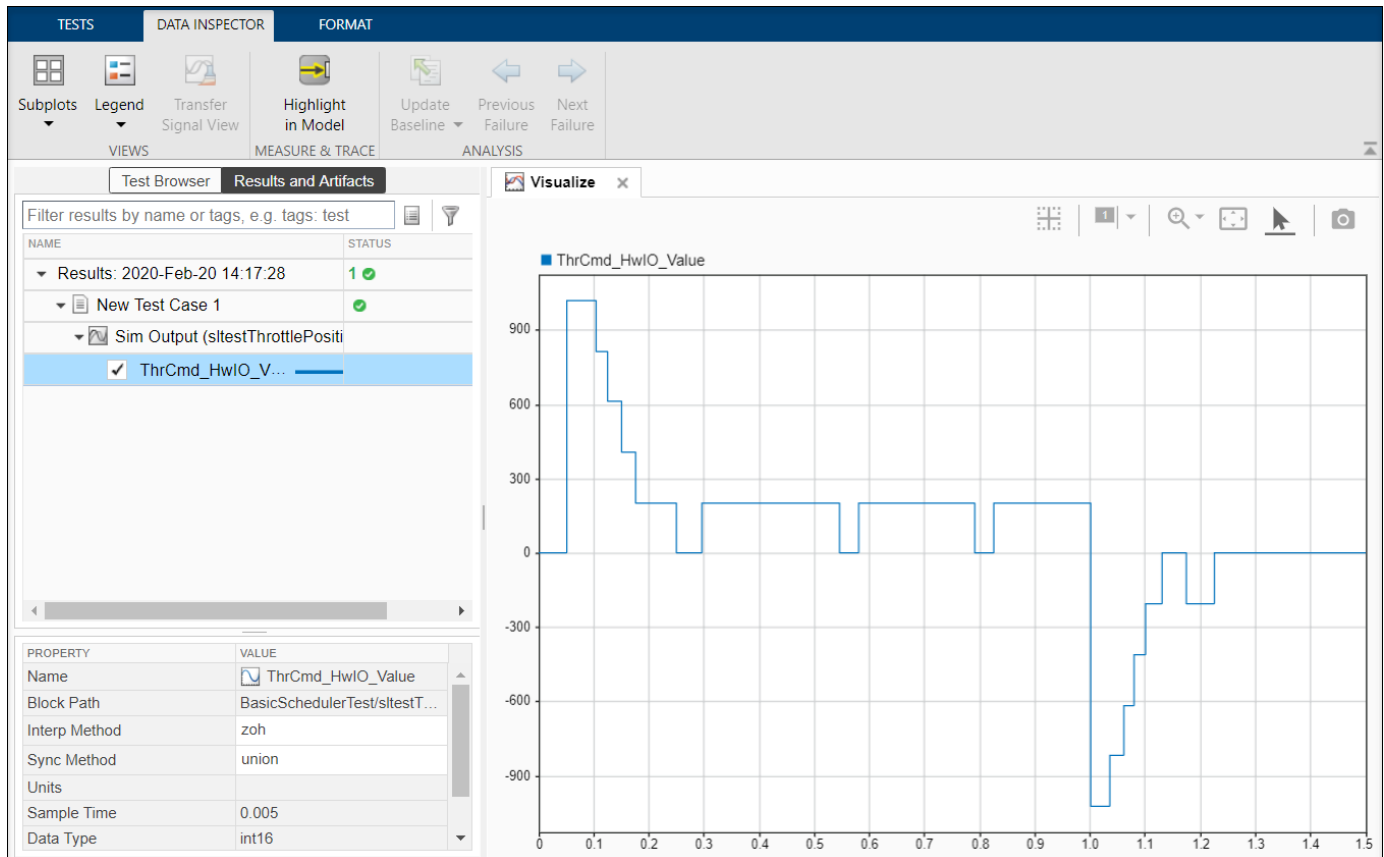
You can use the **Test Manager** to create test cases for Simulink models and code in desktop, SIL, PIL, and HIL (for target computers running Simulink Real-Time™) execution environments. You can create or import test data, run tests, view results, and create reports. You can also import and test C and C++ code, and MATLAB® test scripts.

You can group test cases into test suites, which are saved in a test file. The types of test you can create are simulation tests, equivalence (back-to-back) tests to compare models or generated code, and baseline tests to compare model output to a known baseline. You can run tests directly on your model or use test harnesses, which keep the test environment separate from your model. See “Test Harness and Model Relationship”.

You can specify the model or component to test, as well as:

- Test inputs
- Test outputs, such as logged signals and states
- Baseline criteria for baseline tests
- Iterations to run different combinations of inputs, parameters, baseline criteria, etc.
- Temporal and logical assessments of simulation behavior
- Model parameter and configuration overrides
- Callbacks to run before and after loading the model
- Coverage settings to measure the extent to which model or code execution paths are exercised
- Link requirements to the tests that verify them

Note If your model or test harness contains a To Workspace block, the block variable is not saved in the base workspace after the test finishes running. Upon test completion, the base workspace is restored to its original state.



Open the Test Manager

- In the Simulink toolstrip
 - 1 Open the **Apps** tab.
 - 2 In the Model Verification, Validation, and Test section, click **Simulink Test**.
 - 3 On the **Tests** tab, click **Simulink Test Manager** on the Tests toolstrip.
- At the MATLAB command prompt, enter `sltest.testmanager.view`.

In the Test Manager, the test sections in the **Test Browser** pane contain the options for the test. For descriptions of the options, see “Specify Test Properties in the Test Manager”.

By default, running a test in the Test Manager disables breakpoints for Simulink, Stateflow[®], and Test Sequence blocks. Use **Run with Stepper** in the Test Manager to run a test in the simulation stepper and use the breakpoints there.

Examples

- “Specify Test Properties in the Test Manager”
- “Capture Simulation Data in a Test Case”

- “Create and Run a Back-to-Back Test”
- “Create and Run a Baseline Test”
- “Create a Test Harness”
- “View Test Case Results”

See Also

Topics

“Specify Test Properties in the Test Manager”

“Capture Simulation Data in a Test Case”

“Create and Run a Back-to-Back Test”

“Create and Run a Baseline Test”

“Create a Test Harness”

“View Test Case Results”

Introduced in R2015a

find

Find assessments in `sltest.AssessmentSet` or `sltest.Assessment` object

Syntax

```
asout = find(as,'PropertyName','PropertyValue')
asout = find(as,'PropertyName1','PropertyValue1','-logical','PropertyName2',
'PropertyValue2'...)
asout = find(as,'-regex','PropertyName','PropertyValue')
```

Description

`asout = find(as,'PropertyName','PropertyValue')` returns the results `asout` specified by the properties matching `'PropertyName'`, and `'PropertyValue'`.

`asout = find(as,'PropertyName1','PropertyValue1','-logical','PropertyName2', 'PropertyValue2'...)` returns the results `asout` specified by multiple `'PropertyName'`, `'PropertyValue'` pairs, and the `'-logical'` operator specifying the connective between the pairs. `'-logical'` can be `'-and'` or `'-or'`.

`asout = find(as,'-regex','PropertyName','PropertyValue')` returns assessment results whose `'PropertyName'` matches the regular expression `'PropertyValue'`. When using regular expression search, `'PropertyName'` can be the assessment object `'Name'` or `'BlockPath'`.

Examples

Get Assessments from a Simulation

This example shows how to simulate a model with `verify` statements and obtain assessment results via the programmatic interface.

Get the Assessment Set and One Assessment Result

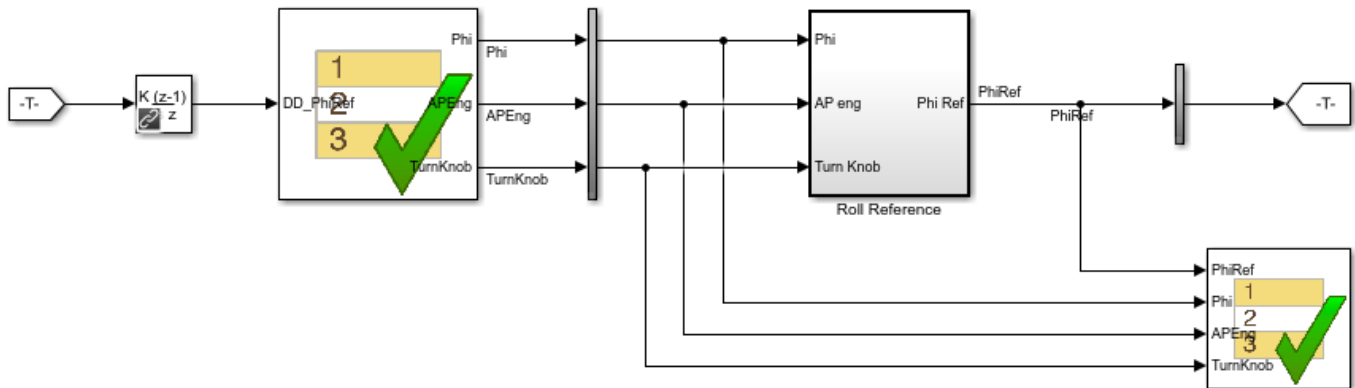
1. Open the model.

```
open_system('sltestRollRefTestExample.slx')

% Turn the command line warning off for verify() statements
warning off Stateflow:Runtime:TestVerificationFailed
```

This model is used to show how verify() statements work in Test Sequence and Test Assessment blocks,

Copyright 2019 The MathWorks, Inc.



Copyright 2016-2020 The MathWorks, Inc.

2. Run the model.

```
s = sim('sltestRollRefTestExample');
```

3. Get the assessment set.

```
as = sltest.getAssessments('sltestRollRefTestExample');
```

4. Get assessment 3 from the assessment set.

```
as3 = get(as,3);
```

Display Results of the Assessment Set and Assessment Result

1. Get summary of the assessment set.

```
asSummary = getSummary(as)
```

```
asSummary =
```

```
struct with fields:
```

```
Total: 6
Untested: 3
Passed: 2
Failed: 1
Result: Fail
```

2. Display the result of assessment 3.

```
disp(as3)
```

```
sltest.Assessment
Package: sltest

Properties:
  Name: 'Simulink:verify_high'
  BlockPath: [1x1 Simulink.SimulationData.BlockPath]
  Values: [1x1 timeseries]
  Result: Fail
```

3. Find untested or failed results in the assessment set.

```
asFailUntested = find(as, 'Result', slTestResult.Fail, '-or', ...
    'Result', slTestResult.Untested)
```

```
asFailUntested =
```

```
sltest.AssessmentSet
Summary:
  Total: 4
  Untested: 3
  Passed: 0
  Failed: 1
  Result: Fail
```

```
Untested Assessments (first 10):
  2 : Untested 'Simulink:verify_high'
  3 : Untested 'Simulink:verifyTKLow'
  4 : Untested 'Simulink:verifyTKNormal'
```

```
Failed Assessments (first 10):
  1 : Fail 'Simulink:verify_high'
```

4. Find assessments under the Test Assessment block, using a regular expression.

```
assessBlock = find(as, '-regexp', 'BlockPath', '.[Aa]ssess')
```

```
assessBlock =
```

```
sltest.AssessmentSet
Summary:
  Total: 6
  Untested: 3
  Passed: 2
  Failed: 1
  Result: Fail
```

```
Untested Assessments (first 10):
  4 : Untested 'Simulink:verify_high'
  5 : Untested 'Simulink:verifyTKLow'
  6 : Untested 'Simulink:verifyTKNormal'
```

```
Passed Assessments (first 10):
```

```
1 : Pass 'Simulink:verify_normal'
2 : Pass 'Simulink:verify_low'
```

```
Failed Assessments (first 10):
3 : Fail 'Simulink:verify_high'
```

Re-enable warnings

warning on `Stateflow:Runtime:TestVerificationFailed`

Input Arguments

as — Assessment object

`sltest.Assessment` | `sltest.AssessmentSet`

Assessment object to search.

Example: `as`

'-logical' — Logical operator

`'-and'` | `'-or'`

Logical operator connecting multiple property names or property values.

Example: `'-and'`

'PropertyName' — Type of property to search

`'Name'` | `'Result'` | `'BlockPath'`

Type of property to search.

Example: `'BlockPath'`

'PropertyValue' — Property value to search

character vector | `slTestResult` enumeration

Property value to search, specified as a character vector. Can be a regular expression when using the `'-regexp'` argument.

When using the `'Result'` property name, `'PropertyValue'` is an enumeration of the assessment result:

- `slTestResult.Fail` for failed assessments
- `slTestResult.Pass` for passed assessments
- `slTestResult.Untested` for untested assessments

Example: `slTestResult.Fail`

Example: `'[Aa]sess'`

'-regexp' — Command to search using regular expression

character vector

Regular expression for `BlockPath` properties search, specified as a character vector.

Example: `'-regexp'`

Output Arguments

asout — Assessment results output

`sltest.assessmentSet` object

Assessment results output from the `find` operation, specified as an `sltest.assessmentSet` object.

Example: `sltest.AssessmentSet`

See Also

`sltest.getAssessments` | `sltest.Assessment` | `sltest.AssessmentSet`

Introduced in R2016b

get

Get assessment of `sltest.AssessmentSet`

Syntax

```
indexResult = get(as,index)
```

Description

`indexResult = get(as,index)` gets the individual assessment result `indexResult` from the `sltest.AssessmentSet` `as`, specified by the integer `index`.

Examples

Get Assessments from a Simulation

This example shows how to simulate a model with `verify` statements and obtain assessment results via the programmatic interface.

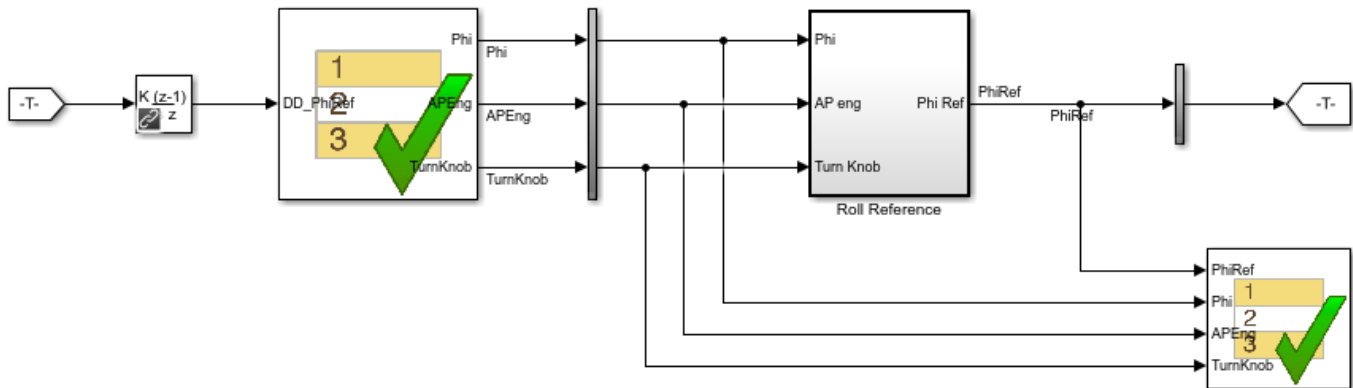
Get the Assessment Set and One Assessment Result

1. Open the model.

```
open_system('sltestRollRefTestExample.slx')  
  
% Turn the command line warning off for verify() statements  
warning off Stateflow:Runtime:TestVerificationFailed
```

This model is used to show how verify() statements work in Test Sequence and Test Assessment blocks,

Copyright 2019 The MathWorks, Inc.



Copyright 2016-2020 The MathWorks, Inc.

2. Run the model.

```
s = sim('sltestRollRefTestExample');
```

3. Get the assessment set.

```
as = sltest.getAssessments('sltestRollRefTestExample');
```

4. Get assessment 3 from the assessment set.

```
as3 = get(as,3);
```

Display Results of the Assessment Set and Assessment Result

1. Get summary of the assessment set.

```
asSummary = getSummary(as)
```

```
asSummary =
```

```
struct with fields:
```

```
Total: 6
Untested: 3
Passed: 2
Failed: 1
Result: Fail
```

2. Display the result of assessment 3.


```
disp(as3)
```

```
sltest.Assessment
Package: sltest

Properties:
  Name: 'Simulink:verify_high'
  BlockPath: [1x1 Simulink.SimulationData.BlockPath]
  Values: [1x1 timeseries]
  Result: Fail
```

3. Find untested or failed results in the assessment set.

```
asFailUntested = find(as, 'Result', slTestResult.Fail, '-or', ...
  'Result', slTestResult.Untested)
```

```
asFailUntested =
```

```
sltest.AssessmentSet
Summary:
  Total: 4
  Untested: 3
  Passed: 0
  Failed: 1
  Result: Fail
```

```
Untested Assessments (first 10):
  2 : Untested 'Simulink:verify_high'
  3 : Untested 'Simulink:verifyTKLow'
  4 : Untested 'Simulink:verifyTKNormal'
```

```
Failed Assessments (first 10):
  1 : Fail 'Simulink:verify_high'
```

4. Find assessments under the Test Assessment block, using a regular expression.

```
assessBlock = find(as, '-regexp', 'BlockPath', '.[Aa]ssess')
```

```
assessBlock =
```

```
sltest.AssessmentSet
Summary:
  Total: 6
  Untested: 3
  Passed: 2
  Failed: 1
  Result: Fail
```

```
Untested Assessments (first 10):
  4 : Untested 'Simulink:verify_high'
  5 : Untested 'Simulink:verifyTKLow'
  6 : Untested 'Simulink:verifyTKNormal'
```

```
Passed Assessments (first 10):
```

```
1 : Pass 'Simulink:verify_normal'  
2 : Pass 'Simulink:verify_low'
```

```
Failed Assessments (first 10):  
3 : Fail 'Simulink:verify_high'
```

Re-enable warnings

warning on `Stateflow:Runtime:TestVerificationFailed`

Input Arguments

as — Assessment set from which to get a single assessment

`sltest.AssessmentSet`

This is the `sltest.AssessmentSet`, from which to get a single assessment.

Example: `sltest.AssessmentSet`

index — Index of single assessment

integer

Index of a single assessment to return to the `sltest.Assessment` object, specified as an integer.

Example: 3

See Also

`sltest.getAssessments` | `sltest.Assessment` | `sltest.AssessmentSet`

Introduced in R2016b

getSummary

Get summary of `sltest.AssessmentSet`

Syntax

```
testOut = getSummary(as)
```

Description

`testOut = getSummary(as)` gets the summary `testOut` of the `sltest.AssessmentSet` `as`.

Examples

Get Assessments from a Simulation

This example shows how to simulate a model with `verify` statements and obtain assessment results via the programmatic interface.

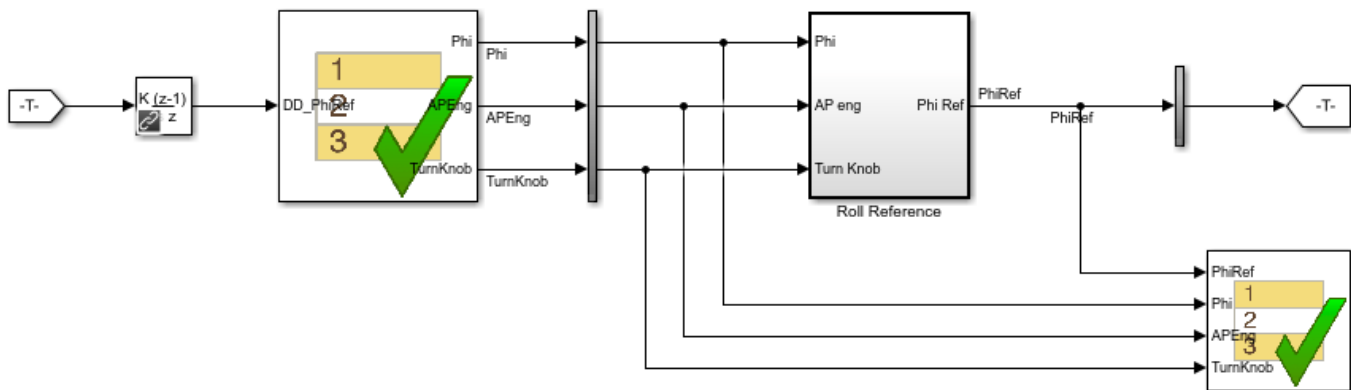
Get the Assessment Set and One Assessment Result

1. Open the model.

```
open_system('sltestRollRefTestExample.slx')  
  
% Turn the command line warning off for verify() statements  
warning off Stateflow:Runtime:TestVerificationFailed
```

This model is used to show how verify() statements work in Test Sequence and Test Assessment blocks,

Copyright 2019 The MathWorks, Inc.



Copyright 2016-2020 The MathWorks, Inc.

2. Run the model.

```
s = sim('sltestRollRefTestExample');
```

3. Get the assessment set.

```
as = sltest.getAssessments('sltestRollRefTestExample');
```

4. Get assessment 3 from the assessment set.

```
as3 = get(as,3);
```

Display Results of the Assessment Set and Assessment Result

1. Get summary of the assessment set.

```
asSummary = getSummary(as)
```

```
asSummary =
```

```
struct with fields:
```

```
Total: 6
Untested: 3
Passed: 2
Failed: 1
Result: Fail
```

2. Display the result of assessment 3.

```
disp(as3)
```

```
sltest.Assessment
Package: sltest

Properties:
  Name: 'Simulink:verify_high'
  BlockPath: [1x1 Simulink.SimulationData.BlockPath]
  Values: [1x1 timeseries]
  Result: Fail
```

3. Find untested or failed results in the assessment set.

```
asFailUntested = find(as, 'Result', slTestResult.Fail, '-or', ...
    'Result', slTestResult.Untested)
```

```
asFailUntested =
```

```
sltest.AssessmentSet
Summary:
  Total: 4
  Untested: 3
  Passed: 0
  Failed: 1
  Result: Fail
```

```
Untested Assessments (first 10):
  2 : Untested 'Simulink:verify_high'
  3 : Untested 'Simulink:verifyTKLow'
  4 : Untested 'Simulink:verifyTKNormal'
```

```
Failed Assessments (first 10):
  1 : Fail 'Simulink:verify_high'
```

4. Find assessments under the Test Assessment block, using a regular expression.

```
assessBlock = find(as, '-regexp', 'BlockPath', '.[Aa]ssess')
```

```
assessBlock =
```

```
sltest.AssessmentSet
Summary:
  Total: 6
  Untested: 3
  Passed: 2
  Failed: 1
  Result: Fail
```

```
Untested Assessments (first 10):
  4 : Untested 'Simulink:verify_high'
  5 : Untested 'Simulink:verifyTKLow'
  6 : Untested 'Simulink:verifyTKNormal'
```

```
Passed Assessments (first 10):
```

```
1 : Pass 'Simulink:verify_normal'  
2 : Pass 'Simulink:verify_low'
```

```
Failed Assessments (first 10):  
3 : Fail 'Simulink:verify_high'
```

Re-enable warnings

warning on `Stateflow:Runtime:TestVerificationFailed`

Input Arguments

as — Assessment set from which to get a summary

`sltest.AssessmentSet`

This is the `sltest.AssessmentSet`, from which to get a summary.

Example: `sltest.AssessmentSet`

Output Arguments

testOut — Assessment summary

`struct`

Summary of the assessment set, specified as a `struct`.

See Also

`sltest.getAssessments` | `sltest.Assessment` | `sltest.AssessmentSet`

Introduced in R2016b

sltest.getAssessments

Returns test assessment set object

Syntax

```
as = sltest.getAssessments(model)
```

Description

`as = sltest.getAssessments(model)` returns `as`, an `sltest.AssessmentSet` from assessments in `model`. Simulate the model before getting the assessment results.

`as` includes results from:

- `verify` statements
- Blocks in the Model Verification library

Examples

Get Assessments from a Simulation

This example shows how to simulate a model with `verify` statements and obtain assessment results via the programmatic interface.

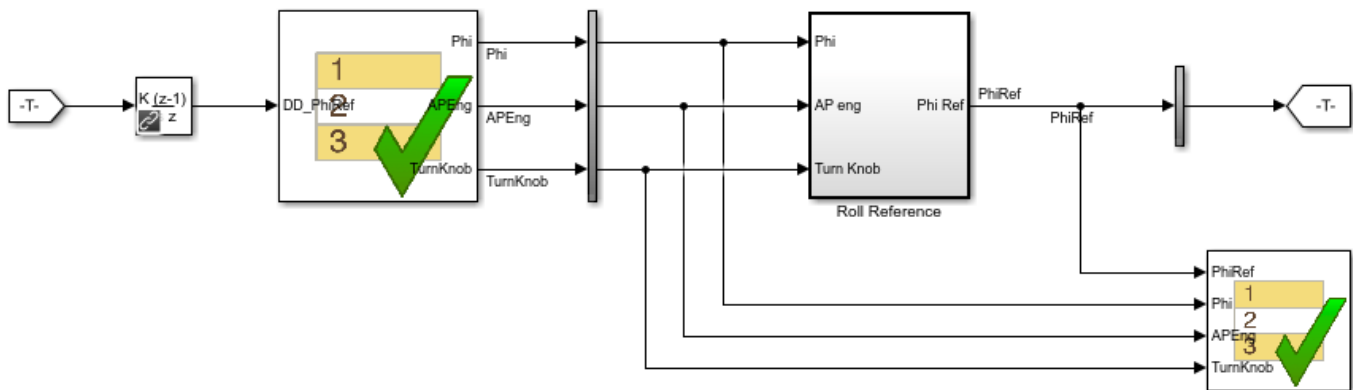
Get the Assessment Set and One Assessment Result

1. Open the model.

```
open_system('sltestRollRefTestExample.slx')  
  
% Turn the command line warning off for verify() statements  
warning off Stateflow:Runtime:TestVerificationFailed
```

This model is used to show how verify() statements work in Test Sequence and Test Assessment blocks,

Copyright 2019 The MathWorks, Inc.



Copyright 2016-2020 The MathWorks, Inc.

2. Run the model.

```
s = sim('sltestRollRefTestExample');
```

3. Get the assessment set.

```
as = sltest.getAssessments('sltestRollRefTestExample');
```

4. Get assessment 3 from the assessment set.

```
as3 = get(as,3);
```

Display Results of the Assessment Set and Assessment Result

1. Get summary of the assessment set.

```
asSummary = getSummary(as)
```

```
asSummary =
```

```
struct with fields:
```

```
Total: 6
Untested: 3
Passed: 2
Failed: 1
Result: Fail
```

2. Display the result of assessment 3.


```
disp(as3)
```

```
sltest.Assessment
Package: sltest

Properties:
  Name: 'Simulink:verify_high'
  BlockPath: [1x1 Simulink.SimulationData.BlockPath]
  Values: [1x1 timeseries]
  Result: Fail
```

3. Find untested or failed results in the assessment set.

```
asFailUntested = find(as, 'Result', slTestResult.Fail, '-or', ...
    'Result', slTestResult.Untested)
```

```
asFailUntested =
```

```
sltest.AssessmentSet
Summary:
  Total: 4
  Untested: 3
  Passed: 0
  Failed: 1
  Result: Fail
```

```
Untested Assessments (first 10):
  2 : Untested 'Simulink:verify_high'
  3 : Untested 'Simulink:verifyTKLow'
  4 : Untested 'Simulink:verifyTKNormal'
```

```
Failed Assessments (first 10):
  1 : Fail 'Simulink:verify_high'
```

4. Find assessments under the Test Assessment block, using a regular expression.

```
assessBlock = find(as, '-regexp', 'BlockPath', '.[Aa]ssess')
```

```
assessBlock =
```

```
sltest.AssessmentSet
Summary:
  Total: 6
  Untested: 3
  Passed: 2
  Failed: 1
  Result: Fail
```

```
Untested Assessments (first 10):
  4 : Untested 'Simulink:verify_high'
  5 : Untested 'Simulink:verifyTKLow'
  6 : Untested 'Simulink:verifyTKNormal'
```

```
Passed Assessments (first 10):
```

```
1 : Pass 'Simulink:verify_normal'  
2 : Pass 'Simulink:verify_low'
```

```
Failed Assessments (first 10):  
3 : Fail 'Simulink:verify_high'
```

Re-enable warnings

warning on [Stateflow:Runtime:TestVerificationFailed](#)

See Also

`sltest.Assessment` | `sltest.AssessmentSet`

Topics

“Examine Model Verification Results by Using Simulation Data Inspector”

Introduced in R2016b

sltest.harness.check

Compare component under test between harness model and main model

Syntax

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

Description

`[CheckResult,CheckDetails] = 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, returning the overall `CheckResult` and additional `CheckDetails` of the comparison. You cannot use `sltest.harness.check` on Subsystem models test harnesses.

Examples

Compare Component Under Test Between Model and Harness

This example shows how to compare a component under test between the main model and the test harness. Comparing the component under test can help you determine if the CUT contains unsynchronized changes.

Check the Controller subsystem in the f14 model against the Controller subsystem in a test harness.

1. Load the model.

```
load_system('f14');
```

2. Create a test harness for Controller.

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

3. Run the comparison.

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

4. View the overall result.

CheckResult

```
CheckResult = logical
    1
```

5. View the details of the comparison.

CheckDetails

```
CheckDetails = struct with fields:
    overall: 1
```

```
    contents: 1
    reason: 'The checksum of the harnessed component and the component in the main model are s

clear('CheckResult','CheckDetails');
close_system('f14',0);
```

Input Arguments

harnessOwner — Model or component

character vector | double

Model or component handle or path, specified as a character vector or double.

Example: 1.9500e+03

Example: 'model_name'

Example: 'model_name/Subsystem'

harnessName — Harness name

character vector

The name of the harness, specified as a character vector.

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. Structure fields contain the comparison results for the overall component, the component contents, the component interface, and a reason for the comparison result. If `sltest.harness.check` returns false, rebuild the test harness and retry `sltest.harness.check`.

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)
sltest.harness.clone(HarnessOwner, HarnessName, Name, Value)
```

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.

`sltest.harness.clone(HarnessOwner, HarnessName, Name, Value)` clones the test harness `HarnessName` associated with `HarnessOwner` using additional options specified by one or more `Name, Value` pair arguments.

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', 'SynchronizationMode', ...
    'SyncOnOpenAndClose')
sltest.harness.clone('f14/Controller', ...
    'ControllerHarness1', 'ControllerHarness2')
```

Clone the test harness `ControllerHarness1` created in the previous step to the `Aircraft Dynamics Model` subsystem and save it as `ControllerHarnessClone`.

```
sltest.harness.clone('f14/Controller', ...
    'ControllerHarness1', 'DestinationOwner', ...
    'f14/Aircraft Dynamics Model', 'Name', ...
    'ControllerHarnessClone')
```

Input Arguments

HarnessOwner — Model or component

character vector | double

Model or component handle or path, specified as a character vector or a double.

Example: 1.9500e+03

Example: 'f14'

Example: 'f14/Controller'

HarnessName — Source harness name

character vector

The name of the source harness, specified as a character vector.

Example: 'ControllerHarness'

NewHarness — Cloned harness name

character vector

The name of the cloned harness, specified as a character vector. If no value is specified, a default value is automatically generated.

Example: 'ControllerHarness2'

Name-Value Pair Arguments

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

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

Example: 'DestinationOwner', 'model3/Controller3', 'Name', 'newClonedHarness'

DestinationOwner — Owner block to which the harness is cloned

harnessOwner (default) | character vector

Owner block to which the test harness is cloned, specified as the comma-separated pair consisting of 'DestinationOwner' and a character vector.

Example: 'DestinationOwner', 'model3/Controller3'

Name — Name of the cloned test harness

autogenerated name (default) | character vector

The name of the cloned test harness, specified as the comma-separated pair consisting of 'Name' and a character vector. If no value is specified for Name, a default value is automatically generated.

Example: 'Name', 'newClonedHarness'

See Also

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

Introduced in R2015b

sltest.harness.close

Close test harness

Syntax

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

Description

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

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

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

Examples

Close a Harness Associated With a Subsystem

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

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

Close a Harness Associated With a Top-level Model

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

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

Input Arguments

modelName — Model name

character vector | double

Model handle or path, specified as a character vector or double.

Example: 1.9500e+03

Example: 'model_name'

harnessOwner — Model or component name

character vector | double

Model or component handle or path, specified as a character vector or double.

Example: 1.9500e+03

Example: 'model_name'

Example: 'model_name/Subsystem'

harnessName — Harness name

character vector

The name of the harness, specified as a character vector.

Example: 'harness_name'

See Also

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

Introduced in R2015a

sltest.harness.convert

Convert test harnesses between internal and external storage

Syntax

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

Description

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

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

Examples

Convert Test Harnesses from External to Internal

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

Input Arguments

modelName — Model name

character vector

Model handle or path, specified as a character vector.

Example: 'model_name'

conversion — Conversion type

'InternalToExternal' | 'ExternalToInternal'

Conversion to perform, specified as a character vector.

Example: 'InternalToExternal'

See Also

`sltest.harness.check` | `sltest.harness.close` | `sltest.harness.create` |
`sltest.harness.delete` | `sltest.harness.export` | `sltest.harness.open` | Model
Reference Conversion Advisor

Introduced in R2016a

sltest.harness.create

Create test harness

Syntax

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

Description

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

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

Examples

Create Harness for a Model

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

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

Create Harness for a Subsystem

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

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

Create a Harness That Uses a Chart Scheduler

Create a harness that uses a Stateflow Chart as the test harness scheduler for the Controller subsystem of the `f14` model.

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

Input Arguments

harnessOwner — Model or component

character vector | double

Model or component handle or path, specified as a character vector or double.

Example: 1.9500e+03

Example: 'model_name'

Example: 'model_name/Subsystem'

Data Types: double | char

Name-Value Pair Arguments

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

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

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

Name — Harness name

character vector

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

Example: 'Name', 'harness_name'

Data Types: char

Description — Harness description

character vector

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

Example: 'Description', 'A test harness'

Data Types: char

Source — Component under test input

'Inport' (default) | 'Signal Editor' | 'From Workspace' | 'From File' | 'Test Sequence' | 'Signal Editor' | 'Chart' | '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 Editor'

Data Types: char

CustomSourcePath — Path to library block for custom source

character vector

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'

Data Types: char

Sink — Harness output

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

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

If your test harness contains a To Workspace block, the block variable is not saved in the base workspace after the test finishes running. Upon test completion, the base workspace is restored to its original state.

Example: 'Sink', 'Scope'

Data Types: char

CustomSinkPath — Path to library block for custom sink

character vector

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/Display'

SeparateAssessment — Separate Test Assessment block

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.

Example: 'SeparateAssessment', true

Data Types: logical

SynchronizationMode — Specifies the synchronization behavior of the component under test

'SyncOnOpenAndClose' (default) | 'SyncOnOpen' | 'SyncOnPushRebuildOnly'

Option to specify when the component under test synchronizes the main model and the test harness. Subsystem model test harnesses are always synchronized with their underlying model.

- 'SyncOnOpenAndClose' rebuilds the component under test from the main model when the test harness opens, and pushes changes from the component under test to the main model when the test harness closes.
- 'SyncOnOpen' rebuilds the component under test from the main model when the test harness opens. It does not push changes from the component under test to the main model when the test harness closes.
- 'SyncOnPushRebuildOnly' rebuilds and pushes changes only when you manually initiate rebuild or push for the entire test harness. For more information, see "Synchronize Changes Between Test Harness and Model".

Example: 'SynchronizationMode', 'SyncOnOpen'

Data Types: char

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. This option is set to true for Subsystem model test harnesses.

false compiles the model and runs other operations to support the harness build. true creates the harness without model compilation.

Example: 'CreateWithoutCompile',false

Data Types: logical

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'`

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. This option is set to normal for Subsystem models.

Example: 'VerificationMode', 'SIL'

Data Types: char

ExistingBuildFolder — Path to folder of existing generated code verified using SIL/PIL`string | character vector`

Path to main build folder of existing generated code verified using SIL/PIL, specified as a string or character vector. If you specify a build folder, the existing code in that folder is used, which enables faster harness creation time. If you do not specify a build folder, the code is regenerated. You cannot set a default value for this property in an `sl_customization` file or using `sltest.harness.setHarnessCreateDefaults`.

Example: 'ExistingBuildFolder', 'C:\TestMdl\SILHarness\Amplifier_ert_rtw'

Data Types: string | char

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

Data Types: logical

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. This option is set to true for Subsystem model test harnesses.

Example: 'RebuildModelData',true

Data Types: `logical`

SaveExternally — Test harnesses saved as separate SLX files

`false` (default) | `true`

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

Example: 'SaveExternally',`true`

Data Types: `logical`

HarnessPath — Path to external test harness file

character vector

If 'SaveExternally' is specified, you can specify a location for the external harness SLX file using a comma-separated pair consisting of 'HarnessPath' and a character vector.

Example: 'HarnessPath', 'C:\MATLAB\SafetyTests'

Data Types: `char`

LogOutputs — Log all harness output signals

`false` (default) | `true`

Log all harness output signals for the component under test, specified as `false` or `true`. When the value is `true`, all signals are logged, except for signals incompatible with logging. Signals are logged during test case execution and return test results. If an output signal does not have a name or a propagated name, it is assigned one in the harness using the format `<component under test name>:<output port number>`. To remove a signal from being logged, you can remove its badge manually.

Example: 'LogOutputs',`true`

Data Types: `logical`

PostCreateCallback — Harness customization after creation

`string` | character vector | cell array of strings | array of character vectors

Use one or more post create callback functions to customize a test harness. The post create callback functions execute after the harness is created. To specify more than one callback, separate them with a comma. The callbacks execute in the order in which you list them. For more information, see “Customize Test Harnesses”.

Example: "PostCreateCallback", "HarnessCallback"

Example: "PostCreateCallback", {"HarnessCallback", "SecondCallback"}

Example: "PostCreateCallback", ["HarnessCallback", "SecondCallback"]

Data Types: `string` | `char`

PostRebuildCallback — Harness customization after rebuild

character vector

Use a post rebuild callback function to customize a test harness. The post rebuild callback function executes after the harness rebuild. For more information, see “Customize Test Harnesses”.

Example: 'PostRebuildCallback','HarnessCustomization'

Data Types: char

ScheduleInitTermReset — Drive model initialize, reset, and terminate ports

false (default) | true

Option to drive model initialize, reset, and terminate ports with the chosen test harness source, specified as the comma-separated pair consisting of 'ScheduleInitTermReset' and false or true. This option only applies to harnesses created for a block diagram. This option is set to false for Subsystem models.

Example: 'ScheduleInitTermReset',true

Data Types: logical

SchedulerBlock — Include scheduler block for periodic signals and function calls

'Test Sequence' | 'MATLAB Function' | 'Chart' | 'None'

Option to include a scheduler block in the test harness, specified as the comma-separated pair consisting of 'SchedulerBlock' and the type of block to use. The block is included if the test harness is created for a model block diagram or a Model block and contains function calls or periodic event ports. To include no scheduler block and connect all ports to harness source blocks, use 'None' .

Example: 'SchedulerBlock','Test Sequence'

Example: 'SchedulerBlock','None'

Data Types: char

AutoShapeInputs — Match scalar and double value source to input signal dimension

false (default) | true

Option to shape scalar and double values to match the dimension of the input signals to the component under test, specified as the comma-separated pair consisting of 'AutoShapeInputs' and false or true. This option only applies to harnesses with Inport, Constant, Signal Editor, From Workspace, or From File blocks. This option is set to false for Subsystem models.

Example: 'AutoShapeInputs',true

Data Types: logical

FunctionInterfaceName — Name of reusable library subsystem function interface

string | character vector

Name of reusable library subsystem function interface, specified as a string or character vector. The test harness is created for the function interface of the reusable library subsystem. You cannot set a default value for this property in an `sl_customization.m` file or by using `sltest.harness.setHarnessCreateDefaults`.

Example: 'FunctionInterfaceName','double_RLS'

Data Types: string | char

Compatibility Considerations

DriveFcnCallWithTestSequence in `sltest.harness.create` is not recommended

Not recommended starting in R2018b

Starting with the R2018b release, you can use the 'SchedulerBlock' option to include a scheduler block when creating a test harness. The name-value pair 'SchedulerBlock', 'Test Sequence' uses a Test Sequence scheduler block and replaces 'DriveFcnCallWithTestSequence', true.

'SchedulerBlock' provides more scheduler options, and creates a simplified block interface compared to 'DriveFcnCallWithTestSequence'. To update your code, for instances of `sltest.harness.create`, replace 'DriveFcnCallWithTestSequence', true with 'SchedulerBlock', 'Test Sequence'.

See Also

`sltest.harness.check` | `sltest.harness.clone` | `sltest.harness.close` |
`sltest.harness.convert` | `sltest.harness.delete` | `sltest.harness.export` |
`sltest.harness.find` | `sltest.harness.load` | `sltest.harness.open` |
`sltest.harness.set`

Introduced in R2015a

sltest.harness.delete

Delete test harness

Syntax

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

Description

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

Examples

Delete a Harness Associated With a Subsystem

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

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

Delete a Harness Associated With a Top-level Model

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

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

Input Arguments

harnessOwner — Model or component

character vector | double

Model or component handle or path, specified as a character vector or double.

Example: 1.9500e+03

Example: 'model_name'

Example: 'model_name/Subsystem'

harnessName — Harness name

character vector

The name of the harness, specified as a character vector.

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

Input Arguments

harnessOwner — Model or component

character vector | double

Model or component handle or path, specified as a character vector or double

Example: `1.9500e+03`

Example: `'model_name'`

Example: `'model_name/Subsystem'`

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

character vector

The name of the harness, specified as a character vector.

Example: `'harness_name'`

modelName — Name of the new model

character vector

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

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.import | sltest.harness.load | sltest.harness.open |
sltest.harness.push | sltest.harness.rebuild | sltest.harness.set

Introduced in R2015a

sltest.harness.find

Find test harnesses in model

Syntax

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

Description

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

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

Examples

Use RegEx to Find Harnesses for a Model Component

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

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

```
harnessList =
```

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

```
    model
    name
    description
    type
    ownerHandle
    ownerFullPath
    ownerType
    isOpen
    canBeOpened
    verificationMode
    saveExternally
    rebuildOnOpen
    rebuildModelData
    graphical
```

```
origSrc
origSink
```

Input Arguments

harnessOwner — Model or component

character vector | double

Model or component handle or path, specified as a character vector or double

Example: 1.9500e+03

Example: 'model_name'

Example: 'model_name/Subsystem'

Name-Value Pair Options

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

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

Example: '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

character vector | regular expression

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

Example: 'Name','sample_harness' 'Name','_[Hh]arnes+'

RegExp — Ability to search using a regular expression

'off' (default) | 'on'

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

Example: 'RegExp','on'

SearchDepth — Subsystem levels to search

all levels (default) | nonnegative integer

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

0 searches `harnessOwner`.

1 searches `harnessOwner` and its subsystems.

2 searches `harnessOwner`, its subsystems, and their subsystems.

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

Example: 'SearchDepth',1

OpenOnly – Search option for open harnesses

'off' (default) | 'on'

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

Example: 'OpenOnly', 'on'

FunctionInterfaceName – Name of the function interface to search for

string | character vector

Name of the function interface for the reusable library subsystem to search for, specified as a string or character vector. Using this Name-Value pair returns a list of all harnesses for the specified function interface.

Example: 'FunctionInterfaceName', 'double_RLS'

Output Arguments**harnessList – List of harnesses and properties**

structure

List of harnesses and properties for the component or model harnessOwner, returned as a structure. The structure fields are shown in the “Use RegExp to Find Harnesses for a Model Component” on page 1-42 example.

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.getHarnessCreateDefaults

Get harness creation default values

Syntax

```
sltest.harness.getHarnessCreateDefaults
```

Description

`sltest.harness.getHarnessCreateDefaults` returns a structure that contains the current default property values for creating new test harnesses. You can customize the default values by using `sltest.harness.setHarnessCreateDefaults` or by creating an `sl_customization` file "Customize Test Harness Creation Default Property Values".

Examples

Display Harness Creation Default Values

Change the default property values for two harness creation options and display the current default values.

```
sltest.harness.setHarnessCreateDefaults(SynchronizationMode=...
    "SyncOnPushRebuildOnly", Sink = "Scope");
sltest.harness.getHarnessCreateDefaults
```

ans =

```
struct with fields:
    Name: ""
    PostCreateCallback: ""
    SaveExternally: 0
    LogOutputs: 0
    Source: "Inport"
    Sink: "Scope"
    Description: ""
    SeparateAssessment: 0
    SynchronizationMode: "SyncOnPushRebuildOnly"
    CreateWithoutCompile: 0
    RebuildOnOpen: 0
    RebuildModelData: 0
    HarnessPath: ""
    PostRebuildCallback: ""
    ScheduleInitTermReset: 0
    SchedulerBlock: "Test Sequence"
    AutoShapeInputs: 0
    CustomSourcePath: ""
```

```
CustomSinkPath: ""  
VerificationMode: "Normal"
```

See Also

`sltest.harness.setHarnessCreateDefaults` | `sltest.harness.create`

Topics

“Customize Test Harness Creation Default Property Values”

Introduced in R2021b

sltest.harness.import

Import Simulink model to test harness

Syntax

```
sltest.harness.import(harnessOwner, 'ImportFileName',
importModel, 'ComponentName', TestedComponent)
sltest.harness.import(harnessOwner, 'ImportFileName',
importModel, 'ComponentName', TestedComponent, Name, Value)
```

Description

`sltest.harness.import(harnessOwner, 'ImportFileName', importModel, 'ComponentName', TestedComponent)` creates a test harness from the Simulink model `importModel`, with a default harness name, associated with `harnessOwner`, with `TestedComponent` the harness component under test.

`sltest.harness.import(harnessOwner, 'ImportFileName', importModel, 'ComponentName', TestedComponent, Name, Value)` uses additional `Name, Value` arguments to specify test harness properties.

Examples

Programmatically Create a Test Harness from a Standalone Model

This example shows how to use `sltest.harness.import` to create a test harness by importing a standalone verification model. You create a test harness for a basic cruise control subsystem.

The standalone model contains a Signal Builder block driving a copy of the `Controller` subsystem, with a subsystem verifying that the throttle output goes to 0 if the brake is applied for three consecutive time steps.

```
mainModel = 'sltestBasicCruiseControl';
harnessModel = 'sltestBasicCruiseControlHarnessModel';
```

1. Load the main model.

```
load_system(mainModel)
```

2. Create a test harness from the standalone model. Create the harness for subsystem `Controller` in the main model, with `Controller` the harness component under test.

```
sltest.harness.import([mainModel '/Controller'], 'ImportFileName', harnessModel, ...
    'ComponentName', [harnessModel '/Controller'], 'Name', ...
    'VerificationSubsystemHarness')
```

3. Return the properties of the new test harness.

```
testHarnessProperties = sltest.harness.find([mainModel '/Controller'])
```

```
testHarnessProperties=1x2 struct array with fields:
    model
    name
    description
    type
    ownerHandle
    ownerFullPath
    ownerType
    isOpen
    canBeOpened
    verificationMode
    saveExternally
    rebuildOnOpen
    rebuildModelData
    postRebuildCallback
    graphical
    origSrc
    origSink
    synchronizationMode
    existingBuildFolder
    functionInterfaceName
    :

close_system(mainModel,0)
```

Input Arguments

harnessOwner — Model or component

character vector | double

Model or component handle or path, specified as a character vector or double

Example: 1.9500e+03

Example: 'model_name'

Example: 'model_name/Subsystem'

importModel — File path

character vector

Path to the standalone model to import as a test harness

Example: 'C:\MATLAB\sltestBasicCruiseControlTestModel'

TestedComponent — Tested component in standalone model

character vector

The name or path and name of the tested component in the standalone model. After import, this component is linked to the harnessOwner component in the main model.

Example: 'Controller'

Name-Value Pair Arguments

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

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

Example: `'Name', 'harness_name', 'RebuildOnOpen', true`

Name — Harness name

character vector

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'`

SynchronizationMode — Specifies the synchronization behavior of the component under test

`'SyncOnOpenAndClose'` (default) | `'SyncOnOpen'` | `'SyncOnPushRebuildOnly'`

Option to specify when the component under test synchronizes the main model and the test harness. Subsystem model test harnesses are always synchronized with their underlying model.

- `'SyncOnOpenAndClose'` rebuilds the component under test from the main model when the test harness opens, and pushes changes from the component under test to the main model when the test harness closes.
- `'SyncOnOpen'` rebuilds the component under test from the main model when the test harness opens. It does not push changes from the component under test to the main model when the test harness closes.
- `'SyncOnPushRebuildOnly'` rebuilds and pushes changes only when you manually initiate rebuild or push for the entire test harness. For more information, see “Synchronize Changes Between Test Harness and Model”.

Example: `'SynchronizationMode', 'SyncOnOpen'`

Data Types: `char`

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`

Data Types: `logical`

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`. This option is set to `true` for Subsystem model test harnesses.

Example: 'RebuildModelData',true

Data Types: logical

SaveExternally — Test harnesses saved as separate SLX files

false (default) | true

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

Example: 'SaveExternally',true

Data Types: logical

HarnessPath — Path to external test harness file

character vector

If 'SaveExternally' is specified, you can specify a location for the external harness SLX file using a comma-separated pair consisting of 'HarnessPath' and a character vector.

Example: 'HarnessPath', 'C:\MATLAB\SafetyTests'

Data Types: char

See Also

sltest.harness.clone | sltest.harness.create | sltest.harness.export |
sltest.harness.push | sltest.harness.rebuild | sltest.harness.set

Topics

“Create Test Harnesses from Standalone Models”

Introduced in R2017a

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

character vector | double

Model or component handle or path, specified as a character vector or double.

Example: 1.9500e+03

Example: 'model_name'

Example: 'model_name/Subsystem'

harnessName — Harness name

character vector

The name of the harness, specified as a character vector.

Example: 'harness_name'

See Also

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

Introduced in R2015a

sltest.harness.move

Move test harness from linked instance to library block or to a different harness owner

Syntax

```
sltest.harness.move(HarnessOwner, HarnessName)
sltest.harness.move(HarnessOwner, HarnessName, NewPath)
sltest.harness.move(HarnessOwner, HarnessName, Name, Value)
```

Description

`sltest.harness.move(HarnessOwner, HarnessName)` moves the test harness `HarnessName` associated with the block `HarnessOwner` from the linked instance to its reference library block. Moving the test harness removes it from the linked instance. This command results in an error if `HarnessName` is not a linked instance.

`sltest.harness.move(HarnessOwner, HarnessName, NewPath)` moves the test harness `harnessName` associated with the block `HarnessOwner` to the destination path specified by `NewPath`.

`sltest.harness.move(HarnessOwner, HarnessName, Name, Value)` moves the test harness `HarnessName` associated with `HarnessOwner` using additional options specified by one or more `Name, Value` pairs.

Examples

Move Test Harness

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

```
% Open the model
open_system sltestHeatpumpLibraryLinkExample
% Move the test harness
sltest.harness.move('sltestHeatpumpLibraryLinkExample/Controller', ...
'Baseline_controller_tests')
```

Move the test harness `Requirements_Tests` from the linked instance of the `Controller` subsystem to the `Plant` subsystem and save it as `Requirements_Tests_Moved`.

```
sltest.harness.move...
('sltestHeatpumpLibraryLinkExample/Controller', ...
'Requirements_Tests', 'DestinationOwner', ...
'sltestHeatpumpLibraryLinkExample/Plant', ...
'Name', 'Requirements_Tests_Moved')
```

Input Arguments

HarnessOwner — Model or component name

character vector | double

Model or component handle or path, specified as a character vector or a double.

Example: 1.9500e+03

Example: 'model_name'

Example: 'model_name/Subsystem'

HarnessName — Harness name

character vector

The name of the harness, specified as a character vector.

Example: 'harness_name'

NewPath — Destination path

character vector

The destination path of the moved test harness, specified as a character vector.

Example: 'model_name/Subsystem2'

Name-Value Pair Arguments

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

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

Example: 'DestinationOwner', 'model3/Controller3', 'Name', 'newMovedHarness'

DestinationOwner — Owner block to which the harness is moved

harnessOwner (default) | character vector

Owner block to which the test harness is moved, specified as the comma-separated pair consisting of 'DestinationOwner' and a character vector.

Example: 'DestinationOwner', 'model3/Controller3'

Name — Name of moved test harness

name of the test harness (default) | character vector

The name of the moved test harness, specified as the comma-separated pair consisting of 'Name' and a character vector. If a value is not specified for Name, the name of the test harness is used by default.

Example: 'Name', 'newMovedHarness'

See Also

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

Introduced in R2016a

sltest.harness.open

Open test harness

Syntax

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

Description

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

Examples

Open a Harness Associated With a Subsystem

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

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

Open a Harness Associated With a Model

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

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

Input Arguments

harnessOwner — Model or component

character vector | double

Model or component handle or path, specified as a character vector or double.

Example: 1.9500e+03

Example: 'model_name'

Example: 'model_name/Subsystem'

harnessName — Harness name

character vector

The name of the harness, specified as a character vector.

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`. Subsystem model test harnesses always push.

Examples

Push Parameters from Harness to Model

Change the denominator value of the Stick Prefilter in the harness. Then, push the change in the `controller_harness` to the `f14` model. Notice that the parameter is updated in the model.

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

sltest.harness.open('f14/Controller','controller_harness')
set_param('controller_harness/Controller/Stick Prefilter',...
    'Denominator',[Ts,2])

blkpath = 'f14/Controller/Stick Prefilter';
disp(['Original denominator: ' get_param(blkpath,'Denominator')])

sltest.harness.push('f14/Controller','controller_harness');
disp(['Updated denominator: ' get_param(blkpath,'Denominator')])
```

Input Arguments

harnessOwner — Model or component

character vector | double

Model or component handle, or path, specified as a character vector or double

Example: 1.9500e+03

Example: 'model_name'

Example: 'model_name/Subsystem'

harnessName — Harness name

character vector

The name of the harness, specified as a character vector.

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. If you specified to use existing generated code for a SIL/PIL subsystem using `sltest.harness.create` or `sltest.harness.set`, the harness rebuild uses that code instead of regenerating it. Subsystem model test harnesses always rebuild.

Examples

Rebuild Parameters from Model to Harness

Change the denominator value of the Stick Prefilter in the main model. Then, rebuild the `controller_harness`, which is associated with the `Controller` subsystem in the `f14` model. Notice that the parameter is updated in the harness.

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

set_param('f14/Controller/Stick Prefilter',...
    'Denominator','[Ts,2]')

sltest.harness.open('f14/Controller','controller_harness')
blkpath = 'controller_harness/Controller/Stick Prefilter';
disp(['Original denominator: ' get_param(blkpath,'Denominator')])

sltest.harness.rebuild('f14/Controller','controller_harness');
disp(['Updated denominator: ' get_param(blkpath,'Denominator')])
```

Input Arguments

harnessOwner — Model or component

character vector | double

Model or component handle, or path, specified as a character vector or double

Example: 1.9500e+03

Example: 'model_name'

Example: 'model_name/Subsystem'

harnessName — Harness name

character vector

The name of the harness, specified as a character vector.

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 the Name of a Test Harness

This example shows how to change the name of a test harness using `sltest.harness.set`.

Create a Test Harness

Load the `f14` model and create a test harness for the `Controller` subsystem.

```
load_system('f14')
sltest.harness.create('f14/Controller', 'Name', 'Harness1')
```

Change the Test Harness Name

Change the name from `Harness1` to `ControllerHarness`.

```
sltest.harness.set('f14/Controller', 'Harness1', 'Name', 'ControllerHarness')
```

Close the Model

```
close_system('f14', 0)
```

Input Arguments

harnessOwner — Model or component

character vector | double

Model or component handle, or path, specified as a character vector or double

Example: `1.9500e+03`

Example: `'model_name'`

Example: `'model_name/Subsystem'`

harnessName — Harness name

character vector

The name of the harness, specified as a character vector.

Example: 'harness_name'

Name-Value Pair Options

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

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

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

Name — New harness name

character vector

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

Example: 'Name', 'new_harness_name'

Description — New harness description

character vector

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

Example: 'Description', 'An updated test harness'

SynchronizationMode — Specifies the synchronization behavior of the component under test

'SyncOnOpenAndClose' (default) | 'SyncOnOpen' | 'SyncOnPushRebuildOnly'

Option to specify when the component under test synchronizes the main model and the test harness. Subsystem model test harnesses are always synchronized with their underlying model.

- 'SyncOnOpenAndClose' rebuilds the component under test from the main model when the test harness opens, and pushes changes from the component under test to the main model when the test harness closes.
- 'SyncOnOpen' rebuilds the component under test from the main model when the test harness opens. It does not push changes from the component under test to the main model when the test harness closes.
- 'SyncOnPushRebuildOnly' rebuilds and pushes changes only when you manually initiate rebuild or push for the entire test harness. For more information, see "Synchronize Changes Between Test Harness and Model".

Example: 'SynchronizationMode', 'SyncOnOpen'

Data Types: char

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

Data Types: logical

ExistingBuildFolder — Path to folder of existing generated code verified using SIL/PIL

string | character vector

Path to main build folder of existing generated code verified using SIL/PIL, specified as a string or character vector. If you specify a build folder, the existing code in that folder is used, which enables faster harness creation time. If you do not specify a build folder, the code is regenerated. You cannot set a default value for this property in an `sl_customization` file or using `sltest.harness.setHarnessCreateDefaults`.

Example: 'ExistingBuildFolder','C:\TestMdl\SILHarness\Amplifier_ert_rtw'

Data Types: string | char

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. This option is set to true for Subsystem model test harnesses.

Example: 'RebuildModelData',true

Data Types: logical

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

PostRebuildCallback — Harness customization after rebuild

character vector

Use a post rebuild callback function to customize a test harness. The post rebuild callback function executes after the harness rebuild. For more information, see “Customize Test Harnesses”.

Example: 'PostRebuildCallback','HarnessCustomization'

FunctionInterfaceName — Name of the function interface to associate to the harness

string | character vector

Name of the function interface to associate with the harness, specified as a string or character vector. This option applies only to reusable library components with function interfaces.

Example: 'FunctionInterfaceName','double_RLS'

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.harness.setHarnessCreateDefaults

Package: sltest.harness

Customize default property values for test harness creation

Syntax

```
sltest.harness.setHarnessCreateDefaults(name = value)
```

Description

`sltest.harness.setHarnessCreateDefaults(name = value)` sets and registers the specified harness creation default values for the specified name-value arguments.

For a list of valid name-value arguments, see `sltest.harness.create`.

As an alternative to using `sltest.harness.setHarnessCreateDefaults`, you can use an `sl_customization.m` file to customize harness creation default values. See “Customize Test Harness Creation Default Property Values”.

If you created an `sl_customization.m` file, you can then use `sltest.harness.setHarnessCreateDefaults` to set additional default values or overwrite the values defined in that file. If you create an `sl_customization.m` file and register it after using `sltest.harness.setHarnessCreateDefaults`, the default values that you previously specified are overwritten with the default values in the file.

Examples

Set Default Value to Save Harness Externally

```
sltest.harness.setHarnessCreateDefaults(SaveExternally = true)
```

See Also

`sltest.harness.getHarnessCreateDefaults` | `sltest.harness.create`

Introduced in R2021b

sltest.harness.showDialog

Package: sltest

Show test harness dialog box

Syntax

```
sltest.harness.showDialog(dialogtype,systemundertest)
```

Description

`sltest.harness.showDialog(dialogtype,systemundertest)` opens a test harness dialog box of the specified type for the specified system under test.

Example

Examples

Open the Create Test Harness Dialog Box

Open the Create Test Harness dialog box for the Controller component of `my_model`.

```
sltest.harness.showDialog('HarnessCreate','my_model/Controller')
```

Input Arguments

dialogtype — Type of test harness dialog box

'HarnessCreate' | 'HarnessManager' | 'HarnessImport'

Type of test harness dialog box specified as one of these values:

- 'HarnessCreate' — Create Test Harness dialog box
- 'HarnessManager' — Manage Test Harness dialog box
- 'HarnessImport' — Import Test Harness dialog box

systemundertest — Simulink model or component being tested

character vector | string | handle

Simulink model or component being tested, specified as a character vector or string of the path to the system under test, or as the handle of the system under test.

See Also

`sltest.harness.create` | `sltest.harness.import` | `sltest.harness.set` | `sltest.harness.push` | `sltest.harness.rebuild` | `sltest.harness.convert`

Topics

“Create Test Harnesses and Select Properties”

“Manage Test Harnesses”

Introduced in R2018b

sltest.import.sldvData

Create test cases from Simulink Design Verifier results

Syntax

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

Description

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

[owner, testHarness, testFile, testCase] = sltest.import.sldvData(dataFile, Name, Value) creates a test harness and test file with additional options specified by one or more Name, Value pair arguments. Specify name-value pair arguments after the data file input argument.

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...
    ('./sldv_output/ShiftLogic/ShiftLogic_sldvdata.mat', ...
    'TestHarnessName', 'CoverageHarness', ...
    'TestFileName', 'CoverageTests')
```

Open the test harness.

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

Open the test file.

```
open(testfile)
```


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)
```

Input Arguments

dataFile — Data file full path name

character vector | string scalar

Path and file name of the data file generated by Simulink Design Verifier analysis, specified as a character vector or string scalar. The input file is a MAT file. When the file is imported into Simulink Test, `sldvData` creates an MLDATX file, and an Excel® file at the location specified in `ExcelFilePath`. If the Excel file already exists, a new sheet is added to the file.

Example: `'ShiftLogic0/ShiftLogic0_sldvdata.mat'`

Name-Value Pair Options

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

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

Example: `'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

character vector | string scalar

The test harness used for running the test cases, specified as the comma-separated pair consisting of `'TestHarnessName'` and the name of a test harness.

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

Example: `'TestHarnessName', 'ModelCoverageTestHarness'`

TestHarnessSource — Source of the new test harness

'Inport' (default) | 'Signal Editor'

The source of the new test harness, specified as the comma-separated pair consisting of `'TestHarnessSource'` and `'Inport'` or `'Signal Editor'`.

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

character vector | string scalar

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

Example: `'TestFileName', 'ModelCoverageTests'`

ExtractedModelPath — Path of extracted model

character vector | string scalar

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

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/'`

TestCase — Test case

character vector | string scalar

The test case to reuse for import operation, specified as the comma-separated pair consisting of `'TestCase'` and the name of the test case. Do not specify any other name-value pair when you use this option.

Example: 'TestCase', 'ModelCoverageTest2'

ExcelFilePath — Path of Excel file

character vector | string scalar

Path of the Excel file, specified as a character vector or string scalar.

Example: 'TestCase', 'ModelCoverageTest2'

Output Arguments**owner — Path of the component under test**

character vector | string scalar

Path of the component under test, returned as a character vector

Example: 'ShiftLogic0/ShiftLogic0_sldvdata'

testHarness — Test harness name

character vector

Name of the test harness for running the test cases, returned as a character vector.

testFile — Test file name

character vector

Name of the test file created or updated using the test cases, returned as a character vector.

testCase — Test case name

character vector | string scalar

Name of the newly created or updated test case, returned as a character vector.

See Also

createInputDataFile

Topics

“Increase Coverage by Generating Test Inputs”

Introduced in R2015b

sltest.testmanager.addTestsForMissingCoverage

Package: sltest.testmanager

Add tests to increase coverage

Syntax

```
tc = sltest.testmanager.addTestsForMissingCoverage(testOpts,cvData)
```

Description

`tc = sltest.testmanager.addTestsForMissingCoverage(testOpts,cvData)` creates a `TestCase` object that increases the test coverage in `cvData` using the test options specified by the `TestOptions` object, `testOpts`. You must have a Simulink Design Verifier license to use this function.

Examples

Add Tests for Missing Coverage to Existing Test Case

This example shows how increase coverage when coverage results for a test case are lower than 100%.

Get the existing test case from the test file and then run the test and get the coverage data. Create a `TestOptions` object and add missing coverage to the coverage data for new test case. Run the new test case and open the Test Manager.

```
load_system("CruiseControlAddCov.slx");

tf = sltest.testmanager.TestFile('CruiseControlAddCov.mldatx');
ts = getTestSuites(tf);
testcaseObj = getTestCases(ts);

tr = run(testcaseObj);
cvdata = getCoverageResults(tr);

testOpts = sltest.testmanager.TestOptions(testcaseObj);
newTestCaseObj = sltest.testmanager.addTestsForMissingCoverage...
    (testOpts,cvdata);
```

```
Warning: Non-scalar in Uniform output, at index 3, output 1.
Set 'UniformOutput' to false.
```

```
Warning: Non-scalar in Uniform output, at index 3, output 1.
Set 'UniformOutput' to false.
```

```
tr = run(newTestCaseObj);

sltest.testmanager.view;
```

In the **Results and Artifacts** panel in the Test Manager, select the first Results. In the **Aggregated Coverage Results** section, the coverage is less than 100%.

Now, select the second Results. In the **Aggregated Coverage Results** section, the coverage increased to 100%.

Clear and close the Test Manager.

```
sltest.testmanager.clear
sltest.testmanager.clearResults
sltest.testmanager.close
```

Input Arguments

testOpts — Options to use when adding for adding missing coverage

TestOptions object

Options to use when adding coverage to a test case, specified as a `sltest.testmanager.TestOptions` object.

cvData — Coverage data

string

Coverage data, specified as the path to the `cvdata` object from the test with missing coverage. For example, suppose that you have a test case, `tc`, that you run to get the coverage data from the test results, `tr`. Use `getCoverageResults` to get the coverage data:

.

```
tr = tc.run();
cvdata = tr.getCoverageResults;
```

Output Arguments

tc — Test case

`sltest.testmanager.TestCase` object

Test case to use to collect missing coverage, returned as an `sltest.testmanager.TestCase` object.

See Also

`sltest.testmanager.TestOptions`

Topics

“Collect Coverage in Tests”

“Increase Test Coverage for a Model”

“Perform Functional Testing and Analyze Test Coverage”

Introduced in R2022a

sltest.testmanager.clear

Clear test files from the Test Manager

Syntax

```
sltest.testmanager.clear
```

Description

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

Examples

Clear Test File from Test Manager

```
% Create test file, test suite, and test case structure
tf = sltest.testmanager.TestFile('API Test File');
ts = createTestSuite(tf, 'API Test Suite');
tc = createTestCase(ts, 'simulation', 'Simulation Test Case');

% Clear test file from test manager
sltest.testmanager.clear;
```

See Also

`sltest.testmanager.view` | `sltest.testmanager.close`

Topics

“Create and Run Test Cases with Scripts”

Introduced in R2015a

sltest.testmanager.clearResults

Clear results from Test Manager

Syntax

```
sltest.testmanager.clearResults
```

Description

`sltest.testmanager.clearResults` clears all results data from the Test Manager **Results and Artifacts** pane.

Examples

Clear Results From Test Manager

```
% Run test files in Test Manager  
sltest.testmanager.run;
```

```
% Clear results from Test Manager  
sltest.testmanager.clearResults
```

See Also

```
sltest.testmanager.clear
```

Topics

“Create and Run Test Cases with Scripts”

Introduced in R2016a

sltest.testmanager.close

Close the Simulink Test Manager

Syntax

```
sltest.testmanager.close
```

Description

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

Examples

Clear Test File and Close Test Manager

```
% Create test file, test suite, and test case structure
tf = sltest.testmanager.TestFile('API Test File');
ts = createTestSuite(tf, 'API Test Suite');
tc = createTestCase(ts, 'simulation', 'Simulation Test Case');

% Clear test file from Test Manager
sltest.testmanager.clear;

% Close Test Manager
sltest.testmanager.close;
```

See Also

`sltest.testmanager.view`

Topics

“Create and Run Test Cases with Scripts”

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
```

## Input Arguments

### srcObjArray — Test case or test suites to copy

object array

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

**targetObj** – Target test file or test suite

object

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

**Output Arguments****objArray** – Test cases or test suites at new location

object array

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

**See Also**

`sltest.testmanager.moveTests`

**Topics**

“Create and Run Test Cases with Scripts”

**Introduced in R2015b**

# sltest.testmanager.createTestForComponent

Create test cases and test harnesses for model or model components

## Syntax

```
[tc,status] = sltest.testmanager.createTestForComponent('TestFile',
tf,'Component',component)
[tc,status] = sltest.testmanager.createTestForComponent('TestFile',
tf,'TopModel',topModel, 'Component',component)
[tc,status] = sltest.testmanager.createTestForComponent(____,Name,Value)
```

## Description

`[tc,status] = sltest.testmanager.createTestForComponent('TestFile', tf, 'Component', component)` creates a test case and a test harness for the specified component in the test file or test suite, `tf`. The component must be a single component and the test file `tf` must already exist. If the test file does not exist, you can include the `CreateTestFile`, `true` Name-Value argument to create one. The returned `status` indicates whether the test case `tc` was created successfully.

`[tc,status] = sltest.testmanager.createTestForComponent('TestFile', tf, 'TopModel', topModel, 'Component', component)` creates a test case and test harness for each component specified in the `component` array. If a test case was not created successfully, the `tc` array contains an `MException` object for that test case. Use this syntax to create more than one test case and harness.

`[tc,status] = sltest.testmanager.createTestForComponent( ____, Name, Value)` creates a test case or test harness for each component, with additional options specified by one or more Name, Value pair arguments. Specify name-value pair arguments after the test file, component, and top model input arguments.

## Examples

### Create Baseline Test for a Model

This example shows how to create a baseline test case for a model and save the inputs in an Excel file. A test harness is created automatically.

Note that this code uses the current folder as the `ExcelFileLocation`. To save the Excel file to another location, change the `ExcelFileLocation` value to a different writable folder.

```
load_system('vdp');

tf = sltest.testmanager.TestFile('MyBaselineTestFile');

sltest.testmanager.createTestForComponent('TestFile',tf,...
 'Component','vdp',...
 'TestType','baseline',...
 'ExcelFileLocation','mybaseline_inputdata.xlsx');
```

## Create Equivalence Test for a Component

Create an equivalence (back-to-back) test case for the Controller component of the `sltestNormalSILEquivalenceExample` model.

```
load_system('sltestNormalSILEquivalenceExample');

tf = sltest.testmanager.TestFile('MyB2BTestFile');

sltest.testmanager.createTestForComponent("TestFile",tf,...
 "Component","sltestNormalSILEquivalenceExample/Controller",...
 "TestType","equivalence",...
 "Simulation1Mode","Normal",...
 "Simulation2Mode","Software-in-the-Loop (SIL)");

Starting build procedure for: Controller
Successful completion of build procedure for: Controller
Creating SIL block ...
Building with 'Microsoft Visual C++ 2019 (C)'.
MEX completed successfully.
```

Build Summary

Top model targets built:

| Model      | Action                      | Rebuild Reason                                   |
|------------|-----------------------------|--------------------------------------------------|
| Controller | Code generated and compiled | Code generation information file does not exist. |

1 of 1 models built (0 models already up to date)  
Build duration: 0h 1m 20.602s

## Set Harness Options When Creating a Test Case

This example shows how to create a test case and test harness for a model component. It also shows how to use the `HarnessOptions` property. The test case uses only a component harness and is set up with no inputs or baseline strategy. To use this test case creation strategy, the `UseComponentInputs` property of `createTestForComponent` function must be false, and the `SLDVTTestGeneration` property must be off. The `HarnessSource` property is ignored and the `FunctionInterfaceName` is deduced from the argument passed to `createTestForComponent`.

Once the test case is set up, you can specify the verification strategy for the component, such as importing test input data for the component into the created test case.

This code creates a test file and harness for the Controller component of the `f14` model. The `HarnessOptions` specify that the harness name is `myHarness`, a Signal Editor is used as the harness source, and the harness syncs with the model when the harness is opened.

```
load_system('f14');

tcObj = sltest.testmanager.createTestForComponent(...
 'CreateTestFile',true,'TestFile','MyHarnessTest', ...
```

```
'Component', 'f14/Controller', 'UseComponentInputs', ...
false, 'HarnessOptions', {'Name', 'myHarness', ...
'Source', 'Signal Editor', 'SynchronizationMode', ...
'SyncOnOpen'});
```

## Create Baseline Tests and Harnesses for Multiple Components

```
load_system("sltestCruiseControl");

[result, status] = sltest.testmanager.createTestForComponent(...
 "CreateTestFile", true, ...
 "TestFile", "myTests.mldatx", ...
 "TopModel", "sltestCruiseControl", ...
 "Component", [...
 "sltestCruiseControl/TargetSpeedThrottle", ...
 "sltestCruiseControlMode/disableCaseDetection", ...
 "sltestCruiseControl/DriverSwRequest", ...
 "sltestCruiseControl"], ...
 "CreateExcelFile", true);
```

## Create Harnesses for Multiple Components

Create only test harnesses, not test cases, for multiple components of a model.

```
load_system('sltestCruiseControl');

[result, status] = sltest.testmanager.createTestForComponent(...
 'CreateTestFile', true, ...
 'TestFile', 'myTests.mldatx', ...
 'TopModel', 'sltestCruiseControl', ...
 'Component', {...
 'sltestCruiseControl/TargetSpeedThrottle', ...
 'sltestCruiseControlMode/disableCaseDetection', ...
 'sltestCruiseControl/DriverSwRequest', ...
 'sltestCruiseControl'}, ...
 'UseComponentInputs', false, ...
 'HarnessOptions', {'SynchronizationMode', 'SyncOnOpen'});
```

## Input Arguments

### tf — Test file or suite to which to add the test cases

sltest.testmanager.TestFile object | sltest.testmanager.TestSuite object | string | character vector

Test file or test suite to which to add the test cases, specified as an `sltest.testmanager.TestFile` or `sltest.testmanager.TestSuite` object, string, or character vector. If `CreateTestFile` is `false`, the test file object must exist. If `CreateTestFile` is `true`, `TestFile` is the path of a new test file, specified as a string or character vector.

Example: 'TestFile', 'myTestFile'

**component — Component or components for which to create test cases and harnesses**

string | character vector | Simulink.BlockPath object | array | cell array

Component or components for which to create test cases and harnesses. To specify a single component, use a `Simulink.BlockPath` object or a string scalar or character vector of the path to the component. To specify more than one component, use an array of `Simulink.BlockPath` objects or a cell array of character vectors or an array of strings where each element is the path to the component. The components can be any block or block diagram in the hierarchy of the top model. You can specify a component multiple times to create multiple test cases and harnesses for that component. This input argument can be any model or block that is supported for test harness generation. See “Test Harness and Model Relationship” for a list of components for which you can build test harnesses.

Example: `'Component','sf_car/shift_logic'`Example: `'Component',{'sldvdemo/Controller','sldvdemo/Pi Controller'};`**topModel — Model name at top of hierarchy**

string | character vector

Model name at the top of the hierarchy, specified as a string or character vector. The top model is the name of the model whose hierarchy contains the components specified in the `component` input. You must specify the top model if your `component` input is an array with multiple components. Specifying the top model is optional if the `component` input is a single component. If the `component` input is a top model, do not use specify the `TopModel`.

Example: `'TopModel','Plant'`**Name-Value Pair Arguments**

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

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

Example: `'CreateTestFile',true`**CreateTestFile — Whether to create a new test file**

false (default) | true

Whether to create a new test file, specified as the comma-separated pair consisting of `'CreateTestFile'` and `true` or `false`.

Example: `'CreateTestFile',true`**TestType — Test case type**

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

Test case type, specified as the comma-separated pair consisting of `'TestType'` and `'baseline'`, `'equivalence'`, or `'simulation'`.

Example: `'TestType','equivalence'`**CreateHarness — Whether to create test harness**

true (default) | false

Whether to create a test harness for a model, specified as the comma-separated pair consisting of 'CreateHarness' and true or false. When the Component is not a top-level model, when you generate test cases using Simulink Design Verifier, or when you create multiple test cases and harnesses, this option is set to true automatically. When component is an export-function model and CreateHarness is true, the function creates a Test Sequence block in the generated test harness.

To generate only a test harness, set “SLDVTTestGeneration” on page 1-0 to off and “UseComponentInputs” on page 1-0 to false. Optionally specify any desired “HarnessOptions” on page 1-0 .

Example: 'CreateHarness',false

### **UseComponentInputs — Option to simulate model to obtain inputs**

true (default) | false

Option to simulate the model to obtain component inputs for the created test harness, specified as true or false. If this property is true, the test harness uses the inputs from the model simulation. If this property is false, the test harness does not use the component inputs from the simulation, and no input signal files or baseline criteria files are created.

Example: 'UseComponentInputs',false

### **FunctionInterface — Function interface for a reusable library subsystem**

string | character vector

Function interface, specified as the comma-separated pair consisting of 'FunctionInterface' and a string or character vector. Specify FunctionInterface to create tests for a reusable library subsystem. The subsystem must be at the top level of the subsystem library and must have a function interface. This property is not supported if component is an array or cell array.

Example: 'FunctionInterface','single'

### **Simulation1Mode — Simulation mode for simulation 1 of equivalence test**

'Normal' | 'Accelerator'

Simulation mode for simulation 1 of an equivalence test, specified as the comma-separated pair consisting of 'Simulation1Mode' and either 'Normal' or 'Accelerator'. If you do not specify a simulation mode, the mode of the system under test is used. The required test harness is created automatically for the simulation mode.

Example: 'Simulation1Mode','Normal'

### **Simulation2Mode — Simulation mode for simulation 2 of equivalence test**

'Normal' | 'Accelerator' | 'Rapid Accelerator' | 'Software-in-the-Loop (SIL)' | 'Processor-in-the-Loop (PIL)'

Simulation mode for simulation 2 of an equivalence test, specified as the comma-separated pair consisting of 'Simulation2Mode' and 'Normal', 'Accelerator', 'Rapid Accelerator', 'Software-in-the-Loop (SIL)', or 'Processor-in-the-Loop (PIL)'. If you do not specify a simulation mode, the mode of the system under test is used. If TestType is 'equivalence' and Simulation2Mode is 'Software-in-the-Loop (SIL)', an extra test harness is created in addition to the test case and test harness, unless the component under test is an atomic subsystem. When equivalence testing an atomic subsystem using normal and SIL or PIL modes, a single test harness is created and used for both modes.

Example: 'Simulation2Mode','Software-in-the-Loop (SIL)'

**InputsLocation — File or directory path for storing logged inputs in MAT-file**

string | character vector

File or directory path for storing logged inputs in a MAT-file, specified as the comma-separated pair consisting of 'InputsLocation' and a character vector or string. Include the file extension `.mat`. If `component` is an array or cell array, `InputsLocation` must be an existing directory path. If you do not specify `InputsLocation`, the function creates a folder in the current directory.

Example: 'InputsLocation', 'C:\MATLAB\inputs\_data.mat'

**BaselineLocation — File or directory path for storing baseline logged outputs in MAT-file**

string | character vector

File or directory path for storing baseline logged output data in a MAT-file, specified as the comma-separated pair consisting of 'BaselineLocation' and a character vector or string. Include the file extension `.mat`. If `component` is an array or cell array, `BaselineLocation` must be an existing directory path. If you do not specify `BaselineLocation`, the function creates a folder in the current directory.

Example: 'BaselineLocation', 'C:\MATLAB\baseline\_data.mat'

**CreateExcelFile — Whether to use Excel format for inputs and outputs**

false (default) | true

Whether to use Excel format for inputs and, for baseline tests only, outputs, specified as the comma-separated pair consisting of 'CreateExcelFile' and either `true` or `false`. If you use the 'ExcelFileLocation' argument to specify the file name and location, you do not need to also use 'CreateExcelFile'.

Example: 'CreateExcelFile', true

**ExcelFileLocation — File or directory path for Excel file**

string | character vector

File path for the Excel file, specified as the comma-separated pair consisting of 'ExcelFileLocation' and a character vector or string. Include the file extension `.xlsx` in the file name. If `component` is an array or cell array, `ExcelFileLocation` must be an existing directory path. If you specify a location, you do not need to also use the 'CreateExcelFile' option. If you do not specify an `ExcelFileLocation`, the function creates a folder in the current directory.

---

**Note** If `SLDVTTestGeneration` is `true` and `HarnessSource` is 'Signal Editor', you cannot save data to an Excel file.

---

Example: 'ExcelFileLocation', 'C:\MATLAB\baseline\_data.xlsx'

**Sheet — Name of Excel sheet to save data to**

string | character vector

Name of Excel sheet in which to save data, specified as the comma-separated pair consisting of 'Sheet' and a character vector or string. If you are creating a single test case and harness and do not specify a `Sheet`, the default sheet is used. When creating multiple test cases and harnesses, if you do not specify a `Sheet`, a default Excel sheet name is used.

Example: 'Sheet', 'MySubsysTest'



**SLDVTTestGeneration — Whether to generate tests using Simulink Design Verifier**

'off' (default) | 'on' | 'EnhancedMCDC'

Whether to generate tests using Simulink Design Verifier, specified as:

- 'off' — Do not use Simulink Design Verifier to generate tests.
- 'on' — Use Simulink Design Verifier to generate tests and use the Simulink Design Verifier options from the model configuration.

'EnhancedMCDC' — Use Simulink Design Verifier to generate tests with the model coverage objectives as enhanced MCDC. The EnhancedMCDC option is valid only if TestType is 'equivalence' and Simulation2Mode is 'Software-in-the-Loop (SIL)' or 'Processor-in-the-Loop (PIL)'.

---

**Note** To generate tests from Simulink Design Verifier, the system under test must be an atomic subsystem.

---

Example: 'SLDVTTestGeneration','on'

**SimulateModelForSLDVTTestGeneration — Whether to simulate top model during test generation**

false (default) | true

Whether to simulate the top model during Simulink Design Verifier test generation, specified as a logical value. When the returned tests have model coverage lower than expected, after you analyze the possible causes and reconfigure the default simulation, set this property to true. Then, rerun Simulink Design Verifier test generation. If the coverage results are still lower than expected, you can iterate the analysis, simulation reconfiguration, and test generation.

Example: 'SimulateModelForSLDVTTestGeneration','true'

**HarnessSource — Input source block for the harness**

'Inport' (default) | 'Signal Editor'

Input source block for the test harness, specified as the comma-separated pair consisting of 'HarnessSource' and either 'Inport' or 'Signal Editor'. If you are not generating any test input data (UseComponentInputs is false), this property is not supported. Instead, specify the value inside the HarnessOptions cell as {'Source', 'Test Sequence'}.

Example: 'HarnessSource','Signal Editor'

**HarnessOptions — Test harness creation options**

cell array

Test harness creation options, specified as a cell array of comma-separated name-value pairs. See `sltest.harness.create` for valid options. Do not include the harnessOwner handle as the first argument in the cell array. The first argument is set automatically to the Component value.

Only specify HarnessOptions for either of these settings:

- UseComponentInputs is false
- SLDVTTestGeneration is on or EnhancedMCDC)

If you are using Simulink Design Verifier and need to use an external harness, use `HarnessOptions` to set only `SaveExternally` to `true`. For this case, you cannot set other options at the same time.

```
Example: 'UseComponentInputs', false, 'HarnessOptions',
{ 'SynchronizationMode', 'SyncOnOpen', 'Source', 'Test
Sequence', 'SeparateAssessment', true }
```

```
Example: 'SLDVTTestGeneration', 'on', 'HarnessOptions', { 'SaveExternally', true }
```

## Output Arguments

### **tc** — Test case

`sltest.testmanager.TestCase` object | cell array

Test case, returned as an `sltest.testmanager.TestCase` object or a cell array of test case objects. If a test or harness creation fails when creating multiple test cases and harnesses, the cell array contains an `MException` or `MSLException` object with failure diagnostics. If `component` is an array or cell array and one test or harness creation fails, the function creates test cases and harnesses for other components even if one component.

### **status** — Test case creation status

cell array

Test case creation status, returned as an array of `true` and `false` values. `true` indicates that the test case was successfully created and `false` indicates that it failed.

## See Also

`sltest.harness.create` | `sltest.testmanager.TestCase` |  
`sltest.testmanager.TestFile`

### Topics

“Batch Equivalence Testing of Multiple Components”

“Create and Run Test Cases with Scripts”

“Generate Tests and Test Harnesses for a Component or Model”

“Create and Run a Back-to-Back Test”

“Library-Based Code Generation for Reusable Library Subsystems” (Embedded Coder)

### Introduced in R2020b

# 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 Editor scenarios 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 Editor scenario.

```
% Open the model for this example
openExample('sldemo_autotrans');

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

testFile =

 TestFile with properties:

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

## Input Arguments

### filePath — Test file name and path

character vector

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

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

### modelName — Model name

character vector

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

Example: 'mymodel'

**testType – Test case type**

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

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

**Output Arguments****testFile – Test file**

`sltest.testmanager.TestFile` object

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

**See Also****Topics**

“Create and Run Test Cases with Scripts”

**Introduced in R2015b**

# sltest.testmanager.exportResults

Export results set from Test Manager

## Syntax

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

## Description

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

## Examples

### Export Results Set From Test Manager

```
% Get the results set object from Test Manager
result = sltest.testmanager.getResultSets

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

## Input Arguments

### **resultObjs** — Results set array

object

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

### **filePath** — Results file path and name

character vector

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

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

## See Also

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

## Topics

“Create and Run Test Cases with Scripts”

**Introduced in R2016a**

## sltest.testmanager.getpref

Get Test Manager preferences

### Syntax

```
settings = sltest.testmanager.getpref(group)
settings = sltest.testmanager.getpref(group,preferences)
```

### Description

`settings = sltest.testmanager.getpref(group)` returns Test Manager preference settings in `group`.

`settings = sltest.testmanager.getpref(group,preferences)` returns Test Manager preferences for one or more preferences. Use `settings = sltest.testmanager.getpref(group)` to get valid preferences for `group`.

### Examples

#### Get Test Suite Section Preferences

Get the preferences for the sections that appear in test suites.

Get preferences for test suite display. A value of 1 means the section appears.

```
settings = sltest.testmanager.getpref('TestSuiteDisplay')
```

```
settings =
 struct with fields:
 TestTag: 1
 Description: 1
 Requirement: 1
 Callback: 1
 Coverage: 1
```

Get the settings for Description and Requirement preferences.

```
settings = sltest.testmanager.getpref('TestSuiteDisplay',{'Description','Requirement'})
```

```
settings =
 struct with fields:
 Description: 1
 Requirement: 1
```

#### Get MATLAB Release Information

Get a list of MATLAB releases available in Test Manager.

Get the list of MATLAB release preferences.

```
settings = sltest.testmanager.getpref('MATLABReleases')
```

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

```
 Name
MATLABRoot
IsDefault
Selected
```

Get the Name value.

```
settings.Name
```

```
ans =
 '14a'
ans =
 'R2017b'
```

### Get Value of Show Logs at Command Prompt

Show the current setting of ShowSimulationLogs IncludeOnCommandPrompt. The value 0 indicates that displaying logs is turned off, and the value 1 indicates it is turned on.

```
settings = sltest.testmanager.getpref...
 ('ShowSimulationLogs', 'IncludeOnCommandPrompt')
```

```
ans =
logical
```

```
 1
```

## Input Arguments

### group — Preference group

```
'TestFileDisplay' | 'TestSuiteDisplay' | 'TestCaseDisplay' | 'MATLABReleases'
```

Preference group name, specified as one of these values:

- 'TestFileDisplay' — File preferences and their display status
- 'TestSuiteDisplay' — Test suite preferences and their display status
- 'TestCaseDisplay' — Test case preferences and their display status
- 'MATLABReleases' — Releases available for testing

### preferences — Preference name

character vector | cell array of character vectors

Preference name, specified as a character vector or a cell array of character vectors. Use `settings = sltest.testmanager.getpref(group)` to get valid preferences for each group.

Example: ('TestSuiteDisplay', 'Description')

Example: ('ShowSimulationLogs', 'IncludeOnCommandPrompt')

## **Output Arguments**

### **settings — Test Manager preferences**

struct

Preference settings, returned as a struct.

## **See Also**

`sltest.testmanager.setpref`

## **Topics**

“Specify Test Properties in the Test Manager”

**Introduced in R2017a**



# sltest.testmanager.getResultSets

Returns result set objects in Test Manager

## Syntax

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

## Description

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

## Examples

### Get Test Suite Result

To work with a test suite result programmatically, use the `sltest.testmanager.ResultSet` function to get the result set object. For example:

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

## Output Arguments

### **rsList** — Result set array

array of objects

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

## See Also

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

## Topics

“Create and Run Test Cases with Scripts”

**Introduced in R2016a**

## sltest.testmanager.getTestFiles

Get test files open in the Test Manager

### Syntax

```
testFiles = sltest.testmanager.getTestFiles
```

### Description

`testFiles = sltest.testmanager.getTestFiles` returns an array of test files that are currently open in the Test Manager. The array contains an `sltest.testmanager.TestFile` object for each test file.

### Examples

#### Get Test Files From Test Manager

You can use the `sltest.testmanager.getTestFiles` function to get `sltest.testmanager.TestFile` objects for each test file open in the Test Manager.

Load test files in the Test Manager that you want to get the objects for.

```
% Get test files from Test Manager
testFiles = sltest.testmanager.getTestFiles
```

### Output Arguments

#### **testFiles** – Test files

`sltest.testmanager.TestFile` object array

Test files that are open in the test manager, returned as an array of `sltest.testmanager.TestFile` objects.

### See Also

`sltest.testmanager.view` | `sltest.testmanager.TestFile`

### Topics

“Create and Run Test Cases with Scripts”

**Introduced in R2016b**

# sltest.testmanager.importResults

Import Test Manager results file

## Syntax

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

## Description

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

## Examples

### Import Results and Generate Report

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

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

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

## Input Arguments

### filePath — File name and path of results set

character vector

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

Example: 'testResults.mldatx'

## Output Arguments

### resultObjs — Results set

object

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

## See Also

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

**Topics**

“Create and Run Test Cases with Scripts”

**Introduced in R2016a**

# sltest.testmanager.load

Load a test file in the Simulink Test manager

## Syntax

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

## Description

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

## Examples

### Load a Test File

Load a test file and view it in the Test Manager.

```
exampleFile = 'sltestParameterOverridesTestSuite.mldatx';

sltest.testmanager.load(exampleFile);
sltest.testmanager.view;
```

## Input Arguments

### filename — File name of test file

character vector

File name of a test file, specified as a character vector. The character vector must fully specify the location of the test file. The file can be a test file (.mldatx), or a MATLAB test file (.m), if that MATLAB test file is a derived class from `sltest.TestCase`.

Example: 'C:\MATLAB\test\_file.mldatx'

## Output Arguments

### tfObj — Test file object

object

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

## See Also

`sltest.TestCase` | `sltest.testmanager.run` | `sltest.testmanager.view` | `sltest.testmanager.TestFile`

## Topics

“Create and Run Test Cases with Scripts”

“Using MATLAB-Based Simulink Tests in the Test Manager”

“Collect Coverage Using MATLAB-Based Simulink Tests”

**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]
```

## Input Arguments

### srcObjArray — Test case or test suites to move

object array

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

### targetObj — Target test file or test suite

object

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

## **Output Arguments**

### **objArray — Test cases or test suites at new location**

object array

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

### **See Also**

`sltest.testmanager.copyTests`

### **Topics**

“Create and Run Test Cases with Scripts”

**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);
```

### Use Custom Report Class to Generate Report

If you create a custom class to customize how the report is generated using the `sltest.testmanager.TestResultReport` class, then generate the report using:

```
% Import existing results or use sltest.testmanager.run to run tests
% and collect results
result = sltest.testmanager.importResults('testResults.mldatx');
filePath = 'testreport.zip';
sltest.testmanager.report(result, filePath, ...
 'Author', 'User', ...
 'Title', 'Test', ...
 'IncludeMLVersion', true, ...
 'IncludeTestResults', int32(0), ...
 'IncludeSimulationSignalPlots', true, ...
 'NumPlotColumnsPerPage', 2, ...
```

```
'CustomReportClass', 'CustomReport', ...
'LaunchReport', true);
```

## Input Arguments

### **resultObj** — Results set object

object

Results set object to get results from, specified as an `sltest.testmanager.ResultSet`, `sltest.testmanager.TestFileResult`, `sltest.testmanager.TestSuiteResult`, `sltest.testmanager.TestCaseResult`, or `sltest.testmanager.TestIterationResult` object.

### **filePath** — File name and path of the generated report

character vector

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

### **Name-Value Pair Arguments**

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

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

Example: `'IncludeTestRequirement', true`

### **Author** — Report author

empty character vector (default)

Name of the report author, specified as a character vector.

Example: `'Test Engineer'`

### **Title** — Report title

`'Test'` (default) | character vector

Title of the report, specified as a character vector.

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.

**NumPlotRowsPerPage — Number of rows of plots to include on report pages**

2 (default)

Number of rows of plots to include on report pages, specified as an integer from 1 to 4. This property is used only if the IncludeSimulationSignalPlots property is true.

**NumPlotColumnsPerPage — Number of columns of plots to include on report pages**

1 (default)

Number of columns of plots to include on report pages, specified as an integer from 1 to 4. This property is used only if the IncludeSimulationSignalPlots property is true.

**IncludeComparisonSignalPlots — Include the comparison plots**

false (default) | true

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

**IncludeMATLABFigures — Option to include figures**

false (default) | true

Option to include the figures opened from a callback script, custom criteria, or by the model in the report, specified as true or false.

**IncludeErrorMessage — Include error messages**

true (default) | false

Choose to include 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

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

**LaunchReport — Open report at completion**

true (default) | false

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

**CustomTemplateFile — Path to document template**

character vector

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

**CustomReportClass — Class name for customized report**

character vector

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

**IncludeCoverageResult — Include coverage result metrics**

false (default) | true

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

**IncludeSimulationMetadata — Include simulation metadata**

false (default) | true

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

**See Also**`sltest.testmanager.ResultSet` | `sltest.testmanager.TestResultReport`**Topics**

“Customize Test Results Reports”

“Create and Run Test Cases with Scripts”

**Introduced in R2015a**

# sltest.testmanager.run

Run tests with Test Manager

## Syntax

```
resultObj = sltest.testmanager.run
resultObj = sltest.testmanager.run(Name,Value)
```

## Description

`resultObj = sltest.testmanager.run` runs all of the Simulink Test test files in the Test Manager.

`resultObj = sltest.testmanager.run(Name,Value)` uses additional options specified by one or more name-value arguments.

## Examples

### Run the Test File in an Example

- 1 Open the model for this example.

```
openExample('sldemo_autotrans');
```

- 2 Create the test file, test suite, and test case structure.

```
tf = sltest.testmanager.TestFile("API Test File");
ts = createTestSuite(tf,"API Test Suite");
tc = createTestCase(ts,"simulation","Simulation Test Case");
```

- 3 Remove the default test suite.

```
tsDel = getTestSuiteByName(tf,"New Test Suite 1");
remove(tsDel);
```

- 4 Assign the system under test to the test case.

```
setProperty(tc,"Model","sldemo_autotrans");
```

- 5 Run the test file and return the results data.

```
ro = sltest.testmanager.run;
```

### Run the Test File with Parallel Computing Toolbox

- 1 Open the model for this example.

```
openExample('sldemo_autotrans');
```

- 2 Create the test file, test suite, and test case structure.

```
tf = sltest.testmanager.TestFile("API Test File");
ts = createTestSuite(tf,"API Test Suite");
tc = createTestCase(ts,"simulation","Simulation Test Case");
```

- 3 Remove the default test suite.

```
tsDel = getTestSuiteByName(tf, "New Test Suite 1");
remove(tsDel);
```

- 4 Assign the system under test to the test case.

```
setProperty(tc, "Model", "sldemo_autotrans");
```

- 5 Run the test file with Parallel Computing Toolbox and return the results data.

```
ro = sltest.testmanager.run("Parallel", true);
```

## Input Arguments

### Name-Value Pair Arguments

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

Example: `Parallel=true, Tags=["safety", "regression"]`

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

Example: `"Parallel", true, "Tags", ["safety", "regression"]`

### **Parallel** — Run with parallel computing

false (default) | true

Specifies whether to run tests with Parallel Computing Toolbox™ or MATLAB Parallel Server™. This option requires Parallel Computing Toolbox or MATLAB Parallel Server license, respectively.

Example: `"Parallel", true`

Data Types: `logical`

### **Tags** — Run only tests with specified tags

string array | cell array of character vectors

Specifies test tags for execution. For more information, see “Tags”.

Example: `"Tags", ["safety"]`

Example: `"Tags", ["safety", "regression"]`

Data Types: `cell` | `char` | `string`

## Output Arguments

### **resultObj** — Results set object

`sltest.testmanager.ResultSet` object

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

## Extended Capabilities

### **Automatic Parallel Support**

Accelerate code by automatically running computation in parallel using Parallel Computing Toolbox™.

To run in parallel, set `Parallel` to `true`.

For more information, see “Run Tests Using Parallel Execution”.

## **See Also**

`sltest.testmanager.load` | `simulate`

## **Topics**

“Create and Run Test Cases with Scripts”

“Test Models Using MATLAB-Based Simulink Tests”

“Using MATLAB-Based Simulink Tests in the Test Manager”

“Collect Coverage Using MATLAB-Based Simulink Tests”

**Introduced in R2015a**

## sltest.testmanager.setpref

Set Test Manager preferences

### Syntax

```
settings = sltest.testmanager.setpref(group,preference,value)
settings = sltest.testmanager.setpref('MATLABReleases','ReleaseList',
releasePrefs)
settings = sltest.testmanager.setpref('MATLABReleases',release,releasePref)
settings =
sltest.testmanager.setpref('ShowSimulationLogs','IncludeOnCommandPrompt',
value)
```

### Description

`settings = sltest.testmanager.setpref(group,preference,value)` sets Test Manager preferences in `group`, specified by `preference`, and `value`.

`settings = sltest.testmanager.setpref('MATLABReleases','ReleaseList',releasePrefs)` updates the specified releases in your preferences with the ones specified by `releasePrefs`. This preference lets you use releases other than the current release for testing.

This syntax replaces the existing list of added releases. Including a release path that is already in the release preferences returns an error. To include that release in the `releasePrefs`, first delete the existing release list.

`settings = sltest.testmanager.setpref('MATLABReleases',release,releasePref)` adds the specified release to the list of releases in Test Manager preferences. Set `releasePref` to `{[]}` to delete that release.

`settings = sltest.testmanager.setpref('ShowSimulationLogs','IncludeOnCommandPrompt',value)` shows the simulation logs at the MATLAB command prompt when `value` is `true`. The default value is `false`.

### Examples

#### Set Test Suite Section Display Preferences

Change the display setting of two Test Manager preferences in test suite sections.

Get test suite display preferences.

```
settings = sltest.testmanager.getpref('TestSuiteDisplay')
settings =
 struct with fields:
```



```

 TestTag: 1
 Description: 1
 Requirement: 1
 Callback: 1
 Coverage: 1

```

Hide the Description and Requirement sections.

```

settings = sltest.testmanager.setpref...
('TestFileDisplay',{ 'Description', 'Requirement' },{false,false})

```

```

settings =

```

```

 struct with fields:

```

```

 TestTag: 1
 Description: 0
 Requirement: 0
 Callback: 1
 Coverage: 1
 TestFileOption: 1

```

### Add and Delete MATLAB Release Preferences

You can add several releases at a time, delete the added releases, or add and delete a single release in your Test Manager MATLAB Release preferences.

Set your preferences to include several releases. Create a struct for each release.

```

r1 = struct('Name','18b',...
 'MATLABRoot','\\mycompany\R2012b\matlab',...
 'Selected',true);
r2 = struct('Name','19a',...
 'MATLABRoot','\\mycompany\R2014a\matlab',...
 'Selected',true);
r3 = struct('Name','20a',...
 'MATLABRoot','\\mycompany\R2015a\matlab',...
 'Selected',true);

```

Add the releases using `sltest.testmanager.setpref`.

```

sltest.testmanager.setpref('MATLABReleases','ReleaseList',{r1,r2,r3});

```

Add another release to the preferences.

```

r4 = struct('Name','19b',...
 'MATLABRoot','\\mycompany\R2013a\matlab',...
 'Selected',true);
sltest.testmanager.setpref('MATLABReleases','19b',{r4});

```

Delete a release from the preferences.

```

sltest.testmanager.setpref('MATLABReleases','18b',{[]});

```

### **Set Simulation Logs Display**

Turn on displaying the simulation logs at the command prompt.

```
sltest.testmanager.setpref('ShowSimulationLogs',...
 'IncludeOnCommandPrompt',true);
```

Revert to not displaying simulation logs output at the command prompt.

```
sltest.testmanager.setpref('ShowSimulationLogs',...
 'IncludeOnCommandPrompt',false);
```

View the current simulation logs display setting.

```
sltest.testmanager.getpref('ShowSimulationLogs',...
 'IncludeOnCommandPrompt');
```

## **Input Arguments**

### **group — Preference group**

'TestFileDisplay' | 'TestSuiteDisplay' | 'TestCaseDisplay'

Preference group name, specified as one of these values:

- 'TestFileDisplay' — File section display preferences
- 'TestSuiteDisplay' — Test suite section display preferences
- 'TestCaseDisplay' — Test case section display preferences
- 'MATLABReleases' — MATLAB releases for testing preference

### **preference — Preference name**

character vector | cell array of character vectors

Preference name, specified as a character vector. Use `settings = sltest.testmanager.getpref(group)` to get valid preferences for a particular group.

Example: ('TestSuiteDisplay','TagText')

Example: ('ShowSimulationLogs','IncludeOnCommandPrompt')

### **value — Preference value**

logical | cell array of logical values

Preference value, specified as `true` to display the preference or `false` to hide it.

Example: `true`

Example: `{true,false}`

### **release — Release to add to or delete from preferences**

character vector

Release to add to or delete from preferences, specified as a character vector.

Example: `'20a'`

### **releasePrefs — Release information**

struct | cell array of structs

Release information, specified as a struct or cell array of structs. In the struct, include, in this order:

- 'Name', releaseName
- 'MATLABRoot', Path
- 'Selected', logical

Example: `struct('Name','20a','MATLABRoot','\\mypath','Selected',true)`

### **releasePref — Release information**

struct | {[]}

Release information, specified as a struct or as {[]}. Use {[]} to delete the release information from the preferences. In the struct, include:

- 'Name', releaseName
- 'MATLABRoot', Path
- 'Selected', Boolean

Example: `struct('Name','20a','MATLABRoot','\\mypath','Selected',true)`

## **Output Arguments**

### **settings — Test Manager preferences**

struct

Preference settings, returned as a struct.

## **See Also**

`sltest.testmanager.getpref`

### **Topics**

“Specify Test Properties in the Test Manager”

“Run Tests in Multiple Releases of MATLAB”

### **Introduced in R2017a**

# sltest.testmanager.TestSpecReport

Generate report of test specifications

## Syntax

```
sltest.testmanager.TestSpecReport(testObj, filePath, Name, Value)
```

## Description

`sltest.testmanager.TestSpecReport(testObj, filePath, Name, Value)` generates a report of the test specifications for the specified `testObj` and saves the report to the specified `filePath` location.

## Examples

### Generate a Test Specification Report

Generate a PDF report that uses the default template. This example reports on test cases from the test file of the `AutopilotTestFile` model. The report specifies the test author and report title. It excludes custom criteria from the report and launches the report after it is generated. Other properties default to `true` and thus, their information is included in the report.

```
testmgrFile = 'AutopilotTestFile.mldatx';
sltest.testmanager.load(testmgrFile);

tfiles = sltest.testmanager.getTestFiles;
tcases = tfiles.getTestSuites.getTestCases;

sltest.testmanager.TestSpecReport(tcases, 'testReport.pdf', ...
 'Author', 'Test File Author', ...
 'Title', 'Test Specification Details', ...
 'IncludeCustomCriteria', false, ...
 'LaunchReport', true);

sltest.testmanager.clear;
sltest.testmanager.clearResults;
```

### Generate a Customized Test Specification Report

Create a custom test case template. After you edit the template as desired, use that template when generating the report. Examples of edits to the custom template include reordering the report sections and changing the report fonts. This example shows how to generate a customized `TestCaseReporter`. Generating a customized `TestSuiteReporter` template is similar and is used to generate both Test Suite and Test File report sections. Customizing templates requires MATLAB Report Generator and Simulink Report Generator licenses.

## Create and Unzip the PDF Template File

```
sltest.testmanager.TestCaseReporter.createTemplate(...
 'MyCustomTemplate', 'pdf');
unzipTemplate('MyCustomTemplate.pdf');
```

## Edit the Template Files

Then, edit the template files in the MyCustomTemplate folder as desired.

## Zip the Template Files

```
zipTemplate('MyCustomTemplate.pdf');
```

## Use Your Custom Template File For a Report

```
testmgrFile = 'AutopilotTestFile.mldatx';
sltest.testmanager.load(testmgrFile);

tfiles = sltest.testmanager.getTestFiles;
tcases = tfiles.getTestSuites.getTestCases;

sltest.testmanager.TestSpecReport(tcases, 'testReport.pdf', ...
 'Author', 'Test Author', 'Title', 'Test', ...
 'LaunchReport', true, ...
 'TestCaseReporterTemplate', 'MyCustomTemplate.pdf');

sltest.testmanager.clear;
sltest.testmanager.clearResults;
```

## Input Arguments

### testObj — Test objects

array of `sltest.testmanager.TestFile` objects | array of `sltest.testmanager.TestSuite` objects | array of `sltest.testmanager.TestCase` objects

Test objects from which to generate the test specification report, specified as an array of `sltest.testmanager.TestFile`, `sltest.testmanager.TestSuite`, or `sltest.testmanager.TestCase` objects. You cannot include a different object types in the same array.

### filePath — File name and path of the report

character vector

File name and path of the generated report, specified as a string or character array. The file path must have one of these file extensions:

- pdf — PDF report
- docx — Word report
- zip — HTML report in a .zip file

Example: "reports/test\_specs/new\_report.pdf"

### **Name-Value Pair Arguments**

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

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

Example: `'IncludeCallbackScripts', false`

#### **Author — Report author**

empty string or character vector (default)

Name of the report author, specified as a string or character vector.

Example: `'Author', 'J. Smith'`

#### **Title — Report title**

`'Test Specification Report'` (default) | string | character vector

Title of the report, specified as a character vector.

#### **IncludeTestDetails — Include test details**

`true` (default) | `false`

Option to include test details in the report, specified as a logical. If `true`, the test details included in the report are test tags, releases, description, and requirements.

#### **IncludeTestFileOptions — Include test file options**

`true` (default) | `false`

Option to include test file options in the report, specified as a logical. If `true`, the test file options included in the report are:

- Whether to close open figures
- Whether to store MATLAB figures
- Whether to generate the report after execution
- Results report generation options
  - Report title
  - Author
  - Whether to include the MATLAB version
  - Types of test results to include (all, failed only, or passed only)
  - Other items to include in the report - test requirements, simulation metadata, error and log messages, plots of simulation output and baseline, plots of criteria and assessments, MATLAB figures, and coverage results
  - Output file format
  - Output file name
  - Custom report class

#### **IncludeCoverageSettings — Include coverage settings**

`true` (default) | `false`

Option to include coverage settings in the report, specified as a logical. If `true`, the coverage settings included in the report are coverage to collect, coverage filter file name, and coverage metrics. Examples of coverage metrics included in the report include decisions, signal range, relational boundaries, saturation on integer overflow, and lookup tables. For more information about collecting coverage, see “Collect Coverage in Tests”.

**IncludeSystemUnderTest — Include system under test**`true (default) | false`

Option to include the system under test in the report, specified as a logical. If `true`, the system under test information included in the report is:

- Model name and image
- Harness name and image
- Test sequence and assessment data (if they exist in the test harness)
- Simulation settings — simulation mode, start time (if overridden), stop time (if overridden), and initial state (if overridden)
- Target settings — target information for real-time test cases

**IncludeConfigSettingsOverrides — Include configuration settings overrides**`true (default) | false`

Option to include configuration settings overrides, specified as a logical. If `true`, the report includes the settings that differ from the model configuration settings.

**IncludeCallbackScripts — Include callback scripts**`true (default) | false`

Option to include callback scripts in the report, specified as a logical.

**IncludeParameterOverrides — Include parameter overrides**`true (default) | false`

Option to include parameter overrides in the report, specified as a logical. If `true`, the report includes the name of the parameter set or workspace variable, the override value, the source of the variable, and the model element.

**IncludeExternalInputs — Include external inputs**`true (default) | false`

Option to include external inputs in the report, specified as a logical. If `true`, the report includes the name, file path, and mapping status of the external inputs.

**IncludeLoggedSignals — Include logged signals**`true (default) | false`

Option to include logged signals, specified as a logical. If `true`, the report includes the name, source, port index, and plot index for each logged signal.

**IncludeBaselineCriteria — Include baseline criteria**`true (default) | false`

Option to include baseline criteria information in the report, specified as a logical. If `true`, the report includes the signal name, absolute tolerance, relative tolerance, leading tolerance, and lagging tolerance for the baseline test.

**IncludeEquivalenceCriteria — Include equivalence criteria**`true (default) | false`

Option to include equivalence criteria information in the report, specified as a logical. If `true`, the report includes the signal name, absolute tolerance, relative tolerance, leading tolerance, and lagging tolerance for the equivalence test.

**IncludeIterations — Include iterations**`true (default) | false`

Option to include iterations information in the report, specified as a logical. If `true`, the report includes the iteration name and the values of the external inputs, parameter set, and logged signal set for each iteration. It also includes the content of the **Iterations Script** section from the Test Manager.

**IncludeLogicalAndTemporalAssessments — Include logical and temporal assessments**`true (default) | false`

Option to include logical and temporal assessments in the report, specified as a logical. If `true`, the report includes the Assessment Callbacks, Assessment Definitions, and Symbols from the Test Case.

**IncludeCustomCriteria — Include custom criteria**`true (default) | false`

Option to include the custom pass/fail criteria script in the report, specified as a logical.

**LaunchReport — Open generated report**`false (default) | true`

Option to open the report after it is generated, specified as a logical.

**TestCaseReporterTemplate — Path to test case reporter template**`character vector`

Path to test case reporter template, specified as a character vector. The template path file name must use a `pdf`, `html`, or `dotx` extension, for a PDF, HTML, or Word template, respectively. The specified template is used instead of the default `TestCaseReporter` template. Using non-default templates is available only if you have a Simulink Report Generator license.

**TestSuiteReporterTemplate — Path to test suite reporter template**`character vector`

Path to test suite reporter template, specified as a character vector. The file name in the template path must use a `pdf`, `html`, or `dotx` extension, for a PDF, HTML, or Word template, respectively. The `TestSuiteReporter` template is used for both test suites and test files. The specified template is used instead of the default `TestSuiteReporter` template. Using non-default templates is available only if you have a Simulink Report Generator license.



## **See Also**

`sltest.testmanager.getTestFiles` | `sltest.testmanager.TestFile` |  
`sltest.testmanager.TestSuite` | `sltest.testmanager.TestCase`

## **Topics**

“Generate Test Specification Reports”  
“Customize Test Specification Reports”

## **Introduced in R2019b**

## 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.

### Examples

#### Create Test Case and View in Test Manager

```
% Open the model for this example
openExample('sldemo_absbrake');

% Create the test file, test suite, and test case structure
tf = sltest.testmanager.TestFile('API Test File');
ts = createTestSuite(tf, 'API Test Suite');
tc = createTestCase(ts, 'baseline', 'Baseline API Test Case');

% Assign the system under test to test case
setProperty(tc, 'Model', 'sldemo_absbrake');

% Open Test Manager to view and edit test case
sltest.testmanager.view;
```

### See Also

`sltest.testmanager.load` | `sltest.testmanager.run`

### Topics

“Create and Run Test Cases with Scripts”

**Introduced in R2015a**

# sltest.testsequence.activateScenario

**Package:** sltest.testsequence

Set Test Sequence block scenario as active

## Syntax

```
sltest.testsequence.activateScenario(blockPath,scenarioName)
```

## Description

`sltest.testsequence.activateScenario(blockPath,scenarioName)` makes the `scenarioName` scenario in the specified Test Sequence block active so it runs during simulation. Only one scenario is active at a time. You can use `activateScenario` only if `sltest.testsequence.setScenarioControlSource` is `sltest.testsequence.ScenarioControlSource.Block`, which sets the active scenario control to a Test Sequence block instead of a variable in the workspace. Use `sltest.testsequence.getScenarioControlSource` to view the current scenario control source setting and `sltest.testsequence.setScenarioControlSource` to change it.

## Examples

### Activate Test Sequence Scenario

Set the Test Sequence block in the `sltestRollRefTestExample` model to use scenarios. Then, add another scenario named `Scenario_2` and activate `Scenario_2` so it runs when the model simulates. The control source is the default, `sltest.testsequence.ScenarioControlSource.Block`. Close the model without saving it.

```
Model = 'sltestRollRefTestExample';
load_system(Model);

sltest.testsequence.useScenario...
('sltestRollRefTestExample/Test Sequence',...
'Scenario_1');

sltest.testsequence.addScenario...
('sltestRollRefTestExample/Test Sequence',...
'Scenario_2');

sltest.testsequence.activateScenario...
('sltestRollRefTestExample/Test Sequence',...
'Scenario_2');

close_system(Model,0)
```

## Input Arguments

**blockPath** — Test Sequence block path

string | character vector

Path to a Test Sequence block, including the block name, specified as a string or character vector.

Example: 'FanSpeedTestHarness/Test Sequence'

**scenarioName — Name of scenario**

character vector | string

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

Example: 'Name', 'Scenario\_1'

**See Also**

`sltest.testsequence.useScenario` | `setScenarioControlSource` |  
`sltest.testsequence.getActiveScenario` | `getAllScenarios`

**Topics**

“Programmatically Create and Run Test Sequence Scenarios”

**Introduced in R2020b**

# sltest.testsequence.addScenario

**Package:** sltest.testsequence

Add new scenario to Test Sequence block

## Syntax

```
sltest.testsequence.addScenario(blockPath,scenarioName)
```

## Description

`sltest.testsequence.addScenario(blockPath,scenarioName)` adds a new scenario with the name specified by `scenarioName` to the Test Sequence block specified by `blockPath`. The specified Test Sequence block must be in scenario mode and the name of the new scenario cannot already exist in the block. The new scenario is added after the last scenario in the block.

## Examples

### Add New Test Sequence Scenario

Set the Test Sequence block in the `sltestRollRefTestExample` model to use scenarios. Then, add a new scenario named `new_Scenario`. Close the model without saving it.

```
Model = 'sltestRollRefTestExample';
load_system(Model);

sltest.testsequence.useScenario...
('sltestRollRefTestExample/Test Sequence',...
 'Scenario_1');

sltest.testsequence.addScenario...
('sltestRollRefTestExample/Test Sequence',...
 'new_Scenario');

close_system(Model,0)
```

## Input Arguments

### **blockPath** — Test Sequence block path

string | character vector

Path to a Test Sequence block, including the block name, specified as a string or character vector.

Example: 'FanSpeedTestHarness/Test Sequence'

### **scenarioName** — Name of scenario

character vector | string

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

Example: 'Name', 'Scenario\_1'

**See Also**

`sltest.testsequence.useScenario | deleteScenario`

**Topics**

“Programmatically Create and Run Test Sequence Scenarios”

**Introduced in R2020b**

# sltest.testsequence.addStep

Add test sequence step

## Syntax

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

## Description

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

## Examples

### Create a New Test Step

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

Set paths and open the model.

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

Open the test harness.

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

Create a new local variable `h`.

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

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

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

## Input Arguments

### **blockPath** — Test Sequence block path

string | character vector

Path to a Test Sequence block, including the block name, specified as a string or character vector.

Example: 'FanSpeedTestHarness/Test Sequence'

### **stepPath** — Test step name and hierarchy level

character vector

Path of the step in the Test Sequence block, specified as a character vector. The path includes the step location in the Test Sequence hierarchy, using `.` to separate hierarchy levels. If the Test Sequence block is using scenarios, add the scenario name that contains the step to the beginning of the step path, for example, `Scenario_2.SystemHeatingTest.InitializeHeating`.

Example: `'SystemHeatingTest.InitializeHeating'`

### **Name-Value Pair Arguments**

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

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

Example: `'Action','out = square(et)','IsWhenStep',false,'Description','Square wave.'` specifies a test step to produce a square wave.

### **Action — Test step actions**

character vector

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

Example: `'Action','out = square(et)'`

### **IsWhenStep — Specifies standard or When decomposition step**

false (default) | true

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

Example: `'IsWhenStep',true`

### **WhenCondition — When decomposition step switch condition**

character vector

Specifies the condition that activates a When decomposition child step. To activate the When step, enter a valid logical expression.

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

### **Description — Description for the test step**

character vector

Test step description, specified as a character vector.

Example: `'Description','This step produces a high-frequency square wave.'`

### **See Also**

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

### **Topics**

“Programmatically Create a Test Sequence”



**Introduced in R2016a**

## sltest.testsequence.addStepAfter

Add test sequence step after existing step

### Syntax

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

### Description

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

### Examples

#### Create a New Test Step

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

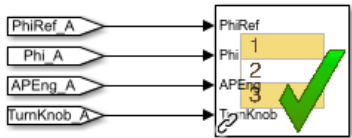
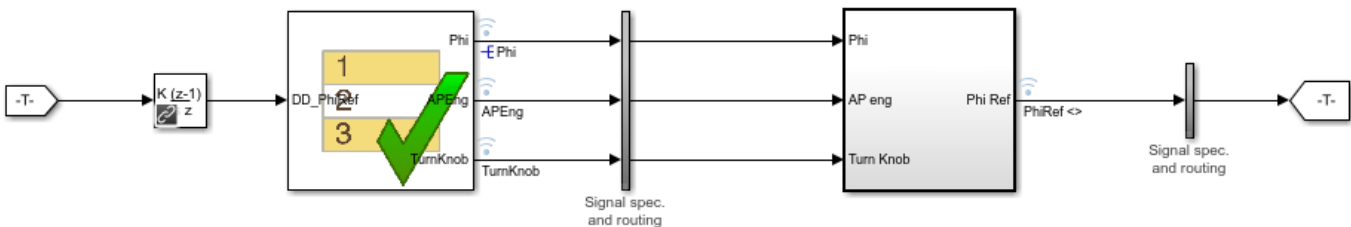
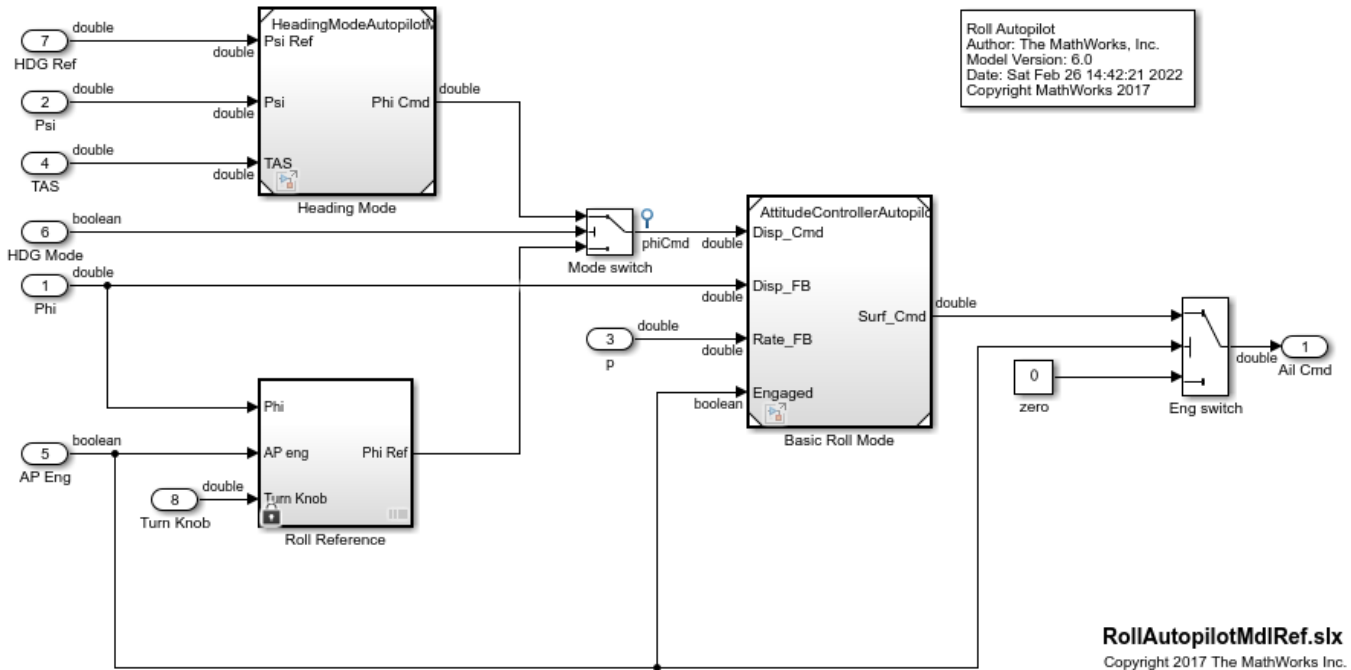
#### Open the Model and the Test Harness

```
rollModel = 'RollAutopilotMdlRef';
testHarness = 'RollReference_Requirement1_3';

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

### Requirements-based Testing for Controller Development

This model is used to show how to perform requirements-based testing using test harnesses, Test Sequence blocks, and the test manager.  
 To view the demo, enter `sltestRequirementsTestingAutopilotDemo` in MATLAB(R).



#### Create a New Local Variable Named h

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

#### Add a Step Named step2 and Set the Value of h to 5

```
sltest.testsequence.addStepAfter...
('RollReference_Requirement1_3/Test Sequence',...
'AttitudeLevels.APEngage_LowRoll.step2',...
```

```
'AttitudeLevels.APEngage_LowRoll.SetLowPhi', ...
'Action', 'h = 5;')
```

## Input Arguments

### **blockPath — Test Sequence block path**

string | character vector

Path to a Test Sequence block, including the block name, specified as a string or character vector.

Example: 'FanSpeedTestHarness/Test Sequence'

### **newStep — New test step name or handle**

character vector

Name of a new test step in the Test Sequence block, specified as a character vector. It is added after `stepPath` and must have the same parent step as `stepPath`.

Example: 'newStep'

Example: 'topStep.midStep.newStep'

### **stepPath — Test step name and hierarchy level**

character vector

Path of the step in the Test Sequence block, specified as a character vector. The path includes the step location in the Test Sequence hierarchy, using `.` to separate hierarchy levels. If the Test Sequence block is using scenarios, add the scenario name that contains the step to the beginning of the step path, for example, `Scenario_2.SystemHeatingTest.InitializeHeating`.

Example: 'SystemHeatingTest.InitializeHeating'

## Name-Value Pair Arguments

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

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

Example: 'Action', 'out = square(et)', 'IsWhenStep', false, 'Description', 'This step produces a square wave.' specifies a test step to produce a square wave.

### **Action — Test step actions**

character vector

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

Example: 'Action', 'out = square(et)'

### **IsWhenStep — Specifies standard or When decomposition step**

false (default) | true

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

Example: 'IsWhenStep', true

**WhenCondition — When decomposition step switch condition**

character vector

Specifies the condition that activates a `When` decomposition child step. To activate the `When` step, enter a valid logical expression.

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

**Description — Description for the test step**

character vector

Test step description, specified as a character vector.

Example: `'Description', 'This step produces a high-frequency square wave.'`

**See Also**

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

**Topics**

“Programmatically Create a Test Sequence”

**Introduced in R2016a**

## **sltest.testsequence.addStepBefore**

Add test sequence step before existing step

### **Syntax**

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

### **Description**

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

### **Examples**

#### **Create a New Test Step**

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

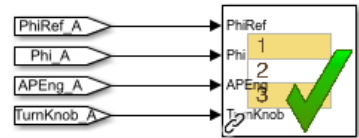
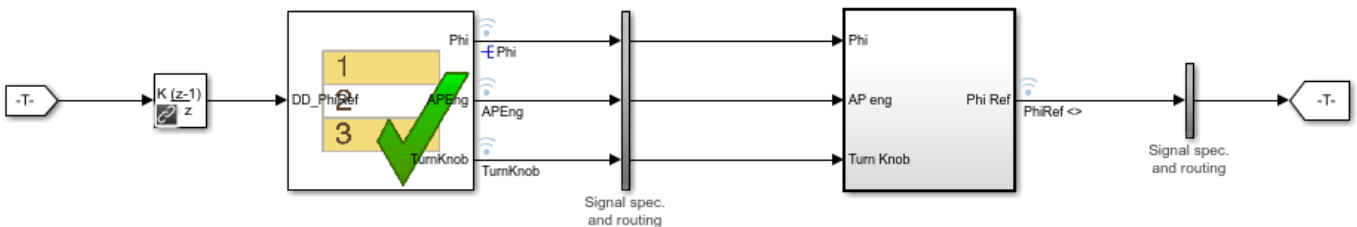
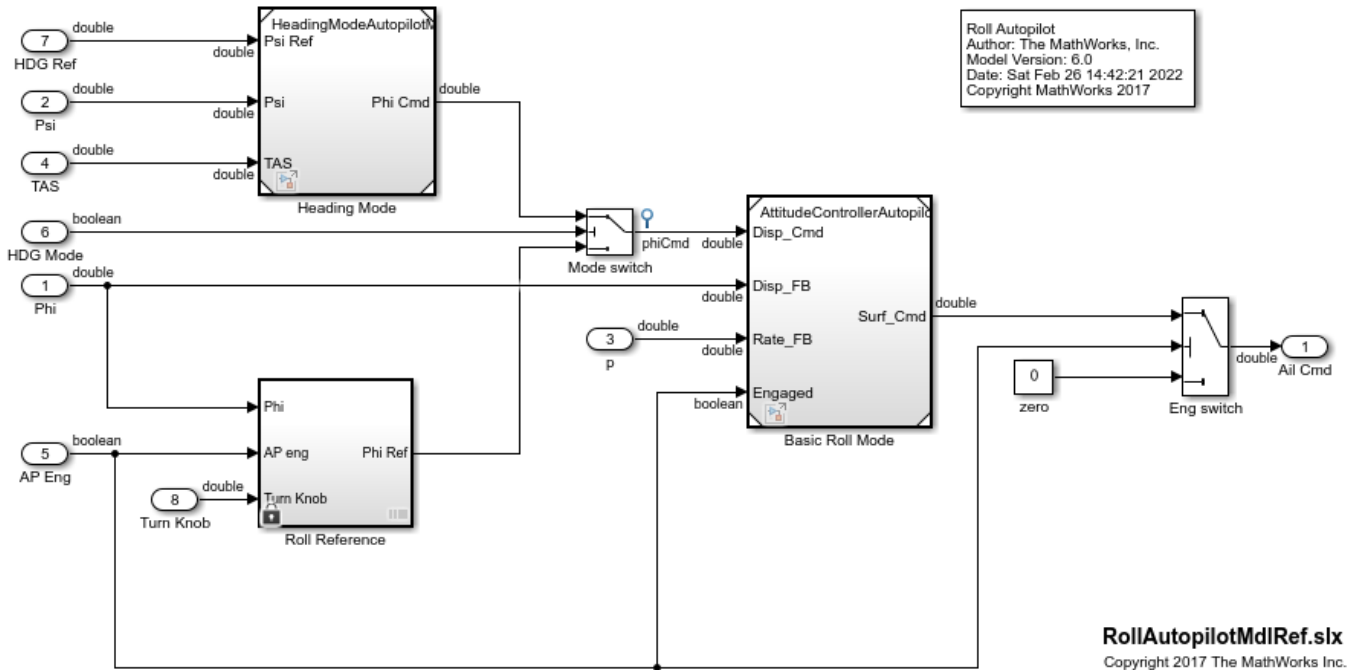
#### **Open the Model and the Test Harness**

```
rollModel = 'RollAutopilotMdlRef';
testHarness = 'RollReference_Requirement1_3';

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

### Requirements-based Testing for Controller Development

This model is used to show how to perform requirements-based testing using test harnesses, Test Sequence blocks, and the test manager.  
 To view the demo, enter `sltestRequirementsTestingAutopilotDemo` in MATLAB(R).



#### Create a New Local Variable Named h

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

#### Add a Step Named step1 and Set the Value of h to 5

```
sltest.testsequence.addStepBefore...
('RollReference_Requirement1_3/Test Sequence',...
'AttitudeLevels.APEngage_LowRoll.step2',...
```

```
'AttitudeLevels.APEngage_LowRoll.SetLowPhi', ...
'Action', 'h = 5;')
```

## Input Arguments

### **blockPath — Test Sequence block path**

string | character vector

Path to a Test Sequence block, including the block name, specified as a string or character vector.

Example: 'FanSpeedTestHarness/Test Sequence'

### **newStep — New test step name or handle**

character vector

Name of a new test step in the Test Sequence block, specified as a character vector. It is added before `stepPath` and must have the same parent step as `stepPath`.

Example: 'newStep'

Example: 'topStep.midStep.newStep'

### **stepPath — Test step name and hierarchy level**

character vector

Path of the step in the Test Sequence block, specified as a character vector. The path includes the step location in the Test Sequence hierarchy, using `.` to separate hierarchy levels. If the Test Sequence block is using scenarios, add the scenario name that contains the step to the beginning of the step path, for example, `Scenario_2.SystemHeatingTest.InitializeHeating`.

Example: 'SystemHeatingTest.InitializeHeating'

## Name-Value Pair Options

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

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

Example: 'Action', 'out = square(et)', 'IsWhenStep', false, 'Description', 'This step produces a square wave.' specifies a test step to produce a square wave.

### **Action — Test step actions**

character vector

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

Example: 'Action', 'out = square(et)'

### **IsWhenStep — Specifies standard or When decomposition step**

false (default) | true

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

Example: 'IsWhenStep', true



**WhenCondition — When decomposition step switch condition**

character vector

Specifies the condition that activates a When decomposition child step. To activate the When step, enter a valid logical expression.

Example: 'WhenCondition', 'a >= 1'

**Description — Description for the test step**

character vector

Test step description, specified as a character vector.

Example: 'Description', 'This step produces a high-frequency square wave.'

**See Also**

sltest.testsequence.addStep | sltest.testsequence.addStepAfter |  
sltest.testsequence.addSymbol | sltest.testsequence.addTransition |  
sltest.testsequence.editStep | sltest.testsequence.findStep

**Topics**

“Programmatically Create a Test Sequence”

**Introduced in R2016a**

## sltest.testsequence.addSymbol

Add symbol to test sequence

### Syntax

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

### Description

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

### Examples

#### Create a New Data Symbol

This example creates a parameter theta in the Test Sequence block.

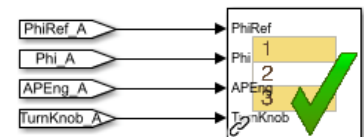
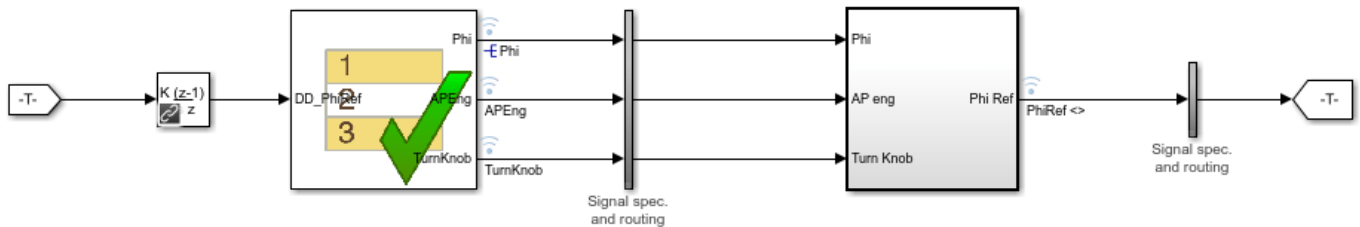
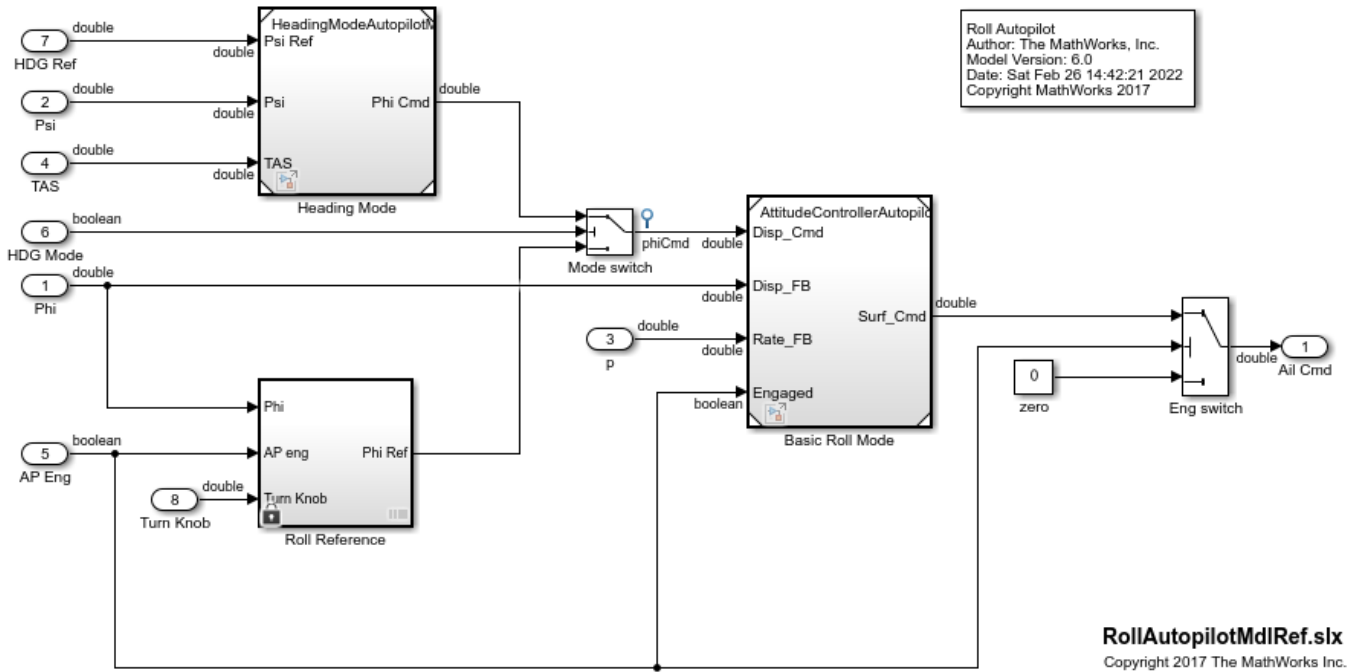
#### Open the Model and the Test Harness

```
rollModel = 'RollAutopilotMdlRef';
testHarness = 'RollReference_Requirement1_3';

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

### Requirements-based Testing for Controller Development

This model is used to show how to perform requirements-based testing using test harnesses, Test Sequence blocks, and the test manager.  
 To view the demo, enter `sltestRequirementsTestingAutopilotDemo` in MATLAB(R).



#### Add a New Parameter

```
sltest.testsequence.addSymbol...
('RollReference_Requirement1_3/Test Sequence',...
'theta', 'Data', 'Parameter');
```

#### Input Arguments

**blockPath** — Test Sequence block path  
 string | character vector

Path to a Test Sequence block, including the block name, specified as a string or character vector.

Example: 'FanSpeedTestHarness/Test Sequence'

**name — Name of new symbol**

character vector

Name of the new symbol, specified as a character vector. The symbol must not already exist in the Test Sequence block.

Example: 'theta'

**kind — Symbol type**

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

Symbol type, specified as a character vector.

Example: 'Data'

**scope — Symbol scope**

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

Symbol scope, specified as a character vector.

Example: 'Parameter'

**See Also**

`sltest.testsequence.deleteSymbol` | `sltest.testsequence.editSymbol` |  
`sltest.testsequence.findSymbol` | `sltest.testsequence.readSymbol` |  
`sltest.testsequence.find`

**Topics**

"Programmatically Create a Test Sequence"

**Introduced in R2016a**

# sltest.testsequence.addTransition

Add new transition to test sequence step

## Syntax

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

## Description

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

## Examples

### Add and Edit a Test Step Transition

This example adds a transition to a test step, then changes the transition's index, condition, and next step of the first transition in the step.

1. Load the model.

```
Model = 'sltestRollRefTestExample';
load_system(Model);
```

2. Add a transition to the step `AttitudeLevels.APEngage.LowRoll`. The transition destination is the step `AttitudeLevels.APEngage_End`.

```
sltest.testsequence.addTransition('sltestRollRefTestExample/Test Sequence',...
'AttitudeLevels.APEngage_LowRoll','TurnKnob ~= 0',...
'AttitudeLevels.APEngagement_End')
```

3. Edit the transition index, condition, and next step of the first transition.

```
sltest.testsequence.editTransition('sltestRollRefTestExample/Test Sequence',...
'AttitudeLevels.APEngage_LowRoll',1,'Index',2,...
'NextStep','AttitudeLevels.APEngage_HighRoll',...
'Condition','duration(DD_PhiRef == 0,sec) >= 5')
```

4. Close the model.

```
close_system(Model,0)
```

## Input Arguments

### blockPath — Test Sequence block path

string | character vector

Path to a Test Sequence block, including the block name, specified as a string or character vector.

Example: 'FanSpeedTestHarness/Test Sequence'

**fromStep — Origination step path**

character vector

Path of an existing step in the Test Sequence block, specified as a character vector, at which the transition originates. The path must include the step name and step hierarchy, using . to separate hierarchy levels. This step must be at same level as toStep.

Example: 'topStep.midStep.step1'

**condition — Condition on which the transition executes**

character vector

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

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

**toStep — Destination step path**

character vector

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

Example: 'topStep.midStep.step2'

**See Also**

sltest.testsequence.addStep | sltest.testsequence.addSymbol |  
sltest.testsequence.deleteTransition | sltest.testsequence.editTransition |  
sltest.testsequence.readTransition | sltest.testsequence.find

**Topics**

"Programmatically Create a Test Sequence"

**Introduced in R2016a**

# sltest.testsequence.deleteScenario

**Package:** sltest.testsequence

Delete scenario from Test Sequence block

## Syntax

```
sltest.testsequence.deleteScenario(blockPath,scenarioName)
```

## Description

`sltest.testsequence.deleteScenario(blockPath,scenarioName)` removes the scenario specified by `scenarioName` from the Test Sequence block specified by `blockPath`. All steps in the deleted scenario are also deleted. Each scenario in a Test Sequence block is identified by a scenario name and an index. When you delete a scenario, the indices of the following scenarios update. For example, if you delete the second scenario, the index of the third scenario changes from 3 to 2.

## Examples

### Delete Test Sequence Scenario

Set the Test Sequence block in the `sltestRollRefTestExample` model to use scenarios. Add a new scenario named `Scenario_2`, and then delete the scenario.

```
Model = 'sltestRollRefTestExample';
load_system(Model);

sltest.testsequence.useScenario...
('sltestRollRefTestExample/Test Sequence',...
'Scenario_1');
sltest.testsequence.addScenario...
('sltestRollRefTestExample/Test Sequence',...
'Scenario_2');

sltest.testsequence.deleteScenario...
('sltestRollRefTestExample/Test Sequence',...
'Scenario_2');

close_system(Model,0)
```

## Input Arguments

### **blockPath** — Test Sequence block path

string | character vector

Path to a Test Sequence block, including the block name, specified as a string or character vector.

Example: 'FanSpeedTestHarness/Test Sequence'

**scenarioName — Name of scenario**

character vector | string

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

Example: 'Name', 'Scenario\_1'

**See Also**

`addScenario` | `getAllScenarios` | `sltest.testsequence.useScenario`

**Topics**

“Programmatically Create and Run Test Sequence Scenarios”

**Introduced in R2020b**



# sltest.testsequence.deleteStep

Delete test sequence step

## Syntax

```
sltest.testsequence.deleteStep(blockPath,stepPath)
```

## Description

`sltest.testsequence.deleteStep(blockPath,stepPath)` deletes a test step specified by `stepPath` from a Test Sequence block specified by `blockPath`. Sub-steps of `stepPath` are also deleted.

## Examples

### Delete a Test Step Programmatically

This example shows how to delete a test step programmatically from a Test Sequence block.

1. Load the model.

```
Model = 'sltestRollRefTestExample';
load_system(Model);
```

2. Delete the step Stop from the Test Sequence block.

```
sltest.testsequence.deleteStep('sltestRollRefTestExample/Test Sequence','Stop');
```

3. Close the model

```
close_system(Model,0);
```

## Input Arguments

### **blockPath** — Test Sequence block path

string | character vector

Path to a Test Sequence block, including the block name, specified as a string or character vector.

Example: 'FanSpeedTestHarness/Test Sequence'

### **stepPath** — Test step name and hierarchy level

character vector

Path of the step in the Test Sequence block, specified as a character vector. The path includes the step location in the Test Sequence hierarchy, using `.` to separate hierarchy levels. If the Test Sequence block is using scenarios, add the scenario name that contains the step to the beginning of the step path, for example, `Scenario_2.SystemHeatingTest.InitializeHeating`.

Example: 'SystemHeatingTest.InitializeHeating'

**See Also**

`sltest.testsequence.addStep` | `sltest.testsequence.deleteSymbol` |  
`sltest.testsequence.deleteTransition`

**Topics**

“Programmatically Create a Test Sequence”

**Introduced in R2017a**

# sltest.testsequence.deleteSymbol

Delete test sequence block symbol

## Syntax

```
sltest.testsequence.deleteSymbol(blockPath, symbolName)
```

## Description

`sltest.testsequence.deleteSymbol(blockPath, symbolName)` deletes `symbolName` from a Test Sequence block specified by `blockPath`.

## Examples

### Delete a Test Sequence Symbol Programmatically

This example shows how to delete a data symbol programmatically from a Test Sequence block. The data symbol `DurationLimit` is a constant.

1. Load the model.

```
Model = 'sltestRollRefTestExample';
load_system(Model);
```

2. Delete the constant `DurationLimit` from the Test Sequence block.

```
sltest.testsequence.deleteSymbol('sltestRollRefTestExample/Test Sequence', ...
 'DurationLimit');
```

3. Close the model.

```
close_system(Model,0);
```

## Input Arguments

### **blockPath** — Test Sequence block path

string | character vector

Path to a Test Sequence block, including the block name, specified as a string or character vector.

Example: 'FanSpeedTestHarness/Test Sequence'

### **symbolName** — Symbol name

character vector

Name of the symbol to delete in the Test Sequence block, specified as a character vector.

Example: 'testOutput'

**See Also**

`sltest.testsequence.addSymbol` | `sltest.testsequence.deleteStep` |  
`sltest.testsequence.deleteTransition`

**Topics**

“Programmatically Create a Test Sequence”

**Introduced in R2017a**

# sltest.testsequence.deleteTransition

Delete test sequence transition

## Syntax

```
sltest.testsequence.deleteTransition(blockPath,stepPath,index)
```

## Description

`sltest.testsequence.deleteTransition(blockPath,stepPath,index)` deletes the transition at the given numerical index from the test step specified by `blockPath` and `stepPath`.

## Examples

### Delete Transition From Test Sequence

This example deletes a transition from the test sequence.

#### Load the Model

```
model = 'sltestRollRefTestExample';
load_system(model);
```

#### Delete the Transition and Close the Model

Delete the transition from the test step `SetKnobAndPhi` under the parent step `TurnKnobAndAttitude`.

```
sltest.testsequence.deleteTransition...
('sltestRollRefTestExample/Test Sequence',...
 'TurnKnobAndAttitude.SetKnobAndPhi',1)

close_system(model,0);
```

## Input Arguments

### **blockPath** — Test Sequence block path

string | character vector

Path to a Test Sequence block, including the block name, specified as a string or character vector.

Example: 'FanSpeedTestHarness/Test Sequence'

### **stepPath** — Test step name and hierarchy level

character vector

Path of the step in the Test Sequence block, specified as a character vector. The path includes the step location in the Test Sequence hierarchy, using `.` to separate hierarchy levels. If the Test Sequence block is using scenarios, add the scenario name that contains the step to the beginning of the step path, for example, `Scenario_2.SystemHeatingTest.InitializeHeating`.

Example: 'SystemHeatingTest.InitializeHeating'

**index – Transition index**

integer

Integer specifying the transition in the test step to be edited. Corresponds to the integer displayed in the Transition cell of the Test Sequence Editor.

Example: 3

**See Also**

`sltest.testsequence.addTransition` | `sltest.testsequence.deleteStep` |  
`sltest.testsequence.deleteSymbol`

**Topics**

“Programmatically Create a Test Sequence”

**Introduced in R2017a**

# sltest.testsequence.editScenario

**Package:** sltest.testsequence

Edit Test Sequence block test scenario properties

## Syntax

```
sltest.testsequence.editScenario(blockPath,scenarioName,'Name',newName)
```

## Description

`sltest.testsequence.editScenario(blockPath,scenarioName,'Name',newName)` changes the name of the scenario specified by `scenarioName` in the Test Sequence block specified by `blockPath` to the specified `newName`. To edit the properties of steps within a scenario, see `sltest.testsequence.editStep`.

## Examples

### Edit Test Sequence Scenario Name

Set the Test Sequence block in the `sltestRollRefTestExample` model to use scenarios. Then, change the name of the scenario named `Scenario_1` to `FirstScenario`.

```
Model = 'sltestRollRefTestExample';
load_system(Model);

% Set block to use scenarios
sltest.testsequence.useScenario...
 ('sltestRollRefTestExample/Test Sequence',...
 'Scenario_1');

% Rename scenario
sltest.testsequence.editScenario...
 ('sltestRollRefTestExample/Test Sequence',...
 'Scenario_1','Name','FirstScenario');

close_system(Model,0)
```

## Input Arguments

### **blockPath** — Test Sequence block path

string | character vector

Path to a Test Sequence block, including the block name, specified as a string or character vector.

Example: 'FanSpeedTestHarness/Test Sequence'

### **scenarioName** — Name of scenario

character vector | string

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

Example: 'Name', 'Scenario\_1'

**newName — New name for scenario**

character vector | string

New name for the scenario, specified as a string or character vector.

Example: 'Name', 'FirstScenario'

**See Also**

`sltest.testsequence.editStep` | `addScenario` | `deleteScenario`

**Topics**

“Programmatically Create and Run Test Sequence Scenarios”

**Introduced in R2020b**



# sltest.testsequence.editStep

Edit test sequence step

## Syntax

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

## Description

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

## Examples

### Add and Edit a Test Step

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

Open the model and test harness.

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

Add a test step named `SquareAndVeryQuick`.

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

Edit the step actions.

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

action =

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

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

Add two substeps to the new step.

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

Change the parent step to a When decomposition.

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

Add a When condition to the substep Step1.

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

## Input Arguments

### **blockPath — Test Sequence block path**

string | character vector

Path to a Test Sequence block, including the block name, specified as a string or character vector.

Example: 'FanSpeedTestHarness/Test Sequence'

### **stepPath — Test step name and hierarchy level**

character vector

Path of the step in the Test Sequence block, specified as a character vector. The path includes the step location in the Test Sequence hierarchy, using . to separate hierarchy levels. If the Test Sequence block is using scenarios, add the scenario name that contains the step to the beginning of the step path, for example, Scenario\_2.SystemHeatingTest.InitializeHeating.

Example: 'SystemHeatingTest.InitializeHeating'

### **Name-Value Pair Arguments**

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

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

Example: 'Action','out = square(et)','IsWhenStep',false,'Description','This step produces a square wave.' specifies a test step to produce a square wave.

### **Name — Test step name**

character vector

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

Example: 'Name','HoldOutput'

### **Action — Test step action**

character vector

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

Example: 'Action','out = square(et)'

### **IsWhenStep — Specifies standard or When decomposition step**

false (default) | true

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

Example: 'IsWhenStep', true

### **WhenCondition – When decomposition step switch condition**

character vector

Character vector specifying the condition that activates a When decomposition child step. This must be a valid logical expression for the When step to activate.

Example: 'WhenCondition', 'a >= 1'

### **Description – Description for the test step**

character vector

Test step description, specified as a character vector.

Example: 'Description', 'This step produces a high-frequency square wave.'

### **See Also**

sltest.testsequence.addStep | sltest.testsequence.deleteStep |  
sltest.testsequence.editSymbol | sltest.testsequence.editTransition |  
sltest.testsequence.findStep

### **Topics**

“Programmatically Create a Test Sequence”

### **Introduced in R2016a**

# sltest.testsequence.editSymbol

Edit symbol in Test Sequence block

## Syntax

```
sltest.testsequence.editSymbol(blockPath,name,Name,Value)
```

## Description

`sltest.testsequence.editSymbol(blockPath,name,Name,Value)` edits a symbol name with properties specified by `Name,Value` in a Test Sequence block specified by `blockPath`. Symbols include data, function calls, messages, and triggers.

## Examples

### Find, read, and edit a Test Sequence block data symbol

This example edits constant `DurationLimit` in the Test Sequence block, changing it to a local variable of single data type.

1. Load the model.

```
Model = 'sltestRollRefTestExample';
load_system(Model)
```

2. Search for data symbols containing the word duration.

```
data_names = sltest.testsequence.findSymbol...
 ('sltestRollRefTestExample/Test Sequence','Name','[Dd]uration',...
 'RegExp','on','Kind','Data')

data_names = 1x1 cell array
 {'DurationLimit'}
```

3. Read the properties of the `DurationLimit` constant.

```
dlProperties = sltest.testsequence.readSymbol...
 ('sltestRollRefTestExample/Test Sequence',data_names{1})

dlProperties = struct with fields:
 Kind: 'Data'
 Scope: 'Constant'
 DataType: 'double'
 Description: ''
 Document: ''
 InitialValue: '5'
 Name: 'DurationLimit'
 Size: ''
 Tag: []
```

4. Change `DurationLimit` to a local variable of `single` data type.

```
sltest.testsequence.editSymbol('sltestRollRefTestExample/Test Sequence', ...
 data_names{1}, 'Scope', 'Local', 'DataType', 'single')
```

5. Close the model.

```
close_system(Model,0)
```

## Input Arguments

### **blockPath** — Test Sequence block path

string | character vector

Path to a Test Sequence block, including the block name, specified as a string or character vector.

Example: 'FanSpeedTestHarness/Test Sequence'

### **name** — Symbol name

character vector

Name of the symbol, specified as a character vector. For the symbol that controls the active scenario, you can change only its name using the 'Name' name-value pair.

Example: 'theta'

### **Name-Value Pair Arguments**

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

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

Find valid name-value pairs by applying `sltest.testsequence.readSymbol` to the existing symbol.

Example: 'DataType', 'single', 'Scope', 'Constant'

## See Also

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

## Topics

"Programmatically Create a Test Sequence"

## Introduced in R2017a

## sltest.testsequence.editTransition

Edit test sequence step transition

### Syntax

```
sltest.testsequence.editTransition(blockPath,stepPath,index,Name,Value)
```

### Description

`sltest.testsequence.editTransition(blockPath,stepPath,index,Name,Value)` edits transition index in `stepPath` of the Test Sequence block `blockPath`. Transition properties are specified by `Name,Value`.

### Examples

#### Add and Edit a Test Step Transition

This example adds a transition to a test step, then changes the transition's index, condition, and next step of the first transition in the step.

1. Load the model.

```
Model = 'sltestRollRefTestExample';
load_system(Model);
```

2. Add a transition to the step `AttitudeLevels.APEngage.LowRoll`. The transition destination is the step `AttitudeLevels.APEngage_End`.

```
sltest.testsequence.addTransition('sltestRollRefTestExample/Test Sequence',...
'AttitudeLevels.APEngage_LowRoll','TurnKnob ~= 0',...
'AttitudeLevels.APEngagement_End')
```

3. Edit the transition index, condition, and next step of the first transition.

```
sltest.testsequence.editTransition('sltestRollRefTestExample/Test Sequence',...
'AttitudeLevels.APEngage_LowRoll',1,'Index',2,...
'NextStep','AttitudeLevels.APEngage_HighRoll',...
'Condition','duration(DD_PhiRef == 0,sec) >= 5')
```

4. Close the model.

```
close_system(Model,0)
```

### Input Arguments

#### **blockPath** — Test Sequence block path

string | character vector

Path to a Test Sequence block, including the block name, specified as a string or character vector.

Example: 'FanSpeedTestHarness/Test Sequence'

**stepPath — Test step name and hierarchy level**

character vector

Path of the step in the Test Sequence block, specified as a character vector. The path includes the step location in the Test Sequence hierarchy, using . to separate hierarchy levels. If the Test Sequence block is using scenarios, add the scenario name that contains the step to the beginning of the step path, for example, Scenario\_2.SystemHeatingTest.InitializeHeating.

Example: 'SystemHeatingTest.InitializeHeating'

**index — Transition index**

integer

Integer specifying the transition in the test step to be edited. Corresponds to the integer displayed in the Transition cell of the Test Sequence Editor.

Example: 3

**Name-Value Pair Arguments**

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

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

Example: 'Condition','error == 1','NextStep','Stop','Index', 3

**Condition — Condition on which the transition executes**

character vector

The condition on which the transition executes, specified as a character vector. To execute the transition, enter a valid logical expression.

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

**NextStep — Destination step of the transition**

character vector

The name of the destination step of the transition, which is next in the sequence if the transition condition is satisfied.

Example: 'RampAngle'

**Index — Transition index**

integer

Integer specifying the new transition index to be applied

Example: 'Index',2

**See Also**

sltest.testsequence.addStep | sltest.testsequence.addTransition |  
sltest.testsequence.deleteTransition | sltest.testsequence.readTransition

**Topics**

“Programmatically Create a Test Sequence”

**Introduced in R2017a**



# sltest.testsequence.find

Find Test Sequence blocks

## Syntax

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

## Description

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

## Examples

### Find Open Test Sequence Blocks

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

Open the model and test harness.

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

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

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

```
blocks =
```

```
1×2 cell array
```

```
Column 1
```

```
'SimpleTrackerHarn...'
```

```
Column 2
```

```
'SimpleTrackerHarn...'
```

## See Also

```
sltest.testsequence.findStep | sltest.testsequence.findSymbol |
sltest.testsequence.getProperty | sltest.testsequence.readStep |
sltest.testsequence.readSymbol | sltest.testsequence.readTransition
```

**Introduced in R2016a**

## sltest.testsequence.findStep

Find test sequence steps

### Syntax

```
steps = sltest.testsequence.findStep(Name,Value)
```

### Description

`steps = sltest.testsequence.findStep(Name,Value)` returns a cell array `steps` listing Test Sequence steps that match properties specified by `Name,Value` pairs.

### Examples

#### Find a test step in a Test Sequence block

This example finds a test step in a Test Sequence block.

1. Load the model.

```
Model = 'sltestRollRefTestExample';
load_system(Model)
```

2. Find test sequence steps that contain the case-insensitive string `apeng`.

```
steps = sltest.testsequence.findStep('sltestRollRefTestExample/Test Sequence',...
 'Name','[Aa][Pp][Ee]ng','RegExp','on')
```

```
steps = 1x10 cell
 Columns 1 through 3
 {'AttitudeLevels...'} {'AttitudeLevels...'} {'AttitudeLevels...'}
 Columns 4 through 6
 {'AttitudeLevels...'} {'AttitudeLevels...'} {'AttitudeLevels...'}
 Columns 7 through 9
 {'AttitudeLevels...'} {'AttitudeLevels...'} {'AttitudeLevels...'}
 Column 10
 {'AttitudeLevels...'}

```

```
steps(3)
```

```
ans = 1x1 cell array
 {'AttitudeLevels.APEngage_LowRoll.EngageAP_Low'}
```

3. Close the model.

```
close_system(Model,0)
```

## Input Arguments

### Name-Value Pair Arguments

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

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

Example: 'Name', '[Aa][Pp][Ee]ng', 'RegExp', 'on'

#### Name — Step name

character vector

The name of the test steps to search

Example: 'Name', 'Engage'

Example: 'Name', '[Aa][Pp][Ee]ng'

#### CaseSensitive — Specify case-sensitive search

'on' | 'off'

Specifies case

Example: 'CaseSensitive', 'on'

#### RegExp — Specify regular expression search

'on' | 'off'

Specify whether to search the step names using Name as a regular expression

Example: 'RegExp', 'on'

## Output Arguments

### steps — Test sequence steps

cell array

Cell array of test steps matching search criteria

Example: 1×10 cell array

## See Also

sltest.testsequence.editStep | sltest.testsequence.find |  
sltest.testsequence.findSymbol | sltest.testsequence.readStep |  
sltest.testsequence.readSymbol | sltest.testsequence.readTransition

**Introduced in R2017a**

# sltest.testsequence.findSymbol

Find Test Sequence block symbols

## Syntax

```
symbols = sltest.testsequence.findSymbol(blockPath,Name,Value)
```

## Description

`symbols = sltest.testsequence.findSymbol(blockPath,Name,Value)` returns symbols in the Test Sequence block `blockPath` matching properties specified by `Name,Value` pairs. Symbols include data, messages, function calls, and triggers.

## Examples

### Find, read, and edit a Test Sequence block data symbol

This example edits constant `DurationLimit` in the Test Sequence block, changing it to a local variable of single data type.

1. Load the model.

```
Model = 'sltestRollRefTestExample';
load_system(Model)
```

2. Search for data symbols containing the word duration.

```
data_names = sltest.testsequence.findSymbol...
 ('sltestRollRefTestExample/Test Sequence', 'Name', '[Dd]uration', ...
 'RegExp', 'on', 'Kind', 'Data')

data_names = 1x1 cell array
 {'DurationLimit'}
```

3. Read the properties of the `DurationLimit` constant.

```
dLProperties = sltest.testsequence.readSymbol...
 ('sltestRollRefTestExample/Test Sequence', data_names{1})

dLProperties = struct with fields:
 Kind: 'Data'
 Scope: 'Constant'
 DataType: 'double'
 Description: ''
 Document: ''
 InitialValue: '5'
 Name: 'DurationLimit'
 Size: ''
 Tag: []
```

4. Change DurationLimit to a local variable of single data type.

```
sltest.testsequence.editSymbol('sltestRollRefTestExample/Test Sequence',...
 data_names{1}, 'Scope', 'Local', 'DataType', 'single')
```

5. Close the model.

```
close_system(Model,0)
```

## Input Arguments

### blockPath — Test Sequence block path

string | character vector

Path to a Test Sequence block, including the block name, specified as a string or character vector.

Example: 'FanSpeedTestHarness/Test Sequence'

### Name-Value Pair Arguments

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

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

Example: 'Kind', 'Message', 'Scope', 'Output'

Example: 'Kind', 'Data', 'Name', '[Aa]ngle', 'RegExp', 'on'

### Kind — Data symbol type

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

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

Example: 'Data'

### Scope — Data symbol scope

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

Data symbol scope, specified as a character vector.

Example: 'Parameter'

### Name — Symbol name

character vector

The name of the test symbols to search

Example: 'Name', 'Engage'

Example: 'Name', '[Dd]uration'

### CaseSensitive — Specify case-sensitive search

'on' | 'off'

Specifies case

Example: 'CaseSensitive', 'on'

**RegExp — Specify regular expression search**

'on' | 'off'

Specify whether to search the step names using Name as a regular expression

Example: 'RegExp', 'on'

**Output Arguments**

**symbols — Block symbols**

cell array

Cell array of Test Sequence block symbols matching search criteria

Example: cell

**See Also**

sltest.testsequence.deleteSymbol | sltest.testsequence.editSymbol |  
sltest.testsequence.find | sltest.testsequence.findStep |  
sltest.testsequence.readSymbol

**Topics**

“Programmatically Create a Test Sequence”

**Introduced in R2017a**

# sltest.testsequence.getActiveScenario

**Package:** sltest.testsequence

Get Test Sequence block active scenario

## Syntax

```
active_scenario = sltest.testsequence.getActiveScenario(blockPath)
```

## Description

`active_scenario = sltest.testsequence.getActiveScenario(blockPath)` returns the name of the active scenario for the Test Sequence block specified by `blockPath`. The active scenario is the scenario that runs during model simulation.

## Examples

### Get Active Test Sequence Scenario

Set the Test Sequence block in the `sltestRollRefTestExample` model to use scenarios. Add another scenario named `new_Scenario` and activate it. Then, get the name of the currently active scenario. You can use `getActiveScenario` only if you have set `sltest.testsequence.setScenarioControlSource` to `sltest.testsequence.ScenarioControlSource.Block`, which sets the active scenario control to a Test Sequence block instead of a variable in the workspace.

```
Model = 'sltestRollRefTestExample';
load_system(Model);

sltest.testsequence.useScenario...
('sltestRollRefTestExample/Test Sequence',...
'Scenario_1');

sltest.testsequence.addScenario...
('sltestRollRefTestExample/Test Sequence',...
'new_Scenario');

sltest.testsequence.activateScenario...
('sltestRollRefTestExample/Test Sequence',...
'new_Scenario');

active_scenario = sltest.testsequence.getActiveScenario...
('sltestRollRefTestExample/Test Sequence')

active_scenario =

 'new_Scenario'
```

Close the model without saving it.

```
close_system(Model,0)
```

## Input Arguments

### **blockPath** — Test Sequence block path

string | character vector

Path to a Test Sequence block, including the block name, specified as a string or character vector.

Example: 'FanSpeedTestHarness/Test Sequence'

## Output Arguments

### **active\_scenario** — Active Test Sequence block scenario name

character vector

Active Test Sequence block scenario name, returned as a character vector.

## See Also

`sltest.testsequence.useScenario` | `setScenarioControlSource` | `activateScenario` | `getAllScenarios`

## Topics

“Programmatically Create and Run Test Sequence Scenarios”

**Introduced in R2020b**



# sltest.testsequence.getAllScenarios

**Package:** sltest.testsequence

Get names of all Test Sequence block scenarios

## Syntax

```
all_scenarios = sltest.testsequence.getAllScenarios(blockPath)
```

## Description

`all_scenarios = sltest.testsequence.getAllScenarios(blockPath)` returns the names of all scenarios in a Test Sequence block specified by `blockPath`.

## Examples

### Get All Test Sequence Scenarios

Set the Test Sequence block in the `sltestRollRefTestExample` model to use scenarios. Add another scenario named `new_Scenario`. Then, get the names of all scenarios in the Test Sequence block.

```
Model = 'sltestRollRefTestExample';
load_system(Model);

sltest.testsequence.useScenario...
('sltestRollRefTestExample/Test Sequence',...
 'Scenario_1');

sltest.testsequence.addScenario...
('sltestRollRefTestExample/Test Sequence',...
 'new_Scenario');

all_scenarios = sltest.testsequence.getAllScenarios...
('sltestRollRefTestExample/

all_scenarios =

 1x2 cell array

 {'Scenario_1'} {'new_Scenario'}
```

Close the model without saving it.

```
close_system(Model,0)
```

## Input Arguments

**blockPath** — Test Sequence block path

string | character vector

Path to a Test Sequence block, including the block name, specified as a string or character vector.

Example: 'FanSpeedTestHarness/Test Sequence'

## **Output Arguments**

**all\_scenarios** — Names of all scenarios in Test Sequence block

cell array

Names of all scenarios in Test Sequence block, returned as a cell array.

## **See Also**

`addScenario` | `sltest.testsequence.useScenario`

## **Topics**

“Programmatically Create and Run Test Sequence Scenarios”

**Introduced in R2020b**

# sltest.testsequence.getProperty

Get Test Sequence block properties

## Syntax

```
blockInfo = sltest.testsequence.getProperty(blockPath)
blockInfo = sltest.testsequence.getProperty(blockPath,propertyName)
```

## Description

`blockInfo = sltest.testsequence.getProperty(blockPath)` returns a `blockInfo` structure containing properties of the Test Sequence block specified by `blockPath`.

`blockInfo = sltest.testsequence.getProperty(blockPath,propertyName)` returns `blockInfo`, containing the value of `propertyName`.

## Examples

### Programmatically Return and Set Test Sequence Block Properties

This example gets and sets properties for a Test Sequence block using the programmatic interface.

1. Load the model.

```
Model = 'sltestRollRefTestExample';
load_system(Model)
```

2. Get properties of the Test Sequence block.

```
blockInfo = sltest.testsequence.getProperty([Model '/Test Sequence'])
```

```
blockInfo = struct with fields:
 Name: 'Test Sequence'
 UpdateMethod: 'INHERITED'
 SampleTime: '-1'
 Description: ''
 Document: ''
 Tag: []
 SupportVariableSizing: 1
 SaturateOnIntegerOverflow: 1
 InputFimath: 'fimath(.....)'
 EmlDefaultFimath: 'Same as MATLAB Default'
 EnableActiveStepData: 0
 ActiveStepDataSymbol: ''
 ActiveStepDataType: 'Enum'
 ScenarioParameter: ''
 Semantics: 'StateflowCompatible'
```

3. Get the Test Sequence block update method.

```
blockUpdateMethod = sltest.testsequence.getProperty(...
 [Model '/Test Sequence'], 'UpdateMethod')
```

```
blockUpdateMethod =
'INHERITED'
```

4. Change the Test Sequence block update method and sample time.

```
sltest.testsequence.setProperty([Model '/Test Sequence'], ...
 'UpdateMethod', 'Discrete', 'SampleTime', '0.1')
```

5. Enable active step data and set the data type to String. Use this data type if you have duplicate step names in a single test sequence or across scenarios in the Test Sequence block.

```
sltest.testsequence.setProperty([Model '/Test Sequence'], ...
 'EnableActiveStepData', 1, 'ActiveStepDataType', 'String');
```

6. Check the changes.

```
blockInfo = sltest.testsequence.getProperty([Model '/Test Sequence'])
```

```
blockInfo = struct with fields:
 Name: 'Test Sequence'
 UpdateMethod: 'DISCRETE'
 SampleTime: '0.1'
 Description: ''
 Document: ''
 Tag: []
 SupportVariableSizing: 1
 SaturateOnIntegerOverflow: 1
 InputFimath: 'fimath(.....'
 EmlDefaultFimath: 'Same as MATLAB Default'
 EnableActiveStepData: 1
 ActiveStepDataSymbol: 'Active_Step'
 ActiveStepDataType: 'String'
 ScenarioParameter: ''
 Semantics: 'StateflowCompatible'
```

7. Close the model.

```
close_system(Model, 0)
```

## Getting the Active Step Enumeration Name

The

The `ActiveStepEnumName` is a property of the `ActiveStepDataSymbol`, which is a property of the Test Sequence block. To obtain the enumeration name of the active step, use this code:

```
ts = sltest.testsequence.getProperty(...
 'model/Test Sequence');
actstep = sltest.testsequence.readSymbol(...
```

```
'model/Test Sequence',ts.ActiveStepDataSymbol);
enum_name = actstep.ActiveStepEnumName
```

## Input Arguments

### blockPath — Test Sequence block path

string | character vector

Path to a Test Sequence block, including the block name, specified as a string or character vector.

Example: 'FanSpeedTestHarness/Test Sequence'

### propertyName — Test Sequence block property name

'Name' | 'UpdateMethod' | 'SampleTime' | 'Description' | 'Document' | 'Tag' |  
'SupportVariableSizing' | 'SaturateOnIntegerOverflow' | 'InputFimath' |  
'FimathForFiConstructors' | 'EnableActiveStepData' | 'ActiveStepDataSymbol' |  
'ActiveStepDataType' | 'ScenarioParameter' | 'Semantics'

Name of a particular Test Sequence block property for which to get a value.

Example: 'Description'

## Output Arguments

### blockInfo — Block properties or property value

struct | character vector | logical

Output of block properties, or the value of a particular block property

Example: struct with fields

Example: char array

Example: logical

## See Also

sltest.testsequence.find | sltest.testsequence.readStep |  
sltest.testsequence.readSymbol | sltest.testsequence.readTransition |  
sltest.testsequence.setProperty

## Topics

“Programmatically Create a Test Sequence”

## Introduced in R2017a

# sltest.testsequence.getScenarioControlSource

**Package:** sltest.testsequence

Get source that controls Test Sequence active scenario

## Syntax

```
scenarioCtrlSource = sltest.testsequence.getScenarioControlSource(blockPath)
```

## Description

`scenarioCtrlSource = sltest.testsequence.getScenarioControlSource(blockPath)` returns the source that controls the active scenario for the Test Sequence block specified by `blockPath`.

## Examples

### Get Source that Controls Active Scenario

Set the Test Sequence block in the `sltestRollRefTestExample` model to use scenarios. Set the control source to the workspace. Then, get the name of the source that controls the active scenario.

```
Model = 'sltestRollRefTestExample';
load_system(Model);

sltest.testsequence.useScenario...
 ('sltestRollRefTestExample/Test Sequence',...
 'Scenario_1');
sltest.testsequence.setScenarioControlSource...
 ('sltestRollRefTestExample/Test Sequence',...
 sltest.testsequence.ScenarioControlSource.Workspace);

scenarioControlSource = ...
 sltest.testsequence.getScenarioControlSource...
 ('sltestRollRefTestExample/Test Sequence')

scenarioCtrlSource =

 ScenarioControlSource enumeration

 Workspace

Close the model without saving it.
close_system(Model,0)
```

## Input Arguments

**blockPath** — Test Sequence block path

string | character vector

Path to a Test Sequence block, including the block name, specified as a string or character vector.

Example: 'FanSpeedTestHarness/Test Sequence'

## Output Arguments

### **scenarioCtrlSource** — Active scenario control source

sltest.testsequence.ScenarioControlSource.Block |  
sltest.testsequence.ScenarioControlSource.Workspace

Active scenario control source, returned as either of these enumerated values:

- `sltest.testsequence.ScenarioControlSource.Block` — The active scenario is controlled by setting it in the Test Sequence block.
- `sltest.testsequence.ScenarioControlSource.Workspace` — The active scenario is controlled by a workspace variable that specifies the index of the active scenario. The variable can be in the base workspace, model workspace, or data dictionary.

## See Also

sltest.testsequence.useScenario | setScenarioControlSource

## Topics

“Programmatically Create and Run Test Sequence Scenarios”

## Introduced in R2020b

## sltest.testsequence.isUsingScenarios

**Package:** sltest.testsequence

Determine whether Test Sequence block uses scenarios

### Syntax

```
TF = sltest.testsequence.isUsingScenarios(blockPath)
```

### Description

TF = sltest.testsequence.isUsingScenarios(blockPath) returns whether the Test Sequence block specified by blockPath uses scenarios.

### Examples

#### Check Whether a Test Sequence Block Uses Scenarios

Check whether the Test Sequence block in the sltestRollRefTestExample model uses scenarios. In this case, the block has not been set to use scenarios, so isUsingScenarios returns false.

```
Model = 'sltestRollRefTestExample';
load_system(Model);

use_scen = sltest.testsequence.isUsingScenarios...
 ('sltestRollRefTestExample/Test Sequence')

use_scen =

 logical

 1
```

Close the model without saving it.

```
close_system(Model,0)
```

### Input Arguments

#### blockPath — Test Sequence block path

string | character vector

Path to a Test Sequence block, including the block name, specified as a string or character vector.

Example: 'FanSpeedTestHarness/Test Sequence'

### See Also

sltest.testsequence.useScenario



**Topics**

“Programmatically Create and Run Test Sequence Scenarios”

**Introduced in R2020b**

## sltest.testsequence.newBlock

Create Test Sequence block

### Syntax

```
blockID = sltest.testsequence.newBlock(blockPath)
```

### Description

`blockID = sltest.testsequence.newBlock(blockPath)` adds a Test Sequence block specified by `blockPath`, returning the handle `blockID`.

### Examples

#### Create a Test Sequence Block and Get Block Properties

This example shows how to create a Test Sequence block programmatically, and get properties for the block, which can be used in Name, Value pairs for `sltest.testsequence.setProperty`.

1. Create a model and a Test Sequence block.

```
new_system('tsb_model');
sltest.testsequence.newBlock('tsb_model/Test Sequence');
```

2. Get properties of the Test Sequence block.

```
block_properties = sltest.testsequence.getProperty...
 ('tsb_model/Test Sequence')

block_properties = struct with fields:
 Name: 'Test Sequence'
 UpdateMethod: 'INHERITED'
 SampleTime: '-1'
 Description: ''
 Document: ''
 Tag: []
 SupportVariableSizing: 1
 SaturateOnIntegerOverflow: 1
 InputFimath: 'fimath(.....)'
 EmlDefaultFimath: 'Same as MATLAB Default'
 EnableActiveStepData: 0
 ActiveStepDataSymbol: ''
 ActiveStepDataType: 'String'
 ScenarioParameter: ''
 Semantics: 'StateflowCompatible'
```

3. Close the model.

```
close_system('tsb_model',0)
```

## Input Arguments

### **blockPath — Test Sequence block path**

string | character vector

Path to a Test Sequence block, including the block name, specified as a string or character vector.

Example: 'FanSpeedTestHarness/Test Sequence'

## Output Arguments

### **blockID — Handle to the block**

double

Block handle, returned as a double.

Example: 190.0021

## See Also

`sltest.testsequence.find` | `sltest.testsequence.getProperty` |  
`sltest.testsequence.setProperty`

## Topics

“Programmatically Create a Test Sequence”

**Introduced in R2017a**

## sltest.testsequence.readStep

Read Test Sequence block steps

### Syntax

```
stepInfo = sltest.testsequence.readStep(blockPath,stepPath)
stepInfo = sltest.testsequence.readStep(blockPath,stepPath,Property)
```

### Description

`stepInfo = sltest.testsequence.readStep(blockPath,stepPath)` returns a struct `stepInfo` of properties for the test step `stepPath` in the Test Sequence block `blockPath`.

`stepInfo = sltest.testsequence.readStep(blockPath,stepPath,Property)` returns the value `stepInfo` of the Property for the test step.

### Examples

#### Read Test Step and Transition Properties

This example reads properties of a test step and a transition in a Test Sequence block.

1. Load the model.

```
Model = 'sltestRollRefTestExample';
load_system(Model)
```

2. Read the properties of the test step `SetMedPhi`, which is a sub-step of `AttitudeLevels.APEngage_MedRoll`.

```
stepInfo = sltest.testsequence.readStep([Model,'/Test Sequence'],...
 'AttitudeLevels.APEngage_MedRoll.SetMedPhi')
```

```
stepInfo = struct with fields:
 Name: 'AttitudeLevels.APEngage_MedRoll.SetMedPhi'
 Action: 'Phi = 11.5;...'
 IsWhenStep: 0
 IsWhenSubStep: 0
 Description: ''
 Index: 1
 TransitionCount: 1
```

3. Read the action of the same step.

```
stepAction = sltest.testsequence.readStep([Model,'/Test Sequence'],...
 'AttitudeLevels.APEngage_MedRoll.SetMedPhi','Action')
```

```
stepAction =
 'Phi = 11.5;
 APEng = false;'
```

4. Read the transition properties for the parent step.

```
xInfo = sltest.testsequence.readTransition([Model, '/Test Sequence'], ...
 'AttitudeLevels.APEngage_MedRoll', 1)
```

```
xInfo = struct with fields:
 Step: 'AttitudeLevels.APEngage_MedRoll'
 Index: 1
 Condition: 'duration(DD_PhiRef == 0,sec) >= DurationLimit'
 NextStep: 'AttitudeLevels.APEngage_HighRoll'
```

5. Close the model.

```
close_system(Model, 0)
```

## Input Arguments

### blockPath — Test Sequence block path

string | character vector

Path to a Test Sequence block, including the block name, specified as a string or character vector.

Example: 'FanSpeedTestHarness/Test Sequence'

### stepPath — Test step name and hierarchy level

character vector

Path of the step in the Test Sequence block, specified as a character vector. The path includes the step location in the Test Sequence hierarchy, using . to separate hierarchy levels. If the Test Sequence block is using scenarios, add the scenario name that contains the step to the beginning of the step path, for example, Scenario\_2.SystemHeatingTest.InitializeHeating.

Example: 'SystemHeatingTest.InitializeHeating'

### Property — Step property

'Name' | 'Action' | 'IsWhenStep' | 'IsWhenSubStep' | 'Description' | 'TransitionCount'

Property of the test step, specified as a character vector.

Example: 'TransitionCount'

## Output Arguments

### stepInfo — Test step properties

struct | character vector | logical | numeric

Properties of the test step.

Example: struct

**See Also**

`sltest.testsequence.deleteStep` | `sltest.testsequence.editStep` |  
`sltest.testsequence.find` | `sltest.testsequence.findStep` |  
`sltest.testsequence.readSymbol` | `sltest.testsequence.readTransition`

**Topics**

“Programmatically Create a Test Sequence”

**Introduced in R2017a**

# sltest.testsequence.readSymbol

Read Test Sequence block symbol properties

## Syntax

```
symbolInfo = sltest.testsequence.readSymbol(blockPath,symbol)
symbolInfo = sltest.testsequence.readSymbol(blockPath,symbol,Property)
```

## Description

`symbolInfo = sltest.testsequence.readSymbol(blockPath,symbol)` returns a struct `symbolInfo` of properties for `symbol` in the Test Sequence block specified by `blockPath`.

`symbolInfo = sltest.testsequence.readSymbol(blockPath,symbol,Property)` returns the value `symbolInfo` of the `Property` for `symbol`.

## Examples

### Find, read, and edit a Test Sequence block data symbol

This example edits constant `DurationLimit` in the Test Sequence block, changing it to a local variable of single data type.

1. Load the model.

```
Model = 'sltestRollRefTestExample';
load_system(Model)
```

2. Search for data symbols containing the word duration.

```
data_names = sltest.testsequence.findSymbol...
 ('sltestRollRefTestExample/Test Sequence', 'Name', '[Dd]uration', ...
 'RegExp', 'on', 'Kind', 'Data')
```

```
data_names = 1x1 cell array
 {'DurationLimit'}
```

3. Read the properties of the `DurationLimit` constant.

```
dlProperties = sltest.testsequence.readSymbol...
 ('sltestRollRefTestExample/Test Sequence', data_names{1})
```

```
dlProperties = struct with fields:
 Kind: 'Data'
 Scope: 'Constant'
 DataType: 'double'
 Description: ''
 Document: ''
 InitialValue: '5'
 Name: 'DurationLimit'
```

```
Size: ''
Tag: []
```

4. Change `DurationLimit` to a local variable of single data type.

```
sltest.testsequence.editSymbol('sltestRollRefTestExample/Test Sequence',...
 data_names{1}, 'Scope', 'Local', 'DataType', 'single')
```

5. Close the model.

```
close_system(Model,0)
```

## Input Arguments

### **blockPath** — Test Sequence block path

string | character vector

Path to a Test Sequence block, including the block name, specified as a string or character vector.

Example: 'FanSpeedTestHarness/Test Sequence'

### **symbol** — Symbol name

character vector

Test Sequence block symbol name, specified as a character vector. Symbols include data, messages, function calls, and triggers used as inputs, outputs, local variables, constants, parameters, or data store memory in the Test Sequence block.

Example: 'DurationLimit'

### **Property** — Symbol property

character vector

Test Sequence block symbol property, specified as a character vector. To find valid properties for a particular symbol, read properties of the symbol using `sltest.testsequence.readSymbol(blockPath, symbol)`.

Example: 'Kind'

Example: 'Scope'

Example: 'DataType' 'Description'

## Output Arguments

### **symbolInfo** — Test Sequence block symbol properties

struct | character vector | logical | numeric

Properties of the Test Sequence block symbol.

Example: struct



## See Also

sltest.testsequence.addSymbol | sltest.testsequence.deleteSymbol |  
sltest.testsequence.findSymbol | sltest.testsequence.readStep |  
sltest.testsequence.readTransition

## Topics

“Programmatically Create a Test Sequence”

**Introduced in R2017a**

## sltest.testsequence.readTransition

Read properties of test sequence transition

### Syntax

```
transitionInfo = sltest.testsequence.readTransition(blockPath,stepPath,index)
transitionInfo = sltest.testsequence.readTransition(blockPath,stepPath,index,
Property)
```

### Description

`transitionInfo = sltest.testsequence.readTransition(blockPath,stepPath,index)` returns a struct `transitionInfo` of properties for the transition `index`, in the test step `stepPath` of the Test Sequence block `blockPath`.

`transitionInfo = sltest.testsequence.readTransition(blockPath,stepPath,index, Property)` returns the value `transitionInfo` of the `Property` for the transition.

### Examples

#### Read Test Step and Transition Properties

This example reads properties of a test step and a transition in a Test Sequence block.

1. Load the model.

```
Model = 'sltestRollRefTestExample';
load_system(Model)
```

2. Read the properties of the test step `SetMedPhi`, which is a sub-step of `AttitudeLevels.APEngage_MedRoll`.

```
stepInfo = sltest.testsequence.readStep([Model,'/Test Sequence'],...
'AttitudeLevels.APEngage_MedRoll.SetMedPhi')
```

```
stepInfo = struct with fields:
 Name: 'AttitudeLevels.APEngage_MedRoll.SetMedPhi'
 Action: 'Phi = 11.5;...'
 IsWhenStep: 0
 IsWhenSubStep: 0
 Description: ''
 Index: 1
 TransitionCount: 1
```

3. Read the action of the same step.

```
stepAction = sltest.testsequence.readStep([Model,'/Test Sequence'],...
'AttitudeLevels.APEngage_MedRoll.SetMedPhi','Action')
```

```
stepAction =
 'Phi = 11.5;
```

```
APEng = false;
```

4. Read the transition properties for the parent step.

```
xInfo = sltest.testsequence.readTransition([Model, '/Test Sequence'], ...
 'AttitudeLevels.APEngage_MedRoll', 1)
xInfo = struct with fields:
 Step: 'AttitudeLevels.APEngage_MedRoll'
 Index: 1
 Condition: 'duration(DD_PhiRef == 0,sec) >= DurationLimit'
 NextStep: 'AttitudeLevels.APEngage_HighRoll'
```

5. Close the model.

```
close_system(Model, 0)
```

## Input Arguments

### **blockPath** — Test Sequence block path

string | character vector

Path to a Test Sequence block, including the block name, specified as a string or character vector.

Example: 'FanSpeedTestHarness/Test Sequence'

### **stepPath** — Test step name and hierarchy level

character vector

Path of the step in the Test Sequence block, specified as a character vector. The path includes the step location in the Test Sequence hierarchy, using `.` to separate hierarchy levels. If the Test Sequence block is using scenarios, add the scenario name that contains the step to the beginning of the step path, for example, `Scenario_2.SystemHeatingTest.InitializeHeating`.

Example: 'SystemHeatingTest.InitializeHeating'

### **index** — Transition index

integer

Integer specifying the transition in the test step to be edited. Corresponds to the integer displayed in the Transition cell of the Test Sequence Editor.

Example: 3

### **Property** — Transition property

character vector

Transition property, specified as a character vector. To find valid properties for a particular symbol, read properties of the symbol using

```
sltest.testsequence.readTransition(blockPath, stepPath, index).
```

Example: 'Step'

Example: 'Index'

Example: 'Condition'

Example: 'NextStep'

## **Output Arguments**

### **transitionInfo – Transition properties**

struct | character vector | numeric

Properties of the test step transition.

Example: struct

## **See Also**

sltest.testsequence.addTransition | sltest.testsequence.deleteTransition |  
sltest.testsequence.editTransition | sltest.testsequence.findStep |  
sltest.testsequence.readStep | sltest.testsequence.readSymbol

## **Topics**

“Programmatically Create a Test Sequence”

**Introduced in R2017a**

# sltest.testsequence.setProperty

Set Test Sequence block properties

## Syntax

```
sltest.testsequence.setProperty(blockPath,Name,Value)
```

## Description

`sltest.testsequence.setProperty(blockPath,Name,Value)` sets properties of the Test Sequence block specified by `blockPath` according to one or more `Name, Value` pairs. Obtain valid properties using `sltest.testsequence.getProperty`.

---

**Note** All properties, except `ActiveStepDataSymbol` and `ScenarioParameter`, are writable. You cannot change the value of `ActiveStepDataSymbol` or `ScenarioParameter` using `sltest.testsequence.setProperty`. You can, however, rename the symbols using `sltest.testsequence.editSymbol`.

---

## Examples

### Create a Test Sequence Block and Get Block Properties

This example shows how to create a Test Sequence block programmatically, and get properties for the block, which can be used in `Name, Value` pairs for `sltest.testsequence.setProperty`.

1. Create a model and a Test Sequence block.

```
new_system('tsb_model');
sltest.testsequence.newBlock('tsb_model/Test Sequence');
```

2. Get properties of the Test Sequence block.

```
block_properties = sltest.testsequence.getProperty...
 ('tsb_model/Test Sequence')

block_properties = struct with fields:
 Name: 'Test Sequence'
 UpdateMethod: 'INHERITED'
 SampleTime: '-1'
 Description: ''
 Document: ''
 Tag: []
 SupportVariableSizing: 1
 SaturateOnIntegerOverflow: 1
 InputFimath: 'fimath(.....)'
 EmlDefaultFimath: 'Same as MATLAB Default'
 EnableActiveStepData: 0
 ActiveStepDataSymbol: ''
 ActiveStepDataType: 'String'
```

```
ScenarioParameter: ''
Semantics: 'StateflowCompatible'
```

3. Close the model.

```
close_system('tsb_model',0)
```

### Programmatically Return and Set Test Sequence Block Properties

This example gets and sets properties for a Test Sequence block using the programmatic interface.

1. Load the model.

```
Model = 'sltestRollRefTestExample';
load_system(Model)
```

2. Get properties of the Test Sequence block.

```
blockInfo = sltest.testsequence.getProperty([Model '/Test Sequence'])
```

```
blockInfo = struct with fields:
 Name: 'Test Sequence'
 UpdateMethod: 'INHERITED'
 SampleTime: '-1'
 Description: ''
 Document: ''
 Tag: []
 SupportVariableSizing: 1
 SaturateOnIntegerOverflow: 1
 InputFimath: 'fimath(.....'
 EmlDefaultFimath: 'Same as MATLAB Default'
 EnableActiveStepData: 0
 ActiveStepDataSymbol: ''
 ActiveStepDataType: 'Enum'
 ScenarioParameter: ''
 Semantics: 'StateflowCompatible'
```

3. Get the Test Sequence block update method.

```
blockUpdateMethod = sltest.testsequence.getProperty(...
 [Model '/Test Sequence'],'UpdateMethod')
```

```
blockUpdateMethod =
'INHERITED'
```

4. Change the Test Sequence block update method and sample time.

```
sltest.testsequence.setProperty([Model '/Test Sequence'],...
 'UpdateMethod','Discrete','SampleTime','0.1')
```

5. Enable active step data and set the data type to String. Use this data type if you have duplicate step names in a single test sequence or across scenarios in the Test Sequence block.

```
sltest.testsequence.setProperty([Model '/Test Sequence'],...
 'EnableActiveStepData',1,'ActiveStepDataType','String');
```

6. Check the changes.

```
blockInfo = sltest.testsequence.getProperty([Model '/Test Sequence'])
blockInfo = struct with fields:
 Name: 'Test Sequence'
 UpdateMethod: 'DISCRETE'
 SampleTime: '0.1'
 Description: ''
 Document: ''
 Tag: []
 SupportVariableSizing: 1
 SaturateOnIntegerOverflow: 1
 InputFimath: 'fimath(.....)'
 EmlDefaultFimath: 'Same as MATLAB Default'
 EnableActiveStepData: 1
 ActiveStepDataSymbol: 'Active_Step'
 ActiveStepDataType: 'String'
 ScenarioParameter: ''
 Semantics: 'StateflowCompatible'
```

7. Close the model.

```
close_system(Model,0)
```

## Setting the Active Step Enumeration Name

The

The `ActiveStepEnumName` is a property of the `ActiveStepDataSymbol`, which is a property of the Test Sequence block. To set the enumeration name of the active step to `NewEnumName`, use this code:

```
ts = sltest.testsequence.getProperty(...
 'model/Test Sequence');
sltest.testsequence.editSymbol(...
 'model/Test Sequence',...
 ts.ActiveStepDataSymbol,...
 'ActiveStepEnumName', 'NewEnumName')
```

## Input Arguments

### blockPath — Test Sequence block path

string | character vector

Path to a Test Sequence block, including the block name, specified as a string or character vector.

Example: 'FanSpeedTestHarness/Test Sequence'

### Name-Value Pair Arguments

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

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

Example: 'Description', 'Temperature cycle', 'EnableActiveStepData', true

Valid name-value pairs are block-specific. Obtain properties for the block using `sltest.testsequence.getProperty`.

### **See Also**

`sltest.testsequence.find` | `sltest.testsequence.getProperty` |  
`sltest.testsequence.newBlock` | `sltest.testsequence.editSymbol`

### **Topics**

“Programmatically Create a Test Sequence”

### **Introduced in R2017a**



# sltest.testsequence.setScenarioControlSource

**Package:** sltest.testsequence

Set source controlling Test Sequence active scenario

## Syntax

```
sltest.testsequence.setScenarioControlSource(blockPath,scenarioControlSource)
```

## Description

`sltest.testsequence.setScenarioControlSource(blockPath,scenarioControlSource)` sets the source, specified by `scenarioControlSource`, that controls the active scenario for the Test Sequence block specified by `blockPath`.

## Examples

### Set Source that Controls Active Scenario

Set the Test Sequence block in the `sltestRollRefTestExample` model to use scenarios. Set the source that controls the active scenario for the Test Sequence block in the `sltestRollRefTestModel` to the workspace. Close the model without saving it.

```
Model = 'sltestRollRefTestExample';
load_system(Model);

sltest.testsequence.useScenario...
('sltestRollRefTestExample/Test Sequence',...
 'Scenario_1');

sltest.testsequence.activateScenario...
('sltestRollRefTestExample/Test Sequence',...
 'Scenario_1');

sltest.testsequence.setScenarioControlSource...
('sltestRollRefTestExample/Test Sequence',...
 sltest.testsequence.ScenarioControlSource.Workspace);

close_system(Model,0)
```

## Input Arguments

### **blockPath** — Test Sequence block path

string | character vector

Path to a Test Sequence block, including the block name, specified as a string or character vector.

Example: 'FanSpeedTestHarness/Test Sequence'

**scenarioControlSource — Active scenario control source**

`sltest.testsequence.ScenarioControlSource.Block` (default) |  
`sltest.testsequence.ScenarioControlSource.Workspace`

Active scenario control source, specified as either of these enumerated values:

- `sltest.testsequence.ScenarioControlSource.Block` — The active scenario is controlled by setting it in the Test Sequence block.
- `sltest.testsequence.ScenarioControlSource.Workspace` — The active scenario is controlled by a workspace variable that specifies the index of the active scenario. The variable can be in the base workspace, model workspace, or data dictionary.

**See Also**

`getScenarioControlSource` | `sltest.testsequence.useScenario`

**Topics**

“Programmatically Create and Run Test Sequence Scenarios”

**Introduced in R2020b**

# sltest.testsequence.useScenario

**Package:** sltest.testsequence

Convert Test Sequence block to scenario mode

## Syntax

```
sltest.testsequence.useScenario(blockPath,scenarioName)
```

## Description

`sltest.testsequence.useScenario(blockPath,scenarioName)` converts the Test Sequence block, specified by `blockPath`, to use scenarios. Scenarios are tabs in the Test Sequence editor that enable you to define and run separate test sequences in a single Test Sequence block. After conversion, all existing steps in the block are moved to a scenario tab named `scenarioName`. If you convert a Test Sequence block to use scenarios, you cannot revert the block to non-scenario mode, even if it contains only one scenario.

## Examples

### Enable Scenarios in Test Sequence Block

Change the Test Sequence block in the `sltestRollRefTestModel` to use scenarios and name the first scenario `Scenario_1`. Then, close the model without saving it.

```
Model = 'sltestRollRefTestExample';
load_system(Model);

sltest.testsequence.useScenario...
 ('sltestRollRefTestExample/Test Sequence',...
 'Scenario_1');

close_system(Model,0)
```

## Input Arguments

### **blockPath** — Test Sequence block path

string | character vector

Path to a Test Sequence block, including the block name, specified as a string or character vector.

Example: 'FanSpeedTestHarness/Test Sequence'

### **scenarioName** — Name of scenario

character vector | string

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

Example: 'Name', 'Scenario\_1'

**See Also**

activateScenario | addScenario | setScenarioControlSource | isUsingScenarios

**Topics**

“Programmatically Create and Run Test Sequence Scenarios”

**Introduced in R2020b**

# sltestiteration

Create test iteration

## Syntax

```
iterObj = sltestiteration
```

## Description

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

## Examples

### Iterate Over Signal Editor Scenarios

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

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

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

 % Set iteration settings
 setTestParam(testItr, 'SignalEditorScenario', ...
 sltest_signalEditorScenarios{k});

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

## Output Arguments

### iterObj — Test iteration

`sltest.testmanager.TestIteration` object

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

## See Also

`sltest.testmanager.TestIteration` | Signal Editor

**Topics**

“Test Iterations”

“Create and Run Test Cases with Scripts”

**Introduced in R2016a**

# Classes

---

# sltest.Assessment

Access assessment from set

## Description

An `sltest.Assessment` object is an individual assessment result from an `sltest.AssessmentSet` object.

## Creation

Create an `sltest.Assessment` object using `result = get(as, index)` where `as` is an `sltest.AssessmentSet` object.

## Properties

### BlockPath — Path to assessment

fully specified Simulink block path

Path to block containing the assessment. For a Test Sequence block, the sub path is a path to the test step containing the assessment. See `Simulink.SimulationData.BlockPath`.

Example: `Simulink.SimulationData.BlockPath`

### Name — Name of assessment

character vector

Name of the assessment, specified as a character vector. For a `verify()` statement, results in the Test Manager are identified by the name.

Example: `'Simulink:verify_low'`

### Values — Assessment timeseries output

timeseries

Output of the assessment, specified as a timeseries.

Example: `Values: [1x1 timeseries]`

### Result — Assessment result

character vector

Result of the assessment.

Example: `'Fail'`

## Object Functions

`disp` Display results of `sltest.AssessmentSet` or `sltest.Assessment`  
`find` Find assessments in `sltest.AssessmentSet` or `sltest.Assessment` object  
`plot` Plot simulation output data in the Simulation Data Inspector



## Examples

### Get Assessments from a Simulation

This example shows how to simulate a model with `verify` statements and obtain assessment results via the programmatic interface.

### Get the Assessment Set and One Assessment Result

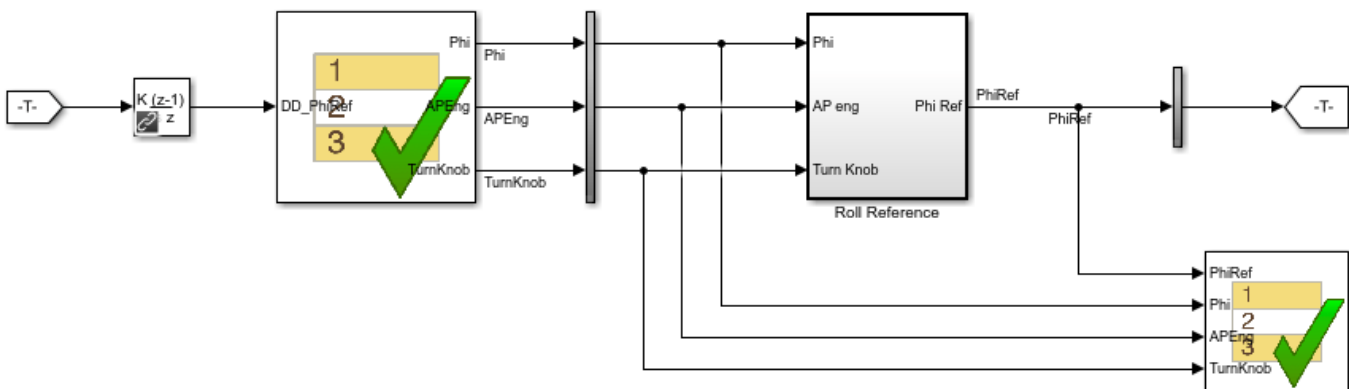
1. Open the model.

```
open_system('sltestRollRefTestExample.slx')
```

```
% Turn the command line warning off for verify() statements
warning off Stateflow:Runtime:TestVerificationFailed
```

This model is used to show how `verify()` statements work in Test Sequence and Test Assessment blocks,

Copyright 2019 The MathWorks, Inc.



Copyright 2016-2020 The MathWorks, Inc.

2. Run the model.

```
s = sim('sltestRollRefTestExample');
```

3. Get the assessment set.

```
as = sltest.getAssessments('sltestRollRefTestExample');
```

4. Get assessment 3 from the assessment set.

```
as3 = get(as,3);
```

## Display Results of the Assessment Set and Assessment Result

1. Get summary of the assessment set.

```
asSummary = getSummary(as)
```

```
asSummary =

 struct with fields:

 Total: 6
 Untested: 3
 Passed: 2
 Failed: 1
 Result: Fail
```

2. Display the result of assessment 3.

```
disp(as3)

sltest.Assessment
Package: sltest

Properties:
 Name: 'Simulink:verify_high'
 BlockPath: [1x1 Simulink.SimulationData.BlockPath]
 Values: [1x1 timeseries]
 Result: Fail
```

3. Find untested or failed results in the assessment set.

```
asFailUntested = find(as, 'Result', slTestResult.Fail, '-or', ...
 'Result', slTestResult.Untested)
```

```
asFailUntested =

sltest.AssessmentSet
Summary:
 Total: 4
 Untested: 3
 Passed: 0
 Failed: 1
 Result: Fail

Untested Assessments (first 10):
 2 : Untested 'Simulink:verify_high'
 3 : Untested 'Simulink:verifyTKLow'
 4 : Untested 'Simulink:verifyTKNormal'

Failed Assessments (first 10):
 1 : Fail 'Simulink:verify_high'
```

4. Find assessments under the Test Assessment block, using a regular expression.

```
assessBlock = find(as, '-regexp', 'BlockPath', '.[Aa]ssess')
```

```
assessBlock =
```

```
sltest.AssessmentSet
Summary:
 Total: 6
 Untested: 3
 Passed: 2
 Failed: 1
 Result: Fail
```

```
Untested Assessments (first 10):
 4 : Untested 'Simulink:verify_high'
 5 : Untested 'Simulink:verifyTKLow'
 6 : Untested 'Simulink:verifyTKNormal'
```

```
Passed Assessments (first 10):
 1 : Pass 'Simulink:verify_normal'
 2 : Pass 'Simulink:verify_low'
```

```
Failed Assessments (first 10):
 3 : Fail 'Simulink:verify_high'
```

Re-enable warnings

```
warning on Stateflow:Runtime:TestVerificationFailed
```

## See Also

`sltest.getAssessments` | `sltest.AssessmentSet`

**Introduced in R2016b**

## sltest.AssessmentSet

Access a set of assessments from a simulation

### Description

The function `as = sltest.getAssessments('model')` creates an `sltest.AssessmentSet` object `as` containing the assessments for `model`. Individual assessment results are obtained using `result = get(as,index)`. `getSummary(as)` returns an overview of the assessment set. `disp` returns an overview of individual assessment results.

### Creation

Create an `sltest.AssessmentSet` object using `sltest.getAssessments`.

### Object Functions

|                         |                                                                                                |
|-------------------------|------------------------------------------------------------------------------------------------|
| <code>disp</code>       | Display results of <code>sltest.AssessmentSet</code> or <code>sltest.Assessment</code>         |
| <code>find</code>       | Find assessments in <code>sltest.AssessmentSet</code> or <code>sltest.Assessment</code> object |
| <code>get</code>        | Get assessment of <code>sltest.AssessmentSet</code>                                            |
| <code>getSummary</code> | Get summary of <code>sltest.AssessmentSet</code>                                               |

### Examples

#### Get Assessments from a Simulation

This example shows how to simulate a model with `verify` statements and obtain assessment results via the programmatic interface.

#### Get the Assessment Set and One Assessment Result

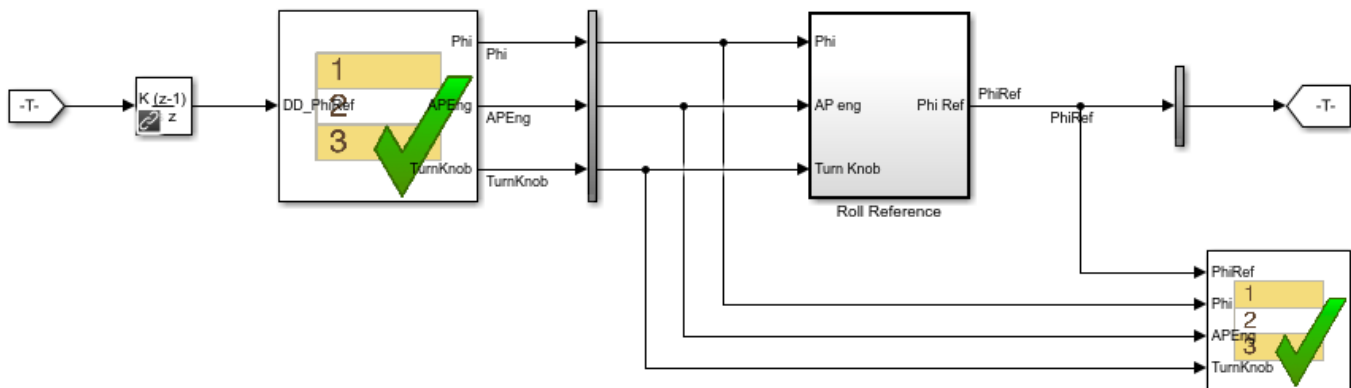
1. Open the model.

```
open_system('sltestRollRefTestExample.slx')

% Turn the command line warning off for verify() statements
warning off Stateflow:Runtime:TestVerificationFailed
```

This model is used to show how verify() statements work in Test Sequence and Test Assessment blocks,

Copyright 2019 The MathWorks, Inc.



Copyright 2016-2020 The MathWorks, Inc.

2. Run the model.

```
s = sim('sltestRollRefTestExample');
```

3. Get the assessment set.

```
as = sltest.getAssessments('sltestRollRefTestExample');
```

4. Get assessment 3 from the assessment set.

```
as3 = get(as,3);
```

### Display Results of the Assessment Set and Assessment Result

1. Get summary of the assessment set.

```
asSummary = getSummary(as)
```

```
asSummary =
```

```
struct with fields:
```

```
Total: 6
Untested: 3
Passed: 2
Failed: 1
Result: Fail
```

2. Display the result of assessment 3.

```
disp(as3)
```

```
sltest.Assessment
Package: sltest

Properties:
 Name: 'Simulink:verify_high'
 BlockPath: [1x1 Simulink.SimulationData.BlockPath]
 Values: [1x1 timeseries]
 Result: Fail
```

3. Find untested or failed results in the assessment set.

```
asFailUntested = find(as, 'Result', slTestResult.Fail, '-or', ...
 'Result', slTestResult.Untested)
```

```
asFailUntested =
```

```
sltest.AssessmentSet
Summary:
 Total: 4
 Untested: 3
 Passed: 0
 Failed: 1
 Result: Fail
```

```
Untested Assessments (first 10):
 2 : Untested 'Simulink:verify_high'
 3 : Untested 'Simulink:verifyTKLow'
 4 : Untested 'Simulink:verifyTKNormal'
```

```
Failed Assessments (first 10):
 1 : Fail 'Simulink:verify_high'
```

4. Find assessments under the Test Assessment block, using a regular expression.

```
assessBlock = find(as, '-regexp', 'BlockPath', '.[Aa]ssess')
```

```
assessBlock =
```

```
sltest.AssessmentSet
Summary:
 Total: 6
 Untested: 3
 Passed: 2
 Failed: 1
 Result: Fail
```

```
Untested Assessments (first 10):
 4 : Untested 'Simulink:verify_high'
 5 : Untested 'Simulink:verifyTKLow'
 6 : Untested 'Simulink:verifyTKNormal'
```

```
Passed Assessments (first 10):
```

```
1 : Pass 'Simulink:verify_normal'
2 : Pass 'Simulink:verify_low'
```

```
Failed Assessments (first 10):
3 : Fail 'Simulink:verify_high'
```

Re-enable warnings

warning on [Stateflow:Runtime:TestVerificationFailed](#)

## See Also

[sltest.getAssessments](#) | [sltest.Assessment](#)

**Introduced in R2016b**

## sltest.io.SimulinkTestSpreadsheet class

**Package:** sltest.io

Read spreadsheet in format used by Simulink Test

### Description

sltest.io.SimulinkTestSpreadsheet inherits from the Simulink.io.FileType base class. It reads spreadsheets formatted in a format used as input to Simulink Test. To import spreadsheets, use this class.

```
classdef SimulinkTestSpreadsheet < Simulink.io.FileType
```

---

**Note** The same methods, operators, functions, and examples for Simulink.io.FileType apply to sltest.io.SimulinkTestSpreadsheet.

---

The sltest.io.SimulinkTestSpreadsheet class is a handle class.

### Class Attributes

|                  |       |
|------------------|-------|
| Abstract         | false |
| HandleCompatible | true  |

For information on class attributes, see “Class Attributes”.

### Properties

#### FileName — File name

character array

File name of the spreadsheet file that contains the signals to import for use as test inputs, specified as a character array.

#### Attributes:

|           |        |
|-----------|--------|
| GetAccess | public |
| SetAccess | public |

### See Also

Simulink.io.FileType | Simulink.io.SignalBuilderSpreadsheet

### Topics

“Microsoft Excel Import, Export, and Logging Format”

“Import Custom File Type”

“Create Custom File Type for Import to Signal Editor”

**Introduced in R2021a**



# sltest.plugins.ModelCoveragePlugin class

**Package:** sltest.plugins

Collect model coverage using the MATLAB Unit Test framework

## Description

The `sltest.plugins.ModelCoveragePlugin` creates coverage reports and allows setting coverage metrics for running Simulink Test and MATLAB-based Simulink test cases with the MATLAB Unit Test framework. Set desired `sltest.plugins.ModelCoveragePlugin` property values, and add the instance of the `sltest.plugins.ModelCoveragePlugin` to the test runner. For MATLAB-based Simulink tests, calls to the `simulate` method collect coverage during the test run. These coverage results are available in the Test Manager results. If you have a license for Parallel Computing Toolbox, you can use the `ModelCoveragePlugin` with parallel test execution.

## Creation

`mcp = sltest.plugins.ModelCoveragePlugin(Properties)` creates a model coverage plugin object `mcp` with specified properties.

You can also import the plugin, then use the class name to create an instance of the plugin:

```
import sltest.plugins.ModelCoveragePlugin
mcp = ModelCoveragePlugin(Properties)
```

## Properties

**RecordModelReferenceCoverage — Specify coverage collection for referenced models**  
false | true

Property that disables or enables coverage collection for models referenced by Model blocks.

Example: 'RecordModelReferenceCoverage', true

### Attributes:

|           |        |
|-----------|--------|
| SetAccess | public |
| GetAccess | public |

**Collecting — Specify coverage collection options**  
CoverageMetrics object

Property that specifies coverage collection options with a `sltest.plugins.coverage.CoverageMetrics` object. If you enable Condition coverage, the Coverage Report shows both Condition and Decision coverage. If you enable MCDC, Condition and Decision coverage are also collected and are included in the Coverage Report.

Example: 'Collecting', covSettings

Example:  
'Collecting', CoverageMetrics('MCD', true, 'Decision', true, 'Condition', true)

**Attributes:**

|           |        |
|-----------|--------|
| SetAccess | public |
| GetAccess | public |

**Producing — Specify model coverage report options**

ModelCoverageReport object

Property that specifies coverage report options with an `sltest.plugins.coverage.ModelCoverageReport`.

Example: 'Producing',mcr

Example: 'Producing',ModelCoverageReport('reports/coverage/modelcoverage')

**Attributes:**

|           |        |
|-----------|--------|
| SetAccess | public |
| GetAccess | public |

**Examples****Collect Model Coverage with MATLAB® Unit Test**

This example shows how to use MATLAB® Unit Test to collect coverage for tests run on a Simulink® model.

You run the tests in the `AutopilotTestFile.mldatx` test file while collecting modified condition/decision (MCDC) coverage.

1. Import the test runner and the plugins for the example.

```
import matlab.unittest.TestRunner
import sltest.plugins.ModelCoveragePlugin
import sltest.plugins.coverage.CoverageMetrics
```

2. Create the model coverage plugin object and the coverage metrics object. In this example, you use MCDC coverage and record coverage for referenced models.

```
mcdcMet = CoverageMetrics('Decision',true,'Condition',true,'MDCD',true);
covSettings = ModelCoveragePlugin('RecordModelReferenceCoverage',true,...
 'Collecting',mcdcMet);
```

3. Create a MATLAB® Unit Test test suite from the test file.

```
tf = sltest.testmanager.TestFile('AutopilotTestFile.mldatx');
APSuite = testsuite(tf.FilePath);
```

4. Create the test runner without any plugins, then add the coverage plugin to the runner.

```
APRun = TestRunner.withNoPlugins();
addPlugin(APRun,covSettings);
```

5. Run the suite.

```
% Turn off the command line warnings.
warning off Stateflow:cdr:VerifyDangerousComparison
```

```
warning off Stateflow:Runtime:TestVerificationFailed
```

```
APResult = run(APRun,APSuite)
```

```
Coverage Report for RollAutopilotMdlRef/Roll Reference
```

```
C:\TEMP\Bdoc22a_1891349_13144\ibC86E06\8\tp8d29381d_3c0c_4725_859c_52e69489b5f4.html
```

```
APResult =
```

```
 TestResult with properties:
```

```
 Name: 'AutopilotTestFile > Basic Design Test Cases/Requirement 1.3 Test'
```

```
 Passed: 0
```

```
 Failed: 1
```

```
 Incomplete: 0
```

```
 Duration: 21.3694
```

```
 Details: [1x1 struct]
```

```
Totals:
```

```
 0 Passed, 1 Failed, 0 Incomplete.
```

```
 21.3694 seconds testing time.
```

6. You can open the link in the command-line output to view the coverage report.

Cleanup. Clear results and re-enable warnings.

```
warning on Stateflow:cdr:VerifyDangerousComparison
```

```
warning on Stateflow:Runtime:TestVerificationFailed
```

```
sltest.testmanager.clearResults;
```

```
sltest.testmanager.clear;
```

```
sltest.testmanager.close;
```

## See Also

```
sltest.plugins.coverage.CoverageMetrics |
```

```
sltest.plugins.coverage.ModelCoverageReport |
```

```
sltest.plugins.MATLABTestCaseIntegrationPlugin |
```

```
sltest.plugins.ToTestManagerLog
```

## Topics

“Test Models Using MATLAB Unit Test”

“Collect Coverage Using MATLAB-Based Simulink Tests”

“Test Models Using MATLAB-Based Simulink Tests”

## Introduced in R2018a

## sltest.plugins.MATLABTestCaseIntegrationPlugin class

**Package:** sltest.plugins

Add simulation and test results for MATLAB-based Simulink tests to Test Manager

### Description

The `sltest.plugins.MATLABTestCaseIntegrationPlugin` adds simulation and test results to the Test Manager test case results for MATLAB-based Simulink tests. A MATLAB-based Simulink test is a MATLAB code (.m) file that is derived from the `sltest.TestCase` class. When you add this plugin to a `TestRunner` object and run the simulation, these results are added to the Test Manager:

- Test results are created for each test suite element of an `sltest.TestCase` class when you run the test case at the MATLAB command line.
- By default, criteria results are added for suite element failures from the `fatalAssertSignalsMatch`, `assertSignalsMatch`, `assumeSignalsMatch`, and `verifySignalsMatch` qualification methods. A comparison run for these failures is added under the corresponding test case result. For passing comparison runs, add the `matlab.unittest.plugins.DiagnosticsOutputPlugin` with passing diagnostics enabled to the `TestRunner`.

Simulation results, which are created using the `simulate` method, are added to the corresponding test case results.

The `sltest.plugins.MATLABTestCaseIntegrationPlugin` class is a `handle` class.

### Creation

To use the `MATLABTestCaseIntegrationPlugin`, add it to the `TestRunner` object:

```
import sltest.plugins.MATLABTestCaseIntegrationPlugin
testRunner.addPlugin...
 (sltest.plugins.MATLABTestCaseIntegrationPlugin());
```

### Examples

#### Prepare a MATLAB-Based Simulink Test for Integration to the Test Manager

- 1 Import the needed plugins and the test runner.

```
import matlab.unittest.TestRunner
import matlab.unittest.plugins.DiagnosticsOutputPlugin
import sltest.plugins.MATLABTestCaseIntegrationPlugin
import sltest.plugins.ToTestManagerLog
```

- 2 Create a `TestSuite` array and basic `TestRunner` object for a test script file, such as one named `mySltestTestCaseFile.m`.

```
suite = testsuite('mySltestTestCaseFile.m');
runner = TestRunner.withNoPlugins;
```

- 3 Add a `MATLABTestCaseIntegrationPlugin` to the `TestRunner` object.

```
runner.addPlugin(MATLABTestCaseIntegrationPlugin);
```

- 4 Run the suite to add the failing diagnostics and simulation results to the Test Manager.

```
results = runner.run(suite);
```

- 5 Add the passing diagnostics to the **Logs** section of the test results in the Test Manager.

```
runner.addPlugin(DiagnosticsOutputPlugin(ToTestManagerLog(), ...
 'IncludingPassingDiagnostics', true));
```

- 6 Rerun the suite to add the diagnostics and simulation results to the Test Manager.

```
results = runner.run(suite);
```

## See Also

`sltest.TestCase` | `assumeSignalsMatch` | `assertSignalsMatch` |  
`fatalAssertSignalsMatch` | `verifySignalsMatch` | `simulate` |  
`matlab.unittest.plugins.DiagnosticsOutputPlugin` |  
`sltest.plugins.ModelCoveragePlugin` | `sltest.plugins.ToTestManagerLog`

## Topics

“Test Models Using MATLAB-Based Simulink Tests”

“Collect Coverage Using MATLAB-Based Simulink Tests”

## Introduced in R2020b

## sltest.plugins.ToTestManagerLog class

**Package:** sltest.plugins

Output stream to write text to Test Manager result Logs for MATLAB-based Simulink tests

### Description

The `sltest.plugins.ToTestManagerLog` creates and writes a text output stream of log information to the **Logs** section of test results in the Test Manager for the current `TestCaseResult`. Use this plugin with the `matlab.unittest.plugins.DiagnosticsOutputPlugin`. The `ToTestManagerLog` plugin generates output only for MATLAB-based Simulink tests, which are derived from `sltest.TestCase` objects.

The `sltest.plugins.ToTestManagerLog` class is a `handle` class.

### Creation

Use `matlab.sltest.plugins.ToTestMangerLog()` with the `matlab.unittest.plugins.DiagnosticsOutputPlugin` to write the diagnostics output to the Test Manager logs. For example,

```
runner.addPlugin(DiagnosticsOutputPlugin...
 (ToTestManagerLog()));
```

### Examples

#### Write MATLAB-Based Simulink Test Log Information to the Test Manager

- 1 Import the needed plugins and the test runner.

```
import matlab.unittest.TestRunner
import matlab.unittest.plugins.DiagnosticsOutputPlugin
import sltest.plugins.MATLABTestCaseIntegrationPlugin
import sltest.plugins.ToTestManagerLog
```

- 2 Create a `TestSuite` array and basic `TestRunner` object for a test script file, such as one named `mySltestTestCaseFile.m`.

```
suite = testsuite('mySltestTestCaseFile.m');
runner = TestRunner.withNoPlugins;
```

- 3 Add a `MATLABTestCaseIntegrationPlugin` to the `TestRunner` object.

```
runner.addPlugin(MATLABTestCaseIntegrationPlugin);
```

- 4 Run the test suite to add the failing diagnostics and simulation results to the Test Manager.

```
results = runner.run(suite);
```

- 5 Add the passing diagnostics to the **Logs** section of the test results in the Test Manager.

```
runner.addPlugin(DiagnosticsOutputPlugin(ToTestManagerLog(), ...
 'IncludingPassingDiagnostics', true));
```

- 6 Rerun the suite to add the diagnostics and simulation results to the Test Manager.

```
results = runner.run(suite);
```

## See Also

[sltest.TestCase](#) | [matlab.unittest.plugins.DiagnosticsOutputPlugin](#) |  
[sltest.plugins.ModelCoveragePlugin](#) |  
[sltest.plugins.MATLABTestCaseIntegrationPlugin](#)

## Topics

“Test Models Using MATLAB-Based Simulink Tests”  
“Collect Coverage Using MATLAB-Based Simulink Tests”

## Introduced in R2020b

## sltest.plugins.TestManagerResultsPlugin class

**Package:** sltest.plugins

Generate enhanced test results with the MATLAB Unit Test framework

### Description

Use the `sltest.plugins.TestManagerResultsPlugin` class to include Test Manager results when using the MATLAB Unit Test framework to run Simulink Test files. Test Case and Test Iteration results appear in the **Details** field of each `TestResult` object.

To publish Test Manager results, configure your test file for reporting and add the `TestReportPlugin` and `TestManagerResultsPlugin` classes to the `TestRunner` object. Test Case and Test Iteration results appear in the **Details** section of the MATLAB Test Report. For more information, see “Test a Model for Continuous Integration Systems”.

### Creation

`tmr = sltest.plugins.TestManagerResultsPlugin` creates an `sltest.plugins.TestManagerResultsPlugin` plugin object `tmr`. You use the plugin object only to direct the `TestRunner` to produce an enhanced test result.

You can also import the plugin, and then use the class name to create the object:

```
import sltest.plugins.TestManagerResultsPlugin
tmr = TestManagerResultsPlugin
```

### Input Arguments

#### Name-Value Pair Options

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

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

Example: `'ExportToFile', 'myfile'`

#### **ExportToFile** — Save results in Simulink Test MLDATX results file

character vector

Optional file to save results in Simulink Test MLDATX format, specified as the comma-separate pair consisting of `'ExportToFile'` and the file name.

You can open the MLDATX results file in the Test Manager by clicking the **Import** button on the toolbar.

Example: `'ExportToFile', 'myfile'`

Example: `'ExportToFile', 'myfile.mldatx'`



## Examples

### Include Test Manager Results in MATLAB Unit Test Results

This example shows how to include Test Manager Results in a `TestResult` object produced through the MATLAB Unit Test framework.

The test case creates a square wave input to a controller subsystem and sweeps through 25 iterations of parameters `a` and `b`. The test compares the `alpha` output to a baseline with a tolerance of `0.0046`. Output that exceeds this tolerance fails the test.

1. Set the path to the test file.

```
testfile = 'f14ParameterSweepTest.mldatx';
```

2. Create the `TestSuite` object.

```
import matlab.unittest.TestSuite
suite = testsuite(testfile);
```

3. Create the `TestRunner` object.

```
import matlab.unittest.TestRunner
runner = TestRunner.withNoPlugins;
```

4. Add the `TestManagerResultsPlugin` to the `TestRunner`.

```
tmr = sltest.plugins.TestManagerResultsPlugin;
addPlugin(runner,tmr)
```

5. Run the test.

```
results = run(runner,suite);
```

6. View results of 19th iteration, a test failure.

```
failure = results(19)
```

```
failure =
```

```
 TestResult with properties:
```

```
 Name: 'f14ParameterSweepTest > New Test Suite 1/Iterations Parameter Sweep(ScriptedIter
 Passed: 0
 Failed: 1
 Incomplete: 0
 Duration: 4.1487
 Details: [1x1 struct]
```

```
Totals:
```

```
 0 Passed, 1 Failed, 0 Incomplete.
 4.1487 seconds testing time.
```

In the `Details` field of the `TestResult` object, test Iteration results appear as a `SimulinkTestManagerResults` object. The `SimulinkTestManagerResults` object contains information such as the type of test case, the cause of the failure, and the values of the parameters that led to the failure.

```
failure.Details.SimulinkTestManagerResults.TestCaseType
```

```
ans =
'Baseline Test'
```

```
failure.Details.SimulinkTestManagerResults.CauseOfFailure
```

```
ans =
'Failed criteria: Baseline'
```

```
failure.Details.SimulinkTestManagerResults.IterationSettings.variableParameters(1)
```

```
ans = struct with fields:
 parameterName: 'a'
 source: 'base workspace'
 value: 2.6000
 displayValue: '2.6'
 simulationIndex: 1
```

```
failure.Details.SimulinkTestManagerResults.IterationSettings.variableParameters(2)
```

```
ans = struct with fields:
 parameterName: 'b'
 source: 'base workspace'
 value: 66
 displayValue: '66'
 simulationIndex: 1
```

## See Also

`matlab.unittest.plugins.TestReportPlugin`

## Topics

“Test Models Using MATLAB Unit Test”

“Export Test Results”

“Output Results for Continuous Integration Systems”

**Introduced in R2018b**

# sltest.plugins.coverage.CoverageMetrics class

**Package:** `sltest.plugins.coverage`

Specify coverage metrics for tests run with MATLAB Unit Test framework

## Description

Use the `sltest.plugins.coverage.CoverageMetrics` class to specify coverage metrics. Pass the coverage metrics object to the model coverage plugin object.

The `sltest.plugins.coverage.CoverageMetrics` class is a `handle` class.

## Class Attributes

`HandleCompatible` `true`

For information on class attributes, see “Class Attributes”.

## Creation

`cmo = sltest.plugins.coverage.CoverageMetrics(Properties)` creates a coverage metrics object with specified properties.

You can also import the plugin, then use the class name to create the object:

```
import sltest.plugins.coverage.CoverageMetrics
cmo = CoverageMetrics(Properties)
```

## Properties

### Decision — Decision coverage

`true` (default) | `false`

Enable or disable decision coverage collection.

Example: `'Decision',true`

### Attributes:

`SetAccess` `public`  
`GetAccess` `public`

### Condition — Condition coverage

`false` (default) | `true`

Enable or disable condition coverage collection. If you enable Condition coverage, the Coverage Report shows both Condition and Decision coverage.

Example: `'Condition',true`

**Attributes:**

|           |        |
|-----------|--------|
| SetAccess | public |
| GetAccess | public |

**MCDC — MCDC coverage**

false (default) | true

Enable or disable modified condition/decision coverage collection. If you enable MCDC, Condition and Decision coverage are also collected and are included in the Coverage Report.

Example: 'MCDC', true

**Attributes:**

|           |        |
|-----------|--------|
| SetAccess | public |
| GetAccess | public |

**LookupTable — Lookup table coverage**

false (default) | true

Enable or disable lookup table coverage collection.

Example: 'LookupTable', true

**Attributes:**

|           |        |
|-----------|--------|
| SetAccess | public |
| GetAccess | public |

**SignalRange — Signal range coverage**

false (default) | true

Enable or disable signal range coverage collection.

Example: 'SignalRange', true

**Attributes:**

|           |        |
|-----------|--------|
| SetAccess | public |
| GetAccess | public |

**SignalSize — Signal size coverage**

false (default) | true

Enable or disable signal size coverage collection.

Example: 'SignalSize', true

**Attributes:**

|           |        |
|-----------|--------|
| SetAccess | public |
| GetAccess | public |

**SimulinkDesignVerifier — Simulink Design Verifier block coverage**

false (default) | true

Enable or disable Simulink Design Verifier block coverage collection.

Example: 'SimulinkDesignVerifier',true

**Attributes:**

|           |        |
|-----------|--------|
| SetAccess | public |
| GetAccess | public |

**SaturationOnIntegerOverflow — Block saturation on integer overflow**

false (default) | true

Enable or disable recording the number of times the block saturates on integer overflow.

Example: 'SaturationOnIntegerOverflow',true

**Attributes:**

|           |        |
|-----------|--------|
| SetAccess | public |
| GetAccess | public |

**RelationalBoundary — Relational boundary coverage**

false (default) | true

Enable or disable relational boundary coverage.

Example: 'RelationalBoundary',true

**Attributes:**

|           |        |
|-----------|--------|
| SetAccess | public |
| GetAccess | public |

## Examples

### Collect Model Coverage with MATLAB® Unit Test

This example shows how to use MATLAB® Unit Test to collect coverage for tests run on a Simulink® model.

You run the tests in the `AutopilotTestFile.mldatx` test file while collecting modified condition/decision (MCDC) coverage.

1. Import the test runner and the plugins for the example.

```
import matlab.unittest.TestRunner
import sltest.plugins.ModelCoveragePlugin
import sltest.plugins.coverage.CoverageMetrics
```

2. Create the model coverage plugin object and the coverage metrics object. In this example, you use MCDC coverage and record coverage for referenced models.

```
mcdcMet = CoverageMetrics('Decision',true,'Condition',true,'MCDC',true);
covSettings = ModelCoveragePlugin('RecordModelReferenceCoverage',true,...
 'Collecting',mcdcMet);
```

3. Create a MATLAB® Unit Test test suite from the test file.

```
tf = sltest.testmanager.TestFile('AutopilotTestFile.mldatx');
APSuite = testsuite(tf.FilePath);
```

4. Create the test runner without any plugins, then add the coverage plugin to the runner.

```
APRun = TestRunner.withNoPlugins();
addPlugin(APRun,covSettings);
```

5. Run the suite.

```
% Turn off the command line warnings.
```

```
warning off Stateflow:cdr:VerifyDangerousComparison
warning off Stateflow:Runtime:TestVerificationFailed
```

```
APResult = run(APRun,APSuite)
```

```
Coverage Report for RollAutopilotMdlRef/Roll Reference
C:\TEMP\Bdoc22a_1891349_13144\ibC86E06\8\tp8d29381d_3c0c_4725_859c_52e69489b5f4.html
```

```
APResult =
```

```
 TestResult with properties:
```

```
 Name: 'AutopilotTestFile > Basic Design Test Cases/Requirement 1.3 Test'
 Passed: 0
 Failed: 1
 Incomplete: 0
 Duration: 21.3694
 Details: [1x1 struct]
```

```
Totals:
```

```
 0 Passed, 1 Failed, 0 Incomplete.
 21.3694 seconds testing time.
```

6. You can open the link in the command-line output to view the coverage report.

Cleanup. Clear results and re-enable warnings.

```
warning on Stateflow:cdr:VerifyDangerousComparison
warning on Stateflow:Runtime:TestVerificationFailed
```

```
sltest.testmanager.clearResults;
sltest.testmanager.clear;
sltest.testmanager.close;
```

## See Also

```
sltest.plugins.ModelCoveragePlugin |
sltest.plugins.coverage.ModelCoverageReport
```

## Topics

“Test Models Using MATLAB Unit Test”

“Types of Model Coverage” (Simulink Coverage)

“Model Objects That Receive Coverage” (Simulink Coverage)

**Introduced in R2018a**

# sltest.plugins.coverage.ModelCoverageReport class

**Package:** sltest.plugins.coverage

**Superclasses:** matlab.mixin.Heterogeneous

Specify model coverage report details for tests run with MATLAB Unit Test

## Description

Use the `sltest.plugins.coverage.ModelCoverageReport` class to specify coverage report details and file location for tests run with the MATLAB Unit Test framework. If you have a license for Parallel Computing Toolbox, you can use the `ModelCoverageReport` with parallel test execution.

The `sltest.plugins.coverage.ModelCoverageReport` class is a handle class.

## Class Attributes

`HandleCompatible` `true`

For information on class attributes, see “Class Attributes”.

## Creation

`mcr = sltest.plugins.coverage.ModelCoverageReport(path)` creates a model coverage report created after the test runs. The input `path` specifies where the report is saved.

You can also import the plugin, then use the class name to create the object:

```
import sltest.plugins.coverage.ModelCoverageReport
mcr = ModelCoverageReport(path)
```

Use `ModelCoveragePlugin` to specify report properties before you run the test:

- 1 Create a `ModelCoverageReport`.
- 2 Create a `ModelCoveragePlugin`, and specify the `ModelCoverageReport` by using the `Producing` property.
- 3 Add the `ModelCoveragePlugin` to the `TestRunner`.
- 4 Run the test.

## Examples

### Set Model Coverage Report Properties for MATLAB Unit Test

This example shows how to specify model coverage report properties when running a Simulink® Test™ test file with MATLAB® Unit Test.

To run the example, set the current folder to a writable folder.

**1. Import classes for the example.**

```
import matlab.unittest.TestSuite
import matlab.unittest.TestRunner
import sltest.plugins.ModelCoveragePlugin
import sltest.plugins.coverage.ModelCoverageReport
```

**2. Create a test suite and test runner.**

Create a MATLAB Unit Test suite from `AutopilotTestFile`. Also create a test runner.

```
ste = testsuite('AutopilotTestFile.mldatx');
trn = TestRunner.withNoPlugins;
```

**3. Specify the report location.**

Create a subfolder in the current folder, and create a `ModelCoverageReport` object specifying the new folder.

```
mkdir('./exReports/coverage');
path = './exReports/coverage';
mcr = ModelCoverageReport(path)

mcr =
 ModelCoverageReport with no properties.
```

**4. Create a Model Coverage Plugin.**

Use the `Producing` property to specify the `ModelCoverageReport` when creating the plugin.

```
mc = ModelCoveragePlugin('Producing',mcr)

mc =
 ModelCoveragePlugin with properties:

 RecordModelReferenceCoverage: '<default>'
 MetricsSettings: '<default>'
```

**5. Add the coverage plugin to the test runner, and run the test.**

```
addPlugin(trn,mc);

% Turn off the command line warnings.
warning off Stateflow:cdr:VerifyDangerousComparison
warning off Stateflow:Runtime:TestVerificationFailed

run(trn,ste)

ans =
 TestResult with properties:

 Name: 'AutopilotTestFile > Basic Design Test Cases/Requirement 1.3 Test'
 Passed: 0
 Failed: 1
 Incomplete: 0
 Duration: 2.0384
 Details: [1x1 struct]
```



```
Totals:
 0 Passed, 1 Failed, 0 Incomplete.
 2.0384 seconds testing time.
```

**Cleanup. Remove temporary folder and clear variables. Enable warnings.**

```
warning on Stateflow:cdr:VerifyDangerousComparison
warning on Stateflow:Runtime:TestVerificationFailed
```

```
rmdir('./exReports','s');
clear('ste','trn','fldr','path','mcr','mc');
```

**See Also**

sltest.plugins.coverage.CoverageMetrics | sltest.plugins.ModelCoveragePlugin

**Topics**

“Test Models Using MATLAB Unit Test”

**Introduced in R2018b**

## sltest.TestCase class

**Package:** sltest

Test case class for MATLAB-based Simulink tests integrated with Test Manager

### Description

The `sltest.TestCase` class enables authoring MATLAB-based Simulink tests. You define a MATLAB-based Simulink test in a `.m` file that inherits from `sltest.TestCase`. Since `sltest.TestCase` is a `matlab.unittest.TestCase`, the `matlab.unittest.TestCase` methods and behaviors are available when authoring a MATLAB-based Simulink test.

The `sltest.TestCase` class is a `handle` class.

### Class Attributes

Abstract true

For information on class attributes, see “Class Attributes”.

### Creation

To create a test that you can load, run, and analyze its results in the Test Manager, create a class definition file that inherits from `sltest.TestCase`. The file contains methods that define the test case.

For debugging at the command line without having to run an `sltest.TestCase` test, use the `forInteractiveUse` static method to create a `TestCase`.

### Methods

#### Public Methods

In addition to the listed methods, the `sltest.TestCase` class can use the methods of the `matlab.unittest.TestCase` class.

|                                                |                                                                                        |
|------------------------------------------------|----------------------------------------------------------------------------------------|
| <code>loadSystem</code>                        | Load model for MATLAB-based Simulink test                                              |
| <code>simulate</code>                          | Simulate model or <code>Simulink.SimulationInput</code> for MATLAB-based Simulink test |
| <code>createSimulationInput</code>             | Create simulation input object                                                         |
| <code>createTemporaryFolder</code>             | Create temporary folder                                                                |
| <code>sltest.TestCase.forInteractiveUse</code> | Create test case for interactive use                                                   |
| <code>assumeSignalsMatch</code>                | Assume two data sets are equivalent                                                    |
| <code>assertSignalsMatch</code>                | Assert two data sets are equivalent                                                    |
| <code>fatalAssertSignalsMatch</code>           | Fatally assert two data sets are equivalent                                            |
| <code>verifySignalsMatch</code>                | Verify two sets of data are equivalent                                                 |

### Examples

## Test Case File Enabled for Use in the Test Manager

This `myBaselineTest.m` class definition file defines a test case that compares a set of baseline values and values produced from a simulation using the `simulate` method. See “Using MATLAB-Based Simulink Tests in the Test Manager” for the full example that includes the baseline data file and other files and commands required to run this test and then load it into the Test Manager.

The first line of the file shows the inheritance from `sltest.TestCase`. The method section, which has a `Test` attribute, contains the `testOne` test case. The test case loads a model, sets some variable values, simulates the harness, and tests whether the simulation and baseline signals match.

```
classdef myBaselineTest < sltest.TestCase
 methods (Test)
 function testOne(testCase)
 testCase.loadSystem('sltest_ratelim');

 in = testCase.createSimulationInput('sltest_ratelim',...
 'WithHarness','sltest_ratelim_Harness1');
 in = in.setVariable('t_gain',[0;2],...
 'Workspace','sltest_ratelim');
 in = in.setVariable('u_gain',[-0.02;-4.02],...
 'Workspace','sltest_ratelim');

 simOut = testCase.simulate(in);
 testCase.verifySignalsMatch(simOut,'baseline1.mat');
 end
 end
end
```

## See Also

`matlab.unittest.TestCase` | `sltest.plugins.ModelCoveragePlugin` |  
`sltest.plugins.MATLABTestCaseIntegrationPlugin` |  
`sltest.plugins.ToTestManagerLog`

## Topics

“Test Models Using MATLAB-Based Simulink Tests”  
 “Using MATLAB-Based Simulink Tests in the Test Manager”  
 “Collect Coverage Using MATLAB-Based Simulink Tests”

## Introduced in R2020b

## sltest.harness.SimulationInput class

**Package:** sltest.harness

Create test harness simulation input for MATLAB-based Simulink tests

### Description

Use objects of the `sltest.harness.SimulationInput` to specify the changes applied to a test harness during simulation. After simulation, the harness is restored to its state before simulation. This object is used with MATLAB-based Simulink tests. `sltest.harness.SimulationInput` is a subclass of `Simulink.SimulationInput`.

The `sltest.harness.SimulationInput` class is a handle class.

### Class Attributes

|                      |      |
|----------------------|------|
| Sealed               | true |
| RestrictsSubclassing | true |

For information on class attributes, see “Class Attributes”.

## Creation

### Description

`in = sltest.harness.SimulationInput(owner, name)` creates and returns a `SimulationInput` object and sets the `HarnessOwner` property to `owner` and the `HarnessName` property to `name`.

Alternatively, use the `createSimulationInput` method of `sltest.TestCase` to create a `sltest.harness.SimulationInput` object.

## Properties

### HarnessOwner — Model or component that owns the harness

string | character vector | model handle | component handle

Model or component that owns the harness, specified as a string, character vector, or a model or component handle.

### HarnessName — Name of the test harness

string | character vector

Name of the test harness for which to a create simulation input object, specified as a string or character vector.

In addition to `HarnessName` and `HarnessOwner` properties, `sltest.harness.SimulationInput` has the properties of `Simulink.SimulationInput`.

## Methods

### Public Methods

|                                                                                  |                                                                                                       |
|----------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------|
| sltest.harness.SimulationInput has the same methods as Simulink.SimulationInput. |                                                                                                       |
| setModelParameter                                                                | Set model parameters to be used for a specific simulation through SimulationInput object, in          |
| setBlockParameter                                                                | Set block parameters to be used for a specific simulation through SimulationInput object, in          |
| setInitialState                                                                  | Configure Simulink.SimulationInput object to set initial state for simulation                         |
| setExternalInput                                                                 | Set external inputs for a simulation through SimulationInput object, in                               |
| setVariable                                                                      | Set variables for a simulation through SimulationInput object, in                                     |
| setPreSimFcn                                                                     | Specify a MATLAB function to run before start of each simulation through SimulationInput object, in   |
| setPostSimFcn                                                                    | Specify a MATLAB function to run after each simulation is complete through SimulationInput object, in |
| applyToModel                                                                     | Apply changes to the model specified through a SimulationInput object, in                             |
| validate                                                                         | Validate the contents of the SimulationInput object, in                                               |
| loadVariablesFromMATFile                                                         | Load variables from MAT-file into a Simulink.SimulationInput object, in                               |

## Examples

### Define Baseline Test in an M-File

This myBaselineTest.m class definition file defines a test case that compares a set of baseline values and values produced from a simulation. This example uses a SimulationInput object to set harness variable values. See “Using MATLAB-Based Simulink Tests in the Test Manager” for the full example that includes the baseline data file and other files and commands required to run this test.

The first line of the file shows the inheritance from sltest.TestCase. The method function section, which has a Test attribute, contains the testOne test case. The test case loads a model, creates an input object, sets some variable values, simulates the harness, and verifies whether the simulation and baseline signals match.

```
classdef myBaselineTest < sltest.TestCase
 methods (Test)
 function testOne(testCase)
 testCase.loadSystem('sltest_ratelim');
 in = sltest.harness.SimulationInput('sltest_ratelim',...
 'sltest_ratelim_Harness1');
 in = in.setVariable('t_gain',[0;2],
 'Workspace','sltest_ratelim');
 in = in.setVariable('u_gain',[-0.02;-4.02],
 'Workspace','sltest_ratelim');
 simOut = testCase.simulate(in);
 testCase.verifySignalsMatch(simOut,'baseline1.mat');
 end
 end
end
```

Then, at the command line, create a test suite, a test runner to run the test, add coverage, and push the test results to the Test Manager. Open the Test Manager to view the results.

```
suite = testsuite('myBaselineTest');
runner = testrunner("textoutput");
runner.addModelCoverage;
runner.addSimulinkTestResults;
runner.run(suite);
```

### **See Also**

[Simulink.SimulationInput](#) | [createSimulationInput](#) | [sltest.TestCase](#) | [addModelCoverage](#) | [addSimulinkTestResults](#)

**Introduced in R2012a**

# sltest.CodeImporter class

Import C or C++ code into Simulink for testing

## Description

Use objects of the `sltest.CodeImporter` class to import a C library or a subset of a library into Simulink for modeling and testing. When you import the code, a Simulink library and a test file are created. Each C-compatible function maps to a C Caller block in the library and each C Caller block, by default, has an attached internal test harness. The created MLDATX test file contains test cases for each imported function. For unit tests, you can only import C code. Additionally, for unit tests, a sandbox is created to isolate the imported C code.

---

**Note** If your code library contains C++ class methods, only the C++ methods that are wrapped in valid C function wrappers are imported into Simulink using the `CodeImporter`.

---

Alternatively, you can use a wizard to set up and import your code into Simulink. In the Test Manager, use **New > Test for C/C++ Code** to open the wizard.

The `sltest.CodeImporter` class is a handle class.

## Class Attributes

`HandleCompatible` true

For information on class attributes, see “Class Attributes”.

## Creation

### Description

`codeimport_obj = sltest.CodeImporter` creates a Simulink Test code importer object with untitled as the Simulink library file name and creates four additional objects, which you access using the `SandboxSettings`, “`ParseInfo`” on page 2-0 , and “`CustomCode`” on page 2-0 , and `Options` properties.

`codeimport_obj = sltest.CodeImporter(libraryfile)` creates a code importer object and uses the specified `libraryfile` as the name of the created Simulink library. It sets the `LibraryFileName` property to `libraryfile`.

## Properties

### TestType — Type of test

"UnitTest" (default) | "IntegrationTest"

Type of test, specified as a string or character vector. To test a subset of your custom code library in isolation, use "UnitTest". The code for a unit test might not be complete and might have undefined

symbols. The `CodeImporter` automatically creates stubs for the undefined symbols. When you specify `"UnitTest"`, a sandbox is also created. To test your entire code library, use `"IntegrationTest"`.

Example: `codeimport_obj.TestType = "IntegrationTest"`

**Attributes:**

|                        |                     |
|------------------------|---------------------|
| <code>GetAccess</code> | <code>public</code> |
| <code>SetAccess</code> | <code>public</code> |

**SandboxSettings — Sandbox configuration for unit tests**

`SandboxSettings` object

Sandbox configuration for unit tests, specified as a `SandboxSettings` object. This property applies only if `TestType` is `UnitTest`. If `TestType` is `IntegrationTest`, sandbox settings are ignored. A default `SandboxSettings` object is created when you create a code importer object. Before using the `createSandbox` method, you can change the default property values. For information on changing the sandbox mode, copying source files to the sandbox folder, and removing pragmas and variable definition header properties, see `sltest.CodeImporter.SandboxSettings`.

Example: `codeimport_obj.SandboxSettings.CopySourceFiles = true`

**Attributes:**

|                        |                      |
|------------------------|----------------------|
| <code>GetAccess</code> | <code>public</code>  |
| <code>SetAccess</code> | <code>private</code> |

**LibraryFileName — Name of created Simulink library file and related artifacts**

`string` | `character vector`

Name of the created Simulink library file and related artifacts, specified as a string or character vector. The related artifacts include the data dictionary, sandbox folder, and test file. For example, if `LibraryFileName` is `"myCodeTest"`

- The Simulink library name is `myCodeTest.slx`.
- The data dictionary name is `myCodeTest.sldd`.
- The sandbox folder name is `myCodeTest_sandbox`.
- The test file is `myCodeTest.mlдатx`.

If you do not specify a library file name, it defaults to `untitled`.

Example: `codeimport_obj.LibraryFileName = "myCodeTest"`

**Attributes:**

|                        |                     |
|------------------------|---------------------|
| <code>GetAccess</code> | <code>public</code> |
| <code>SetAccess</code> | <code>public</code> |

**OutputFolder — Folder for created library and related artifacts files**

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

Folder for the created library and related artifacts files, specified as a string or character vector. The default is an empty string, which is interpreted as the current working folder. You can alternatively specify the `OutputFolder` using MATLAB path commands, enclosed in `$`, such as `$pwd$`, which is the current working folder.



Example: `codeimport_obj.OutputFolder = "C:/myMATLABFiles/CodeImports"`

Example: `codeimport_obj.OutputFolder = "$pwd"`

**Attributes:**

|                        |                     |
|------------------------|---------------------|
| <code>GetAccess</code> | <code>public</code> |
| <code>SetAccess</code> | <code>public</code> |

**CustomCode — C or C++ code files and associated properties to import**

`CustomCode` object

C or C++ code files and their associated properties to import, specified as a `CustomCode` object. Use this property to change code importing options, such as source and header files, folder paths, libraries, and compiler and linker flags. For information, see `Simulink.CodeImporter.CustomCode`.

**Attributes:**

|                        |                      |
|------------------------|----------------------|
| <code>GetAccess</code> | <code>public</code>  |
| <code>SetAccess</code> | <code>private</code> |

**Options — Additional import options**

`Simulink.CodeImporter.Options` object

Additional import options, specified as a `Simulink.CodeImporter.Options` object. Use this object to change the default values for the size of an argument passed by a pointer to a function, creating a test harness when importing code, or changing the Simulink library browser name. For information, see `Simulink.CodeImporter.Options`.

**Attributes:**

|                        |                      |
|------------------------|----------------------|
| <code>GetAccess</code> | <code>public</code>  |
| <code>SetAccess</code> | <code>private</code> |

**ParseInfo — Code parsing information**

`ParseInfo` object

The `ParseInfo` object properties have empty values before you call the `parse` method. After parsing, `ParseInfo` includes whether the code successfully parsed, and information about the available functions, entry functions, and types in the code. For information, see `Simulink.CodeImporter.ParseInfo`.

**Attributes:**

|                        |                      |
|------------------------|----------------------|
| <code>GetAccess</code> | <code>public</code>  |
| <code>SetAccess</code> | <code>private</code> |

## Methods

### Public Methods

In addition to the `createSandbox` method, the `sltest.CodeImporter` class can use the `addToProject`, `parse`, `import`, and `view` methods of the `Simulink.CodeImporter` class.

|                            |                                                          |
|----------------------------|----------------------------------------------------------|
| <code>createSandbox</code> | Create sandbox for C code unit testing                   |
| <code>addToProject</code>  | Add custom code and imported artifacts to MATLAB project |

|        |                                                                                    |
|--------|------------------------------------------------------------------------------------|
| import | Import custom code, functions, and types into Simulink                             |
| parse  | Analyze custom code for functions, types, global variables, and their dependencies |
| view   | Launch Simulink Code Importer wizard                                               |

## Examples

### C Code Unit Testing

This example assumes that you have existing C code files to test and you do not want to test your entire C code library.

- 1 Create an `sltest.CodeImporter` object and specify `myCodeTest` as the Simulink library file name.

```
codeimport_obj = sltest.CodeImporter("myCodeTest");
```

- 2 Assign the source files to the `CustomCode` property.

```
codeimport_obj.CustomCode.SourceFiles = {"myCode1.c"};
```

- 3 Create the sandbox. The `createSandbox` method creates stub files automatically for any undefined symbols.

```
codeimport_obj.createSandbox;
```

After you create the sandbox, you can inspect the created stubs and, if desired, manually stub the symbols using the files in the `manualstub` subfolder of the `sandbox` folder. Then, repeat the `createSandbox` command to update the sandbox.

- 4 Parse the generated sandbox

```
codeimport_obj.parse;
```

- 5 Import the code to a Simulink library model. The `import` method creates a test file by default.

```
codeimport_obj.import;
```

- 6 Open the Test Manager. Then, open the created `myCodeTest.mldatx` test file, and run the created test cases.

### C or C++ Code Library Testing

This example assumes you have an existing C or C++ code library to test and that all C++ methods are in C wrapper functions.

- 1 Create an `sltest.CodeImporter` object and specify `myCodeLibTest` as the Simulink library name. Set the `TestType` to `IntegrationTest`.

```
codeimport_obj = sltest.CodeImporter("myCodeLibTest");
codeimport_obj.TestType = "IntegrationTest";
```

- 2 Create a `CustomCode` object that specifies the header files. Use `custcode.<propertyname> = <value>` syntax to specify other types of files and properties. Then, assign the `CustomCode` object to the `CustomCode` property.

```
codeimport_obj.CustomCode.SourceFiles = {"myCode1.c", "myCode2.c"};
codeimport_obj.InterfaceHeaders = {"myCode_a.h", "myCode_b.h"};
```

- 3 Parse the specified code.

```
codeimport_obj.parse;
```

- 4 Import the code to a Simulink library model. The `import` method creates a test file by default.

```
codeimport_obj.import;
```

- 5 Open the Test Manager. Then, open the created `myCodeLibTest.mldatx` test file, and run the created test cases.

## See Also

[sltest.CodeImporter.SandboxSettings](#) | [createSandbox](#) | [Simulink.CodeImporter](#) | [Simulink.CodeImporter.ParseInfo](#) | [Simulink.CodeImporter.Options](#) | [Simulink.CodeImporter.CustomCode](#)

## Topics

[“Conduct Unit Testing on Imported Custom Code by Using the Wizardfilename”](#)

[“Import Custom Code for Unit Testing Using API Commands”](#)

## Introduced in R2021a

## sltest.CodeImporter.SandboxSettings class

**Package:** sltest.CodeImporter

Sandbox settings for C code testing

### Description

Use an object of the `sltest.CodeImporter.SandboxSettings` class to configure a sandbox for C code unit testing. Creating an `sltest.CodeImporter` object also creates a `SandboxSettings` object with default values. `SandboxSettings` objects do not apply when the `TestType` setting of the `sltest.CodeImporter` object is `IntegrationTest`.

### Properties

#### Mode — Sandbox generation mode

"GenerateAggregatedHeader" (default) | "GeneratePreprocessedSource" | "UseOriginalCode"

Sandbox generation mode, specified as one of these values:

- "GenerateAggregatedHeader" — For a single C code source file, generates a sandbox that includes a minimal, aggregated header file and stubs for undefined symbols. The header file contains the symbols required for compilation.
- "GeneratePreprocessedSource" — Generates a sandbox containing the preprocessed version of the C code source files and creates stubs for undefined symbols.
- "UseOriginalCode" — Generates a sandbox using the original C code without any modifications. However, if the code has undefined symbols, the `CodeImporter` creates stub files.

#### Attributes:

|           |        |
|-----------|--------|
| GetAccess | public |
| SetAccess | public |

#### CopySourceFiles — Whether to copy source files into the sandbox

true or 1 (default) | false or 0

Whether to copy specified source files into the sandbox, specified as a numeric or logical 1 (true) or 0 (false). When `CopySourceFiles` is 1 or true,

- If `Mode` is `GenerateAggregatedHeader`, the `CodeImporter` copies the specified source file to the sandbox `src` folder.
- If `Mode` is `UseOriginalCode`, the `CodeImporter` copies the required header and source files to the sandbox `include` and `src` folders, respectively.
- If `Mode` is `GeneratePreprocessedSource`, the `CopySourceFiles` setting is ignored.

#### Attributes:

|           |        |
|-----------|--------|
| GetAccess | public |
| SetAccess | public |

**RemoveAllPragma — Whether to remove pragmas from code files**

false or 0 (default) | true or 1

Whether to remove pragmas from code files, specified as a numeric or logical 1 (true) or 0 (false). When RemoveAllPragma is true or 1,

- If Mode is GenerateAggregatedHeader, the CodeImporter removes all #pragma instances from the aggregated header file.
- If Mode is GeneratePreprocessedSource, the CodeImporter removes all #pragma instances from the preprocessed source.
- If Mode is UseOriginalCode, RemoveAllPragma is ignored.

**Attributes:**

|           |        |
|-----------|--------|
| GetAccess | public |
| SetAccess | public |

**RemoveVariableDefinitionInHeader — Whether to remove global definitions**

false or 0 (default) | true or 1

Whether to remove all global variable definitions in the generated header file and replace them with extern declarations, specified as a numeric or logical 1 (true) or 0 (false).

RemoveVariableDefinitionInHeader applies only to GenerateAggregatedHeader mode.

When RemoveVariableDefinitionInHeader is true or 1,

- If Mode is GenerateAggregatedHeader, the global variable definitions are replaced in the aggregated header file.
- If Mode is GeneratePreprocessedSource or UseOriginalCode, RemoveVariableDefinitionInHeader is ignored.

---

**Note** If a source or header file that contains a global variable definition is included multiple times in the source file you are importing, set RemoveVariableDefinitionHeader to true or 1. The corresponding setting in the wizard is **Remove variable definition in header file**.

---

**Attributes:**

|           |        |
|-----------|--------|
| GetAccess | public |
| SetAccess | public |

**Examples****Change Sandbox Settings**

Change the sandbox mode and variable definition header settings.

```
codeimport_obj = sltest.CodeImporter('myCodeTest');
codeimport_obj.CustomCode.SourceFiles = {'myCode1.c', 'myCode2.c'};
```

```
codeimport_obj.SandboxSettings.Mode = "GeneratePreprocessedSource";
codeimport_obj.SandboxSettings.RemoveVariableDefinitionInHeader = true;
```

### **See Also**

`createSandbox` | `sltest.CodeImporter`

### **Topics**

“Conduct Unit Testing on Imported Custom Code by Using the Wizardfilename”

“Import Custom Code for Unit Testing Using API Commands”

**Introduced in R2021a**

# sltest.testmanager.Assessment class

**Package:** `sltest.testmanager`

Logical or temporal assessment object

## Description

Use objects of the `sltest.testmanager.Assessment` class to specify the name, summary, requirements, and symbols for a logical or temporal assessment. You can also specify whether the assessment runs when its parent test case runs.

The `sltest.testmanager.Assessment` class is a handle class.

## Creation

You do not create an `sltest.testManager.Assessment` object explicitly. Create assessments in the Test Manager, then use `getAssessments` to obtain the Assessment objects, and `addAssessment` to add them to another test case.

## Properties

### Name — Name of assessment

`string` | character vector

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

#### Attributes:

|                          |                     |
|--------------------------|---------------------|
| <code>GetAccess</code>   | <code>public</code> |
| <code>SetAccess</code>   | <code>public</code> |
| <code>Dependent</code>   | <code>true</code>   |
| <code>NonCopyable</code> | <code>true</code>   |

### Enabled — Whether assessment runs

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

Whether the assessment runs when the parent test case runs, specified as a numeric or logical 1 (`true`) or 0 (`false`).

#### Attributes:

|                          |                     |
|--------------------------|---------------------|
| <code>GetAccess</code>   | <code>public</code> |
| <code>SetAccess</code>   | <code>public</code> |
| <code>Dependent</code>   | <code>true</code>   |
| <code>NonCopyable</code> | <code>true</code>   |

### Info — Summary of the assessment

`string` | character vector

Summary line of the assessment, returned as a string or character vector.

**Attributes:**

|             |         |
|-------------|---------|
| GetAccess   | public  |
| SetAccess   | private |
| Dependent   | true    |
| NonCopyable | true    |

**Requirements — Requirements associated with the assessment**  
struct

Requirements associated with the assessment, returned as a struct.

**Attributes:**

|             |         |
|-------------|---------|
| GetAccess   | public  |
| SetAccess   | private |
| Dependent   | true    |
| NonCopyable | true    |

**Symbols — Symbol objects used in the assessment**

array of `sltest.testmanager.AssessmentSymbol` objects

Symbol objects used in the assessment, specified as an array of `sltest.testmanager.AssessmentSymbol` objects.

**Attributes:**

|             |         |
|-------------|---------|
| GetAccess   | public  |
| SetAccess   | private |
| Dependent   | true    |
| NonCopyable | true    |

**Methods****Public Methods**

`remove` Remove assessment from test case  
`getTestCase` Get parent test case of assessment

**Examples****Get Assessments and Add to Test Case**

Load an existing test file that contains assessments. Get the assessments and then create a new test case and add the assessments to the new test case.

```
tf = sltest.testmanager.load('test_traffic.mldatx');
ts = getTestSuites(tf);
tc = getTestCases(ts);

assessObjs = getAssessments(tc);
tc_New = createTestCase(ts);
tc_NewAssess = addAssessment(tc_New, assessObjs);
```

Clear and close the Test Manager.



```
sltest.testmanager.clear
sltest.testmanager.clearResults
sltest.testmanager.close
```

## See Also

sltest.testmanager.AssessmentSymbol | addAssessment | getAssessments |  
addAssessmentSymbol | getAssessmentSymbols

## Topics

“Logical and Temporal Assessments”

“Assess Simulation and Compare Output Data”

“Assess Temporal Logic by Using Temporal Assessments”

“Logical and Temporal Assessment Syntax”

## Introduced in R2022a

## sltest.testmanager.AssessmentSymbol class

**Package:** sltest.testmanager

Symbol used in assessment

### Description

An instance of `sltest.testmanager.AssessmentSymbol` that specifies the name, scope, and value of the symbol used in an assessment.

The `sltest.testmanager.AssessmentSymbol` class is a handle class.

### Creation

You do not create an `sltest.testManager.AssessmentSymbol` object explicitly. For assessments in the Test Manager, use `getAssessmentSymbols` to obtain the `AssessmentSymbol` object. Then, use `addAssessmentSymbol` to add the symbols to an assessment in another test case. By default when you use `addAssessment`, the associated symbols are also copied.

### Properties

#### Name — Symbol name

string | character vector

Symbol name, returned as a string.

#### Attributes:

|             |         |
|-------------|---------|
| GetAccess   | public  |
| SetAccess   | private |
| Dependent   | true    |
| NonCopyable | true    |

#### Scope — Symbol scope

Signal | Parameter | Variable | Unresolved

Symbol scope, which indicates the type of the assessment symbol, returned as one of these values:

- Signal
- Parameter
- Variable
- Unresolved

**Attributes:**

|             |         |
|-------------|---------|
| GetAccess   | public  |
| SetAccess   | private |
| Dependent   | true    |
| NonCopyable | true    |

**Value — Symbol value**

struct

Symbol value, returned as a struct.

**Attributes:**

|             |         |
|-------------|---------|
| GetAccess   | public  |
| SetAccess   | private |
| Dependent   | true    |
| NonCopyable | true    |

**Examples****Get Assessment Symbols and Add to Test Case**

Load an existing test file and get the assessments and symbols from the test file. Create a new test case and add the assessments, but not the symbols, to the new test case. Then, add the associated assessment symbols to the test case.

```
tf = sltest.testmanager.load('test_traffic.mldatx');
ts = getTestSuites(tf);
tc = getTestCases(ts);

assessObjs = getAssessments(tc);
symbolsObjs = getAssessmentSymbols(tc);

tc_New = createTestCase(ts);
tc_NewAssess = addAssessment(tc_New, assessObjs, ...
 "AddSymbols", false);

addedSymbolsObj = addAssessmentSymbol(tc_New, symbolsObjs);
```

Clear and close the Test Manager.

```
sltest.testmanager.clear
sltest.testmanager.clearResults
sltest.testmanager.close
```

**See Also**

sltest.testmanager.Assessment | addAssessment | getAssessments |  
addAssessmentSymbol | getAssessmentSymbols

**Introduced in R2022a**

## sltest.testmanager.BaselineCriteria class

**Package:** sltest.testmanager

Add or modify baseline criteria

### Description

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

The `sltest.testmanager.BaselineCriteria` class is a handle class.

### Class Attributes

HandleCompatible true

For information on class attributes, see “Class Attributes”.

### Creation

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

### Properties

#### Name — Name of baseline criteria

character vector

Name of the baseline criteria, returned as a character vector.

#### Attributes:

|             |         |
|-------------|---------|
| GetAccess   | public  |
| SetAccess   | private |
| Dependent   | true    |
| NonCopyable | true    |

Data Types: character vector

#### AbsTol — Absolute tolerance

scalar

Absolute tolerance for the baseline criteria set, specified as a scalar.

#### Attributes:

|             |        |
|-------------|--------|
| GetAccess   | public |
| SetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

Data Types: scalar

### Active — Enabled indicator

0 | 1

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

#### Attributes:

|             |        |
|-------------|--------|
| GetAccess   | public |
| SetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

Data Types: logical

### FilePath — File path

character vector

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

#### Attributes:

|             |         |
|-------------|---------|
| GetAccess   | public  |
| SetAccess   | private |
| Dependent   | true    |
| NonCopyable | true    |

Data Types: character vector

### RelTol — Relative tolerance

scalar

Relative tolerance for the baseline criteria set, specified as a scalar.

#### Attributes:

|             |        |
|-------------|--------|
| GetAccess   | public |
| SetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

Data Types:

### LeadingTol — Leading tolerance

scalar

Leading time tolerance for the baseline criteria set, specified as a scalar.

#### Attributes:

|             |        |
|-------------|--------|
| GetAccess   | public |
| SetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

Data Types:

**LaggingTol – Lagging time tolerance**

scalar

Lagging time tolerance for the baseline criteria set, specified as a scalar.

**Attributes:**

|             |        |
|-------------|--------|
| GetAccess   | public |
| SetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

Data Types:

**ExcelSpecifications – Sheet and range information for Excel baseline file**

1-by-N array

Sheet and range information for Microsoft Excel baseline file, returned as a 1-by-N array, where each row has a Sheet and Range value. Specify Range as shown in the table.

| Ways to specify Range                           | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|-------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ' <i>Corner1:Corner2</i> '<br>Rectangular Range | Specify the range using the syntax ' <i>Corner1:Corner2</i> ', where <i>Corner1</i> and <i>Corner2</i> are two opposing corners that define the region. For example, 'D2:H4' represents the 3-by-5 rectangular region between the two corners D2 and H4 on the worksheet. The 'Range' name-value pair argument is not case-sensitive, and uses Excel A1 reference style (see Excel help).<br><br><b>Example:</b> 'Range', 'Corner1:Corner2'                             |
| ' '<br>Unspecified or Empty                     | If unspecified, the importing function automatically detects the used range.<br><br><b>Example:</b> 'Range', ''<br><br><b>Note:</b> <i>Used Range</i> refers to the rectangular portion of the spreadsheet that actually contains data. The importing function automatically detects the used range by trimming leading and trailing rows and columns that do not contain data. Text that is only white space is considered data and is captured within the used range. |
| ' <i>Row1:Row2</i> '<br>Row Range               | You can identify the range by specifying the beginning and ending rows using Excel row designators. Then <code>readtable</code> automatically detects the used column range within the designated rows. For instance, the importing function interprets the range specification '1:7' as an instruction to read all columns in the used range in rows 1 through 7 (inclusive).<br><br><b>Example:</b> 'Range', '1:7'                                                    |

| Ways to specify Range             | Description                                                                                                                                                                                                                                                                                                                                                                                                                |
|-----------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 'Column1:Column2'<br>Column Range | You can identify the range by specifying the beginning and ending columns using Excel column designators. Then <code>readtable</code> automatically detects the used row range within the designated columns. For instance, the importing function interprets the range specification 'A:F' as an instruction to read all rows in the used range in columns A through F (inclusive).<br><br><b>Example:</b> 'Range', 'A:F' |
| 'NamedRange'<br>Excel Named Range | In Excel, you can create names to identify ranges in the spreadsheet. For instance, you can select a rectangular portion of the spreadsheet and call it 'myTable'. If such named ranges exist in a spreadsheet, then the importing function can read that range using its name.<br><br><b>Example:</b> 'Range', 'myTable'                                                                                                  |

**Attributes:**

|             |        |
|-------------|--------|
| GetAccess   | public |
| SetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

Data Types: array

**Methods****Public Methods**

|                       |                                                                      |
|-----------------------|----------------------------------------------------------------------|
| addExcelSpecification | Add a Microsoft Excel sheet to baseline criteria or test case inputs |
| getSignalCriteria     | Get signal criteria                                                  |
| remove                | Remove baseline criteria                                             |

**Examples****Add Baseline Criteria and Change Tolerance**

This example captures a baseline for a test and changes the absolute tolerance from 0 to 9.

```
% Open the model for this example
openExample('sldemo_absbrake');

% Create the test file, test suite, and test case structure
tf = sltest.testmanager.TestFile('API Test File');
ts = createTestSuite(tf, 'API Test Suite');
tc = createTestCase(ts, 'baseline', 'Baseline API Test Case');

% Remove the default test suite
tsDel = getTestSuiteByName(tf, 'New Test Suite 1');
remove(tsDel);
```

```
% Assign the system under test to the test case
setProperty(tc, 'Model', 'sldemo_absbrake');

% Capture the baseline criteria
baseline = captureBaselineCriteria(tc, 'baseline_API.mat', true);

% Set the baseline criteria tolerance for a signal
sc = getSignalCriteria(baseline);
sc(1).AbsTol = 9;
```

## See Also

sltest.testmanager.TestCase | sltest.testmanager.SignalCriteria

## Topics

“Set Signal Tolerances”

“Create and Run Test Cases with Scripts”

**Introduced in R2015b**



# sltest.testmanager.ComparisonRunResult class

**Package:** sltest.testmanager

Access result of a comparison test

## Description

You can access the results of a data comparison (such as a baseline or equivalence test) with instances of `sltest.testmanager.ComparisonRunResult`.

## Creation

`getComparisonResult(t)` creates instances of `sltest.testmanager.ComparisonRunResult`, where `t` is a test results object.

## Properties

### Outcome — Comparison result

`sltest.testmanager.ComparisonSignalOutcomes` object

Pass or fail result of the run comparison, specified as a `sltest.testmanager.ComparisonSignalOutcomes` object.

### Attributes:

|           |         |
|-----------|---------|
| SetAccess | private |
| GetAccess | public  |

## Methods

### Public Methods

`getComparisonSignalResults` Get test signal comparison result from comparison run result

## Examples

### Get Comparison Results of a Baseline Test

This example shows how to programmatically get the comparison results of the second iteration of a baseline test case.

1. Get the path to the test file, then run the test file.

```
extf = 'sltestTestCaseRealTimeReuseExample.mldatx';
tf = sltest.testmanager.TestFile(extf);
ro = run(tf);
```

2. Get the test iteration results.

```
tfr = getTestFileResults(ro);
tsr = getTestSuiteResults(tfr);
```

```
tcr = getTestCaseResults(tsr);
tir = getIterationResults(tcr);
```

3. Get the comparison run result of iteration 2.

```
cr2 = getComparisonResult(tir(2))
```

```
cr2 =
```

```
ComparisonRunResult with properties:
```

```
Outcome: Passed
```

4. Get the comparison signal result of the run result.

```
cr2sig = getComparisonSignalResults(cr2)
```

```
cr2sig =
```

```
1x2 ComparisonSignalResult array with properties:
```

```
Outcome
Baseline
ComparedTo
Difference
```

5. Clear the results and the Test Manager.

```
sltest.testmanager.clearResults;
sltest.testmanager.clear;
sltest.testmanager.close;
```

## See Also

**Introduced in R2017b**

# sltest.testmanager.ComparisonSignalResult class

**Package:** sltest.testmanager

Access signal comparison results from a baseline or equivalence result

## Description

You can access the results of a data comparison (such as a baseline or equivalence test) with instances of `sltest.testmanager.ComparisonSignalResult`.

## Creation

`getComparisonSignalResults(cr)` creates instances of `sltest.testmanager.ComparisonSignalResult`, where `cr` is a `sltest.testmanager.ComparisonRunResult` object.

## Properties

### Outcome — Comparison result

`sltest.testmanager.ComparisonSignalOutcomes` object

Pass or fail result of the run comparison, specified as a `sltest.testmanager.ComparisonSignalOutcomes` object.

#### Attributes:

|           |         |
|-----------|---------|
| SetAccess | private |
| GetAccess | public  |

### Baseline — Baseline signal

`Simulink.sdi.Signal` object

Baseline signal, specified as a `Simulink.sdi.Signal` object.

#### Attributes:

|           |         |
|-----------|---------|
| SetAccess | private |
| GetAccess | public  |

### ComparedTo — Output signal

`Simulink.sdi.Signal` object

Output signal, specified as a `Simulink.sdi.Signal` object.

#### Attributes:

|           |         |
|-----------|---------|
| SetAccess | private |
| GetAccess | public  |

### Difference — Baseline - output delta

`Simulink.sdi.Signal` object

Difference signal between baseline and output, specified as a `Simulink.sdi.Signal` object.

**Attributes:**

|           |         |
|-----------|---------|
| SetAccess | private |
| GetAccess | public  |

**Methods****Public Methods**

`getComparisonSignalResults` Get test signal comparison result from comparison run result

**Examples****Get Comparison Results of a Baseline Test**

This example shows how to programmatically get the comparison results of the second iteration of a baseline test case.

1. Get the path to the test file, then run the test file.

```
extf = 'sltestTestCaseRealTimeReuseExample.mldatx';
tf = sltest.testmanager.TestFile(extf);
ro = run(tf);
```

2. Get the test iteration results.

```
tfr = getTestFileResults(ro);
tsr = getTestSuiteResults(tfr);
tcr = getTestCaseResults(tsr);
tir = getIterationResults(tcr);
```

3. Get the comparison run result of iteration 2.

```
cr2 = getComparisonResult(tir(2))
```

```
cr2 =
```

```
ComparisonRunResult with properties:
```

```
Outcome: Passed
```

4. Get the comparison signal result of the run result.

```
cr2sig = getComparisonSignalResults(cr2)
```

```
cr2sig =
```

```
1×2 ComparisonSignalResult array with properties:
```

```
Outcome
Baseline
ComparedTo
```

Difference

5. Clear the results and the Test Manager.

```
sltest.testmanager.clearResults;
sltest.testmanager.clear;
sltest.testmanager.close;
```

## **See Also**

**Introduced in R2017b**

## sltest.testmanager.CoverageSettings class

**Package:** sltest.testmanager

Modify coverage settings

### Description

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

The sltest.testmanager.CoverageSettings class is a handle class.

### Class Attributes

HandleCompatible true

For information on class attributes, see “Class Attributes”.

### Creation

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

### Properties

#### RecordCoverage — Enable coverage collection

false (default) | true

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

Coverage collection is enabled or disabled in the Test Manager **Coverage Settings**. The corresponding UI option is **Record coverage for system under test**.

Example: 'Decision',true

#### Attributes:

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

#### Md\RefCoverage — Collect coverage for referenced models

false (default) | true

Coverage collection for referenced models is enabled or disabled in the Test Manager **Coverage Settings**. The corresponding UI option is **Record coverage for referenced models**.

Example: 'Decision',true

**Attributes:**

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

**MetricSettings — Coverage metric setting selection**

string

Selection of coverage settings that are enabled or disabled, specified as a string. For the set of possible strings, see the parameter info for CovMetricSettings in **Coverage metric settings** and `sltest.plugins.coverage.CoverageMetrics`. Coverage metric settings can be modified only at the test file level.

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

Example: "dw"

**Attributes:**

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

**CoverageFilterFilename — Coverage filter to use for coverage analysis**

string array

The file name of a coverage filter file or files on the MATLAB path, specified as a string array. The file names specify the coverage filters that override filter files specified in the model configuration settings. An empty value, "", attaches no coverage filter. For more information on coverage filters, see "Coverage Filtering" (Simulink Coverage). The coverage filter setting propagates down from test files to test suites and test cases, and its results are displayed at the Result Set level.

Specify the coverage filter file name in the Test Manager, in the **Coverage filter filename** table of the **Coverage Settings** section.

Example: "covfilter.cvf"

Example: ["covfilter.cvf1";"covfilter2.cvf"]

Example: ""

**Attributes:**

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

**Examples**

**Enable MDC and Signal Range Coverage Metrics and Specify a Coverage Filter**

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

% Enable MDC and signal range coverage metrics
cov.MetricSettings = "mr";

% Specify a coverage filter
cov.CoverageFilterFilename = "covfilter.cvf";
```

**See Also**

sltest.testmanager.TestFile | sltest.testmanager.TestSuite |  
sltest.testmanager.TestCase | sltest.plugins.coverage.CoverageMetrics

**Topics**

“Collect Coverage in Tests”  
“Create and Run Test Cases with Scripts”

**Introduced in R2016a**



# sltest.testmanager.CustomCriteria class

**Package:** sltest.testmanager

Add or modify custom criteria

## Description

An instance of `sltest.testmanager.CustomCriteria` is a test case custom criteria that evaluates the simulation output, returning a pass or fail result.

The `sltest.testmanager.CustomCriteria` class is a handle class.

## Class Attributes

HandleCompatible true

For information on class attributes, see “Class Attributes”.

## Creation

`obj = getCustomCriteria(tc)` creates an `sltest.testmanager.CustomCriteria` object for a test case object `tc`.

## Properties

### Enabled — Enable or disable custom criteria

false (default) | true

Property that enables or disables the custom criteria for evaluation, specified as a logical.

Example: true

### Attributes:

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

### Callback — Criteria script

character array

Property that defines the custom criteria script, specified as a character array.

Example: `test.verifyEqual(lastPhi,0,['Final: ',num2str(lastPhi),'.'])`

**Attributes:**

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

**Examples****Get and Set Custom Criteria in Test Case**

Create a test case object from the test suite ts.

```
tc = ts.getTestCaseByName('Requirement 1.3 Test');
```

Get the custom criteria from the test case tc.

```
tcCriteria = getCustomCriteria(tc);
```

Set the custom criteria script.

```
tcCriteria.Callback = 'test.verifyEqual(lastPhi,0);'
```

Enable the custom criteria.

```
tcCriteria.Enabled = true;
```

**See Also**

getCustomCriteria

**Topics**

“Process Test Results with Custom Scripts”

“Custom Criteria Programmatic Interface Example”

“Create and Run Test Cases with Scripts”

**Introduced in R2016b**

# sltest.testmanager.CustomCriteriaResult class

**Package:** `sltest.testmanager`

View custom criteria test result

## Description

An instance of `sltest.testmanager.CustomCriteriaResult` is a test result of the evaluation of custom criteria.

The `sltest.testmanager.CustomCriteriaResult` class is a handle class.

## Class Attributes

`HandleCompatible` `true`

For information on class attributes, see “Class Attributes”.

## Creation

`obj = getCustomCriteriaResult(tcr)` creates an `sltest.testmanager.CustomCriteriaResult` object for a test case result object `tcr`.

`obj = getCustomCriteriaResult(tir)` creates an `sltest.testmanager.CustomCriteriaResult` object for a test iteration result object `tir`.

## Properties

### Outcome — Result of custom criteria evaluation

`sltest.testmanager.TestResultOutcomes` object.

Custom criteria result, returned as an `sltest.testmanager.TestResultOutcomes` object.

Example: Passed

### Attributes:

|                          |                     |
|--------------------------|---------------------|
| <code>SetAccess</code>   | <code>public</code> |
| <code>GetAccess</code>   | <code>public</code> |
| <code>Dependent</code>   | <code>true</code>   |
| <code>NonCopyable</code> | <code>true</code>   |

### DiagnosticRecord — Criteria script

`sltest.testmanager.DiagnosticRecord` object.

Diagnostic record of the custom criteria result, returned as an `sltest.testmanager.DiagnosticRecord` object.

Example: `DiagnosticRecord`

**Attributes:**

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

**Examples****Get Custom Criteria Result from Test Case Result**

Run the test case `tc`, creating a result set `tcResultSet`.

```
tcResultSet = run(tc);
```

Get the test case result from the result set.

```
tcResult = getTestCaseResults(tcResultSet);
```

Get the custom criteria result from the test case result.

```
ccResult = getCustomCriteriaResult(tcResult);
```

**See Also**

`getCustomCriteria`

**Topics**

["Process Test Results with Custom Scripts"](#)

["Custom Criteria Programmatic Interface Example"](#)

["Create and Run Test Cases with Scripts"](#)

**Introduced in R2016b**

# sltest.testmanager.DiagnosticRecord class

**Package:** sltest.testmanager

View custom criteria diagnostic information

## Description

An instance of `sltest.testmanager.DiagnosticRecord` displays diagnostic information returned from custom criteria analysis.

The `sltest.testmanager.DiagnosticRecord` class is a handle class.

## Class Attributes

HandleCompatible true

For information on class attributes, see “Class Attributes”.

## Creation

`obj = getCustomCriteriaResult(tcResult)` creates an `sltest.testmanager.CustomCriteriaResult` object, which has a property `DiagnosticRecord`. `DiagnosticRecord` is an `sltest.testmanager.DiagnosticRecord` object for the test case result object `tcResult`.

## Properties

### Outcome — Record outcome of diagnostic

`sltest.testmanager.TestResultOutcomes` object.

Outcome of diagnostic, returned as an `sltest.testmanager.TestResultOutcomes` object.

Example: Passed

### Attributes:

|             |         |
|-------------|---------|
| SetAccess   | private |
| GetAccess   | public  |
| Dependent   | true    |
| NonCopyable | true    |

### TestDiagnosticResult — Record test diagnostic results

cell array

Diagnostic record of the custom criteria result, returned as a cell array.

Example: 'Final: 0.'

**Attributes:**

|             |         |
|-------------|---------|
| SetAccess   | private |
| GetAccess   | public  |
| Dependent   | true    |
| NonCopyable | true    |

**FrameworkDiagnosticResult — Record framework diagnostic results**

cell array

Framework diagnostic record of the custom criteria result, returned as a cell array.

Example: 'verifyEqual passed...'

**Attributes:**

|             |         |
|-------------|---------|
| SetAccess   | private |
| GetAccess   | public  |
| Dependent   | true    |
| NonCopyable | true    |

**Event — Record event name**

character vector

Name of the recorded event, returned as a character vector.

Example: VerificationPassed

**Attributes:**

|             |         |
|-------------|---------|
| SetAccess   | private |
| GetAccess   | public  |
| Dependent   | true    |
| NonCopyable | true    |

**Report — Record diagnostic information**

character vector

Report of the diagnostic result, returned as a character vector.

**Attributes:**

|             |         |
|-------------|---------|
| SetAccess   | private |
| GetAccess   | public  |
| Dependent   | true    |
| NonCopyable | true    |

**Exception — Capture error information**

Mexception object

If the custom criteria returns an error, it constructs an Mexception object containing information about the error.

Example: MException

**Attributes:**

|             |         |
|-------------|---------|
| SetAccess   | private |
| GetAccess   | public  |
| Dependent   | true    |
| NonCopyable | true    |

**Examples****Get Custom Criteria Result from Test Case Result**

Run the test case tc, creating a result set tcResultSet.

```
tcResultSet = run(tc);
```

Get the test case result from the result set.

```
tcResult = getTestCaseResults(tcResultSet);
```

Get the custom criteria result from the test case result.

```
ccResult = getCustomCriteriaResult(tcResult);
```

Display the diagnostic result

```
ccResult.DiagnosticRecord
```

```
ans =
```

```
DiagnosticRecord with properties:
```

```

 Outcome: Passed
 TestDiagnosticResult: {'Final: 0.'}
 FrameworkDiagnosticResult: {'verifyEqual passed....'}
 Event: 'VerificationPassed'
 Report: '=====...'
```

**See Also**

getCustomCriteria

**Topics**

“Process Test Results with Custom Scripts”

“Custom Criteria Programmatic Interface Example”

“Create and Run Test Cases with Scripts”

**Introduced in R2016b**

## sltest.testmanager.EquivalenceCriteria class

**Package:** sltest.testmanager

Add or modify equivalence criteria

### Description

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

The `sltest.testmanager.EquivalenceCriteria` class is a `handle` class.

### Class Attributes

`HandleCompatible` true

For information on class attributes, see “Class Attributes”.

## Creation

### Description

`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.

### Attributes:

|                          |                     |
|--------------------------|---------------------|
| <code>GetAccess</code>   | <code>public</code> |
| <code>SetAccess</code>   | <code>public</code> |
| <code>Dependent</code>   | <code>true</code>   |
| <code>NonCopyable</code> | <code>true</code>   |

Data Types: `logical`

## Methods

### Public Methods

|                                |                             |
|--------------------------------|-----------------------------|
| <code>getSignalCriteria</code> | Get signal criteria         |
| <code>remove</code>            | Remove equivalence criteria |

## Examples



## Add Equivalence Criteria to Test Case

```
% Open the model for this example
openExample('sldemo_absbrake');

% Create the test file, test suite, and test case structure
tf = sltest.testmanager.TestFile('API Test File');
ts = createTestSuite(tf, 'API Test Suite');
tc = createTestCase(ts, 'equivalence', 'Equivalence Test Case');

% Remove the default test suite
tsDel = getTestSuiteByName(tf, 'New Test Suite 1');
remove(tsDel);

% Assign the system under test to the test case
% for Simulation 1 and Simulation 2
setProperty(tc, 'Model', 'sldemo_absbrake', 'SimulationIndex', 1);
setProperty(tc, 'Model', 'sldemo_absbrake', 'SimulationIndex', 2);

% Capture equivalence criteria
eq = captureEquivalenceCriteria(tc);
```

## See Also

sltest.testmanager.TestCase

## Topics

“Create and Run Test Cases with Scripts”

**Introduced in R2015b**

## sltest.testmanager.LoggedSignal class

**Package:** `sltest.testmanager`

Create or modify logged signals for use as simulation outputs

### Description

An instance of `sltest.testmanager.LoggedSignal` stores a logged signal for use in a `sltest.testmanager.LoggedSignalSet` object. You can use the logged signal for data comparison with baseline criteria, equivalence criteria, custom criteria, or in iterations.

### Creation

#### Description

`obj = addLoggedSignal(lgset,BlockPath,PortIndex)` creates and adds a `LoggedSignal` object to a `LoggedSignalSet` object. You must open or load the model to add signals from the model.

`obj = addDataStoreSignal(lgset,BlockPath)` creates and adds an `sltest.testmanager.LoggedSignal` object to a set when the `LoggedSignal` object derives from a data store or `Simulink.Signal` object. You must open or load the model to add a `LoggedSignal` from the model.

`objs = getLoggedSignals(lgset)` creates and returns a vector of the `LoggedSignal` objects that are stored in a `LoggedSignalSet`.

#### Input Arguments

##### **lgset — Logged signal set**

`sltest.testmanager.LoggedSignalSet` object

Object that can contain one or more `LoggedSignal` objects.

##### **BlockPath — Block path object**

`Simulink.BlockPath` object | character vector

`Simulink.BlockPath` object that uniquely identifies the block that outputs the signal.

##### **PortIndex — Output port index**

integer

Index of the output port for the block designated by `BlockPath`, starting from 1.

### Properties

#### **Name — Signal name**

character vector

Name of the signal. This property is read-only.

**BlockPath — Block path object**

Simulink.BlockPath object | character vector

Simulink.BlockPath object that uniquely identifies the block that outputs the signal. This property is read-only.

**PortIndex — Output port index**

integer

Index of the output port for the block designated by BlockPath, starting from 1. This property is read-only.

**Source — Block path information**

character vector

Name of the block path for the object. If the signal corresponds to a Simulink.Signal object, the field displays 'base workspace' or 'model workspace' to describe the location of the object. This property is read-only.

**Active — On/off flag**

boolean

Indicates whether the signal is logged during test case execution.

**PlotIndices — Signal subplot indices**

character array

Indices for subplot location.

**Methods****Public Methods**

remove Remove a logged signal

**Examples****Add Signals to a Test Case**

Open a model and create a signal set.

```
% Open model for this example
openExample('sldemo_absbrake')

% Create test case
tf = sltest.testmanager.TestFile(strcat(pwd, '\mytf.mldatx'));
ts = sltest.testmanager.TestSuite(tf, 'myts');
tc = sltest.testmanager.TestCase(ts, 'baseline', 'mytc');

% Create signal set
lgset = tc.addLoggedSignalSet;
```

Select the Vehicle Speed block and enter gcb. Use the returned path to create a Simulink.BlockPath object.

```
% Add signals to set
bPath = Simulink.BlockPath('sldemo_absbrake/Vehicle speed');
sig1 = lgset.addLoggedSignal(bPath,1);
sig2 = lgset.addLoggedSignal(bPath,2);

setProperty(tc, 'Model', 'sldemo_absbrake');

% Save test file
saveToFile(tf);
```

## See Also

[Simulink.BlockPath](#) | [addLoggedSignal](#) | [getLoggedSignals](#) | [sltest.testmanager.LoggedSignalSet](#) | [addDataStoreSignal](#)

## Topics

“Create and Run Test Cases with Scripts”

“Capture Simulation Data in a Test Case”

## Introduced in R2019a

# sltest.testmanager.LoggedSignalSet class

**Package:** `sltest.testmanager`

Create or modify a set of logged signals

## Description

An instance of `sltest.testmanager.LoggedSignalSet` stores a set of `sltest.testmanager.LoggedSignal` objects. You can use the logged signals for data comparison with baseline criteria, equivalence criteria, custom criteria, or in iterations.

## Creation

### Description

`obj = addLoggedSignalSet(tc,Name,Value)` creates and adds a `LoggedSignalSet` object to an `sltest.testmanager.TestCase` object.

`objs = getLoggedSignalSets(tc,Name,Value)` creates and returns a vector of the `LoggedSignalSet` objects that are stored in a test case object.

### Input Arguments

#### **tc — Test case**

`sltest.testmanager.TestCase` object

Name of the test case object.

#### **Name-Value Pair Arguments**

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

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

#### **Name — Name of the signal set**

character vector

Name of the logged signal set.

Example: `obj = addLoggedSignalSet(tc,'Name','mylgset');`

#### **SimulationIndex — Simulation index**

1 (default) | 2

When the test case is an equivalence test, this index specifies the simulation that contains the signal set.

Example: `obj = getLoggedSignalSets(tc_equiv,'SimulationIndex',2);`

## Properties

### Name — Name of the signal set

character vector

Name of the signal set.

### Active — On/off flag

1 (default) | 0

Indicates whether the signals contained in the set are used during test case execution.

## Methods

### Public Methods

|                                 |                                                     |
|---------------------------------|-----------------------------------------------------|
| <code>addDataStoreSignal</code> | Add a data store or Simulink.Signal object to a set |
| <code>addLoggedSignal</code>    | Add a logged signal to a set                        |
| <code>getLoggedSignals</code>   | Return logged signals contained in a set            |
| <code>remove</code>             | Remove a logged signal set                          |

## Examples

### Add a Signal Set to a Test Case

Open a model and create a signal set.

```
% Open model for this example
openExample('sldemo_absbrake')

% Create test case
tf = sltest.testmanager.TestFile(strcat(pwd, '\mytf.mldatx'));
ts = sltest.testmanager.TestSuite(tf, 'myts');
tc = sltest.testmanager.TestCase(ts, 'baseline', 'mytc');

% Create signal set
lgset = tc.addLoggedSignalSet;
```

### See Also

`sltest.testmanager.EquivalenceCriteria` | `addLoggedSignalSet` | `getLoggedSignalSets` | `sltest.testmanager.LoggedSignal`

### Topics

“Create and Run Test Cases with Scripts”

“Capture Simulation Data in a Test Case”

### Introduced in R2019a

# sltest.testmanager.Options class

**Package:** sltest.testmanager

Return and specify test file options

## Description

Get instances of sltest.testmanager.Options to view test file options, including report generation options. For test files, you can also set options. View options using:

- getOptions (TestCase)
- getOptions (TestSuite)
- getOptions (TestFile)

The sltest.testmanager.Options class is a handle class.

## Class Attributes

HandleCompatible true

For information on class attributes, see “Class Attributes”.

## Creation

obj = getOptions(test) returns the test file options object associated with the test case, suite, or file.

## Input Arguments

### test — Test case, suite, or file

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

Test case, suite, or file, specified as an sltest.testmanager.TestCase, sltest.testmanager.TestSuite, or sltest.testmanager.TestFile object.

## Properties

### Author — Report author

character vector

Author of the report, specified as a character vector.

### Attributes:

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

**CloseFigures — Option to close figures at end of test**`true (default) | false`

Option to close figures at the end of the test, specified as `true` to close the figures and `false` to leave them open.

**Attributes:**

|                          |                     |
|--------------------------|---------------------|
| <code>SetAccess</code>   | <code>public</code> |
| <code>GetAccess</code>   | <code>public</code> |
| <code>Dependent</code>   | <code>true</code>   |
| <code>NonCopyable</code> | <code>true</code>   |

**CustomReportClass — Custom report generation class**`character vector`

Custom report generation class, specified as a character vector. For information, see “Customize Test Results Reports”.

**CustomTemplateFile — Path name of report generation custom template file**`character vector`

Path name of report generation custom template file, specified as a character vector. For information, see “Customize Test Results Reports”.

**Attributes:**

|                          |                     |
|--------------------------|---------------------|
| <code>SetAccess</code>   | <code>public</code> |
| <code>GetAccess</code>   | <code>public</code> |
| <code>Dependent</code>   | <code>true</code>   |
| <code>NonCopyable</code> | <code>true</code>   |

**GenerateReport — Option to generate a report at end of test**`false (default) | true`

Option to generate a report at the end of the test, specified as `true` or `false`.

**Attributes:**

|                          |                     |
|--------------------------|---------------------|
| <code>SetAccess</code>   | <code>public</code> |
| <code>GetAccess</code>   | <code>public</code> |
| <code>Dependent</code>   | <code>true</code>   |
| <code>NonCopyable</code> | <code>true</code>   |

**IncludeComparisonSignalPlots — Option to include simulation output and baseline plots in reports**`true (default) | false`

Option to include simulation output and baseline plots in report, specified as `true` or `false`.

**Attributes:**

|                          |                     |
|--------------------------|---------------------|
| <code>SetAccess</code>   | <code>public</code> |
| <code>GetAccess</code>   | <code>public</code> |
| <code>Dependent</code>   | <code>true</code>   |
| <code>NonCopyable</code> | <code>true</code>   |



**IncludeCoverageResult — Option to include coverage results in reports**

false (default) | true

Option to include coverage results in reports, specified as true or false.

**Attributes:**

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

**IncludeErrorMessage — Option to include error and log messages in reports**

true (default) | false

Option to include error and log messages in reports, specified as true or false.

**Attributes:**

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

**IncludeMATLABFigures — Option to include figures generated from MATLAB code in reports**

false (default) | true

Option to include figures generated from MATLAB code in reports, specified as true or false. Specify the MATLAB code as custom criteria on the test case or as a callback on the test case, suite, or file. You must also set SaveFigures to true for this setting to apply.

**Attributes:**

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

**IncludeMLVersion — Option to include MATLAB version in report**

true (default) | false

Option to include the MATLAB version you are running in the report, specified as true or false.

**IncludeSimulationMetadata — Option to include simulation metadata in reports**

false (default) | true

Option to include simulation metadata in reports, specified as true or false.

**Attributes:**

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

**IncludeSimulationSignalPlots — Option to include criteria and assessment plots in reports**`false (default) | true`

Option to include criteria and assessment plots in reports, specified as `true` or `false`.

**Attributes:**

|                          |                     |
|--------------------------|---------------------|
| <code>SetAccess</code>   | <code>public</code> |
| <code>GetAccess</code>   | <code>public</code> |
| <code>Dependent</code>   | <code>true</code>   |
| <code>NonCopyable</code> | <code>true</code>   |

**NumPlotRowsPerPage — Number of rows of plots to include on report pages**`2 (default)`

Number of rows of plots to include on report pages, specified as an integer from 1 to 4. This property is used only if the `IncludeSimulationSignalPlots` property is `true`.

**Attributes:**

|                          |                     |
|--------------------------|---------------------|
| <code>SetAccess</code>   | <code>public</code> |
| <code>GetAccess</code>   | <code>public</code> |
| <code>Dependent</code>   | <code>true</code>   |
| <code>NonCopyable</code> | <code>true</code>   |

**NumPlotColumnsPerPage — Number of columns of plots to include on report pages**`1 (default)`

Number of columns of plots to include on report pages, specified as an integer from 1 to 4. This property is used only if the `IncludeSimulationSignalPlots` property is `true`.

**Attributes:**

|                          |                     |
|--------------------------|---------------------|
| <code>SetAccess</code>   | <code>public</code> |
| <code>GetAccess</code>   | <code>public</code> |
| <code>Dependent</code>   | <code>true</code>   |
| <code>NonCopyable</code> | <code>true</code>   |

**IncludeTestRequirement — Option to include test requirements in reports**`true (default) | false`

Option to include test requirements in reports, specified as `true` or `false`.

**Attributes:**

|                          |                     |
|--------------------------|---------------------|
| <code>SetAccess</code>   | <code>public</code> |
| <code>GetAccess</code>   | <code>public</code> |
| <code>Dependent</code>   | <code>true</code>   |
| <code>NonCopyable</code> | <code>true</code>   |

**IncludeTestResults — Test results to include in the report**`'failed' (default) | 'passed' | 'all' | enumerated  
sltest.testmanager.TestResultsIncludedInReport value`

Test results to include in the report, specified as `'failed'`, `'passed'`, or `'all'`. You can alternatively use an enumerated value:

- `sltest.testmanager.TestResultsIncludedInReport.AllTests`
- `sltest.testmanager.TestResultsIncludedInReport.FailedOnly`
- `sltest.testmanager.TestResultsIncludedInReport.PassedOnly`

**Attributes:**

|                          |                     |
|--------------------------|---------------------|
| <code>SetAccess</code>   | <code>public</code> |
| <code>GetAccess</code>   | <code>public</code> |
| <code>Dependent</code>   | <code>true</code>   |
| <code>NonCopyable</code> | <code>true</code>   |

**ReportFormat — Output format for report**

`'pdf'` (default) | `'zip'` | `'doc'` | enumerated `sltest.testmanager.ReportFileFormat` value

Output format for report, specified as `'pdf'`, `'zip'`, or `'docx'`. You can alternatively use an enumerated value:

- `sltest.testmanager.ReportFileFormat.doc`
- `sltest.testmanager.ReportFileFormat.pdf`
- `sltest.testmanager.ReportFileFormat.zip`

**Attributes:**

|                          |                     |
|--------------------------|---------------------|
| <code>SetAccess</code>   | <code>public</code> |
| <code>GetAccess</code>   | <code>public</code> |
| <code>Dependent</code>   | <code>true</code>   |
| <code>NonCopyable</code> | <code>true</code>   |

**ReportPath — Path name of file to save report to**

character vector

Path name of file to save report to, specified as a character vector.

**Attributes:**

|                          |                     |
|--------------------------|---------------------|
| <code>SetAccess</code>   | <code>public</code> |
| <code>GetAccess</code>   | <code>public</code> |
| <code>Dependent</code>   | <code>true</code>   |
| <code>NonCopyable</code> | <code>true</code>   |

**SaveFigures — Option to save MATLAB figures with test results**

`false` (default) | `true`

Option to save MATLAB figures with test results, specified as `true` or `false`. If you want to include figures in results or reports, set this option to `true`.

**Attributes:**

|                          |                     |
|--------------------------|---------------------|
| <code>SetAccess</code>   | <code>public</code> |
| <code>GetAccess</code>   | <code>public</code> |
| <code>Dependent</code>   | <code>true</code>   |
| <code>NonCopyable</code> | <code>true</code>   |

**Title — Title of report**

character vector

Title of the report, specified as a character vector.

**Attributes:**

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

**Examples****Get and Set Test File Options**

```
% Create the test file, test suite, and test case structure
tf = sltest.testmanager.TestFile('API Test File');
ts = createTestSuite(tf, 'API Test Suite');
tc = createTestCase(ts, 'baseline', 'Baseline API Test Case');
```

```
% Get the test file options
opt = getOptions(tf);
```

```
% Set the title for the report, save figures, and include
% 3 rows of plots per page. Columns per page default to 2.
opt.Title = 'ABC Co. Test Results';
opt.SaveFigures = true;
opt.IncludeSimulationSignalPlots = true;
opt.NumPlotRowsPerPage = 3;
```

**See Also**

`getOptions (TestSuite)` | `getOptions (TestCase)` | `getOptions (TestFile)`

**Topics**

“Create, Store, and Open MATLAB Figures”  
“Export Test Results”

**Introduced in R2017a**

# sltest.testmanager.ParameterOverride class

**Package:** sltest.testmanager

Add or modify parameter override

## Description

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

The `sltest.testmanager.ParameterOverride` class is a handle class.

## Class Attributes

HandleCompatible true

For information on class attributes, see “Class Attributes”.

## Creation

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

## Properties

### Name — Parameter override name

string scalar | character vector

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

### Attributes:

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

### Value — Override value

numeric scalar | vector | string scalar

Value of the parameter override, specified as a scalar or vector value. If the value is a string scalar, it is evaluated as a MATLAB expression when the test executes.

### Attributes:

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

**Enabled — Enabled indicator**

0 | 1

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

**Attributes:**

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

**Source — Parameter override source**

character vector

Source of the parameter override, returned as a character vector. Two examples of sources are the base workspace and a Model block. The parameter is read-only.

**Attributes:**

|             |         |
|-------------|---------|
| SetAccess   | private |
| GetAccess   | public  |
| Dependent   | true    |
| NonCopyable | true    |

**Workspace — Workspace of the parameter**

character vector

Workspace of the parameter to override, returned as a character vector. The parameter is read-only.

**Attributes:**

|             |         |
|-------------|---------|
| SetAccess   | private |
| GetAccess   | public  |
| Dependent   | true    |
| NonCopyable | true    |

**Methods****Public Methods**

remove Remove parameter override

**Examples****Add Parameter Override to Test Case**

Open the model for this example.

```
openExample('sldemo_absbrake');
```

Create the test file, test suite, and test case structure.

```
tf = sltest.testmanager.TestFile("API Test File");
ts = createTestSuite(tf,"API Test Suite");
tc = createTestCase(ts,"baseline","Baseline API Test Case");
```

Remove the default test suite.

```
tsDel = getTestSuiteByName(tf, "New Test Suite 1");
remove(tsDel);
```

Assign the system under test to the test case.

```
setProperty(tc, "Model", "sldemo_absbrake");
```

Capture the baseline criteria.

```
baseline = captureBaselineCriteria(tc, "baseline_API.mat", true);
```

Test a new model parameter by overriding it in the test case parameter set.

```
ps = addParameterSet(tc, "Name", "API Parameter Set");
po = addParameterOverride(ps, "m", 55);
```

## See Also

sltest.testmanager.TestCase | sltest.testmanager.ParameterSet

## Topics

“Create and Run Test Cases with Scripts”

**Introduced in R2015b**

## sltest.testmanager.ParameterSet class

**Package:** sltest.testmanager

Add or modify parameter set

### Description

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

The sltest.testmanager.ParameterSet class is a handle class.

### Class Attributes

HandleCompatible true

For information on class attributes, see “Class Attributes”.

### Creation

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. If the parameter set was created from a MAT file, the name is derived from the MAT file name and is read-only.

#### Attributes:

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

#### FilePath — File path

character vector

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

#### Attributes:

|             |         |
|-------------|---------|
| SetAccess   | private |
| GetAccess   | public  |
| Dependent   | true    |
| NonCopyable | true    |



**Active – Active overrides indicator**

0 | 1

Indicates whether to use the overrides in the parameter set during test execution. If Active is 0, the overrides are not used. If Active is 1, the overrides are used.

**Attributes:**

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

**ExcelSpecifications – Sheet and range information for Excel file**

1-by-N array

Sheet and range information for Microsoft Excel file, returned as a 1-by-N array, where each row has a Sheet and Range value. Specify Range as shown in the table.

| Ways to specify Range                  | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|----------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 'Corner1:Corner2'<br>Rectangular Range | Specify the range using the syntax ' <i>Corner1:Corner2</i> ', where <i>Corner1</i> and <i>Corner2</i> are two opposing corners that define the region. For example, 'D2:H4' represents the 3-by-5 rectangular region between the two corners D2 and H4 on the worksheet. The 'Range' name-value pair argument is not case-sensitive, and uses Excel A1 reference style (see Excel help).<br><br><b>Example:</b> 'Range', 'Corner1:Corner2'                             |
| ' '<br>Unspecified or Empty            | If unspecified, the importing function automatically detects the used range.<br><br><b>Example:</b> 'Range', ''<br><br><b>Note:</b> <i>Used Range</i> refers to the rectangular portion of the spreadsheet that actually contains data. The importing function automatically detects the used range by trimming leading and trailing rows and columns that do not contain data. Text that is only white space is considered data and is captured within the used range. |
| 'Row1:Row2'<br>Row Range               | You can identify the range by specifying the beginning and ending rows using Excel row designators. Then <code>readtable</code> automatically detects the used column range within the designated rows. For instance, the importing function interprets the range specification '1:7' as an instruction to read all columns in the used range in rows 1 through 7 (inclusive).<br><br><b>Example:</b> 'Range', '1:7'                                                    |

| Ways to specify Range             | Description                                                                                                                                                                                                                                                                                                                                                                                                                |
|-----------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 'Column1:Column2'<br>Column Range | You can identify the range by specifying the beginning and ending columns using Excel column designators. Then <code>readtable</code> automatically detects the used row range within the designated columns. For instance, the importing function interprets the range specification 'A:F' as an instruction to read all rows in the used range in columns A through F (inclusive).<br><br><b>Example:</b> 'Range', 'A:F' |
| 'NamedRange'<br>Excel Named Range | In Excel, you can create names to identify ranges in the spreadsheet. For instance, you can select a rectangular portion of the spreadsheet and call it 'myTable'. If such named ranges exist in a spreadsheet, then the importing function can read that range using its name.<br><br><b>Example:</b> 'Range', 'myTable'                                                                                                  |

**Attributes:**

|             |        |
|-------------|--------|
| GetAccess   | public |
| SetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

Data Types: array

**Methods****Public Methods**

|                       |                                           |
|-----------------------|-------------------------------------------|
| addParameterOverride  | Add parameter override to parameter set   |
| export                | Export parameter set to Excel spreadsheet |
| getParameterOverrides | Get parameter overrides                   |
| remove                | Remove parameter set                      |

**Examples****Add Parameter Override to Test Case**

```
% Open the model for this example
openExample('sldemo_absbrake');

% Create the test file, test suite, and test case structure
tf = sltest.testmanager.TestFile('API Test File');
ts = createTestSuite(tf, 'API Test Suite');
tc = createTestCase(ts, 'baseline', 'Baseline API Test Case');

% Assign the system under test to the test case
setProperty(tc, 'Model', 'sldemo_absbrake');

% Test a new model parameter by overriding it in the test case
% parameter set
```

```
ps = addParameterSet(tc, 'Name', 'API Parameter Set');
po = addParameterOverride(ps, 'm', 55);
```

**See Also**

sltest.testmanager.TestCase

**Topics**

“Create and Run Test Cases with Scripts”

**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.

The sltest.testmanager.ResultSet class is a handle class.

### Class Attributes

HandleCompatible true

For information on class attributes, see “Class Attributes”.

### Creation

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

### Properties

#### CoverageResults — Coverage analysis results

row vector of cvdata objects

Coverage analysis results, returned as a row vector of aggregated cvdata objects across all simulations. For information on cvdata objects, see cvdata.

#### Attributes:

|             |         |
|-------------|---------|
| SetAccess   | private |
| GetAccess   | public  |
| Dependent   | true    |
| NonCopyable | true    |

#### Duration — Length of time the test ran, in seconds

duration

Length of time the test ran, in seconds, returned as a duration.

#### Attributes:

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

#### Release — Release in which the test was run

character vector

Release in which the test was run, returned as a character vector.

**Attributes:**

|           |         |
|-----------|---------|
| SetAccess | private |
| GetAccess | public  |

**NumDisabled — Number of disabled tests**

integer

Number of test cases that were disabled in the results set.

**Attributes:**

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

**NumFailed — Number of failed tests**

integer

Number of failed tests contained in the results set.

**Attributes:**

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

**NumPassed — Number of passed tests**

integer

Number of passed tests contained in the results set.

**Attributes:**

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

**NumTestCaseResults — Number of test case result children**

integer

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

**Attributes:**

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

**NumTestSuiteResults — Number of test suite result children**

integer

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

**Attributes:**

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

**NumTestFileResults – Number of test file result children**

integer

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

**Attributes:**

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

**NumTotal – Total number of tests**

integer

Total number of tests in the results set.

**Attributes:**

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

**Outcome – Test outcome**

Passed | Failed

Test outcome, returned as Passed or Failed.

**Attributes:**

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

**StartTime – Time the test began to run**

datetime

Time the test began to run, returned as a datetime.

**Attributes:**

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

**StopTime — Time the test completed**

datetime

Time the test completed, returned as a datetime.

**Attributes:**

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

**UserData — Custom data stored with result set**

any data type

Custom data stored with the result set, specified as any type of data. Use this field to add custom information about the results, such as the settings used to obtain the results.

**Attributes:**

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

**Methods****Public Methods**

|                     |                                                                |
|---------------------|----------------------------------------------------------------|
| getTestCaseResults  | Get test case results object                                   |
| getTestSuiteResults | Get test suite results object                                  |
| getTestFileResults  | Get test suite results object                                  |
| getCoverageResults  | Get coverage results                                           |
| remove              | Remove result set                                              |
| clearSimulationData | Delete specified simulation test results in result set objects |

**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;
```

**See Also**

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

**Topics**

“Create and Run Test Cases with Scripts”

**Introduced in R2015a**



# sltest.testmanager.SignalCriteria class

**Package:** sltest.testmanager

Add or modify signal criteria

## Description

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

The `sltest.testmanager.SignalCriteria` class is a `handle` class.

## Class Attributes

`HandleCompatible` `true`

For information on class attributes, see “Class Attributes”.

## Creation

### Description

`obj = getAllSignalCriteria` creates a `sltest.testmanager.SignalCriteria` object for a baseline or equivalence test case object.

## Properties

### AbsTol — Absolute tolerance

scalar

Absolute tolerance at a signal level, specified as a scalar. Set this value on the signal to override the value set in the baseline or equivalence criteria set.

### Attributes:

|                          |                     |
|--------------------------|---------------------|
| <code>GetAccess</code>   | <code>public</code> |
| <code>SetAccess</code>   | <code>public</code> |
| <code>Dependent</code>   | <code>true</code>   |
| <code>NonCopyable</code> | <code>true</code>   |

Data Types: `scalar`

### BlockPath — Signal block path

character vector

Signal block path, returned as a character vector. This property is read-only.

**Attributes:**

|             |         |
|-------------|---------|
| GetAccess   | public  |
| SetAccess   | private |
| Dependent   | true    |
| NonCopyable | true    |

Data Types: char

**DataSource — Signal data source**

character vector

Signal data source, returned as a character vector. This property is read-only.

**Attributes:**

|             |         |
|-------------|---------|
| GetAccess   | public  |
| SetAccess   | private |
| Dependent   | true    |
| NonCopyable | true    |

Data Types: char

**Enabled — Enabled indicator**

0 | 1

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

**Attributes:**

|             |        |
|-------------|--------|
| GetAccess   | public |
| SetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

Data Types: logical

**InterpMethod — Interpolation method**

'zoh' | 'linear'

The method of interpolation used to align signal data, specified as 'zoh' or 'linear'. The method can be one of the following:

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

**Attributes:**

|             |        |
|-------------|--------|
| GetAccess   | public |
| SetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

Data Types: char

**LaggingTol — Lagging time tolerance**

scalar

Lagging time tolerance at a signal level, specified as a scalar. Set this value on the signal to override the value set in the baseline or equivalence criteria set. LaggingTol is in seconds.

**Attributes:**

|             |        |
|-------------|--------|
| GetAccess   | public |
| SetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

Data Types: scalar

**LeadingTol — Leading time tolerance**

scalar

Leading time tolerance at a signal level, specified as a scalar. Set this value on the signal to override the value set in the baseline or equivalence criteria set. LeadingTol is in seconds.

**Attributes:**

|             |        |
|-------------|--------|
| GetAccess   | public |
| SetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

Data Types: scalar

**Name — Signal name**

character vector

Signal name, returned as a character vector.

**Attributes:**

|             |         |
|-------------|---------|
| GetAccess   | public  |
| SetAccess   | private |
| Dependent   | true    |
| NonCopyable | true    |

Data Types: char

**RelTol — Relative tolerance**

scalar

Relative tolerance at a signal level, specified as a scalar. Set this value on the signal to override the value set in the baseline or equivalence criteria set.

**Attributes:**

|             |        |
|-------------|--------|
| GetAccess   | public |
| SetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

Data Types: scalar

**SID — Signal identifier**

character vector

Signal identifier, returned as a character vector.

**Attributes:**

|             |         |
|-------------|---------|
| GetAccess   | public  |
| SetAccess   | private |
| Dependent   | true    |
| NonCopyable | true    |

Data Types: char

**SyncMethod — Time synchronization method**

'union' | 'intersection'

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

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

**Attributes:**

|             |        |
|-------------|--------|
| GetAccess   | public |
| SetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

Data Types: char

**Methods****Public Methods**

remove Remove signal criteria

**Examples**

## Set Absolute Tolerance in Baseline Criteria

```
% Open the model for this example
openExample('sldemo_absbrake');

% Create the test file, test suite, and test case structure
tf = sltest.testmanager.TestFile('API Test File');
ts = createTestSuite(tf,'API Test Suite');
tc = createTestCase(ts,'baseline','Baseline API Test Case');

% Remove the default test suite
tsDel = getTestSuiteByName(tf,'New Test Suite 1');
remove(tsDel);

% Assign the system under test to the test case
setProperty(tc,'Model','sldemo_absbrake');

% Capture the baseline criteria
baseline = captureBaselineCriteria(tc,'baseline_API.mat',true);

% Test a new model parameter by overriding it in the test case
% parameter set
ps = addParameterSet(tc,'Name','API Parameter Set');
po = addParameterOverride(ps,'m',55);

% Set the baseline criteria tolerance for one signal
sc = getSignalCriteria(baseline);
sc(1).AbsTol = 9;
```

## See Also

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

## Topics

“Create and Run Test Cases with Scripts”  
“Set Signal Tolerances”

## Introduced in R2015b

## sltest.testmanager.TestCase class

**Package:** sltest.testmanager

Create or modify test case

### Description

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

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

For MATLAB-based Simulink tests, test methods defined in the class file correspond to test cases. See “Test Models Using MATLAB-Based Simulink Tests” for more information.

The sltest.testmanager.TestCase class is a handle class.

### Class Attributes

HandleCompatible true

For information on class attributes, see “Class Attributes”.

## Creation

### Description

obj = sltest.testmanager.TestCase(parent, testtype, name) creates a sltest.testmanager.TestCase object as a child of the specified Parent. The TestType defaults to baseline and the test case is automatically assigned a default Name.

obj = sltest.testmanager.TestCase(parent, testtype, name) creates a sltest.testmanager.TestCase object with the specified TestType and test case Name..

## Properties

### Parent — Parent test suite

sltest.testmanager.TestSuite object

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

### Attributes:

|             |         |
|-------------|---------|
| GetAccess   | public  |
| SetAccess   | private |
| Dependent   | true    |
| NonCopyable | true    |

Data Types: sltest.testmanager.TestSuite

**TestType — Test case type**

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

The test case type, specified as 'baseline', 'equivalence', or 'simulation'.

- Baseline tests compare outputs from a simulation to expected results stored as baseline data.
- Equivalence tests compare the outputs from two different simulations. Simulations can run in different modes, such as normal simulation and software-in-the-loop.
- Simulation tests run the system under test and capture simulation data. If the system under test contains blocks that verify simulation, such as Test Sequence and Test Assessment blocks, pass/fail results are reflected in the simulation test results.

**Attributes:**

|             |         |
|-------------|---------|
| GetAccess   | public  |
| SetAccess   | private |
| Dependent   | true    |
| NonCopyable | true    |

Data Types: char

**Name — Test case name**

character vector

Name of the test case, specified as a character vector. If you do not specify a name, a unique name is created.

**Attributes:**

|             |        |
|-------------|--------|
| GetAccess   | public |
| SetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

Data Types: char

**Description — Test case description**

character vector

Test case description text, returned as a character vector.

**Attributes:**

|             |        |
|-------------|--------|
| GetAccess   | public |
| SetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

Data Types: char

**Enabled — Test execution indicator**

true or 1 | false or 0

Indicates if the test case will execute, specified as a logical value true or 1, or false or 0.

**Attributes:**

|             |        |
|-------------|--------|
| GetAccess   | public |
| SetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

Data Types: logical

**ReasonForDisabling – Disabled description**

character vector

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

**Attributes:**

|             |        |
|-------------|--------|
| GetAccess   | public |
| SetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

Data Types: char

**Requirements – Test file requirements**

structure array

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

**Attributes:**

|             |         |
|-------------|---------|
| GetAccess   | public  |
| SetAccess   | private |
| Dependent   | true    |
| NonCopyable | true    |

Data Types: struct

**RunOnTarget – Target indicator**

cell array

Indicates if the test case simulation runs on a target, returned as a cell array of logical values. For more information on real-time testing, see “Test Models in Real Time”

**Attributes:**

|             |         |
|-------------|---------|
| GetAccess   | public  |
| SetAccess   | private |
| Dependent   | true    |
| NonCopyable | true    |

Data Types: logical

**Tags – Tags for categorizing**

character vector | string array

Tags to use for categorizing, specified as a character vector or string array.



**Attributes:**

|             |        |
|-------------|--------|
| GetAccess   | public |
| SetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

Data Types: char | string

**TestFile – Parent test file**

sltest.testmanager.TestFile object

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

**Attributes:**

|             |         |
|-------------|---------|
| GetAccess   | public  |
| SetAccess   | private |
| Dependent   | true    |
| NonCopyable | true    |

Data Types: sltest.testmanager.TestFile

**TestPath – Test hierarchy**

character vector

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

**Attributes:**

|             |         |
|-------------|---------|
| GetAccess   | public  |
| SetAccess   | private |
| Dependent   | true    |
| NonCopyable | true    |

Data Types: char

**Methods****Public Methods**

|                            |                                                      |
|----------------------------|------------------------------------------------------|
| addBaselineCriteria        | Add baseline criteria to test case                   |
| addInput                   | Add input file to test case                          |
| addIteration               | Add test iteration to test case                      |
| addLoggedSignalSet         | Add logged signal set to a test case                 |
| addParameterSet            | Add parameter set                                    |
| captureBaselineCriteria    | Capture baseline criteria and add to test case       |
| captureEquivalenceCriteria | Capture equivalence criteria and add to test case    |
| convertTestType            | Convert test from one type to another                |
| copySimulationSettings     | Copy simulation setting in equivalence test case     |
| createInputDataFile        | Create file as basis for test case input signal data |
| deleteIterations           | Delete test iterations that belong to test case      |
| getAssessmentsCallback     | Get test case assessments callback                   |
| getBaselineCriteria        | Get baseline criteria                                |
| getCoverageSettings        | Get coverage settings                                |

|                        |                                                  |
|------------------------|--------------------------------------------------|
| getCustomCriteria      | Get custom criteria that belong to test case     |
| getEquivalenceCriteria | Get equivalence criteria from test case          |
| getInputs              | Get test case inputs                             |
| getIterations          | Get test iterations that belong to test case     |
| getLoggedSignalSets    | Get logged signal set from a test case           |
| getOptions             | Get test file options                            |
| getParameterSets       | Get test case parameter sets                     |
| getProperty            | Get test case property                           |
| getTestCaseResults     | Get test case results history                    |
| remove (test case)     | Remove test case                                 |
| run                    | Run test case                                    |
| setProperty            | Set test case property                           |
| setAssessmentsCallback | Set test case assessment callback                |
| addAssessment          | Add assessments to a test case                   |
| getAssessments         | Get assessments associated with test case        |
| addAssessmentSymbol    | Add assessment symbols to a test case            |
| getAssessmentSymbols   | Get assessment symbols associated with test case |

## Examples

### Create New Test File, Test Suite, and Test Case

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

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

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

testcase =

 TestCase with properties:

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

### See Also

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

### Topics

“Create and Run Test Cases with Scripts”  
 “Collect Coverage Using MATLAB-Based Simulink Tests”  
 “Author Class-Based Unit Tests in MATLAB”

“Test Models Using MATLAB-Based Simulink Tests”

**Introduced in R2015b**

## sltest.testmanager.TestCaseResult class

**Package:** sltest.testmanager

**Superclasses:** sltest.testmanager.TestIterationResult

Access test case results data

### Description

An `sltest.testmanager.TestCaseResult` enables you to access results from executing test cases or test files.

The `sltest.testmanager.TestCaseResult` class is a handle class.

### Class Attributes

HandleCompatible true

For information on class attributes, see “Class Attributes”.

### Creation

`tcr = getTestCaseResults(ResultSet)` returns `tcr` a test case result from a `ResultSet`.

### Properties

#### Duration — Length of time the test case ran, in seconds

`duration`

Length of time the test case ran, in seconds, returned as a duration.

#### Attributes:

|             |         |
|-------------|---------|
| SetAccess   | private |
| GetAccess   | public  |
| Dependent   | true    |
| NonCopyable | true    |

#### NumDisabledIterations — Number of disabled tests

`integer`

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

#### Attributes:

|             |         |
|-------------|---------|
| SetAccess   | private |
| GetAccess   | public  |
| Dependent   | true    |
| NonCopyable | true    |

#### NumFailedIterations — Number of failed tests

`integer`

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

**Attributes:**

|             |         |
|-------------|---------|
| SetAccess   | private |
| GetAccess   | public  |
| Dependent   | true    |
| NonCopyable | true    |

**NumIncompleteIterations – Number of incomplete tests**

integer

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

**Attributes:**

|             |         |
|-------------|---------|
| SetAccess   | private |
| GetAccess   | public  |
| Dependent   | true    |
| NonCopyable | true    |

**NumPassedIterations – Number of passed tests**

integer

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

**Attributes:**

|             |         |
|-------------|---------|
| SetAccess   | private |
| GetAccess   | public  |
| Dependent   | true    |
| NonCopyable | true    |

**NumTotalIterations – Total number of tests**

integer

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

**Attributes:**

|             |         |
|-------------|---------|
| SetAccess   | private |
| GetAccess   | public  |
| Dependent   | true    |
| NonCopyable | true    |

**Outcome – Outcome of test case result**

enum

Outcome of an individual test case result, returned as an enumerated value. Valid values are:

- Disabled
- Failed
- Incomplete
- Passed
- Running

- Scheduled
- Untested

**Attributes:**

|             |         |
|-------------|---------|
| SetAccess   | private |
| GetAccess   | public  |
| Dependent   | true    |
| NonCopyable | true    |

**Parent — Parent of the result object**

object

Parent of the result. The parent of a test case result is a test suite result or result set object.

**Attributes:**

|             |         |
|-------------|---------|
| SetAccess   | private |
| GetAccess   | public  |
| Dependent   | true    |
| NonCopyable | true    |

**Release — Release in which the test was run**

character vector

Release in which the test was run, returned as a character vector.

**Attributes:**

|             |         |
|-------------|---------|
| SetAccess   | private |
| GetAccess   | public  |
| Dependent   | true    |
| NonCopyable | true    |

**RunOnTarget — Target indicator**

cell array

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

**Attributes:**

|             |         |
|-------------|---------|
| SetAccess   | private |
| GetAccess   | public  |
| Dependent   | true    |
| NonCopyable | true    |

**StartTime — Time the test case began to run**

datetime

Time the test case began to run, returned as a datetime.

**Attributes:**

|             |         |
|-------------|---------|
| SetAccess   | private |
| GetAccess   | public  |
| Dependent   | true    |
| NonCopyable | true    |

**StopTime — Time the test case completed**

datetime

Time the test case completed, returned as a datetime.

**Attributes:**

|             |         |
|-------------|---------|
| SetAccess   | private |
| GetAccess   | public  |
| Dependent   | true    |
| NonCopyable | true    |

**Tags — Tags to filter test file results**

string array

Tags to filter the test file results. Use tags to view a subset of the test results. See “Tags” for more information.

**Attributes:**

|             |         |
|-------------|---------|
| SetAccess   | private |
| GetAccess   | public  |
| Dependent   | true    |
| NonCopyable | true    |

**TestCasePath — Result hierarchy path**

character vector

The hierarchy path in the parent result set.

**Attributes:**

|             |         |
|-------------|---------|
| SetAccess   | private |
| GetAccess   | public  |
| Dependent   | true    |
| NonCopyable | true    |

**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.

**Attributes:**

|             |         |
|-------------|---------|
| SetAccess   | private |
| GetAccess   | public  |
| Dependent   | true    |
| NonCopyable | true    |

**TestFilePath — Test file path**

character vector

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

**Attributes:**

|             |         |
|-------------|---------|
| SetAccess   | private |
| GetAccess   | public  |
| Dependent   | true    |
| NonCopyable | true    |

**TestSequenceScenario — Test sequence scenario**

struct

Test sequence scenario used in the test case, returned as a struct. The struct contains two fields, `TestSequenceBlock` and `TestSequenceScenario`. The `TestSequenceBlock` field is the path of the Test Sequence block that contains the scenario. The `TestSequenceScenario` is the name of the scenario that ran during the test. The test sequence information is returned in the `TestCaseResult` object only if the test case did not include iterations. If iterations were included, the `TestSequenceScenario` is returned in a `TestIterationResults` object.

**Attributes:**

|             |         |
|-------------|---------|
| SetAccess   | private |
| GetAccess   | public  |
| Dependent   | true    |
| NonCopyable | true    |

**ErrorMessages — Error messages**

struct

Error messages produced by the test case, returned as a struct. The struct contains two fields, `TestCase` and `Simulation`. The `TestCase` field is a string array that contains the names of the test cases. The `Simulation` field is a struct with two fields, `simulationIndex` and `messages`. The `messages` field is a string array that contains the text of the error messages.

**Attributes:**

|             |         |
|-------------|---------|
| SetAccess   | private |
| GetAccess   | public  |
| Dependent   | true    |
| NonCopyable | true    |

**LogMessages — Log messages**

struct

Log messages produced by the test case, returned as a struct. The struct contains two fields, `TestCase` and `Simulation`. The `TestCase` field is a string array that contains the names of the test cases. The `Simulation` field is a struct with two fields, `simulationIndex` and `messages`. The `messages` field is a string array that contains the text of the log messages.



**Attributes:**

|             |         |
|-------------|---------|
| SetAccess   | private |
| GetAccess   | public  |
| Dependent   | true    |
| NonCopyable | true    |

**UserData — Custom data stored with test case results**

any data type

Custom data stored with the test case results, specified as any type of data. Use this field to add custom information, such as the settings used to obtain the results.

**Attributes:**

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

**Methods****Public Methods**

|                         |                                                           |
|-------------------------|-----------------------------------------------------------|
| getBaselineRun          | Get test case baseline dataset                            |
| getTestCase             | Get test case that produced result                        |
| getComparisonResult     | Get test data comparison result                           |
| getComparisonRun        | Get test case signal comparison results                   |
| getCoverageResults      | Get coverage results                                      |
| getCustomCriteriaPlots  | Get plots from test case custom criteria                  |
| getCustomCriteriaResult | Get custom criteria results from test case result         |
| getIterationResults     | Get iteration results                                     |
| getInputRuns            | Get inputs from simulations captured with the test result |
| getOutputRuns           | Get test case simulation output results                   |
| getSimulationPlots      | Get plots from test case callbacks                        |
| getVerifyRuns           | Get test case verify statement                            |

**Examples****Get Test Case Result From Results Set**

Run a test file from the test manager and obtain a results set. In this example, the Outcome of the first test case is Failed because a model is not specified.

```
% Create a test file, test suite and test case
tf = sltest.testmanager.TestFile('My Test File');
ts = createTestSuite(tf,'Baseline Test Suite');
tc = createTestCase(ts,'baseline','Baseline Test Case');

% Run test file and obtain the output results set
resultset = sltest.testmanager.run;

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

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

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

% Get error messages
tcr.ErrorMessages.Simulation.messages

% View the Outcome field of the first test case
tcr(1).Outcome
```

## See Also

sltest.testmanager.TestSuiteResult | sltest.testmanager.TestFileResult

## Topics

“Create and Run Test Cases with Scripts”

“Programmatically Create and Run Test Sequence Scenarios”

**Introduced in R2015a**

# sltest.testmanager.TestFile class

**Package:** sltest.testmanager

Create or modify test file

## Description

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

For MATLAB-based Simulink tests, test files correspond to the file name. See “Test Models Using MATLAB-Based Simulink Tests” for more information.

The `sltest.testmanager.TestFile` class is a `handle` class.

## Class Attributes

|                  |      |
|------------------|------|
| HandleCompatible | true |
|------------------|------|

For information on class attributes, see “Class Attributes”.

## Creation

### Description

`obj = sltest.testmanager.TestFile(filePath)` opens the `sltest.testmanager.TestFile` that exists with the same file name and at the specified `filePath` location. If the test file does not exist at the location, a new one is created with a default test suite and test case as children of the test file. The default test case type is a baseline test case.

`obj = sltest.testmanager.TestFile(filePath,mode)` overwrites the existing test file if `mode` is set to `true`. The default for `mode` is `false`.

## Properties

### FilePath — File path and name

character vector

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

Example: 'C:\MATLAB\test\_file.mldatx'

### Attributes:

|             |         |
|-------------|---------|
| GetAccess   | public  |
| SetAccess   | private |
| Dependent   | true    |
| NonCopyable | true    |

Data Types: char

**Description — Test file description**

character vector

Test file description text, specified as a character vector.

**Attributes:**

|             |        |
|-------------|--------|
| GetAccess   | public |
| SetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

Data Types: char

**Dirty — Unsaved changes indicator**

0 | 1

Indicates if the test file has unsaved changes, returned as 0 or 1. If the value is 0, there are no unsaved changes. The value is 1 if there are unsaved changes.

**Attributes:**

|             |         |
|-------------|---------|
| GetAccess   | public  |
| SetAccess   | private |
| Dependent   | true    |
| NonCopyable | true    |

Data Types: logical

**Enabled — Test file execution indicator**

true | false

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

|             |        |
|-------------|--------|
| GetAccess   | public |
| SetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

Data Types: logical

**Name — Test file name**

character vector

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

**Attributes:**

|             |         |
|-------------|---------|
| GetAccess   | public  |
| SetAccess   | private |
| Dependent   | true    |
| NonCopyable | true    |

Data Types: char

**ReasonForDisabling — Disabled description**

character vector

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

**Attributes:**

|                          |                     |
|--------------------------|---------------------|
| <code>GetAccess</code>   | <code>public</code> |
| <code>SetAccess</code>   | <code>public</code> |
| <code>Dependent</code>   | <code>true</code>   |
| <code>NonCopyable</code> | <code>true</code>   |

Data Types: char

**Requirements — Test file requirements**

structure array

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

**Attributes:**

|                          |                      |
|--------------------------|----------------------|
| <code>GetAccess</code>   | <code>public</code>  |
| <code>SetAccess</code>   | <code>private</code> |
| <code>Dependent</code>   | <code>true</code>    |
| <code>NonCopyable</code> | <code>true</code>    |

Data Types: struct

**Tags — Tags for categorizing**

character vector | string array

Tags to use for categorizing, specified as a character vector or string array.

**Attributes:**

|                          |                     |
|--------------------------|---------------------|
| <code>GetAccess</code>   | <code>public</code> |
| <code>SetAccess</code>   | <code>public</code> |
| <code>Dependent</code>   | <code>true</code>   |
| <code>NonCopyable</code> | <code>true</code>   |

Data Types: char | string

**Methods****Public Methods**

|                                     |                                                                              |
|-------------------------------------|------------------------------------------------------------------------------|
| <code>close</code>                  | Close test file in Test Manager                                              |
| <code>convertTestType</code>        | Convert test from one type to another                                        |
| <code>createTestForSubsystem</code> | (To be removed) Create test harness and test case for subsystem in test file |
| <code>createTestSuite</code>        | Create new test suite                                                        |
| <code>getCoverageSettings</code>    | Get coverage settings                                                        |
| <code>getOptions</code>             | Get and set test file options                                                |
| <code>getProperty</code>            | Get test file property                                                       |
| <code>getTestCaseByName</code>      | Get test case object by name                                                 |

|                                 |                                             |
|---------------------------------|---------------------------------------------|
| <code>getTestCases</code>       | Get test cases in a test file               |
| <code>getAllTestCases</code>    | Get all test cases in a test file           |
| <code>getTestSuiteByName</code> | Get test suite object by name               |
| <code>getTestSuites</code>      | Get test suites at first level of test file |
| <code>run</code>                | Run test cases in test file                 |
| <code>saveToFile</code>         | Save test file                              |
| <code>setProperty</code>        | Set test file property                      |

## Examples

### Create New Test File

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

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

```
testfile =
```

```
 TestFile with properties:
```

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

### See Also

`sltest.testmanager.TestSuite` | `sltest.testmanager.TestCase` | `sltest.TestCase`

### Topics

“Create and Run Test Cases with Scripts”

“Test Models Using MATLAB-Based Simulink Tests”

“Using MATLAB-Based Simulink Tests in the Test Manager”

“Collect Coverage Using MATLAB-Based Simulink Tests”

### Introduced in R2015b

# sltest.testmanager.TestFileResult class

**Package:** sltest.testmanager

**Superclasses:** sltest.testmanager.ResultSet

Access test file results data

## Description

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

The sltest.testmanager.TestFileResult class is a handle class.

### Class Attributes

HandleCompatible true

For information on class attributes, see “Class Attributes”.

## Creation

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

## Properties

### Duration — Length of time the test ran, in seconds

duration

Length of time the test ran, in seconds, returned as a duration.

#### Attributes:

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

### NumDisabled — Number of disabled tests

integer

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

#### Attributes:

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

**NumFailed — Number of failed tests**

integer

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

**Attributes:**

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

**NumIncomplete — Number of incomplete tests**

integer

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

**Attributes:**

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

**NumPassed — Number of passed tests**

integer

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

**Attributes:**

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

**NumTestCaseResults — Number of test case result children**

integer

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

**Attributes:**

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

**NumTestSuiteResults — Number of test suite result children**

integer

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



**Attributes:**

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

**NumTotal — Total number of tests**

integer

The total number of tests in the test file result.

**Attributes:**

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

**Parent — Parent of the result object**

object

Parent of the result. The parent of a test file result is a result set object.

**Attributes:**

|             |         |
|-------------|---------|
| SetAccess   | private |
| GetAccess   | public  |
| Dependent   | true    |
| NonCopyable | true    |

**Release — Release in which the test was run**

character vector

Release in which the test was run, returned as a character vector.

**Attributes:**

|             |         |
|-------------|---------|
| SetAccess   | private |
| GetAccess   | public  |
| Dependent   | true    |
| NonCopyable | true    |

**StartTime — Time the test began to run**

datetime

Time the test began to run, returned as a datetime.

**Attributes:**

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

**StopTime — Time the test completed**

datetime

Time the test completed, returned as a datetime.

**Attributes:**

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

**Tags — Tags to filter test file results**

string array

Tags to filter the test file results. Use tags to view a subset of the test results. See “Tags” for more information.

**Attributes:**

|             |         |
|-------------|---------|
| SetAccess   | private |
| GetAccess   | public  |
| Dependent   | true    |
| NonCopyable | true    |

**TestFilePath — Result hierarchy path**

character vector

The hierarchy path in the result set.

**Attributes:**

|             |         |
|-------------|---------|
| SetAccess   | private |
| GetAccess   | public  |
| Dependent   | true    |
| NonCopyable | true    |

**TestFilePath — Test file path**

character vector

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

**Attributes:**

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

**UserData — Custom data stored with test file results**

any data type

Custom data stored with the test file results, specified as any type of data. Use this field to add custom information, such as the settings used to obtain the results.

**Attributes:**

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

**Methods****Public Methods**

|                     |                                  |
|---------------------|----------------------------------|
| getTestSuiteResults | Get test suite results object    |
| getCoverageResults  | Get coverage results             |
| getCleanupPlots     | Get plots from cleanup callbacks |
| getSetupPlots       | Get plots from setup callbacks   |

**Examples****Get Test File Result From Results Set**

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

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

**See Also**

sltest.testmanager.TestSuiteResult | sltest.testmanager.TestCaseResult

**Topics**

“Create and Run Test Cases with Scripts”

**Introduced in R2016a**

## sltest.testmanager.TestInput class

**Package:** sltest.testmanager

Add or modify test input

### Description

Instances of `sltest.testmanager.TestInput` are sets of signal input data that can be mapped to override the inputs in the system under test.

The `sltest.testmanager.TestInput` class is a `handle` class.

### Class Attributes

`HandleCompatible` `true`

For information on class attributes, see “Class Attributes”.

## Creation

### Description

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

## Properties

### 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.

### Attributes:

|                          |                     |
|--------------------------|---------------------|
| <code>GetAccess</code>   | <code>public</code> |
| <code>SetAccess</code>   | <code>public</code> |
| <code>Dependent</code>   | <code>true</code>   |
| <code>NonCopyable</code> | <code>true</code>   |

Data Types: `logical`

### ExcelSpecifications — Sheet and range information for Excel baseline file

1-by-N array

Sheet and range information for Microsoft Excel baseline file, returned as a 1-by-N array, where each row has a Sheet and Range value. Specify Range as shown in the table.

| Ways to specify Range                  | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|----------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 'Corner1:Corner2'<br>Rectangular Range | Specify the range using the syntax ' <i>Corner1:Corner2</i> ', where <i>Corner1</i> and <i>Corner2</i> are two opposing corners that define the region. For example, 'D2:H4' represents the 3-by-5 rectangular region between the two corners D2 and H4 on the worksheet. The 'Range' name-value pair argument is not case-sensitive, and uses Excel A1 reference style (see Excel help).<br><br><b>Example:</b> 'Range', 'Corner1:Corner2'                             |
| ' '<br>Unspecified or Empty            | If unspecified, the importing function automatically detects the used range.<br><br><b>Example:</b> 'Range', ''<br><br><b>Note:</b> <i>Used Range</i> refers to the rectangular portion of the spreadsheet that actually contains data. The importing function automatically detects the used range by trimming leading and trailing rows and columns that do not contain data. Text that is only white space is considered data and is captured within the used range. |
| 'Row1:Row2'<br>Row Range               | You can identify the range by specifying the beginning and ending rows using Excel row designators. Then <code>readtable</code> automatically detects the used column range within the designated rows. For instance, the importing function interprets the range specification '1:7' as an instruction to read all columns in the used range in rows 1 through 7 (inclusive).<br><br><b>Example:</b> 'Range', '1:7'                                                    |
| 'Column1:Column2'<br>Column Range      | You can identify the range by specifying the beginning and ending columns using Excel column designators. Then <code>readtable</code> automatically detects the used row range within the designated columns. For instance, the importing function interprets the range specification 'A:F' as an instruction to read all rows in the used range in columns A through F (inclusive).<br><br><b>Example:</b> 'Range', 'A:F'                                              |
| 'NamedRange'<br>Excel Named Range      | In Excel, you can create names to identify ranges in the spreadsheet. For instance, you can select a rectangular portion of the spreadsheet and call it 'myTable'. If such named ranges exist in a spreadsheet, then the importing function can read that range using its name.<br><br><b>Example:</b> 'Range', 'myTable'                                                                                                                                               |

**Attributes:**

|             |        |
|-------------|--------|
| GetAccess   | public |
| SetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

Data Types: array

**FilePath — File path**

character vector

File path of the test input, returned as a character vector.

Example: 'C:\MATLAB\sltestExampleInputs.xlsx'

**Attributes:**

|             |         |
|-------------|---------|
| GetAccess   | public  |
| SetAccess   | private |
| Dependent   | true    |
| NonCopyable | true    |

Data Types: char

**InputString — Input**

character vector

Input evaluated during test case execution in the LoadExternalInput configuration parameter of the system under test, specified as a character vector.

Example: 'Acceleration.getElement(1),Acceleration.getElement(2)'

**Attributes:**

|             |        |
|-------------|--------|
| GetAccess   | public |
| SetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

Data Types:

**Name — Test input name**

character vector

Name of the test input, returned as a character vector.

Example: 'sltestExampleInputs.xlsx'

**Attributes:**

|             |         |
|-------------|---------|
| GetAccess   | public  |
| SetAccess   | private |
| Dependent   | true    |
| NonCopyable | true    |

Data Types: char

**MappingStatus — Input mapping status**

character vector

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

Example: 'Successfully mapped inputs.'

**Attributes:**

|             |         |
|-------------|---------|
| GetAccess   | public  |
| SetAccess   | private |
| Dependent   | true    |
| NonCopyable | true    |

Data Types: char

**Methods****Public Methods**

|                       |                                                                      |
|-----------------------|----------------------------------------------------------------------|
| addExcelSpecification | Add a Microsoft Excel sheet to baseline criteria or test case inputs |
| map                   | Map test input to system under test                                  |
| remove                | Remove test input                                                    |

**Examples****Add Microsoft® Excel® Data as Input**

This example shows how to add data from a Microsoft® Excel® spreadsheet and map it to a test case. Only the two sheets that have data are added and mapped.

**Load the Example Model**

```
open_system('sltestExcelExample');
```

**Create a New Test File**

```
tf = sltest.testmanager.TestFile('input_test_file.mldatx');
```

**Get the Test Suite and Test Case Objects**

```
ts = getTestSuites(tf);
tc = getTestCases(ts);
```

**Add the Example Model as the System Under Test**

```
setProperty(tc, 'Model', 'sltestExcelExample');
```

**Add Excel® Data to Inputs Section and Specify Sheets to Add**

```
excelfile = 'sltestExampleInputs.xlsx';
input = addInput(tc, excelfile, 'Sheets', ["Acceleration", "Braking"]);
```

**Map the Input Signal for the Sheets by Block Name**

```
map(input(1), 0);
map(input(2), 0);
```

**See Also**

sltest.testmanager.TestCase | addInput

**Topics**

“Create and Run Test Cases with Scripts”

**Introduced in R2015b**



# sltest.testmanager.TestIteration class

**Package:** sltest.testmanager

Create or modify test iteration

## Description

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

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

```

1
2 %% Iterate over all External Inputs.
3
4 % Determine the number of possible iterations
5 numSteps = length(sltest_externalInputs);
6
7 % Create each iteration
8 for k = 1 : numSteps
9 % Set up a new iteration object
10 testItr = sltestiteration();
11
12 % Set iteration settings
13 setTestParam(testItr, 'ExternalInput', sltest_externalInputs{k});
14
15 % Register the iteration to run in this test case
16 addIteration(sltest_testCase, testItr); % You can pass in an optional iteration name
17 end
18

```

Iteration Templates [Generate an iteration script using templates](#)

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

The `sltest.testmanager.TestIteration` class is a handle class.

### Class Attributes

HandleCompatible

true

For information on class attributes, see “Class Attributes”.

## Creation

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

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

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

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

## Properties

### Name — Iteration name

empty (default) | character vector

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

Example: 'Iteration 1a'

#### Attributes:

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

### ModelParams — Model parameter overrides

empty (default) | cell array

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

#### Attributes:

|             |           |
|-------------|-----------|
| SetAccess   | protected |
| GetAccess   | public    |
| Dependent   | true      |
| NonCopyable | true      |

### TestParams — Test parameter settings

empty (default) | cell array

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

#### Attributes:

|             |           |
|-------------|-----------|
| SetAccess   | protected |
| GetAccess   | public    |
| Dependent   | true      |
| NonCopyable | true      |

**Variables – Model variable overrides**

empty (default) | cell array

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

**Attributes:**

|             |           |
|-------------|-----------|
| SetAccess   | protected |
| GetAccess   | public    |
| Dependent   | true      |
| NonCopyable | true      |

**Enabled – Run iteration with test case**

false (default) | true

Option to run the iteration with the test case, specified as a logical.

**Attributes:**

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

**Methods****Public Methods**

|                     |                                    |
|---------------------|------------------------------------|
| getIterationResults | Get test iteration results history |
| setModelParam       | Set model parameter for iteration  |
| setTestParam        | Set test case iteration parameter  |
| setVariable         | Set model variable override        |

**Examples****Model Parameter Sweep**

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

```
% Open the model for this example
openExample('sldemo_absbrake');

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

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

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

```

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

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

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

```

### Iterate Over Parameter Sets

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

```

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

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

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

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

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

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

```

### Alternatives

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

### See Also

`sltest.testmanager.TestIterationResult`

#### Topics

“Test Iterations”

“Create and Run Test Cases with Scripts”

### Introduced in R2016a

# sltest.testmanager.TestIterationResult class

**Package:** sltest.testmanager

Access test iteration result data

## Description

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

The `sltest.testmanager.TestIterationResult` class is a `handle` class.

## Class Attributes

`HandleCompatible` true

For information on class attributes, see “Class Attributes”.

## Creation

### Description

The function `sltest.testmanager.run` creates a `sltest.testmanager.ResultSet` object, which contains the test case result object. For an example, see “Get Test Iteration Results” on page 2-131

## Properties

### Outcome — Outcome of test iteration result

0 | 1 | 2 | 3

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

### Attributes:

|                          |                      |
|--------------------------|----------------------|
| <code>GetAccess</code>   | <code>public</code>  |
| <code>SetAccess</code>   | <code>private</code> |
| <code>Dependent</code>   | <code>true</code>    |
| <code>NonCopyable</code> | <code>true</code>    |

Data Types: `integer`

### Duration — Length of time the test iteration ran, in seconds

`duration`

Length of time the test iteration ran, in seconds, returned as a duration.

**Attributes:**

|             |         |
|-------------|---------|
| GetAccess   | public  |
| SetAccess   | private |
| Dependent   | true    |
| NonCopyable | true    |

Data Types: duration

**StartTime — Time the test iteration began to run**

datetime

Time the test iteration began to run, returned as a datetime.

**Attributes:**

|             |         |
|-------------|---------|
| GetAccess   | public  |
| SetAccess   | private |
| Dependent   | true    |
| NonCopyable | true    |

Data Types: datetime

**StopTime — Time the test iteration completed**

datetime

Time the test completed, returned as a datetime.

**Attributes:**

|             |         |
|-------------|---------|
| GetAccess   | public  |
| SetAccess   | private |
| Dependent   | true    |
| NonCopyable | true    |

Data Types: datetime

**TestFilePath — Test file path**

character vector

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

**Attributes:**

|             |         |
|-------------|---------|
| GetAccess   | public  |
| SetAccess   | private |
| Dependent   | true    |
| NonCopyable | true    |

Data Types: char

**TestCasePath — Result hierarchy path**

character vector

The hierarchy path in the parent result set.

**Attributes:**

|             |         |
|-------------|---------|
| GetAccess   | public  |
| SetAccess   | private |
| Dependent   | true    |
| NonCopyable | true    |

Data Types: char

**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.

**Attributes:**

|             |         |
|-------------|---------|
| GetAccess   | public  |
| SetAccess   | private |
| Dependent   | true    |
| NonCopyable | true    |

Data Types: char

**TestSequenceScenario — Test sequence scenario**

struct

Test sequence scenario used in the test iteration, returned as a struct. The struct contains two fields, `TestSequenceBlock` and `TestSequenceScenario`. The `TestSequenceBlock` field is the path of the Test Sequence block containing the scenario that ran for this iteration. The `TestSequenceScenario` is the name of that scenario. The test sequence information is returned in a `TestIterationResult` object only if the test case included iterations. If iterations were not included, the `TestSequenceScenario` is returned in a `TestCaseResults` object.

**Attributes:**

|             |         |
|-------------|---------|
| GetAccess   | public  |
| SetAccess   | private |
| Dependent   | true    |
| NonCopyable | true    |

Data Types: struct

**RunOnTarget — Target indicator**

cell array

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

**Attributes:**

|             |         |
|-------------|---------|
| GetAccess   | public  |
| SetAccess   | private |
| Dependent   | true    |
| NonCopyable | true    |

Data Types: logical

**Parent — Parent of the result object**`sltest.testmanager.TestCaseResult` object

Parent of the result. The parent of a test iteration result is a test case result object.

**Attributes:**

|                          |                      |
|--------------------------|----------------------|
| <code>GetAccess</code>   | <code>public</code>  |
| <code>SetAccess</code>   | <code>private</code> |
| <code>Dependent</code>   | <code>true</code>    |
| <code>NonCopyable</code> | <code>true</code>    |

Data Types: `sltest.testmanager.TestCaseResult`

**ErrorMessages — Error messages**

array of strings

Error messages produced by the iteration, returned as a array of strings.

**Attributes:**

|                          |                      |
|--------------------------|----------------------|
| <code>GetAccess</code>   | <code>public</code>  |
| <code>SetAccess</code>   | <code>private</code> |
| <code>Dependent</code>   | <code>true</code>    |
| <code>NonCopyable</code> | <code>true</code>    |

Data Types: `string`

**LogMessages — Log messages**

array of strings

Log messages produced by the iteration, returned as a array of strings.

**Attributes:**

|                          |                      |
|--------------------------|----------------------|
| <code>GetAccess</code>   | <code>public</code>  |
| <code>SetAccess</code>   | <code>private</code> |
| <code>Dependent</code>   | <code>true</code>    |
| <code>NonCopyable</code> | <code>true</code>    |

Data Types: `string`

**UserData — Custom data stored with test iteration results**

any data type

Custom data stored with the test iteration results, specified as any type of data. Use this field to add custom information, such as the settings used to obtain the results.

**Methods****Public Methods**

|                                  |                                              |
|----------------------------------|----------------------------------------------|
| <code>getBaselineRun</code>      | Get test iteration baseline dataset          |
| <code>getComparisonResult</code> | Get test data comparison result              |
| <code>getComparisonRun</code>    | Get test iteration signal comparison results |
| <code>getCoverageResults</code>  | Get coverage results                         |



|                         |                                                 |
|-------------------------|-------------------------------------------------|
| getCustomCriteriaPlots  | Get plots from custom criteria                  |
| getCustomCriteriaResult | Get custom criteria results from test iteration |
| getOutputRuns           | Get test iteration simulation output results    |
| getSimulationPlots      | Get plots from callbacks                        |
| getTestIteration        | Get test iteration that produced result         |
| getVerifyRuns           | Get test iteration verify statement             |

## Examples

### Get Test Iteration Results

```
% Open the model for this example
openExample('sldemo_absbrake');

% Create the test file, suite, and case
tf = sltest.testmanager.TestFile...
 ('Get Test Iteration Results File');
ts = createTestSuite(tf,'Test Suite');
tc = createTestCase(ts,'baseline','Test Case');

% Remove the default test suite
tsDel = getTestSuiteByName(tf,'New Test Suite 1');
remove(tsDel);

% Assign the system under test to the test case
setProperty(tc,'Model','sldemo_absbrake');

% Specify iterations
vars = 32 : 0.5 : 34;

for k = 1 : length(vars)

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

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

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

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

% Run the test and capture results
resultset = run(tf);
tfr = getTestFileResults(resultset);
tsr = getTestSuiteResults(tfr);
tcr = getTestCaseResults(tsr);
tir = getIterationResults(tcr);
```

```
% Get the test case type from first iteration
testType = tir(1).TestCaseType;
```

### **See Also**

sltest.testmanager.TestIteration

### **Topics**

“Test Iterations”

“Programmatically Create and Run Test Sequence Scenarios”

“Create and Run Test Cases with Scripts”

**Introduced in R2016a**

## sltest.testmanager.TestOptions class

**Package:** sltest.testmanager

Options for adding test coverage

### Description

Use an `sltest.testmanager.TestOptions` object to specify the options to use when you add missing coverage to a test. Use this object with `sltest.testmanager.addTestsForMissingCoverage`.

The `sltest.testmanager.TestOptions` class is a `handle` class.

### Creation

`testOpts = sltest.testmanager.TestOptions()` creates a `TestOptions` object with default properties. After creating the object, set the `TestFilePath`, `TestCase`, `Parent`, `HarnessOwner`, and `HarnessName` properties.

`testOpts = sltest.testmanager.TestOptions(tc)` creates a `TestOptions` object for the `sltest.testmanager.TestCase` object, `tc`. Use this syntax to append added coverage to the test case. The `TestFilePath`, `Parent`, `HarnessOwner`, and `HarnessName` properties are automatically populated.

`testOpts = sltest.testmanager.TestOptions(harnessOwner, harnessName)` creates a `TestOptions` object for the test harness, `harnessName`, that is associated with `model` or `model` element, `harnessOwner`. Use this syntax to reuse a test harness for a new test case.

### Public Properties

#### Model — Model name

`string`

Model name, specified as a string.

#### Attributes:

|                        |                     |
|------------------------|---------------------|
| <code>GetAccess</code> | <code>public</code> |
| <code>SetAccess</code> | <code>public</code> |

#### TestFilePath — Test file name

`string`

Test file name, specified as a string. If you specify values for either or both the `TestCase` and `Parent` properties, the `TestFilePath` must be the same as the `FilePath` of the existing test case or parent. If you do not specify values for both `TestCase` and `Parent`, you must specify a `TestFilePath` for a new or existing test file.

Example: `"TestFilePath", "myTests/baseline_test2.mldatx"`

**Attributes:**

|           |        |
|-----------|--------|
| GetAccess | public |
| SetAccess | public |

**TestCase — Name of new or existing test case**

`sltest.testmanager.TestCase` object

Name of new or existing test case for which to add coverage, specified as an `sltest.testmanager.TestCase` object.

**Attributes:**

|           |        |
|-----------|--------|
| GetAccess | public |
| SetAccess | public |

**Parent — Test suite that contains the test case**

`sltest.testmanager.TestSuite` object

Test suite that contains the test case, specified as a `sltest.testmanager.TestSuite` object.

**Attributes:**

|           |        |
|-----------|--------|
| GetAccess | public |
| SetAccess | public |

**TestCaseType — Type of test**

"baseline" (default) | "equivalence" | "simulation"

Type of test, specified for new test cases or iterations as "baseline", "equivalence", or "simulation". For existing test cases, this property is automatically set to the type of the specified test case.

**Attributes:**

|           |        |
|-----------|--------|
| GetAccess | public |
| SetAccess | public |

**SrcSimIndex — Simulation index for equivalence tests**

1 | 2 | [1,2]

Simulation index for equivalence tests, specified as 1, 2, or [1,2]. Specifying 1 or 2 adds the missing coverage for the first or second simulation, respectively. Specifying [1,2] adds coverage for both simulations.

**Attributes:**

|           |        |
|-----------|--------|
| GetAccess | public |
| SetAccess | public |

**HarnessOwner — Harness owner**

string

Harness owner, specified as a string. `HarnessOwner` is the model or model element with which the harness is associated. Specify a harness owner only if you are creating a new harness. For existing harnesses, this property is automatically set to the owner of the specified harness.

**Attributes:**

|           |        |
|-----------|--------|
| GetAccess | public |
| SetAccess | public |

**HarnessName — Harness name**

string

Harness name, specified as a string. Specify a harness name only if you are creating a new harness. For existing harnesses, this property is automatically set to the name of the specified harness.

**Attributes:**

|           |        |
|-----------|--------|
| GetAccess | public |
| SetAccess | public |

**HarnessSource — Harness input source**

"Inport" (default) | "Signal Editor"

Harness input source, specified as "Inport" or "Signal Editor".) Specify the source only if you are creating a new harness. For existing harnesses, this property is automatically set to the source of the specified harness.

**Attributes:**

|           |        |
|-----------|--------|
| GetAccess | public |
| SetAccess | public |

**Examples****Create an Empty Test Options Object**

Create a TestOptions object to add coverage by using a new test case.

```
testOpts = sltest.testmanager.TestOptions()
```

```
testOpts =
```

```
 TestOptions with properties:
```

```
 Model: ""
 TestFilePath: ""
 TestCase: [0x0 sltest.testmanager.TestCase]
 Parent: [0x0 sltest.testmanager.TestSuite]
 TestCaseType: Baseline
 SrcSimIndex: 1
 HarnessOwner: ""
 HarnessName: ""
 HarnessSource: INPORT
```

**Add Tests for Missing Coverage to Existing Test Case**

This example shows how increase coverage when coverage results for a test case are lower than 100%.

Get the existing test case from the test file and then run the test and get the coverage data. Create a `TestOptions` object and add missing coverage to the coverage data for new test case. Run the new test case and open the Test Manager.

```
load_system("CruiseControlAddCov.slx");

tf = sltest.testmanager.TestFile('CruiseControlAddCov.mldatx');
ts = getTestSuites(tf);
testcaseObj = getTestCases(ts);

tr = run(testcaseObj);
cvdata = getCoverageResults(tr);

testOpts = sltest.testmanager.TestOptions(testcaseObj);
newTestCaseObj = sltest.testmanager.addTestsForMissingCoverage...
 (testOpts,cvdata);

Warning: Non-scalar in Uniform output, at index 3, output 1.
Set 'UniformOutput' to false.

Warning: Non-scalar in Uniform output, at index 3, output 1.
Set 'UniformOutput' to false.

tr = run(newTestCaseObj);

sltest.testmanager.view;
```

In the **Results and Artifacts** panel in the Test Manager, select the first Results. In the **Aggregated Coverage Results** section, the coverage is less than 100%.

Now, select the second Results. In the **Aggregated Coverage Results** section, the coverage increased to 100%.

Clear and close the Test Manager.

```
sltest.testmanager.clear
sltest.testmanager.clearResults
sltest.testmanager.close
```

## See Also

`sltest.testmanager.addTestsForMissingCoverage`

### Topics

“Collect Coverage in Tests”

“Increase Test Coverage for a Model”

“Perform Functional Testing and Analyze Test Coverage”

**Introduced in R2022a**

# sltest.testmanager.TestResultReport class

**Package:** sltest.testmanager

Customize generated results report

## Description

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

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

The sltest.testmanager.TestResultReport class is a handle class.

## Class Attributes

HandleCompatible true

For information on class attributes, see “Class Attributes”.

## Creation

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

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

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

## Input Arguments

### resultObjects — Results set object

object

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

### reportFilePath — File name and path of the generated report

character vector

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

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

## Properties

### AuthorName — Author name

empty character vector (default)

#### Attributes:

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

### BodyFontColor — Body paragraph font color

'Black' (default) | character vector

Body paragraph text font color, specified as a character vector.

Example: 'Red'

#### Attributes:

|             |           |
|-------------|-----------|
| SetAccess   | protected |
| GetAccess   | public    |
| Dependent   | true      |
| NonCopyable | true      |

### BodyFontName — Body paragraph font style name

'Arial' (default) | character vector

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

Example: 'Times New Roman'

#### Attributes:

|             |           |
|-------------|-----------|
| SetAccess   | protected |
| GetAccess   | public    |
| Dependent   | true      |
| NonCopyable | true      |

### BodyFontSize — Body paragraph font size

'12pt' (default) | character vector

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

Example: '14pt'

#### Attributes:

|             |           |
|-------------|-----------|
| SetAccess   | protected |
| GetAccess   | public    |
| Dependent   | true      |
| NonCopyable | true      |

### ChapterIndent — First level indentation width

'3mm' (default) | character vector

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



Example: '5mm'

**Attributes:**

|             |           |
|-------------|-----------|
| SetAccess   | protected |
| GetAccess   | public    |
| Dependent   | true      |
| NonCopyable | true      |

**ChapterIndentL2 – Second level indentation width**

'6mm' (default) | character vector

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

Example: '8mm'

**Attributes:**

|             |           |
|-------------|-----------|
| SetAccess   | protected |
| GetAccess   | public    |
| Dependent   | true      |
| NonCopyable | true      |

**ChapterIndentL3 – Third level indentation width**

'8mm' (default) | character vector

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

Example: '10mm'

**Attributes:**

|             |           |
|-------------|-----------|
| SetAccess   | protected |
| GetAccess   | public    |
| Dependent   | true      |
| NonCopyable | true      |

**CustomTemplateFile – Template file name and path**

empty character vector (default)

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

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

**Attributes:**

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

**HeadingFontColor – Section heading font color**

'Black' (default) | character vector

Section heading text font color, specified as a character vector.

Example: 'Blue'

**Attributes:**

|             |           |
|-------------|-----------|
| SetAccess   | protected |
| GetAccess   | public    |
| Dependent   | true      |
| NonCopyable | true      |

**HeadingFontName — Section heading font style name**

'Arial' (default) | character vector

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

Example: 'Times New Roman'

**Attributes:**

|             |           |
|-------------|-----------|
| SetAccess   | protected |
| GetAccess   | public    |
| Dependent   | true      |
| NonCopyable | true      |

**HeadingFontSize — Section heading font size**

'14pt' (default) | character vector

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

Example: '16pt'

**Attributes:**

|             |           |
|-------------|-----------|
| SetAccess   | protected |
| GetAccess   | public    |
| Dependent   | true      |
| NonCopyable | true      |

**IconFileOutcomeDisabled — Disabled test result icon**

empty character vector (default)

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

Example: 'C:\MATLAB\disabled\_test\_icon.png'

**Attributes:**

|             |           |
|-------------|-----------|
| SetAccess   | protected |
| GetAccess   | public    |
| Dependent   | true      |
| NonCopyable | true      |

**IconFileOutcomeFailed — Failed test result icon**

empty character vector (default)

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

Example: 'C:\MATLAB\failed\_test\_icon.png'

**Attributes:**

|             |           |
|-------------|-----------|
| SetAccess   | protected |
| GetAccess   | public    |
| Dependent   | true      |
| NonCopyable | true      |

**IconFileOutcomeIncomplete — Incomplete test result icon**

empty character vector (default)

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

Example: 'C:\MATLAB\incomplete\_test\_icon.png'

**Attributes:**

|             |           |
|-------------|-----------|
| SetAccess   | protected |
| GetAccess   | public    |
| Dependent   | true      |
| NonCopyable | true      |

**IconFileOutcomeMisaligned — Misaligned test result icon**

empty character vector (default)

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

Example: 'C:\MATLAB\misaligned\_test\_icon.png'

**Attributes:**

|             |           |
|-------------|-----------|
| SetAccess   | protected |
| GetAccess   | public    |
| Dependent   | true      |
| NonCopyable | true      |

**IconFileOutcomePassed — Passed test result icon**

empty character vector (default)

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

Example: 'C:\MATLAB\passed\_test\_icon.png'

**Attributes:**

|             |           |
|-------------|-----------|
| SetAccess   | protected |
| GetAccess   | public    |
| Dependent   | true      |
| NonCopyable | true      |

**IconFileTestCaseResult — Test case result icon**

empty character vector (default)

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

Example: 'C:\MATLAB\test\_case\_result\_icon.png'

**Attributes:**

|             |           |
|-------------|-----------|
| SetAccess   | protected |
| GetAccess   | public    |
| Dependent   | true      |
| NonCopyable | true      |

**IconFileTestFileResult — Test file result icon**

empty character vector (default)

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

Example: 'C:\MATLAB\test\_file\_result\_icon.png'

**Attributes:**

|             |           |
|-------------|-----------|
| SetAccess   | protected |
| GetAccess   | public    |
| Dependent   | true      |
| NonCopyable | true      |

**IconFileTestIterationResult — Iteration result icon**

empty character vector (default)

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

Example: 'C:\MATLAB\iteration\_result\_icon.png'

**Attributes:**

|             |           |
|-------------|-----------|
| SetAccess   | protected |
| GetAccess   | public    |
| Dependent   | true      |
| NonCopyable | true      |

**IconFileTestSuiteResult — Test suite result icon**

empty character vector (default)

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

Example: 'C:\MATLAB\test\_suite\_result\_icon.png'

**Attributes:**

|             |           |
|-------------|-----------|
| SetAccess   | protected |
| GetAccess   | public    |
| Dependent   | true      |
| NonCopyable | true      |

**IconModelReference — Model reference icon**

empty character vector (default)

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

Example: 'C:\MATLAB\model\_reference\_icon.png'

**Attributes:**

|             |           |
|-------------|-----------|
| SetAccess   | protected |
| GetAccess   | public    |
| Dependent   | true      |
| NonCopyable | true      |

**IconTopLevelModel — Top-level model icon**

empty character vector (default)

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

Example: 'C:\MATLAB\top\_level\_model\_icon.png'

**Attributes:**

|             |           |
|-------------|-----------|
| SetAccess   | protected |
| GetAccess   | public    |
| Dependent   | true      |
| NonCopyable | true      |

**IncludeComparisonSignalPlots — Include comparison signal plots**

false (default) | true

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

**Attributes:**

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

**IncludeCoverageResult — Include coverage results**

false (default) | true

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

**Attributes:**

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

**IncludeErrorMessages — Include error messages**

true (default) | false

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

**Attributes:**

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

**IncludeMWVersion — Include MATLAB version**`true (default) | false`

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

**Attributes:**

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

**IncludeSimulationMetaData — Include simulation metadata**`false (default) | true`

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

**Attributes:**

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

**IncludeSimulationSignalPlots — Include simulation signal plots**`false (default) | true`

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

**Attributes:**

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

**IncludeTestRequirement — Include test requirement**`true (default) | false`

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

**Attributes:**

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

**IncludeTestResults — Include all or subset of test results**

2 (default) | 0 | 1

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

**Attributes:**

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

**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.

**Attributes:**

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

**ReportTitle — Report title**

empty character vector (default)

Title of the report, specified as a character vector

Example: 'Test Case Report'

**Attributes:**

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

**SectionSpacing — Spacing between sections**

'2mm' (default) | character vector

Spacing between sections, specified in millimeters as a character vector.

Example: '5mm'

**Attributes:**

|             |           |
|-------------|-----------|
| SetAccess   | protected |
| GetAccess   | public    |
| Dependent   | true      |
| NonCopyable | true      |

**SignalPlotHeight — Plot height**

'600px' (default) | character vector

Plot height, specified in pixels as a character vector.

Example: '500px'

**Attributes:**

|             |           |
|-------------|-----------|
| SetAccess   | protected |
| GetAccess   | public    |
| Dependent   | true      |
| NonCopyable | true      |

**SignalPlotWidth — Plot width**

'500px' (default) | character vector

Plot width, specified in pixels as a character vector.

Example: '400px'

**Attributes:**

|             |           |
|-------------|-----------|
| SetAccess   | protected |
| GetAccess   | public    |
| Dependent   | true      |
| NonCopyable | true      |

**TableFontColor — Table font color**

'Black' (default) | character vector

Table font color, specified as a character vector.

Example: 'Blue'

**Attributes:**

|             |           |
|-------------|-----------|
| SetAccess   | protected |
| GetAccess   | public    |
| Dependent   | true      |
| NonCopyable | true      |

**TableFontName — Table font style name**

'Arial' (default) | character vector

Table font-style name, specified as a character vector.

Example: 'Times New Roman'



**Attributes:**

|             |           |
|-------------|-----------|
| SetAccess   | protected |
| GetAccess   | public    |
| Dependent   | true      |
| NonCopyable | true      |

**TableFontSize — Table font size**

'7.5pt' (default) | character vector

Table font size, specified in points as a character vector.

Example: '10pt'

**Attributes:**

|             |           |
|-------------|-----------|
| SetAccess   | protected |
| GetAccess   | public    |
| Dependent   | true      |
| NonCopyable | true      |

**TitleFontColor — Title font color**

'Black' (default) | character vector

Title font color, specified as a character vector.

Example: 'Blue'

**Attributes:**

|             |           |
|-------------|-----------|
| SetAccess   | protected |
| GetAccess   | public    |
| Dependent   | true      |
| NonCopyable | true      |

**TitleFontName — Title font style name**

'Arial' (default) | character vector

Title font-style name, specified as a character vector.

Example: 'Times New Roman'

**Attributes:**

|             |           |
|-------------|-----------|
| SetAccess   | protected |
| GetAccess   | public    |
| Dependent   | true      |
| NonCopyable | true      |

**TitleFontSize — Title font size**

'16pt' (default) | character vector

Title font size, specified in points as a character vector.

Example: '20pt'

**Attributes:**

|             |           |
|-------------|-----------|
| SetAccess   | protected |
| GetAccess   | public    |
| Dependent   | true      |
| NonCopyable | true      |

**Methods****Public Methods**

|                                 |                                                      |
|---------------------------------|------------------------------------------------------|
| genTableRowsForResultMetaInfo   | Generate test result metadata table                  |
| addReportBody                   | Add main report body                                 |
| addReportTOC                    | Add report table of contents                         |
| addTitlePage                    | Add report title page                                |
| genBaselineInfoTable            | Generate baseline dataset information table          |
| genCoverageTable                | Generate coverage collection table                   |
| genHyperLinkToToC               | Generate link to table of contents                   |
| genIterationSettingTable        | Generate iteration settings table                    |
| genMetadataBlockForTestResult   | Generate result metadata section                     |
| genParameterOverridesTable      | Generate test case parameter overrides table         |
| genRequirementLinksTable        | Generate requirement links table                     |
| genResultSetBlock               | Generate results set section                         |
| genRunBlockForTestCaseResult    | Generate test case configuration and results section |
| genSignalSummaryTable           | Generate signal output and comparison data           |
| genSimulationConfigurationTable | Generate test case simulation configuration table    |
| genTableRowsForResultMetaInfo   | Generate test result metadata table                  |
| genTestCaseResultBlock          | Generate test case result section                    |
| genTestSuiteResultBlock         | Generate test suite result section                   |
| layoutReport                    | Incorporates parts of report into one document       |
| plotOneSignalToFile             | Save signal plot to file                             |

**Examples****Inherit Class and Customize a Report**

```
% class definition
classdef CustomReport < sltest.testmanager.TestResultReport
 % This custom class used by Test Manager
 % adds a custom message in the title page

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

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

 % Call the superclass method to get the default behavior
```

```
 addTitlePage@sltest.testmanager.TestResultReport(obj);

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

## Use Custom Report Class to Generate Report

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

## See Also

[sltest.testmanager.report](#) | [sltest.testmanager.ResultSet](#)

## Topics

“Customize Test Results Reports”  
“Export Test Results”  
“Create and Run Test Cases with Scripts”

## Introduced in R2016a

## sltest.testmanager.TestSuite class

**Package:** sltest.testmanager

Create or modify test suite

### Description

Instances of `sltest.testmanager.TestSuite` can contain other test suites and test cases. For MATLAB-based Simulink tests, test suites are implemented using class and method parameters. See “Test Models Using MATLAB-Based Simulink Tests” for more information.

The `sltest.testmanager.TestSuite` class is a `handle` class.

### Class Attributes

`HandleCompatible` true

For information on class attributes, see “Class Attributes”.

## Creation

### Description

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

## Properties

### Parent — Parent test file or test suite

object

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

### Attributes:

|                          |         |
|--------------------------|---------|
| <code>GetAccess</code>   | public  |
| <code>SetAccess</code>   | private |
| <code>Dependent</code>   | true    |
| <code>NonCopyable</code> | true    |

Data Types:

### Name — Test suite name

character vector

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

**Attributes:**

|             |        |
|-------------|--------|
| GetAccess   | public |
| SetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

Data Types:

**Description — Test suite description**

character vector

Test suite description text, specified as a character vector.

**Attributes:**

|             |        |
|-------------|--------|
| GetAccess   | public |
| SetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

Data Types: char

**Enabled — Test suite execution indicator**

true (default) | false

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

**Attributes:**

|             |        |
|-------------|--------|
| GetAccess   | public |
| SetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

Data Types: logical

**ReasonForDisabling — Disabled description**

character vector

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

**Attributes:**

|             |        |
|-------------|--------|
| GetAccess   | public |
| SetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

Data Types: char

**Requirements — Test suite requirements**

structure array

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

**Attributes:**

|             |         |
|-------------|---------|
| GetAccess   | public  |
| SetAccess   | private |
| Dependent   | true    |
| NonCopyable | true    |

Data Types: struct

**Tags — Tags for categorizing**

character vector | string array

Tags to use for categorizing, specified as a character vector or string array.

**Attributes:**

|             |        |
|-------------|--------|
| GetAccess   | public |
| SetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

Data Types: char | string

**TestFile — Parent test file**

sltest.testmanager.TestFile object

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

**Attributes:**

|             |         |
|-------------|---------|
| GetAccess   | public  |
| SetAccess   | private |
| Dependent   | true    |
| NonCopyable | true    |

Data Types: object

**TestPath — Test hierarchy**

character vector

Test file and test suite hierarchy, returned as a character vector.

**Attributes:**

|             |         |
|-------------|---------|
| GetAccess   | public  |
| SetAccess   | private |
| Dependent   | true    |
| NonCopyable | true    |

Data Types: char

**Methods****Public Methods**

convertTestType Convert test from one type to another

|                        |                                                                               |
|------------------------|-------------------------------------------------------------------------------|
| createTestCase         | Create test case                                                              |
| createTestForSubsystem | (To be removed) Create test harness and test case for subsystem in test suite |
| createTestSuite        | Create test suite                                                             |
| getCoverageSettings    | Get coverage settings                                                         |
| getOptions             | Get test file options                                                         |
| getProperty            | Get test suite property                                                       |
| getTestCaseByName      | Get test case object by name                                                  |
| getTestCases           | Get test cases at first level of test suite                                   |
| getTestSuiteByName     | Get test suite object by name                                                 |
| getTestSuites          | Get test suites at first level of test suite                                  |
| remove                 | Remove test suite                                                             |
| run                    | Run test cases in test suite                                                  |
| setProperty            | Set test suite property                                                       |

## Examples

### Create New Test File and Suite

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

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

testsuite =

 TestSuite with properties:

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

## See Also

[sltest.testmanager.TestFile](#) | [sltest.testmanager.TestCase](#) | [testsuite](#)

### Topics

“Create and Run Test Cases with Scripts”  
 “Use Parameters in Class-Based Tests”  
 “Test Models Using MATLAB-Based Simulink Tests”

### Introduced in R2015b

## sltest.testmanager.TestSuiteResult class

**Package:** sltest.testmanager

**Superclasses:** sltest.testmanager.ResultSet

Access test suite results data

### Description

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

The `sltest.testmanager.TestSuiteResult` class is a handle class.

### Class Attributes

HandleCompatible true

For information on class attributes, see “Class Attributes”.

### Creation

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

### Properties

#### Duration — Length of time the test suite ran, in seconds

`duration`

Length of time the test suite ran, in seconds, returned as a duration.

#### Attributes:

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

#### NumDisabled — Number of disabled tests

`integer`

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

#### Attributes:

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |



**NumFailed — Number of failed tests**

integer

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

**Attributes:**

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

**NumPassed — Number of passed tests**

integer

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

**Attributes:**

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

**NumTestCaseResults — Number of test case result children**

integer

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

**Attributes:**

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

**NumTestSuiteResults — Number of test suite result children**

integer

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

**Attributes:**

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

**NumTotal — Total number of tests**

integer

The total number of tests in the test suite result.

**Attributes:**

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

**Parent — Parent of the result object**

object

Parent of the result. The parent of a test suite result is another test suite result, test file result, or result set object.

**Attributes:**

|             |         |
|-------------|---------|
| SetAccess   | private |
| GetAccess   | public  |
| Dependent   | true    |
| NonCopyable | true    |

**Release — Release in which the test was run**

character vector

Release in which the test was run, returned as a character vector.

**Attributes:**

|             |           |
|-------------|-----------|
| SetAccess   | protected |
| GetAccess   | public    |
| Dependent   | true      |
| NonCopyable | true      |

**StartTime — Time the test suite began to run**

datetime

Time the test suite began to run, returned as a datetime.

**Attributes:**

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

**StopTime — Time the test suite completed**

datetime

Time the test suite completed, returned as a datetime.

**Attributes:**

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

**Tags — Tags to filter test file results**

string array

Tags to filter the test file results. Use tags to view a subset of the test results. See “Tags” for more information.

**Attributes:**

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

**TestFilePath — Test file path**

character vector

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

**Attributes:**

|             |         |
|-------------|---------|
| SetAccess   | private |
| GetAccess   | public  |
| Dependent   | true    |
| NonCopyable | true    |

**TestSuitePath — Result hierarchy path**

character vector

The hierarchy path in the parent result set.

**Attributes:**

|             |         |
|-------------|---------|
| SetAccess   | private |
| GetAccess   | public  |
| Dependent   | true    |
| NonCopyable | true    |

**UserData — Custom data stored with test suite results**

any data type

Custom data stored with the test suite results, specified as any type of data. Use this field to add custom information, such as the settings used to obtain the results.

**Attributes:**

|             |        |
|-------------|--------|
| SetAccess   | public |
| GetAccess   | public |
| Dependent   | true   |
| NonCopyable | true   |

**Methods****Public Methods**

|                    |                                                |
|--------------------|------------------------------------------------|
| getCleanupPlots    | Get plots from cleanup callbacks of test suite |
| getCoverageResults | Get coverage results                           |

|                     |                               |
|---------------------|-------------------------------|
| getSetupPlots       | Plots from setup callbacks    |
| getTestCaseResults  | Get test case results object  |
| getTestSuiteResults | Get test suite results object |

## Examples

### Get Test Suite Result From Results Set

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

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

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

### See Also

sltest.testmanager.TestFileResult | sltest.testmanager.TestCaseResult

### Topics

“Create and Run Test Cases with Scripts”

**Introduced in R2015a**

# Methods

---

## assertSignalsMatch

**Class:** `sltest.TestCase`

**Package:** `sltest`

Assert two data sets are equivalent

### Syntax

```
assertSignalsMatch(testCase,actual,expected)
assertSignalsMatch(____,diagnostic)
assertSignalsMatch(____,Name,Value)
```

### Description

`assertSignalsMatch(testCase,actual,expected)` filters test content by asserting that the actual and expected signal data values are equivalent. When an assertion fails, the test point or test file stops running, and that test is marked as failed. This qualification is useful when a failure at the assertion point renders the rest of the current test method invalid, but does not prevent proper execution of other test methods. Often, you use assertions to check that preconditions of the current test are not violated or that fixtures are set up correctly. For more information, see `matlab.unittest.qualifications.Assertable`.

`assertSignalsMatch( ____,diagnostic)` returns diagnostic information when the actual and expected data values are not equivalent.

`assertSignalsMatch( ____,Name,Value)` filters test content with additional options specified by one or more `Name,Value` pair arguments.

### Input Arguments

#### **testCase** — Instance of test case

`sltest.TestCase` object

Instance of the test case, specified as an `sltest.TestCase` object.

#### **actual** — Actual values to compare to expected values

time series data | string | character array

Actual values to compare to expected values, specified as time series data, a string, or character array. The data for each actual value must pair data value with time value. The data must be in a format supported by the Simulation Data Inspector. The Simulation Data Inspector requires data in a format that associates sample values with time. Supported formats include `timeseries`, `Structure with time`, and `Dataset`.

Example: `'C:/matlab/data/actualData.mat'`

#### **expected** — Expected values to use as the baseline for the comparison

time series data | string scalar | character array

Expected values to use as the baseline for the comparison. The data for each expected value must pair data value with time value. The data must be in a format supported by the Simulation Data

Inspector. The Simulation Data Inspector requires data in a format that associates sample values with time. Supported formats include `timeseries`, `Structure with time`, and `Dataset`.

Example: `'C:/matlab/data/expectedData.mat'`

### **diagnostic – Diagnostic information**

string | character vector | function handle | `matlab.unittest.diagnostics.Diagnostic` instance

Diagnostic information to display when the assertion that the actual and expected values are equivalent fails, specified as a string, character array, a function handle, or an instance of a `matlab.unittest.diagnostics.Diagnostic` class.

Example: `'Simulation output does not match.'`

### **Name-Value Pair Arguments**

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

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

In addition to the listed Name-Value pairs, you can use the `Simulink.sdi.constraints.MatchesSignal` Name-Value pairs.

Example: `'AbsTol',.02`

### **AbsTol – Absolute tolerance**

0 (default) | scalar

Absolute tolerance, specified as the comma-separated pair consisting of `'AbsTol'` and the scalar value of the tolerance. The tolerance specifies the magnitude of the difference between the actual and expected values.

Example: `'AbsTol',1e-9`

### **RelTol – Relative tolerance**

0 (default) | scalar

Relative tolerance, specified as the comma-separated pair consisting of `'RelTol'` and the scalar value of the tolerance. The relative tolerance specifies the magnitude of the difference between the actual and expected values, relative to the expected value.

Example: `'RelTol',.002`

### **TimeTol – Time tolerance**

0 (default) | scalar

Time tolerance, specified as the comma-separated pair consisting of `'TimeTol'` and the scalar value of the tolerance.

Example: `'TimeTol',.2`

## Attributes

|        |        |
|--------|--------|
| Access | public |
| Sealed | true   |

To learn about attributes of methods, see [Method Attributes](#).

## Examples

### Assert that Back-to-Back Simulations Match

Create a test case for interactive use. Then, simulate the model in normal mode to obtain the expected values, and simulate in rapid accelerator mode to obtain the actual values. Use `assertSignalsMatch` to compare the values.

```
testCase = ...
 sltest.TestCase.forInteractiveUse;

expected = testCase.simulate('myModel',...
 'SimulationMode','Normal');
actual = testCase.simulate('myModel',...
 'SimulationMode','Rapid-Accelerator');

testCase.assertSignalsMatch(actual,expected)
```

### Assert that Simulation Outputs Match Within a Specified Tolerance

Create a test case for interactive use. Simulate the model in rapid accelerator mode to obtain the actual values. Use `assertSignalsMatch` to compare the actual values to the baseline values saved in a MAT-file. Set a relative tolerance.

```
testCase = ...
 sltest.TestCase.forInteractiveUse;

actual = testCase.simulate('myModel',...
 'SimulationMode','Rapid-Accelerator');

testCase.assertSignalsMatch(actual,'baseline.mat',...
 'RelTol',0.001)
```

### Assert that Simulations Match and Display Custom Diagnostic Message

Create a test case for interactive use. Then, simulate the model in rapid accelerator mode to obtain the actual values. Use `assertSignalsMatch` to compare the actual values to the baseline values saved in a MAT-file. Specify the diagnostic message to display if the assertion fails.

```
testCase = ...
 sltest.TestCase.forInteractiveUse;
```



```
actual = testCase.simulate('myModel',...
 'SimulationMode','Rapid-Accelerator');

testCase.assertSignalsMatch(actual,'baseline.mat',...
 'Rapid-Accel output did not match.')
```

## See Also

sltest.TestCase | Simulink.sdi.constraints.MatchesSignal |  
matlab.unittest.qualifications.Assertable | fatalAssertSignalsMatch |  
verifySignalsMatch | assumeSignalsMatch |  
matlab.unittest.constraints.AbsoluteTolerance |  
matlab.unittest.constraints.RelativeTolerance

## Topics

“Test Models Using MATLAB-Based Simulink Tests”

## Introduced in R2020b

## assumeSignalsMatch

**Class:** `sltest.TestCase`

**Package:** `sltest`

Assume two data sets are equivalent

### Syntax

```
assumeSignalsMatch(testCase,actual,expected)
assumeSignalsMatch(____,diagnostic)
assumeSignalsMatch(____,Name,Value)
```

### Description

`assumeSignalsMatch(testCase,actual,expected)` assumes that the actual and expected signal data values are equivalent. When an assumption fails, the test stops running at the test point or test file, and the test is marked as incomplete. For more information, see `matlab.unittest.qualifications.Assumable`.

`assumeSignalsMatch( ____,diagnostic)` returns diagnostic information when the actual and expected data values are not equivalent.

`assumeSignalsMatch( ____,Name,Value)` filters test content with additional options specified by one or more Name, Value pair arguments.

### Input Arguments

**testCase** — Instance of test case

`sltest.TestCase` object

Instance of the test case, specified as an `sltest.TestCase` object.

**actual** — Actual values to compare to expected values

time series data | string | character array

Actual values to compare to expected values, specified as time series data, a string, or character array. The data for each actual value must pair data value with time value. The data must be in a format supported by the Simulation Data Inspector. The Simulation Data Inspector requires data in a format that associates sample values with time. Supported formats include `timeseries`, `Structure with time`, and `Dataset`.

Example: `'C:/matlab/data/actualData.mat'`

**expected** — Expected values to use as the baseline for the comparison

time series data | string scalar | character array

Expected values to use as the baseline for the comparison. The data for each expected value must pair data value with time value. The data must be in a format supported by the Simulation Data Inspector. The Simulation Data Inspector requires data in a format that associates sample values with time. Supported formats include `timeseries`, `Structure with time`, and `Dataset`.

Example: 'C:/matlab/data/expectedData.mat'

### **diagnostic** — Diagnostic information

string | character vector | function handle | matlab.unittest.diagnostics.Diagnostic instance

Diagnostic information to display when the assumption that the actual and expected values are equivalent fails, specified as a string, character array, a function handle, or an instance of a `matlab.unittest.diagnostics.Diagnostic` class.

Example: 'Simulation output does not match.'

### **Name-Value Pair Arguments**

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

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

In addition to the listed Name-Value pairs, you can use the `Simulink.sdi.constraints.MatchesSignal` Name-Value pairs.

Example: 'AbsTol',.02

### **AbsTol** — Absolute tolerance

0 (default) | scalar

Absolute tolerance, specified as the comma-separated pair consisting of 'AbsTol' and the scalar value of the tolerance. The tolerance specifies the magnitude of the difference between the actual and expected values.

Example: 'AbsTol',1e-9

### **RelTol** — Relative tolerance

0 (default) | scalar

Relative tolerance, specified as the comma-separated pair consisting of 'RelTol' and the scalar value of the tolerance. The relative tolerance specifies the magnitude of the difference between the actual and expected values, relative to the expected value.

Example: 'RelTol',.002

### **TimeTol** — Time tolerance

0 (default) | scalar

Time tolerance, specified as the comma-separated pair consisting of 'TimeTol' and the scalar value of the tolerance.

Example: 'TimeTol',.2

## **Attributes**

|        |        |
|--------|--------|
| Access | public |
| Sealed | true   |

To learn about attributes of methods, see Method Attributes.

## Examples

### Assume that Back-to-Back Simulations Match

Create a test case for interactive use. Then, simulate the model in normal mode to obtain the expected values, and simulate in rapid accelerator mode to obtain the actual values. Use `assumeSignalsMatch` to compare the values.

```
testCase = ...
 sltest.TestCase.forInteractiveUse;

expected = testCase.simulate('myModel', ...
 'SimulationMode', 'Normal');
actual = testCase.simulate('myModel', ...
 'SimulationMode', 'Rapid-Accelerator');

testCase.assumeSignalsMatch(actual, expected)
```

### Assume that Simulation Outputs Match Within a Specified Tolerance

Create a test case for interactive use. Simulate the model in rapid accelerator mode to obtain the actual values. Use `assumeSignalsMatch` to compare the actual values to the baseline values saved in a MAT-file. Set an absolute tolerance.

```
testCase = ...
 sltest.TestCase.forInteractiveUse;

actual = testCase.simulate('myModel', ...
 'SimulationMode', 'Rapid-Accelerator');

testCase.assumeSignalsMatch(actual, 'baseline.mat', ...
 'AbsTol', 1e-12)
```

### Assume that Simulations Match and Display Custom Diagnostic Message

Create a test case for interactive use. Then, simulate the model in rapid accelerator mode to obtain the actual values. Use `assumeSignalsMatch` to compare the actual values to the baseline values saved in a MAT-file. Specify the diagnostic message to display if the assumption fails.

```
testCase = ...
 sltest.TestCase.forInteractiveUse;

actual = testCase.simulate('myModel', ...
 'SimulationMode', 'Rapid-Accelerator');

testCase.assumeSignalsMatch(actual, 'baseline.mat', ...
```

```
'Rapid-Accel output did not match.')
```

## See Also

`sltest.TestCase` | `matlab.unittest.qualifications.Assumable` |  
`matlab.unittest.constraints.AbsoluteTolerance` |  
`Simulink.sdi.constraints.MatchesSignal` | `fatalAssertSignalsMatch` |  
`verifySignalsMatch` | `assertSignalsMatch` |  
`matlab.unittest.constraints.RelativeTolerance` |  
`Simulink.sdi.constraints.MatchesSignal`

## Topics

“Test Models Using MATLAB-Based Simulink Tests”

## Introduced in R2020b

## createTemporaryFolder

**Class:** `sltest.TestCase`

**Package:** `sltest`

Create temporary folder

### Syntax

```
folder = createTemporaryFolder(testCase)
```

### Description

`folder = createTemporaryFolder(testCase)` returns a temporary folder. This folder is deleted based on the scope of the test case. For example, for a folder created by a method, that folder is deleted when the method is complete and goes out of scope. Temporary folders are useful if you want to use a separate folder to store methods of a test case. See “Create Temporary Folder for Each Test Method” on page 3-10 “Write Plugins to Extend TestRunner” for information on test case scope.

### Input Arguments

**testCase — Instance of test case**

`sltest.TestCase` object

Instance of the test case, specified as an `sltest.TestCase` object.

### Output Arguments

**folder — Temporary folder path**

character vector

Temporary folder, returned as a character vector.

### Attributes

Access public

To learn about attributes of methods, see Method Attributes.

### Examples

#### Create Temporary Folder for Each Test Method

```
classdef tempFolder < sltest.TestCase
 properties
 Folder;
 end
```

```
methods (TestMethodSetup)
 function setup(testCase)
 testCase.Folder = ...
 testCase.createTemporaryFolder();
 end
end

methods (Test)
 function test1(testCase)
 testCase.simulate('myModel1',...
 'InFolder',testCase.Folder);
 end

 function test2(testCase)
 testCase.simulate('myModel2',...
 'InFolder',testCase.Folder);
 end
end
end
```

## See Also

sltest.TestCase

## Topics

“Test Models Using MATLAB-Based Simulink Tests”

**Introduced in R2020b**

## createSimulationInput

Create simulation input object

### Syntax

```
inputObj = createSimulationInput(model)
inputObj = createSimulationInput(model, 'WithHarness', harness)
```

### Description

`inputObj = createSimulationInput(model)` creates a `Simulink.SimulationInput` or `sltest.harness.SimulationInput` object for the specified model. If the model is not loaded, `createSimulationInput` uses `testCase.loadSystem` to load the model when you simulate it.

`inputObj = createSimulationInput(model, 'WithHarness', harness)` creates an `sltest.harness.SimulationInput` object for the specified test harness.

### Input Arguments

#### **model** — Model name

string | character vector

Name of the model, specified as a string or character vector, without the model extension. For example, for the `myModel.slx` model, enter only `myModel`.

Example: `'RollAutopilotModelRef'`

#### **harness** — Harness name

string | character vector

Name of the harness, specified as a comma-separated pair consisting of `'WithHarness'` and the name of the test harness for which to create input, specified as a string or character vector.

Example: `'WithHarness', 'RollAutopilotModelRef_harness1'`

### Output Arguments

#### **inputObj** — Simulation input object

`Simulink.SimulationInput` object | `sltest.harness.SimulationInput` object

Simulation input object, returned as a `Simulink.SimulationInput` or `sltest.harness.SimulationInput` object. To use the simulation input with a test harness, use the `WithHarness` syntax to return an `sltest.harness.SimulationInput` object. If you do not use the `WithHarness` option, this function returns a `Simulink.SimulationInput`. You use these objects to pass inputs to a harness or model, respectively.



## Attributes

|        |        |
|--------|--------|
| Access | public |
| Sealed | true   |

To learn about attributes of methods, see [Method Attributes](#).

## Examples

### Create Simulation Input For a Harness

Create a MATLAB-based Simulink test file named `testSim.m`.

In the file, define the test case in the `testOne` function. The function creates the simulation input for a harness called `myExample_harness1`, simulates using that input, and compares the simulation output to a baseline file.

```
classdef testSim < sltest.TestCase
 methods (Test)
 function testOne(testCase)
 in = testCase.createSimulationInput('myExample',...
 'WithHarness','myExample_Harness1');
 simOut = testCase.simulate(in);
 testCase.verifySignalsMatch(simOut,'myExampleBaseline.mat');
 end
 end
end
```

## See Also

[Simulink.SimulationInput](#) | [sltest.harness.SimulationInput](#)

## Topics

- “Test Models Using MATLAB-Based Simulink Tests”
- “Using MATLAB-Based Simulink Tests in the Test Manager”
- “Collect Coverage Using MATLAB-Based Simulink Tests”

**Introduced in R2021a**

## fatalAssertSignalsMatch

**Class:** `sltest.TestCase`

**Package:** `sltest`

Fatally assert two data sets are equivalent

### Syntax

```
fatalAssertSignalsMatch(testCase,actual,expected)
fatalAssertSignalsMatch(____,diagnostic)
fatalAssertSignalsMatch(____,Name,Value)
```

### Description

`fatalAssertSignalsMatch(testCase,actual,expected)` fatally asserts that the actual and expected signal data values are equivalent. When a fatal assertion fails, the whole test session is stopped. This is useful when the test contains so many errors that it does not make sense to continue the test session.

`fatalAssertSignalsMatch( ____,diagnostic)` returns diagnostic information when the actual and expected data values are not equivalent..

`fatalAssertSignalsMatch( ____,Name,Value)` fatally asserts with additional options specified by one or more `Name,Value` pair arguments.

### Input Arguments

**testCase** — Instance of test case

`sltest.TestCase` object

Instance of the test case, specified as an `sltest.TestCase` object.

**actual** — Actual values to compare to expected values

time series data | string | character array

Actual values to compare to expected values, specified as time series data, a string, or character array. The data for each actual value must pair data value with time value. The data must be in a format supported by the Simulation Data Inspector. The Simulation Data Inspector requires data in a format that associates sample values with time. Supported formats include `timeseries`, `Structure with time`, and `Dataset`.

Example: `'C:/matlab/data/actualData.mat'`

**expected** — Expected values to use as the baseline for the comparison

time series data | string scalar | character array

Expected values to use as the baseline for the comparison. The data for each expected value must pair data value with time value. The data must be in a format supported by the Simulation Data Inspector. The Simulation Data Inspector requires data in a format that associates sample values with time. Supported formats include `timeseries`, `Structure with time`, and `Dataset`.

Example: 'C:/matlab/data/expectedData.mat'

### diagnostic — Diagnostic information

string | character vector | function handle | matlab.unittest.diagnostics.Diagnostic instance

Diagnostic information to display when the fatal assertion that the actual and expected values are equivalent fails, specified as a string, character array, a function handle, or an instance of a `matlab.unittest.diagnostics.Diagnostic` class.

Example: 'Simulation output does not match.'

### Name-Value Pair Arguments

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

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

In addition to the listed Name-Value pairs, you can use the `Simulink.sdi.constraints.MatchesSignal` Name-Value pairs.

Example: 'AbsTol',.02

### AbsTol — Absolute tolerance

0 (default) | scalar

Absolute tolerance, specified as the comma-separated pair consisting of 'AbsTol' and the scalar value of the tolerance. The tolerance specifies the magnitude of the difference between the actual and expected values.

Example: 'AbsTol',1e-9

### RelTol — Relative tolerance

0 (default) | scalar

Relative tolerance, specified as the comma-separated pair consisting of 'RelTol' and the scalar value of the tolerance. The relative tolerance specifies the magnitude of the difference between the actual and expected values, relative to the expected value.

Example: 'RelTol',.002

### TimeTol — Time tolerance

0 (default) | scalar

Time tolerance, specified as the comma-separated pair consisting of 'TimeTol' and the scalar value of the tolerance.

Example: 'TimeTol',.2

## Attributes

|        |        |
|--------|--------|
| Access | public |
| Sealed | true   |

To learn about attributes of methods, see Method Attributes.

## Examples

### Fatal Assert that Back-to-Back Simulations Match

Create a test case for interactive use. Then, simulate the model in normal mode to obtain the expected values, and simulate in rapid accelerator mode to obtain the actual values. Use `fatalAssertSignalsMatch` to compare the values.

```
testCase = ...
 sltest.TestCase.forInteractiveUse;

expected = testCase.simulate('myModel', ...
 'SimulationMode', 'Normal');
actual = testCase.simulate('myModel', ...
 'SimulationMode', 'Rapid-Accelerator');

testCase.fatalAssertSignalsMatch(actual, expected)
```

### Fatal Assert that Simulation Outputs Match Within a Specified Tolerance

Create a test case for interactive use. Simulate the model in rapid accelerator mode to obtain the actual values. Use `fatalAssertSignalsMatch` to compare the actual values to the baseline values saved in a MAT-file. Set a relative tolerance.

```
testCase = ...
 sltest.TestCase.forInteractiveUse;

actual = testCase.simulate('myModel', ...
 'SimulationMode', 'Rapid-Accelerator');

testCase.fatalAssertSignalsMatch(actual, 'baseline.mat', ...
 'RelTol', 0.001)
```

### Fatal Assert that Simulations Match and Display Custom Diagnostic Message

Create a test case for interactive use. Then, simulate the model in rapid accelerator mode to obtain the actual values. Use `fatalAssertSignalsMatch` to compare the actual values to the baseline values saved in a MAT-file. Specify the diagnostic message to display if the assertion fails.

```
testCase = ...
 sltest.TestCase.forInteractiveUse;

actual = testCase.simulate('myModel', ...
 'SimulationMode', 'Rapid-Accelerator');

testCase.fatalAssertSignalsMatch(actual, 'baseline.mat', ...
```

```
'Rapid-Accel output did not match.')
```

## See Also

[sltest.TestCase](#) | [matlab.unittest.qualifications.FatalAssertable](#) | [Simulink.sdi.constraints.MatchesSignal](#) | [verifySignalsMatch](#) | [assumeSignalsMatch](#) | [matlab.unittest.constraints.AbsoluteTolerance](#) | [matlab.unittest.constraints.RelativeTolerance](#) | [assertSignalsMatch](#)

## Topics

“Test Models Using MATLAB-Based Simulink Tests”

## Introduced in R2020b

## sltest.TestCase.forInteractiveUse

**Class:** `sltest.TestCase`

**Package:** `sltest`

Create test case for interactive use

### Syntax

```
testCase = sltest.TestCase.forInteractiveUse
```

### Description

`testCase = sltest.TestCase.forInteractiveUse` returns a test case instance configured for interactive use at the MATLAB command line. Use this method at the MATLAB command line to debug a `TestCase` object without having to write a complete test file. You cannot use this method within a test file.

### Output Arguments

**testCase** — Instance of test case

`sltest.TestCase` object

Instance of a test case, returned as an `sltest.TestCase` object. This instance is configured to print qualification failures and successes to the standard output, which is usually the screen.

### Attributes

Static yes

To learn about attributes of methods, see [Method Attributes](#).

### Examples

#### Create Test Case for Debugging at Command Line

```
testCase = ...
 sltest.TestCase.forInteractiveUse;

expected = testCase.simulate('myModel', ...
 'SimulationMode', 'Normal');
actual = testCase.simulate('myModel', ...
 'SimulationMode', 'Rapid-Accelerator');

testCase.verifySignalsMatch(actual, expected)
```

### See Also

`sltest.TestCase` | `assertSignalsMatch` | `fatalAssertSignalsMatch` | `assumeSignalsMatch` | `verifySignalsMatch`

**Topics**

“Test Models Using MATLAB-Based Simulink Tests”

“Using MATLAB-Based Simulink Tests in the Test Manager”

“Collect Coverage Using MATLAB-Based Simulink Tests”

**Introduced in R2020b**

# loadSystem

**Class:** `sltest.TestCase`

**Package:** `sltest`

Load model for MATLAB-based Simulink test

## Syntax

```
[modelName,modelhandle,modelpath] = loadSystem(testCase,model)
[modelname,modelhandle,modelpath] = loadSystem(____,Name,Value)
```

## Description

`[modelName,modelhandle,modelpath] = loadSystem(testCase,model)` loads the specified model and returns the name of the model, the model handle, and the path to the model.

`loadSystem` loads a Simulink model within the scope of a single `TestCase` method class. The life cycle of the model is tied to the test case. When the test case goes out of scope, the testing framework closes the model without saving any changes to the model and clears any base workspace variables loaded by the model. Previously loaded models remain loaded. Variable values of previously loaded models might be overridden if, for example, the model being opened has model callbacks. To retain the variable values of opened models, add `PreservingBaseWorkspace` as an input Name-Value pair.

`[modelName,modelhandle,modelpath] = loadSystem( ____,Name,Value)` loads the model and test case with additional options specified by one or more Name, Value pairs.

## Input Arguments

### **testCase** — Instance of test case

`sltest.TestCase` object

Instance of the test case, specified as an `sltest.TestCase` object.

### **model** — Name of model

string | character vector

Name of model to load, specified as a string or character vector.

Example: `'sltestCar'`

### **Name-Value Pair Arguments**

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

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

Example: `'PreservingBaseWorkspace',true`



**PreservingBaseWorkspace — Whether to preserve existing base workspace variable values**

false (default) | true

Whether to preserve existing base workspace variable values, specified as the comma-separated pair consisting of 'PreservingBaseWorkspace' and true or false.

**IncludingReferencedModels — Whether to load model and its referenced models**

false (default) | true

Whether to load the model and its referenced models, specified as the comma-separated pair consisting of 'IncludingReferencedModels' and true or false.

**Output Arguments****modelName — Name of loaded model**

string | character vector

Name of the loaded model, returned as a string or character vector. If 'IncludingReferencedModels' is true, then loadSystem returns the model name and the names of its referenced models.

**modelhandle — Handle of loaded model**

string | character vector

Handle of the loaded model, returned as a string or character vector. If IncludeReferencedModels is true, returns the handle of the model and the handles of its referenced models.

**modelpath — Path of loaded model**

string | character vector

Path of the loaded model, returned as a string or character vector. If IncludeReferencedModels is true, returns the path of the model and the paths of its referenced models.

**Attributes**

|        |        |
|--------|--------|
| Access | public |
|--------|--------|

To learn about attributes of methods, see Method Attributes.

**Examples****Load Model**

```
classdef simTest < sltest.TestCase
 methods (Test)
 function testOne(testCase)
 [modelName,modelhandle,modelpath] = ...
 testCase.loadSystem('myModel');
 end
 end
end
```

**Load Model and Referenced Models**

```
classdef simTest < sltest.TestCase
 methods (Test)
 function testOne(testCase)
 [modelName,modelhandle,modelpath] =...
 testCase.loadSystem('myModel',...
 'IncludingReferencedModels',true);
 end
 end
end
```

**See Also**

sltest.TestCase

**Topics**

“Test Models Using MATLAB-Based Simulink Tests”

“Using MATLAB-Based Simulink Tests in the Test Manager”

“Collect Coverage Using MATLAB-Based Simulink Tests”

**Introduced in R2020b**

# simulate

**Class:** `sltest.TestCase`

**Package:** `sltest`

Simulate model or `Simulink.SimulationInput` for MATLAB-based Simulink test

## Syntax

```
simulate(testcase,model)
simulate(testcase,siminput)
simulate(testcase,Name,Value)
```

## Description

`simout = simulate(testcase,model)` simulates the specified model and returns a `Simulink.SimulationOutput` object.

`simout = simulate(testcase,siminput)` simulates using a `Simulink.SimulationInput` object as the `siminput`.

`simout = simulate(testcase,Name,Value)` simulates the model with additional options specified by one or more `Name, Value` pair arguments.

## Input Arguments

### testCase — Instance of test case

`sltest.TestCase` object

Instance of the test case, specified as an `sltest.TestCase` object.

### model — Name of model

scalar string | character vector

Name of the model to simulate, specified as a scalar string or character vector, without the model extension. For example, for the `myModel.slx` model, enter only `myModel`.

### siminput — Input object to simulate

`sltest.harness.SimulationInput` object | `Simulink.SimulationInput` object

Input object to simulate, specified as a `sltest.harness.SimulationInput` or `Simulink.SimulationInput` object.

## Name-Value Pair Arguments

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

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

Example: `'InFolder','C:\matlab\myTest'`

**InFolder — Name of folder**

string | character vector

Name of folder that contains the model to simulate, specified as the comma-separated pair consisting of 'InFolder' and the name of the folder, specified as a string or character vector.

**WithHarness — Name of harness**

string | character vector

Name of the harness to simulate, specified as the comma-separated pair of consisting of 'WithHarness', specified as a string or character vector.

---

**Note** You can use the `sim` function name-value pairs with the `simulate` method.

---

**Output Arguments****simout — Simulation result object**

Simulink.SimulationOutput object

Simulation results, returned as a Simulink.SimulationOutput object.

**Attributes**

|        |        |
|--------|--------|
| Access | public |
| Sealed | true   |

To learn about attributes of methods, see Method Attributes.

**Examples****Simulate a Test Case Using Model Input**

```
classdef simTest < sltest.TestCase
 methods (Test)
 function testOne(testCase)
 simout = testCase.simulate('myModel');
 end
 end
end
```

**Simulate Test Case Using a SimulationInput Object**

```
classdef simTest < sltest.TestCase
 methods(Test)
 function testOne(testCase)
 simInput = Simulink.SimulationInput('myModel');
 simOut = testCase.simulate(simInput);
 end
 end
end
```

```

end
end

```

### Simulate a Test Case Using a Model in a Specified Folder

```

classdef simTest < sltest.TestCase
 methods(Test)
 function testOne(testCase)
 simout = testCase.simulate('myModel',...
 'InFolder','C:/matlab/newTestFolder');
 end
 end
end
end

```

### Simulate a Test Case Using Harness Input

```

classdef simTest < sltest.TestCase
 methods (Test)
 function testOne(testCase)
 simout = testCase.simulate('myModel',...
 'WithHarness','mymodel_Harness1');
 end
 end
end
end

```

## See Also

[sltest.TestCase](#) | [Simulink.SimulationInput](#) | [Simulink.SimulationOutput](#) | [sim](#)

## Topics

“Test Models Using MATLAB-Based Simulink Tests”  
 “Using MATLAB-Based Simulink Tests in the Test Manager”  
 “Collect Coverage Using MATLAB-Based Simulink Tests”

## Introduced in R2020b

# verifySignalsMatch

**Class:** `sltest.TestCase`

**Package:** `sltest`

Verify two sets of data are equivalent

## Syntax

```
verifySignalsMatch(testCase,actual,expected)
verifySignalsMatch(____,diagnostic)
verifySignalsMatch(____,Name,Value)
```

## Description

`verifySignalsMatch(testCase,actual,expected)` verifies that the actual and expected signal data values are equivalent. When a verification fails, the failure is recorded and the test runs to completion. For more information, see `matlab.unittest.qualifications.Verifiable`.

`verifySignalsMatch( ____,diagnostic)` returns diagnostic information when the actual and expected data values are not equivalent.

`verifySignalsMatch( ____,Name,Value)` filters test content with additional options specified by one or more `Name,Value` pair arguments.

## Input Arguments

### **testCase** — Instance of test case

`sltest.TestCase` object

Instance of the test case, specified as an `sltest.TestCase` object.

### **actual** — Actual values to compare to expected values

time series data | string | character array

Actual values to compare to expected values, specified as time series data, a string, or character array. The data for each actual value must pair data value with time value. The data must be in a format supported by the Simulation Data Inspector. The Simulation Data Inspector requires data in a format that associates sample values with time. Supported formats include `timeseries`, `Structure with time`, and `Dataset`.

Example: `'C:/matlab/data/actualData.mat'`

### **expected** — Expected values to use as the baseline for the comparison

time series data | string scalar | character array

Expected values to use as the baseline for the comparison. The data for each expected value must pair data value with time value. The data must be in a format supported by the Simulation Data Inspector. The Simulation Data Inspector requires data in a format that associates sample values with time. Supported formats include `timeseries`, `Structure with time`, and `Dataset`.

Example: `'C:/matlab/data/expectedData.mat'`

**diagnostic — Diagnostic information**

string | character vector | function handle | `matlab.unittest.diagnostics.Diagnostic` instance

Diagnostic information to display when the verification that the actual and expected values are equivalent fails, specified as a string, character array, a function handle, or an instance of a `matlab.unittest.diagnostics.Diagnostic` class.

Example: 'Simulation output does not match.'

**Name-Value Pair Arguments**

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

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

In addition to the listed Name-Value pairs, you can use the `Simulink.sdi.constraints.MatchesSignal` Name-Value pairs.

Example: 'AbsTol',.02

**AbsTol — Absolute tolerance**

0 (default) | scalar

Absolute tolerance, specified as the comma-separated pair consisting of 'AbsTol' and the scalar value of the tolerance. The tolerance specifies the magnitude of the difference between the actual and expected values.

Example: 'AbsTol',1e-9

**RelTol — Relative tolerance**

0 (default) | scalar

Relative tolerance, specified as the comma-separated pair consisting of 'RelTol' and the scalar value of the tolerance. The relative tolerance specifies the magnitude of the difference between the actual and expected values, relative to the expected value.

Example: 'RelTol',.002

**TimeTol — Time tolerance**

0 (default) | scalar

Time tolerance, specified as the comma-separated pair consisting of 'TimeTol' and the scalar value of the tolerance.

Example: 'TimeTol',.2

**Attributes**

|        |        |
|--------|--------|
| Access | public |
| Sealed | true   |

To learn about attributes of methods, see [Method Attributes](#).

## Examples

### Verify that Back-to-Back Simulations Match

Create a test case for interactive use. Then, simulate the model in normal mode to obtain the expected values, and simulate in rapid accelerator mode to obtain the actual values. Use `verifySignalsMatch` to compare the values.

```
testCase =...
 sltest.TestCase.forInteractiveUse;

expected = testCase.simulate('myModel',...
 'SimulationMode','Normal');
actual = testCase.simulate('myModel',...
 'SimulationMode','Rapid-Accelerator');

testCase.verifySignalsMatch(actual,expected)
```

### Verify that Simulation Outputs Match Within a Specified Tolerance

Create a test case for interactive use. Simulate the model in rapid accelerator mode to obtain the actual values. Use `verifySignalsMatch` to compare the actual values to the baseline values saved in a MAT-file. Set a relative tolerance.

```
testCase =...
 sltest.TestCase.forInteractiveUse;

actual = testCase.simulate('myModel',...
 'SimulationMode','Rapid-Accelerator');

testCase.verifySignalsMatch(actual,'baseline.mat',...
 'RelTol',0.001)
```

### Verify that Simulations Match and Display Custom Diagnostic Message

Create a test case for interactive use. Then, simulate the model in rapid accelerator mode to obtain the actual values. Use `verifySignalsMatch` to compare the actual values to the baseline values saved in a MAT-file. Specify the diagnostic message to display if the verification fails.

```
testCase =...
 sltest.TestCase.forInteractiveUse;

actual = testCase.simulate('myModel',...
 'SimulationMode','Rapid-Accelerator');

testCase.verifySignalsMatch(actual,'baseline.mat',...
 'Rapid-Accel output did not match.')
```

## See Also

`sltest.TestCase` | `matlab.unittest.qualifications.Verifiable` |  
`Simulink.sdi.constraints.MatchesSignal` | `fatalAssertSignalsMatch` |



`assumeSignalsMatch` | `matlab.unittest.constraints.AbsoluteTolerance` |  
`matlab.unittest.constraints.RelativeTolerance` | `assertSignalsMatch`

**Topics**

“Test Models Using MATLAB-Based Simulink Tests”

**Introduced in R2020b**

## addBaselineCriteria

**Class:** `sltest.testmanager.TestCase`

**Package:** `sltest.testmanager`

Add baseline criteria to test case

### Syntax

```
base = addBaselineCriteria(tc, file)
base = addBaselineCriteria(tc, file, 'RefreshIfExists', true)

base = addBaselineCriteria(tc, excel, 'SeparateBaselines', false)
base = addBaselineCriteria(tc, excel, 'Sheets', sheets, Name, Value)
```

### Description

`base = addBaselineCriteria(tc, file)` adds a MAT-file, Simulation Data Inspector (SDI) MLDATX file, or Microsoft Excel file as baseline criteria to a baseline test case. If `file` is an Excel file that has multiple sheets, each is added to the test case as a separate baseline set.

`base = addBaselineCriteria(tc, file, 'RefreshIfExists', true)` adds the baseline criteria to the test case, replacing the baseline criteria if the test case already had one.

`base = addBaselineCriteria(tc, excel, 'SeparateBaselines', false)` adds all sheets in the Excel file as a single baseline set.

`base = addBaselineCriteria(tc, excel, 'Sheets', sheets, Name, Value)` specifies the sheets from the Excel to include in the baseline criteria and uses additional options specified by one or more `Name, Value` pair arguments.

### Input Arguments

#### **tc** — Test case

`sltest.testmanager.TestCase` object

Test case that you want to add the baseline criteria to, specified as an `sltest.testmanager.TestCase` object.

#### **file** — Excel, MAT, or SDI MLDATX file name and path

character vector

File and path name of the baseline criteria file, specified as a character vector. You can specify a MAT-file, Simulation Data Inspector MLDATX file, or a Microsoft Excel file.

Example: `'C:\MATLAB\baseline_API.mat'`

#### **excel** — Excel file name and path

character vector

File and path name of the Excel file to use as the baseline criteria, specified as a character vector.

Example: 'C:\MATLAB\baseline.xlsx'

### **sheets — Names of Excel sheets to add**

character vector | string | array of strings

Names of sheets from Excel file to add, specified as a character vector, string, or array of strings.

Example: 'signals', ["Heater", "Plant"]

### **Name-Value Pair Arguments**

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

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

Example: 'Ranges', 'B1:C4', 'RefreshIfExists', false

### **Ranges — Range of cells from sheet**

character vector | string | array of strings

Ranges of cells from the sheets that you add as baseline criteria, specified as a character vector, a string, or an array of strings. The ranges you specify must correspond to the sheets you specify. For example, if you specify one sheet, specify one range. If you specify a cell array of sheets, each value in the 'Ranges' cell array must correspond to a sheet in the 'Sheets' cell array. Specify an empty range to use the entire sheet.

Example: 'B2:C30', "D2:E30", ["B2:C30", "D2:E30", "B2:C30"],  
["B2:C30", "", "D2:E30"]

### **RefreshIfExists — Replace baseline criteria**

false (default) | true

Option to replace the test case baseline criteria, specified as a Boolean. Use `false` to return an error if the test case already has baseline criteria, that is, to prevent overwriting the baseline. Use `true` to add the baseline criteria, replacing the existing baseline.

### **SeparateBaselines — Specify separate baselines**

true (default) | false

Option to use each sheet in specified by the 'Sheets' argument as a separate baseline, specified as `true` or `false`.

## **Output Arguments**

### **base — Baseline criteria**

sltest.testmanager.BaselineCriteria object | array of  
sltest.testmanager.BaselineCriteria objects

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

## **Examples**

### Add Baseline Criteria to Test Case

```
% Open the model for this example
openExample('sldemo_absbrake');

% Create the test file, test suite, and test case structure
tf = sltest.testmanager.TestFile('API Test File');
ts = createTestSuite(tf, 'API Test Suite');
tc = createTestCase(ts, 'baseline', 'Baseline API Test Case');

% Remove the default test suite
tsDel = getTestSuiteByName(tf, 'New Test Suite 1');
remove(tsDel);

% Assign the system under test to the test case
setProperty(tc, 'Model', 'sldemo_absbrake');

% Add baseline criteria from file
baseline = addBaselineCriteria(tc, 'C:\MATLAB\baseline_API.mat');
```

### Use Microsoft Excel File as Baseline

Use an Excel file as baseline, overwriting the existing baseline on the test case.

```
% Open the model for this example
openExample('sldemo_absbrake');

% Create the test file, test suite, and test case structure
tf = sltest.testmanager.TestFile('Excel Test File');
ts = createTestSuite(tf, 'Excel Test Suite');
tc = createTestCase(ts, 'baseline', 'Baseline Excel Test Case');

% Remove the default test suite
tsDel = getTestSuiteByName(tf, 'New Test Suite 1');
remove(tsDel);

% Assign the system under test to the test case
setProperty(tc, 'Model', 'sldemo_absbrake');

% Add baseline criteria from file
baseline = addBaselineCriteria(tc, ...
 'C:\MATLAB\myexcel.xlsx', 'RefreshIfExists', true);
```

### Add One Baseline from a Microsoft Excel File

Use an Excel file as baseline, creating one baseline even if the Excel file has multiple sheets.

```
% Open the model for this example
openExample('sldemo_absbrake');

% Create the test file, test suite, and test case structure
tf = sltest.testmanager.TestFile('Excel Test File');
ts = createTestSuite(tf, 'Excel Test Suite');
tc = createTestCase(ts, 'baseline', 'Baseline Excel Test Case');

% Remove the default test suite
tsDel = getTestSuiteByName(tf, 'New Test Suite 1');
remove(tsDel);

% Assign the system under test to the test case
setProperty(tc, 'Model', 'sldemo_absbrake');

% Add baseline criteria from file
```

```
baseline = addBaselineCriteria(tc,...
 'C:\MATLAB\myexcel.xlsx','SeparateBaselines',false);
```

## Specify Sheets and Cell Range in a Baseline

Select three sheets from an Excel file to use as the baseline. For each sheet, specify a range of cells.

```
% Open the model for this example
openExample('sldemo_absbrake');

% Create the test file, test suite, and test case structure
tf = sltest.testmanager.TestFile('Excel Test File');
ts = createTestSuite(tf,'Excel Test Suite');
tc = createTestCase(ts,'baseline','Baseline Excel Test Case');

% Remove the default test suite
tsDel = getTestSuiteByName(tf,'New Test Suite 1');
remove(tsDel);

% Assign the system under test to the test case
setProperty(tc,'Model','sldemo_absbrake');

% Create sheets and ranges arrays
sheets = ["HotTemp", "ColdTemp", "NominalTemp"];
ranges = ["B2:C30", "D2:E30", "B2:C30"];

% Add baseline criteria from file, using the sheets and cell ranges specified
baseline = addBaselineCriteria(tc,...
 'C:\MATLAB\myexcel.xlsx','Sheets',sheets,'Ranges',ranges);
```

## See Also

`sltest.testmanager.BaselineCriteria`

### Topics

“Baseline Criteria”

“Create and Run Test Cases with Scripts”

### Introduced in R2015b

## addDataStoreSignal

**Class:** `sltest.testmanager.LoggedSignalSet`

**Package:** `sltest.testmanager`

Add a data store or `Simulink.Signal` object to a set

### Syntax

```
obj = addDataStoreSignal(lgset,BlockPath)
```

### Description

`obj = addDataStoreSignal(lgset,BlockPath)` creates and adds an `sltest.testmanager.LoggedSignal` object to a set when the `LoggedSignal` object derives from a data store or `Simulink.Signal` object. You must open or load the model to add a `LoggedSignal` from the model.

### Input Arguments

**lgset — Logged signal set**

`sltest.testmanager.LoggedSignalSet` object

Logged signal set object contained in a test case.

**BlockPath — Block path object**

`Simulink.BlockPath` object

`Simulink.BlockPath` object that uniquely identifies a Data Store Write block and the associated data store memory or associated `Simulink.Signal` object.

### Examples

**Add a Global Data Store to a Signal Set**

Open a model and create a signal set.

```
% Open model for this example
openExample('sldemo mdlref_dsm')

% Create test case
tf = sltest.testmanager.TestFile(strcat(pwd, '\mytf.mldatx'));
ts = sltest.testmanager.TestSuite(tf, 'myts');
tc = sltest.testmanager.TestCase(ts, 'baseline', 'mytc');

% Create signal set
mylgset = tc.addLoggedSignalSet;
```

Identify the global data store paths for these model components. To programmatically determine the path, select each component individually and in the MATLAB Command Window, enter `gcb`:

- Select Model block `sldemo_mdref_dsm_bot2`
- In `sldemo_mdref_dsm_bot2`, select the Data Store Write block.

Use the returned paths to create a `Simulink.BlockPath` object for the global data store.

```
% Add signal to set
bPath = Simulink.BlockPath({'sldemo_mdref_dsm/A1',...
 'sldemo_mdref_dsm_bot2/DSW'});
sig1 = mylgset.addDataStoreSignal(bPath);

% Check signal was added successfully
sigs = mylgset.getLoggedSignals
```

### Add Local Data Store Memory to a Signal Set

Begin with the model and signal set created in the previous example.

Identify the global data store paths for these model components. To programmatically determine the path, select each component individually and in the MATLAB Command Window, enter `gcb`:

- Select Model block `sldemo_mdref_dsm_bot`
- In `sldemo_mdref_dsm_bot`, select subsystem `PositiveSS` subsystem
- In subsystem `PositiveSS`, select the Data Store Write block

Use the returned paths to create a `Simulink.BlockPath` object for the local data store.

```
% Add signal to set
bPath = Simulink.BlockPath({'sldemo_mdref_dsm/A',...
 'sldemo_mdref_dsm_bot/PositiveSS',...
 'sldemo_mdref_dsm_bot/PositiveSS/DSW'});
sig2 = mylgset.addDataStoreSignal(bPath);

% Check that signal was added successfully
sigs = mylgset.getLoggedSignals;
```

### See Also

`gcb` | `sltest.testmanager.LoggedSignal`

### Topics

“Create and Run Test Cases with Scripts”  
 “Capture Simulation Data in a Test Case”

### Introduced in R2019a

# addExcelSpecification

**Package:** sltest.testmanager

Add a Microsoft Excel sheet to baseline criteria or test case inputs

## Syntax

```
addExcelSpecification(obj, 'Sheet', sheet)
addExcelSpecification(obj, 'Sheet', sheet, 'Range', range)
```

## Description

`addExcelSpecification(obj, 'Sheet', sheet)` adds the specified Excel sheet to the baseline criteria or test case inputs `obj`.

`addExcelSpecification(obj, 'Sheet', sheet, 'Range', range)` adds the cells in the specified range to the baseline criteria or test case inputs.

## Examples

### Add a Sheet to Baseline Criteria

Create the test file, test suite, and test case structure.

```
tf = sltest.testmanager.TestFile('Add Excel Test');
ts = createTestSuite(tf, 'Add Excel Suite');
tc = createTestCase(ts, 'baseline', 'Baseline Excel Test Case');
```

Add baseline criteria from an Excel file. Specifying two sheets creates two baseline criteria.

```
base = addBaselineCriteria(tc, 'C:\MATLAB\baseline.xlsx', 'Sheets', {'Optics', 'Converter'});
```

Add the sheet X20out to the first set.

```
base(1).addExcelSpecification('Sheet', 'X20out');
```

Show the contents of the Sheet property of the Excel specifications for each baseline criteria. The first set now includes the X20out sheet.

```
base(1).ExcelSpecifications().Sheet
base(2).ExcelSpecifications().Sheet
```

```
ans =
 'Optics'
```

```
ans =
 'X20out'
```



```
ans =
 'Converter'
```

## Input Arguments

### obj — Object to which to add Excel sheet or cell data

baseline criteria object | test case object

Object to which to add Excel sheet or cell data, specified as a baseline criteria or test case input object.

### sheet — Excel sheet to add

character vector | string

Excel sheet to add to baseline criteria or test case inputs, specified as a character vector.

Example: 'Optics'

### range — Range of cells to add

character vector | string

Range of cells from the specified sheet to add to test case inputs, specified as a character vector or string in one of these forms:

| Ways to specify Range                  | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|----------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 'Corner1:Corner2'<br>Rectangular Range | Specify the range using the syntax ' <i>Corner1:Corner2</i> ', where <i>Corner1</i> and <i>Corner2</i> are two opposing corners that define the region. For example, 'D2:H4' represents the 3-by-5 rectangular region between the two corners D2 and H4 on the worksheet. The 'Range' name-value pair argument is not case-sensitive, and uses Excel A1 reference style (see Excel help).<br><br><b>Example:</b> 'Range', 'Corner1:Corner2'                             |
| ' '<br>Unspecified or Empty            | If unspecified, the importing function automatically detects the used range.<br><br><b>Example:</b> 'Range', ''<br><br><b>Note:</b> <i>Used Range</i> refers to the rectangular portion of the spreadsheet that actually contains data. The importing function automatically detects the used range by trimming leading and trailing rows and columns that do not contain data. Text that is only white space is considered data and is captured within the used range. |
| 'Row1:Row2'<br>Row Range               | You can identify the range by specifying the beginning and ending rows using Excel row designators. Then <code>readtable</code> automatically detects the used column range within the designated rows. For instance, the importing function interprets the range specification '1:7' as an instruction to read columns in the used range in rows 1 through 7 (inclusive).<br><br><b>Example:</b> 'Range', '1:7'                                                        |

| Ways to specify Range             | Description                                                                                                                                                                                                                                                                                                                                                                                                             |
|-----------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 'Column1:Column2'<br>Column Range | You can identify the range by specifying the beginning and ending columns using Excel column designators. Then <code>readtable</code> automatically detects the used row range within the designated columns. For instance, the importing function interprets the range specification 'A:F' as an instruction to read rows in the used range in columns A through F (inclusive).<br><br><b>Example:</b> 'Range' , 'A:F' |
| 'NamedRange'<br>Excel Named Range | In Excel, you can create names to identify ranges in the spreadsheet. For instance, you can select a rectangular portion of the spreadsheet and call it 'myTable'. If such named ranges exist in a spreadsheet, then the importing function can read that range using its name.<br><br><b>Example:</b> 'Range' , 'myTable'                                                                                              |

Example: 'A1:C20'

## See Also

`sltest.testmanager.BaselineCriteria` | `sltest.testmanager.TestInput`

## Topics

"Baseline Criteria"

"Inputs"

"Use External Excel or MAT-File Data in Test Cases"

## Introduced in R2017b

# addInput

**Class:** `sltest.testmanager.TestCase`

**Package:** `sltest.testmanager`

Add input file to test case

## Syntax

```
input = addInput(tc, file, Name, Value)
```

## Description

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

## Input Arguments

### tc — Test case

`sltest.testmanager.TestCase` object

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

### file — Input file name and path

character vector

Name and path of MAT-file or Microsoft Excel input file, specified as a character vector.

### Name-Value Pair Arguments

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

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

Example: `'Sheets', 'mysheet', 'Ranges', 'C1:F10', 'CreateIterations', false`

### Pairs for MAT-Files and Microsoft Excel Files

#### SimulationIndex — Test case simulation number

1 | 2

Test case simulation number that the inputs apply to, specified as 1 or 2. This setting applies to equivalence tests.

Example: `'SimulationIndex', 2`

#### CreateIterations — Create a table iteration from the input

true (default) | false

Option to add the input file to the iteration table under **Iterations** in the test case, specified as Boolean.

Example: 'CreateIterations', false

### Pairs to Use Only with Microsoft Excel Files

#### Sheets — Names of sheets to use as inputs

character vector | string | array of strings

Names of sheets from Excel file to use as test case inputs, specified as a character vector, string, or array of strings.

Example: 'testinputs', ["Heater", "Plant"]

#### Ranges — Range of cells from sheet

character vector | string | array of strings

Ranges of cells from the sheets that you added as inputs, specified as a character vector, string, or array of strings. You can specify 'Ranges' only if you also specify 'Sheets'. The ranges you specify must correspond to the sheets. For example, if you specify one sheet, specify one range. If you specify a cell array of sheets, each value in the 'Ranges' cell array must correspond with a sheet in the 'Sheets' cell array.

You can specify 'Ranges' as shown in the table.

| Ways to specify Range                  | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|----------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 'Corner1:Corner2'<br>Rectangular Range | Specify the range using the syntax ' <i>Corner1:Corner2</i> ', where <i>Corner1</i> and <i>Corner2</i> are two opposing corners that define the region. For example, 'D2:H4' represents the 3-by-5 rectangular region between the two corners D2 and H4 on the worksheet. The 'Range' name-value pair argument is not case-sensitive, and uses Excel A1 reference style (see Excel help).<br><br><b>Example:</b> 'Range', 'Corner1:Corner2'                             |
| ' '<br>Unspecified or Empty            | If unspecified, the importing function automatically detects the used range.<br><br><b>Example:</b> 'Range', ''<br><br><b>Note:</b> <i>Used Range</i> refers to the rectangular portion of the spreadsheet that actually contains data. The importing function automatically detects the used range by trimming leading and trailing rows and columns that do not contain data. Text that is only white space is considered data and is captured within the used range. |

| Ways to specify Range             | Description                                                                                                                                                                                                                                                                                                                                                                                                                |
|-----------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 'Row1:Row2'<br>Row Range          | You can identify the range by specifying the beginning and ending rows using Excel row designators. Then <code>readtable</code> automatically detects the used column range within the designated rows. For instance, the importing function interprets the range specification '1:7' as an instruction to read all columns in the used range in rows 1 through 7 (inclusive).<br><br><b>Example:</b> 'Range', '1:7'       |
| 'Column1:Column2'<br>Column Range | You can identify the range by specifying the beginning and ending columns using Excel column designators. Then <code>readtable</code> automatically detects the used row range within the designated columns. For instance, the importing function interprets the range specification 'A:F' as an instruction to read all rows in the used range in columns A through F (inclusive).<br><br><b>Example:</b> 'Range', 'A:F' |
| 'NamedRange'<br>Excel Named Range | In Excel, you can create names to identify ranges in the spreadsheet. For instance, you can select a rectangular portion of the spreadsheet and call it 'myTable'. If such named ranges exist in a spreadsheet, then the importing function can read that range using its name.<br><br><b>Example:</b> 'Range', 'myTable'                                                                                                  |

Example: 'B2:C30', "D2:E30", ["B2:C30", "D2:E30", "B2:C30"]

### SeparateInputs — Specify separate inputs

true (default) | false

Option to use each sheet in the Excel file or specified by the 'Sheets' argument as a separate input, specified as true or false.

## Output Arguments

### input — Test input

sltest.testmanager.TestInput object | array of sltest.testmanager.TestInput objects

Test input, returned as an sltest.testmanager.TestInput object or an array of sltest.testmanager.TestInput objects.

## Examples

### Add Microsoft® Excel® Data as Input

This example shows how to add data from a Microsoft® Excel® spreadsheet and map it to a test case. Only the two sheets that have data are added and mapped.

**Load the Example Model**

```
open_system('sltestExcelExample');
```

**Create a New Test File**

```
tf = sltest.testmanager.TestFile('input_test_file.mldatx');
```

**Get the Test Suite and Test Case Objects**

```
ts = getTestSuites(tf);
tc = getTestCases(ts);
```

**Add the Example Model as the System Under Test**

```
setProperty(tc, 'Model', 'sltestExcelExample');
```

**Add Excel® Data to Inputs Section and Specify Sheets to Add**

```
excelfile = 'sltestExampleInputs.xlsx';
input = addInput(tc, excelfile, 'Sheets', ["Acceleration", "Braking"]);
```

**Map the Input Signal for the Sheets by Block Name**

```
map(input(1), 0);
map(input(2), 0);
```

**Specify Sheets and Ranges for Microsoft Excel File**

This example shows the syntax to add Excel file sheets and range.

```
% Create test file
tf = sltest.testmanager.TestFile('Excel Input Test File');

% Create test suite and test case
ts = createTestSuite(tf, 'Excel Test Suite');
tc = createTestCase(ts, 'baseline', 'Excel Input Test Case');

% Add Excel data to Inputs section, specifying sheets and range
input = addInput(tc, 'C:\MyHomeDir\myexcel.xlsx', ...
 'Sheets', ["Optics", "Torque", "Throttle"], ...
 'Ranges', ["B1:C20", "", "D1:G10"]);
```

**See Also**

`sltest.testmanager.TestCase`

**Topics**

“Create and Run Test Cases with Scripts”

“Inputs”

“Create Data Files for Test Case Input”

“Use External Excel or MAT-File Data in Test Cases”

**Introduced in R2015b**

# addIteration

**Class:** `sltest.testmanager.TestCase`

**Package:** `sltest.testmanager`

Add test iteration to test case

## Syntax

```
addIteration(tc,iter)
addIteration(____,name)
```

## Description

`addIteration(tc,iter)` adds a test iteration to the test case. The Test Manager gives the iteration a unique name.

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

## Input Arguments

### **tc** – Test case to add iteration to

`sltest.testmanager.TestCase` object

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

### **iter** – Test iteration to add

`sltest.testmanager.TestIteration` object

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

### **name** – Test iteration name

character vector

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

Example: 'Test Iteration 5'

## Examples

### **Iterate Over Parameter Sets**

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

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

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

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

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

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

### **See Also**

sltest.testmanager.TestIteration

### **Topics**

“Test Iterations”

“Create and Run Test Cases with Scripts”

**Introduced in R2016a**



# getTestCase

**Class:** `sltest.testmanager.Assessment`

**Package:** `sltest.testmanager`

Get parent test case of assessment

## Syntax

```
tc = sltest.testmanager.Assessment.getTestCase(assessmentObj)
```

## Description

`tc = sltest.testmanager.Assessment.getTestCase(assessmentObj)` returns the test case object that is the parent of the specified assessment object.

## Input Arguments

**assessmentObj** — **Assessment for which to get its parent**

`sltest.testmanager.Assessment` object

Assessment for which to get its parent, specified as an `sltest.testmanager.Assessment` object.

## Output Arguments

**tc** — **Parent test case of assessment**

`sltest.testmanager.TestCase` object

Parent test case of assessment, returned as an `sltest.testmanager.TestCase` object.

## Examples

### Get Parent Test Case of a Test Assessment

This example shows how to obtain the parent test case of a test assessment. Load a test file and obtain its test suites and test cases, then display the test case that contains the first assessment.

```
tf = sltest.testmanager.load('test_traffic.mldatx');
```

```
ts = tf.getTestSuites;
tc = tf.getTestSuites.getTestCases;
assessmentObjs = tc.getAssessments;
```

```
tcParent = assessmentObjs(1).getTestCase
```

```
tcParent =
 TestCase with properties:
```

```
 Name: 'New Test Case 1'
 TestFile: [1x1 sltest.testmanager.TestFile]
```

```
 TestPath: 'test_traffic > New Test Suite 1 > New Test Case 1'
 TestType: 'simulation'
 RunOnTarget: {[0]}
 Parent: [1x1 sltest.testmanager.TestSuite]
 Requirements: [0x1 struct]
 Description: ''
 Enabled: 1
 Tags: [0x0 string]
```

### See Also

sltest.testmanager.Assessment | remove (Assessment) |  
sltest.testmanager.AssessmentSymbol | addAssessment | getAssessments |  
addAssessmentSymbol | getAssessmentSymbols

**Introduced in R2022a**

## remove

**Class:** `sltest.testmanager.Assessment`

**Package:** `sltest.testmanager`

Remove assessment from test case

### Syntax

```
sltest.testmanager.Assessment.remove(assessment)
```

### Description

`sltest.testmanager.Assessment.remove(assessment)` removes the specified assessment object from its parent test case.

### Input Arguments

**assessment** — **Assessment to remove**

`sltest.testmanager.Assessment` object

Assessment to remove from its parent test case, specified as an `sltest.testmanager.Assessment` object.

### Examples

#### Remove Test Assessment

This example shows how to remove a test assessment.

Load the test file and open the Test Manager.

```
tf = sltest.testmanager.load('test_traffic.mldatx');
sltest.testmanager.view;
```

Select the test case and then open the **Logical and Temporal Assessment** section to view a summary of the existing assessments.

▼ LOGICAL AND TEMPORAL ASSESSMENTS\*

▶ ASSESSMENT CALLBACK\*

Extend Result

| EN...                    | NAME      | ASSESSMENT                                                                                                                                                                                                                |
|--------------------------|-----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <input type="checkbox"/> | Waiti...  | ▶ At any point of time, <b>NumCars</b> must be greater than or equal to <b>int32(0)</b>                                                                                                                                   |
| <input type="checkbox"/> | Both I... | ▶ At any point of time, <b>Light1 == Red   Light2 == Red</b> must be true                                                                                                                                                 |
| <input type="checkbox"/> | Light1... | ▶ At any point of time, if <b>Light1 == Yellow</b> becomes true and stays true for at most <b>YellowDelayWithTol</b> seconds then, starting from rising edge of trigger, with no delay, <b>Light1 == Red</b> must be true |

Get the assessments.

```
ts = tf.getTestSuites;
tc = ts.getTestCases;
assessObjs = tc.getAssessments;
```

Remove the second assessment and then view the updated Test Manager.

```
assessObjs(2).remove;
```

The Test Manager updates and shows that the second assessment is removed.

▼ LOGICAL AND TEMPORAL ASSESSMENTS\*

▶ ASSESSMENT CALLBACK\*

Extend Result

| EN...                    | NAME      | ASSESSMENT                                                                                                                                                                                                                |
|--------------------------|-----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <input type="checkbox"/> | Waiti...  | ▶ At any point of time, <b>NumCars</b> must be greater than or equal to <b>int32(0)</b>                                                                                                                                   |
| <input type="checkbox"/> | Light1... | ▶ At any point of time, if <b>Light1 == Yellow</b> becomes true and stays true for at most <b>YellowDelayWithTol</b> seconds then, starting from rising edge of trigger, with no delay, <b>Light1 == Red</b> must be true |

```
sltest.testmanager.clear
sltest.testmanager.close
```

**See Also**

```
sltest.testmanager.Assessment | getTestCase |
sltest.testmanager.AssessmentSymbol | addAssessment | getAssessments |
addAssessmentSymbol | getAssessmentSymbols
```

**Introduced in R2022a**

## addAssessment

**Class:** `sltest.testmanager.TestCase`

**Package:** `sltest.testmanager`

Add assessments to a test case

### Syntax

```
addedAssessments = addAssessment(tc, assessmentsObj)
addedAssessments = addAssessment(tc, assessmentsObj, 'AddSymbols' = true)
addedAssessments = addAssessment(tc, assessmentsObj, 'AddSymbols' = false)
```

### Description

`addedAssessments = addAssessment(tc, assessmentsObj)` adds an array of assessment objects, including the symbols uses in the assessments, to the specified test case.

`addedAssessments = addAssessment(tc, assessmentsObj, 'AddSymbols' = true)` behaves the same as the previous syntax.

`addedAssessments = addAssessment(tc, assessmentsObj, 'AddSymbols' = false)` adds assessment objects to the test case, but does not add the associated symbols. The symbols in the assessment have a `Scope` value of `Unresolved`. You must associate the symbols with an item in the assessment before you run the test case.

### Input Arguments

#### **tc — Test case**

`sltest.testmanager.TestCase` object

Test case for which to get assessments, specified as an `sltest.testmanager.TestCase` object.

#### **assessmentObj — Assessments to add to test case**

array of `sltest.testmanager.Assessment` objects

Assessments to add to the test case, specified as an array of `sltest.testmanager.Assessment` objects.

### Output Arguments

#### **addedAssessments — Assessments added to test case**

array of `sltest.testmanager.Assessment` objects

Assessments added to the test case, returned as an array of `sltest.testmanager.Assessment` objects.

### Examples

### Get Assessments and Add to Test Case

Load an existing test file that contains assessments. Get the assessments and then create a new test case and add the assessments to the new test case.

```
tf = sltest.testmanager.load('test_traffic.mldatx');
ts = getTestSuites(tf);
tc = getTestCases(ts);

assessObjs = getAssessments(tc);
tc_New = createTestCase(ts);
tc_NewAssess = addAssessment(tc_New, assessObjs);
```

Clear and close the Test Manager.

```
sltest.testmanager.clear
sltest.testmanager.clearResults
sltest.testmanager.close
```

### Copy Assessments to All Test Cases in a Test Suite

Load an existing test file that contains a test suite and a test case with assessments.

```
tf = sltest.testmanager.load('test_traffic.mldatx');
ts = getTestSuites(tf);
tc = getTestCases(ts);
assessObjs = getAssessments(tc);
```

Create a new test suite with three test cases.

```
tsNew = tf.createTestSuite;
tsNew.createTestCase;
tsNew.createTestCase;
tsNew.createTestCase;
```

Use `arrayfun` to copy the assessments from the original test case to all of the new test cases in the new test suite.

```
arrayfun(@(x)addAssessment(x, assessObjs), ...
 tsNew.getTestCases, 'UniformOutput', false);
```

Clear and close the Test Manager.

```
sltest.testmanager.clear
sltest.testmanager.clearResults
sltest.testmanager.close
```

### See Also

`sltest.testmanager.Assessment` | `remove (Assessment)` | `getTestCase` | `sltest.testmanager.AssessmentSymbol` | `getAssessments` | `addAssessmentSymbol` | `getAssessmentSymbols`

### Topics

“Logical and Temporal Assessments”

“Assess Temporal Logic by Using Temporal Assessments”  
“Logical and Temporal Assessment Syntax”

**Introduced in R2022a**



# addAssessmentSymbol

**Class:** `sltest.testmanager.TestCase`

**Package:** `sltest.testmanager`

Add assessment symbols to a test case

## Syntax

```
addedSymbols = addAssessmentSymbol(tc,assessmentSymbols)
```

## Description

`addedSymbols = addAssessmentSymbol(tc,assessmentSymbols)` adds assessment symbol objects to the specified test case.

## Input Arguments

**tc — Test case**

`sltest.testmanager.TestCase` object

Test case for which to add assessment symbols, specified as an `sltest.testmanager.TestCase` object.

## Output Arguments

**addedSymbols — Assessment symbols added to test case**

array of `sltest.testmanager.AssessmentSymbol` objects

Assessment symbols added to the test case, returned as an array of `sltest.testmanager.AssessmentSymbol` objects.

## Examples

### Get Assessment Symbols and Add to Test Case

Load an existing test file and get the assessments and symbols from the test file. Create a new test case and add the assessments, but not the symbols, to the new test case. Then, add the associated assessment symbols to the test case.

```
tf = sltest.testmanager.load('test_traffic.mldatx');
ts = getTestSuites(tf);
tc = getTestCases(ts);

assessObjs = getAssessments(tc);
symbolsObjs = getAssessmentSymbols(tc);

tc_New = createTestCase(ts);
tc_NewAssess = addAssessment(tc_New,assessObjs,...
```

```
 "AddSymbols", false);
addedSymbolsObj = addAssessmentSymbol(tc_New, symbolsObjs);
```

Clear and close the Test Manager.

```
sltest.testmanager.clear
sltest.testmanager.clearResults
sltest.testmanager.close
```

### **See Also**

```
sltest.testmanager.Assessment | remove (Assessment) | getTestCase |
sltest.testmanager.AssessmentSymbol | addAssessment | getAssessments |
getAssessmentSymbols
```

### **Topics**

“Logical and Temporal Assessments”

“Assess Temporal Logic by Using Temporal Assessments”

### **Introduced in R2022a**

# getAssessments

**Class:** `sltest.testmanager.TestCase`

**Package:** `sltest.testmanager`

Get assessments associated with test case

## Syntax

```
assessments = getAssessments(tc)
```

## Description

`assessments = getAssessments(tc)` returns the assessments for a test case object.

## Input Arguments

**tc — Test case**

`sltest.testmanager.TestCase` object

Test case for which to get assessments, specified as an `sltest.testmanager.TestCase` object.

## Output Arguments

**assessments — Assessments associated with the test case**

array of `sltest.testmanager.Assessment` objects

Assessments associated with the test case, returned as an array of `sltest.testmanager.Assessment` objects.

## Examples

### Get Assessments from Test Case

Get the assessments from a test case and view the details of the first assessment.

```
tf = sltest.testmanager.load('test_traffic.mldatx');

ts = tf.getTestSuites;
tc = tf.getTestSuites.getTestCases;
assessObjs = tc.getAssessments;
assessObjs(1)

ans =
 Assessment with properties:
 Name: 'Waiting cars'
 Enabled: 0
 Info: "At any point of time, NumCars must be greater than or equal to int32(0)"
 Requirements: [0x1 struct]
```

Symbols: [1x1 sltest.testmanager.AssessmentSymbol]

### **See Also**

sltest.testmanager.Assessment | remove (Assessment) | getTestCase |  
sltest.testmanager.AssessmentSymbol | addAssessment | addAssessmentSymbol |  
getAssessmentSymbols

### **Topics**

“Logical and Temporal Assessments”  
“Logical and Temporal Assessment Syntax”

### **Introduced in R2022a**

# getAssessmentSymbols

**Class:** `sltest.testmanager.TestCase`

**Package:** `sltest.testmanager`

Get assessment symbols associated with test case

## Syntax

```
assessmentSymbols = getAssessmentSymbols(tc)
```

## Description

`assessmentSymbols = getAssessmentSymbols(tc)` returns the assessment symbols for a test case object.

## Input Arguments

**tc — Test case**

`sltest.testmanager.TestCase` object

Test case that contains the assessment symbols, specified as an `sltest.testmanager.TestCase` object.

## Output Arguments

**assessmentSymbols — Assessment symbols used in the test case**

array of `sltest.testmanager.AssessmentSymbol` objects

Assessment symbols used in the test case, returned as an array of `sltest.testmanager.AssessmentSymbol` objects.

## Examples

### Get Assessment Symbols

Get the symbols associated with the assessments in a test case and view the first symbol in the returned `AssessmentSymbol` object.

```
tf = sltest.testmanager.load('test_traffic.mldatx');

ts = tf.getTestSuites;
tc = tf.getTestSuites.getTestCases;

symObjs = tc.getAssessmentSymbols;
symObjs(1)

ans =
 AssessmentSymbol with properties:
```

```
Name: 'Light1'
Scope: Signal
Value: [1x1 struct]
```

### See Also

`sltest.testmanager.Assessment` | `remove (Assessment)` | `getTestCase` |  
`sltest.testmanager.AssessmentSymbol` | `addAssessment` | `getAssessments` |  
`addAssessmentSymbol`

### Topics

“Logical and Temporal Assessments”  
“Logical and Temporal Assessment Syntax”

### Introduced in R2022a

# getTestCaseResults

**Class:** `sltest.testmanager.TestCase`

**Package:** `sltest.testmanager`

Get test case results history

## Syntax

```
tcresult = getTestCaseResults(tc)
```

## Description

`tcresult = getTestCaseResults(tc)` returns the test case results history for the specified test case, `tc`. The test case history includes the results for all runs of the test case in the Test Manager.

## Input Arguments

**tc — Test case**

`sltest.testmanager.TestCase` object

Test case for which to obtain result history, specified as an `sltest.testmanager.TestCase` object.

## Output Arguments

**tcresult — Test case result history**

array of `sltest.testmanager.TestCaseResult` objects

Test case result history, returned as an array of `sltest.testmanager.TestCaseResult` objects. Each object in the array contains the results for a single test case run.

## Examples

### Obtain Test Case Results

This example shows how to create a test file, test suite, and simulation test case programmatically. It also shows how to obtain the results. The test case runs on the `HeatPumpScenario` model.

Clear existing test files from the Test Manager.

```
sltest.testmanager.clear;
```

Create a new test file, test suite, and test case.

```
tf = sltest.testmanager.TestFile('TestFile1');
ts = createTestSuite(tf, 'TestSuite1');
tc = createTestCase(ts, 'simulation', 'TestCase1');
```

Remove the default test suite so that only the created test suite is used.

```
tsDel = tf.getTestSuites();
remove(tsDel(1));
```

Assign the system under test to the test case.

```
setProperty(tc, 'Model', 'HeatPumpScenario');
tcresult = run(tc);
```

Obtain the test case results

```
tcresultobj = getTestCaseResults(tc);
```

### **See Also**

getTestCase

**Introduced in R2019b**



# addLoggedSignal

**Class:** `sltest.testmanager.LoggedSignalSet`

**Package:** `sltest.testmanager`

Add a logged signal to a set

## Syntax

```
obj = addLoggedSignal(lgset,BlockPath,PortIndex)
obj = addLoggedSignal(____, 'LeafElement', busleaf)
```

## Description

`obj = addLoggedSignal(lgset,BlockPath,PortIndex)` creates and adds an `sltest.testmanager.LoggedSignal` object to a `sltest.testmanager.LoggedSignalSet` object. You must open or load the model to add signals from the model.

`obj = addLoggedSignal( ____, 'LeafElement', busleaf)` specifies the bus leaf element to include as a logged signal. `busleaf` is a string. If the signal has nested buses, specify the full path to the leaf ('parentLeaf.childLeaf').

## Input Arguments

### **lgset — Logged signal set**

`sltest.testmanager.LoggedSignalSet` object

Logged signal set object contained in a test case.

### **BlockPath — Block path object**

`Simulink.BlockPath` object | character vector

`Simulink.BlockPath` object that uniquely identifies the block that outputs the signal.

### **PortIndex — Output port index**

integer

Index of the output port for the block designated by `BlockPath`, starting from 1.

## Examples

### **Add Signals to a Signal Set**

Open a model and create a signal set.

```
openExample('sldemo_absbrake');

% Create test case
tf = sltest.testmanager.TestFile(strcat(pwd, '\mytf.mldatx'));
ts = sltest.testmanager.TestSuite(tf, 'myts');
```

```
tc = sltest.testmanager.TestCase(ts, 'baseline', 'mytc');
```

```
% Create signal set
mylgset = tc.addLoggedSignalSet;
```

Select the Vehicle Speed block and enter gcb. Use the returned path to create a Simulink.BlockPath object.

```
% Add signals to the set
bPath = Simulink.BlockPath('sldemo_absbrake/Vehicle speed');
sig1 = mylgset.addLoggedSignal(bPath,1);
sig2 = mylgset.addLoggedSignal(bPath,2);
```

```
setProperty(tc, 'Model', 'sldemo_absbrake');
```

## See Also

gcb

## Topics

“Create and Run Test Cases with Scripts”

“Capture Simulation Data in a Test Case”

**Introduced in R2019a**

# addLoggedSignalSet

**Class:** `sltest.testmanager.TestCase`

**Package:** `sltest.testmanager`

Add logged signal set to a test case

## Syntax

```
obj = addLoggedSignalSet(tc,Name,Value)
```

## Description

`obj = addLoggedSignalSet(tc,Name,Value)` creates and adds a `LoggedSignalSet` object to an `sltest.testmanager.TestCase` object.

## Input Arguments

### **tc** — Test case

`sltest.testmanager.TestCase` object

Test case object.

### **Name-Value Pair Arguments**

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

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

### **Name** — Name of the signal set

character vector

Name of the logged signal set.

```
Example: obj = addLoggedSignalSet(tc,'Name','mylgset');
```

### **SimulationIndex** — Simulation index

1 (default) | 2

When the test case is an equivalence test, this index specifies the simulation that contains the signal set.

```
Example: obj = getLoggedSignalSets(tc_equiv,'SimulationIndex',2);
```

## Examples

### **Add a Signal Set to a Test Case**

Open a model and create a test case.

```
openExample('sldemo_absbrake')

% Create test case
tf = sltest.testmanager.TestFile(strcat(pwd, '\mytf.mldatx'));
ts = sltest.testmanager.TestSuite(tf, 'myts');
tc = sltest.testmanager.TestCase(ts, 'baseline', 'mytc');

% Create a signal set
lgset = tc.addLoggedSignalSet;
```

## See Also

`sltest.testmanager.EquivalenceCriteria` | `sltest.testmanager.LoggedSignalSet`

## Topics

“Create and Run Test Cases with Scripts”

“Capture Simulation Data in a Test Case”

## Introduced in R2019a

# addParameterOverride

**Class:** `sltest.testmanager.ParameterSet`

**Package:** `sltest.testmanager`

Add parameter override to parameter set

## Syntax

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

## Description

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

`ovr = addParameterOverride(ps,Name,Value,blockPath)` adds a parameter override that overrides a block or block mask parameter.

## Input Arguments

### **ps** — Parameter set

`sltest.testmanager.ParameterSet` object

Parameter set to which you want to add the override, specified as a `sltest.testmanager.ParameterSet` object.

### **Name** — Parameter name

string scalar | character vector

Name of the parameter to override, specified as a string or character vector. If `Name` is a block parameter, use the programmatic name.

Data Types: `char` | `string`

### **Value** — Parameter value

numeric | string scalar

Parameter value to override, specified as a numeric or string scalar. If `Value` is a string scalar, it is evaluated as a MATLAB expression when the test executes.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `string`

### **blockPath** — Block path

string scalar | character vector

Block path of the parameter to override, specified as a string scalar or character vector. Specify `blockPath` only if the parameter to override is a block or block mask parameter stored in a mask workspace. The combination of `Name` and `blockPath` properties must be unique within a parameter set.

Data Types: char | string

## Output Arguments

### **ovr** — Parameter override

`sltest.testmanager.ParameterOverride` object

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

## Examples

### **Add Parameter Override to Parameter Set**

Open the model for this example.

```
openExample('sldemo_absbrake');
```

Create the test file, test suite, and test case structure.

```
tf = sltest.testmanager.TestFile("API Test File");
ts = createTestSuite(tf,"API Test Suite");
tc = createTestCase(ts,"baseline","Baseline API Test Case");
```

Remove the default test suite.

```
tsDel = getTestSuiteByName(tf,"New Test Suite 1");
remove(tsDel);
```

Assign the system under test to the test case.

```
setProperty(tc,"Model","sldemo_absbrake");
```

Capture the baseline criteria.

```
baseline = captureBaselineCriteria(tc,"baseline_API.mat",true);
```

Test a new model parameter by overriding it in the test case parameter set.

```
ps = addParameterSet(tc,"Name","API Parameter Set");
po = addParameterOverride(ps,"m",55);
```

### **Add Parameter Override to Override Block Parameter**

Open the model for this example.

```
openExample('sldemo_absbrake');
```

Create the test file, test suite, and test case structure.

```
tf = sltest.testmanager.TestFile("API Test File");
ts = createTestSuite(tf,"API Test Suite");
tc = createTestCase(ts,"baseline","Baseline API Test Case");
```

Remove the default test suite.

```
tsDel = getTestSuiteByName(tf, "New Test Suite 1");
remove(tsDel);
```

Assign the system under test to the test case.

```
setProperty(tc, "Model", "sldemo_absbrake");
```

Capture the baseline criteria.

```
baseline = captureBaselineCriteria(tc, "baseline_API.mat", true);
```

Test a new value of the Constant block labeled `Desired relative slip` by overriding it in the test case parameter set.

```
ps = addParameterSet(tc, "Name", "API Parameter Set");
po = addParameterOverride(ps, "Value", 0.22, ...
"sldemo_absbrake/Desired relative slip");
```

## See Also

### Topics

“Create and Run Test Cases with Scripts”

**Introduced in R2015b**

## export

**Class:** `sltest.testmanager.ParameterSet`

**Package:** `sltest.testmanager`

Export parameter set to Excel spreadsheet

### Syntax

```
export(ps, filename)
export(ps, filename, sheet)
```

### Description

`export(ps, filename)` exports `sltest.testmanager.ParameterSet` object data to an Excel file. The data for each parameter is exported into the spreadsheet columns, Parameter, Value, and BlockPath. For example, this spreadsheet shows three parameters from different model workspaces: the top model (which is unnamed), the `MdlRefLeaf1` model workspace, and the `MdlRefLeaf2` model workspace.

|   | A          | B      | C           |
|---|------------|--------|-------------|
| 1 | Parameter: | Value: | BlockPath:  |
| 2 | a          |        | 1           |
| 3 | a          | 3      | MdlRefLeaf1 |
| 4 | a          | 7      | MdlRefLeaf2 |

**Note** By default, the parameter set data is exported to the first sheet in the specified file and existing data in that sheet is overwritten.

`export(ps, filename, sheet)` exports the parameter set data to the specified sheet in the Excel file.

### Input Arguments

**ps — Parameter set to export**

`sltest.testmanager.ParameterSet` object

Parameter set to export, specified as an `sltest.testmanager.ParameterSet` object.

**filename — File name**

string | character vector

File name of the Excel workbook, specified as a string or character vector. If the file does not exist, it is created.

**sheet — Name of Excel workbook sheet**

string | character vector



Name of Excel workbook sheet, specified as a string or character vector. If you do not specify a sheet name, the parameter set data is exported to the first sheet in the specified file and existing data in that sheet is overwritten. If you specify a sheet name that does not exist, that sheet is created.

## Examples

### Export Parameter Set to Excel

Export test parameters to an Excel spreadsheet.

Add a parameter set a to a test case (tc), then override the value of parameter a in that parameter set. Export the parameter set to a sheet in an Excel file.

```
ps = addParameterSet(tc, 'Name', 'Param Set');
addParameterOverride(ps, 'a', 1);
export(ps, 'myPSfile.xlsx', 'Sheet2');
```

### See Also

`sltest.testmanager.ParameterSet | addParameterOverride`

### Topics

“Override Model Parameters in a Test Case”

### Introduced in R2020b

## addParameterSet

**Class:** `sltest.testmanager.TestCase`

**Package:** `sltest.testmanager`

Add parameter set

### Syntax

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

### Description

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

### Input Arguments

#### **tc** — Test case

`sltest.testmanager.TestCase` object

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

#### **Name-Value Pair Arguments**

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

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

Example: `'SimulationIndex',2`

#### **Name** — Parameter set name

auto-generated unique name (default) | character vector

Name of the parameter set, specified as a character vector. 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

character vector

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

#### **SimulationIndex** — Simulation number

1 (default) | 2

Simulation number to which the parameter set applies, 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.

### Sheets — Excel spreadsheet sheet names

string array

Excel spreadsheet names for which you want to create parameter sets, specified as a string array. If you do specify `Sheets`, a parameter set is created for each sheet in the file.

## Output Arguments

### pset — Parameter set

`sltest.testmanager.ParameterSet` object

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

## Examples

### Add Parameter Set to Test Case

```
% Open the model for this example
openExample('sldemo_absbrake');

% Create the test file, test suite, and test case structure
tf = sltest.testmanager.TestFile('API Test File');
ts = createTestSuite(tf,'API Test Suite');
tc = createTestCase(ts,'baseline','Baseline API Test Case');

% Remove the default test suite
tsDel = getTestSuiteByName(tf,'New Test Suite 1');
remove(tsDel);

% Assign the system under test to the test case
setProperty(tc,'Model','sldemo_absbrake');

% Test a new model parameter by overriding it in the test case
% parameter set
ps = addParameterSet(tc,'Name','API Parameter Set');
```

## See Also

### Topics

“Create and Run Test Cases with Scripts”

### Introduced in R2015b

## createInputDataFile

**Class:** sltest.testmanager.TestCase

**Package:** sltest.testmanager

Create file as basis for test case input signal data

### Syntax

```
input = createInputDataFile(tc, file)
input = createInputDataFile(tc, file, Name, Value)
```

### Description

`input = createInputDataFile(tc, file)` creates an input file for a test case. The file includes the signals based on the inport blocks in the model specified for the test case `tc`. You enter the time and signal data either in Microsoft Excel or, for MAT-files, using the signal editor in the Test Manager.

For information on the file format the Test Manager uses for Microsoft Excel files, see “Format Test Case Data in Excel”.

`input = createInputDataFile(tc, file, Name, Value)` uses additional arguments specified by one or more `Name, Value` pair arguments.

### Examples

#### Create Input File Template for Signal Data

Create the input file template for a test case, using the Excel file format. Name the sheet for the template `Optics`. Creating the file also adds it as input in the test case. After you create the file, edit it to populate it with signal data.

```
% Create test file
tf = sltest.testmanager.TestFile('Excel Input Test File');

% Create test suite and test case
ts = createTestSuite(tf, 'Excel Test Suite');
tc = createTestCase(ts, 'baseline', 'Excel Input Test Case');

% Assign the system under test to the test case
setProperty(tc, 'Model', 'sltestExcelExample');

% Generate Excel file template and add it to Inputs section, specifying the sheet name
input = createInputDataFile(tc, 'C:\MyHomeDir\myexcel.xlsx', 'Sheet', 'Optics');
```

### Input Arguments

#### **tc** — Test case

`sltest.testmanager.TestCase` object

Test case that you want to create the template input file from, specified as a `sltest.testmanager.TestCase` object.

**file — New input file name and path**

character vector

Name and path of MAT-file or Microsoft Excel to create, specified as a character vector.

---

**Note** In the generated Excel file, except for the last dimension, cells containing a zero value are not written because zeros are implicit in the MATLAB language. For string data, an empty string is written, and for enumerated data, the default value is written. This approach improves the performance of reading and writing Excel files.

---

**Name-Value Pair Arguments**

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

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

Example: `'Sheet', 'mysheet', 'Range', 'C1:F10', 'CreateIterations', false`

**Pairs for MAT-Files and Microsoft Excel Files****CreateIterations — Create a table iteration from the input**

true (default) | false

Option to create a table iteration from the input, specified as Boolean.

Example: `'CreateIterations', false`

**Pairs Only for Microsoft Excel Files****Sheet — Name of sheet for inputs**

character vector

Name to give the sheet in the new Excel file, specified as a character vector.

Example: `'Sheet', 'testinputs'`

**Range — Range of cells in sheet**

character vector

Ranges of cells to add the inputs to in the sheet, specified as a character vector. You can specify `'Range'` only if you also specify `'Sheet'`.

Example: `'Range', 'B2:C30'`

**Output Arguments****input — Test input file**

sltest.testmanager.TestInput object

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

**See Also**

`addInput` | `sltest.testmanager.TestCase`

**Topics**

“Format Test Case Data in Excel”

“Create Data Files for Test Case Input”

“Use External Excel or MAT-File Data in Test Cases”

**Introduced in R2018a**

# addReportBody

**Class:** `sltest.testmanager.TestResultReport`

**Package:** `sltest.testmanager`

Add main report body

## Syntax

```
addReportBody(obj)
```

## Description

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

This method also calls:

- `genResultSetBlock`
- `genTestSuiteResultBlock`
- `genTestCaseResultBlock`

## Input Arguments

**obj** — Test report object

`sltest.testmanager.TestResultReport` object

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

## See Also

`sltest.testmanager.TestResultReport`

## Topics

“Export Test Results”

“Create and Run Test Cases with Scripts”

**Introduced in R2016a**

## addReportTOC

**Class:** `sltest.testmanager.TestResultReport`

**Package:** `sltest.testmanager`

Add report table of contents

### Syntax

`addReportTOC(obj)`

### Description

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

#### Summary

| Name                                                                                                                                                                                                                                          | Outcome | Duration<br>(Seconds) |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------|-----------------------|
| <a href="#">Results: 2015-Dec-01 11:46:00</a>                                                                                                                                                                                                 | 2 ✓     | 11                    |
| <ul style="list-style-type: none"> <li> <ul style="list-style-type: none"> <li><a href="#">test1</a></li> </ul> </li> </ul>                                                                                                                   | 2 ✓     | 11                    |
| <ul style="list-style-type: none"> <li> <ul style="list-style-type: none"> <li> <ul style="list-style-type: none"> <li><a href="#">New Test Suite 1</a></li> </ul> </li> </ul> </li> </ul>                                                    | 2 ✓     | 11                    |
| <ul style="list-style-type: none"> <li> <ul style="list-style-type: none"> <li> <ul style="list-style-type: none"> <li> <ul style="list-style-type: none"> <li><a href="#">New Test Case 1</a></li> </ul> </li> </ul> </li> </ul> </li> </ul> | 2 ✓     | 10                    |
| <ul style="list-style-type: none"> <li> <ul style="list-style-type: none"> <li><a href="#">Iteration</a></li> </ul> </li> </ul>                                                                                                               | ✓       | 8                     |
| <ul style="list-style-type: none"> <li> <ul style="list-style-type: none"> <li> <ul style="list-style-type: none"> <li><a href="#">Iteration 1</a></li> </ul> </li> </ul> </li> </ul>                                                         | ✓       | 2                     |

### Input Arguments

#### **obj** — Test report object

`sltest.testmanager.TestResultReport` object

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

### See Also

`sltest.testmanager.TestResultReport`

#### Topics

“Export Test Results”

“Create and Run Test Cases with Scripts”

**Introduced in R2016a**



# addTitlePage

**Class:** `sltest.testmanager.TestResultReport`

**Package:** `sltest.testmanager`

Add report title page

## Syntax

`addTitlePage(obj)`

## Description

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

## Report Generated by Test Manager

---

|                |                             |
|----------------|-----------------------------|
| <b>Title:</b>  | <b>Test</b>                 |
| <b>Author:</b> | <b>Test Author</b>          |
| <b>Date:</b>   | <b>01-Dec-2015 11:50:23</b> |

---

### Test Environment

|           |          |
|-----------|----------|
| Platform: | PCWIN64  |
| MATLAB:   | (R2016b) |

## Input Arguments

**obj** — Test report object

`sltest.testmanager.TestResultReport` object

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

## See Also

`sltest.testmanager.TestResultReport`

### Topics

“Export Test Results”

“Create and Run Test Cases with Scripts”

**Introduced in R2016a**

## captureBaselineCriteria

**Class:** `sltest.testmanager.TestCase`

**Package:** `sltest.testmanager`

Capture baseline criteria and add to test case

### Syntax

```
baseline = captureBaselineCriteria(tc, file, append)
baseline = captureBaselineCriteria(tc, file, append, Name, Value)
```

### Description

`baseline = captureBaselineCriteria(tc, file, append)` runs the system under test and captures a baseline criteria set as a MAT-file, Simulation Data Inspector (SDI) MLDATX file, or Microsoft Excel file. The function returns a baseline criteria object, `sltest.testmanager.BaselineCriteria`. Use this function only if the test type is a baseline test case.

`baseline = captureBaselineCriteria(tc, file, append, Name, Value)` uses additional options specified by one or more `Name, Value` pair arguments.

### Input Arguments

#### **tc — Test case**

`sltest.testmanager.TestCase` object

Test case to capture baseline criteria in, specified as an `sltest.testmanager.TestCase` object.

#### **file — Excel, MAT, or SDI MLDATX file name and path**

character vector

File and path name of the baseline criteria file, specified as a character vector. You can specify a MAT-file, Simulation Data Inspector MLDATX file, or a Microsoft Excel file.

Example: `'C:\MATLAB\baseline_API.mat'`

#### **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 existing criteria.

#### **Name-Value Pair Arguments**

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

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

Example: 'Sheet', 'mysheet', 'Range', 'C1:F10'

### **Pairs for MAT-Files, Simulation Data Inspector (SDI) .mldatx file, and Microsoft Excel Files**

#### **Release — Simulink release to capture the baseline in**

character vector | string array

Simulink release to capture the baseline data in, specified as a character vector or string array. Use a release specified in your preferences. For more information, see `sltest.testmanager.getpref` and `sltest.testmanager.setpref`.

Example: 'Release', 'R2017a'

#### **CaptureForIterations — Capture baseline data for test case iterations**

false (default) | true

Whether to capture baseline data for test case iterations, specified as a Boolean.

Example: 'CaptureForIterations', true

### **Pairs Only for Microsoft Excel Files**

#### **Sheet — Name of sheet to capture baseline criteria to**

character vector | string array

Name to sheet to capture baseline criteria to, specified as a character vector or string array.

Example: 'Sheet', 'testinputs'

#### **Range — Range of cells**

character vector | string array

Ranges of cells to capture baseline criteria to, specified as a character vector or string array. You can specify 'Range' only if you also specify 'Sheet'.

Example: 'Range', 'B2:C30'

## **Output Arguments**

### **baseline — Baseline criteria object**

object

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

## **Examples**

### **Capture Baseline Criteria**

```
% Open the model for this example
openExample('sldemo_absbrake');

% Create the test file, test suite, and test case structure
tf = sltest.testmanager.TestFile('API Test File');
ts = createTestSuite(tf, 'API Test Suite');
```

```
tc = createTestCase(ts,'baseline','Baseline API Test Case');

% Remove the default test suite
tsDel = getTestSuiteByName(tf,'New Test Suite 1');
remove(tsDel);

% Assign the system under test to the test case
setProperty(tc,'Model','sldemo_absbrake');

% Capture the baseline criteria
baseline = captureBaselineCriteria(tc,'baseline_API.mat',true);
```

## See Also

### Topics

“Create and Run Test Cases with Scripts”

“Capture Baseline Criteria”

**Introduced in R2015b**

# captureEquivalenceCriteria

**Class:** `sltest.testmanager.TestCase`

**Package:** `sltest.testmanager`

Capture equivalence criteria and add to test case

## Syntax

```
eq = captureEquivalenceCriteria(tc,replaceAll)
```

## Description

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

## Input Arguments

### **tc — Test case**

`sltest.testmanager.TestCase` object

Test case to capture equivalence criteria in, specified as an `sltest.testmanager.TestCase` object.

### **replaceAll — Replace equivalence criteria**

`true` | `false`

Replace existing equivalence criteria if criteria already exist in the test case, specified as a Boolean. `true` replaces existing criteria, and `false` produces an error if criteria already exist in the test case.

## Output Arguments

### **eq — Equivalence criteria object**

object

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

## Examples

### **Add Equivalence Criteria to Test Case**

```
% Open the model for this example
openExample('sldemo_absbrake');

% Create the test file, test suite, and test case structure
tf = sltest.testmanager.TestFile('API Test File');
```

```
ts = createTestSuite(tf, 'API Test Suite');
tc = createTestCase(ts, 'equivalence', 'Equivalence Test Case');

% Remove the default test suite
tsDel = getTestSuiteByName(tf, 'New Test Suite 1');
remove(tsDel);

% Assign the system under test to the test case
% for Simulation 1 and Simulation 2
setProperty(tc, 'Model', 'sldemo_absbrake', 'SimulationIndex', 1);
setProperty(tc, 'Model', 'sldemo_absbrake', 'SimulationIndex', 2);

% Capture equivalence criteria
eq = captureEquivalenceCriteria(tc);
```

## See Also

### Topics

“Create and Run Test Cases with Scripts”

**Introduced in R2015b**

# close

**Class:** `sltest.testmanager.TestFile`

**Package:** `sltest.testmanager`

Close test file in Test Manager

## Syntax

```
close(tf)
```

## Description

`close(tf)` closes the test file in the Test Manager and does not save unsaved changes.

## Input Arguments

**tf** — Test file

`sltest.testmanager.TestFile` object

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

## Examples

### Close Test File

```
% Create test file in Test Manager
tf = sltest.testmanager.TestFile('My Test File');

% Close test file
close(tf);
```

## See Also

### Topics

“Create and Run Test Cases with Scripts”

**Introduced in R2015b**

## convertTestType

**Class:** `sltest.testmanager.TestCase`

**Package:** `sltest.testmanager`

Convert test from one type to another

### Syntax

```
convertTestType(tc, testType)
```

### Description

`convertTestType(tc, testType)` converts the test case type to a different type.

If you convert certain test case types to another type, then you can lose information about the original test case:

- Baseline to simulation or equivalence — baseline criteria is lost
- Equivalence to simulation or baseline — equivalence criteria is lost for Simulation 1 and 2

### Input Arguments

#### **tc — Test case**

`sltest.testmanager.TestCase` object

Test case that you want to convert to a different type, specified as a `sltest.testmanager.TestCase` object.

#### **testType — Test case type**

`sltest.testmanager.TestCaseTypes.Baseline` |  
`sltest.testmanager.TestCaseTypes.Equivalence` |  
`sltest.testmanager.TestCaseTypes.Simulation`

Test case type that you want to convert to, specified as a `sltest.testmanager.TestCaseTypes` enumeration. Specify:

- `sltest.testmanager.TestCaseTypes.Baseline` to convert to a baseline test case
- `sltest.testmanager.TestCaseTypes.Equivalence` to convert to an equivalence test case
- `sltest.testmanager.TestCaseTypes.Simulation` to convert to a simulation test case

### Examples

#### **Change Baseline Test Case to Simulation**

```
% Open the model for this example
openExample('sldemo_absbrake');
```

```
% Create new test file with test suite and default test case
```



```

tf = sltest.testmanager.TestFile('API Test File');
ts = getTestSuites(tf);
tc = getTestCases(ts);

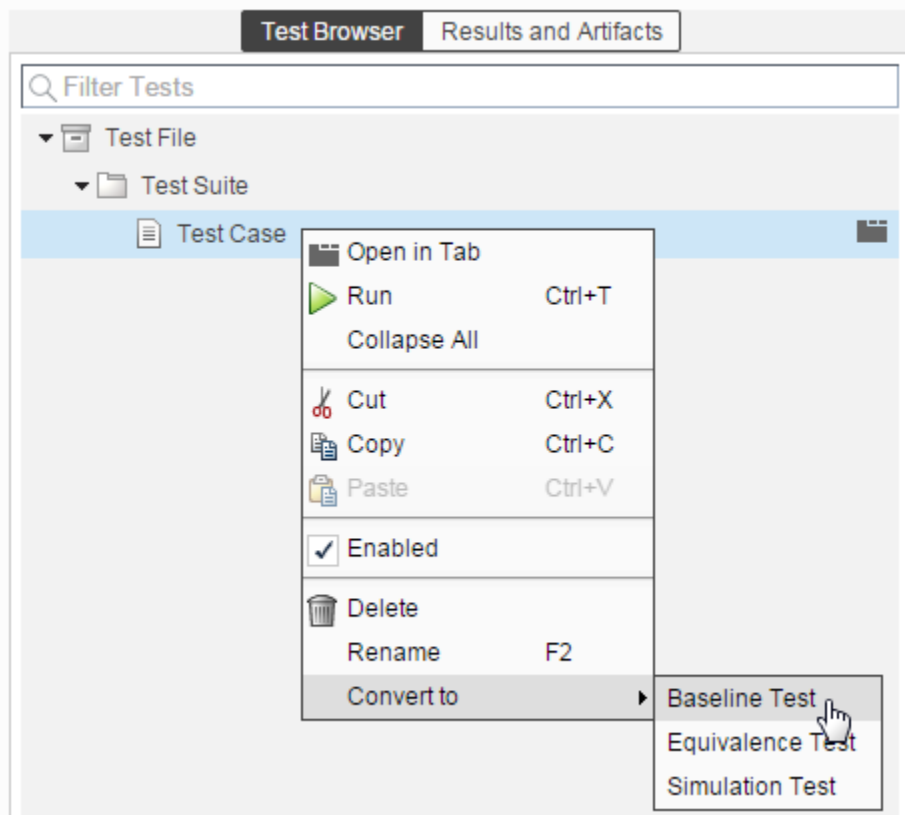
% Assign system under test to test case
setProperty(tc, 'Model', 'sldemo_absbrake');

% Convert test case to simulation type
convertTestType(tc, sltest.testmanager.TestCaseTypes.Simulation);

```

## Alternatives

You can also convert the test case type using the context menu in the **Test Browser** pane. Right-click the test case, select **Convert to**, and then select the test case type you want to convert the test case to.



## See Also

`sltest.testmanager.TestCase`

## Topics

“Create and Run Test Cases with Scripts”

## Introduced in R2016b

## convertTestType

**Class:** `sltest.testmanager.TestFile`

**Package:** `sltest.testmanager`

Convert test from one type to another

### Syntax

```
convertTestType(tf, testType)
```

### Description

`convertTestType(tf, testType)` converts the test case type to a different type. The function converts all test cases contained in the test file. If you want to convert a single test case, then use the `convertTestType (TestCase)` method.

If you convert certain test case types to another type, then you can lose information about the original test case:

- Baseline to simulation or equivalence — baseline criteria is lost
- Equivalence to simulation or baseline — equivalence criteria is lost for Simulation 1 and 2

### Input Arguments

#### **tf — Test file**

`sltest.testmanager.TestFile` object

Test file that contains the test cases you want to convert to a different type, specified as a `sltest.testmanager.TestFile` object.

#### **testType — Test case type**

`sltest.testmanager.TestCaseTypes.Baseline` |  
`sltest.testmanager.TestCaseTypes.Equivalence` |  
`sltest.testmanager.TestCaseTypes.Simulation`

Test case type that you want to convert to, specified as a `sltest.testmanager.TestCaseTypes` enumeration. Specify:

- `sltest.testmanager.TestCaseTypes.Baseline` to convert to a baseline test case
- `sltest.testmanager.TestCaseTypes.Equivalence` to convert to an equivalence test case
- `sltest.testmanager.TestCaseTypes.Simulation` to convert to a simulation test case

### Examples

#### **Change Baseline Test Cases to Simulation**

```
% Create new test file with test suite and default test case
tf = sltest.testmanager.TestFile('API Test File');
```

```

ts = getTestSuites(tf);
tc1 = getTestCases(ts);

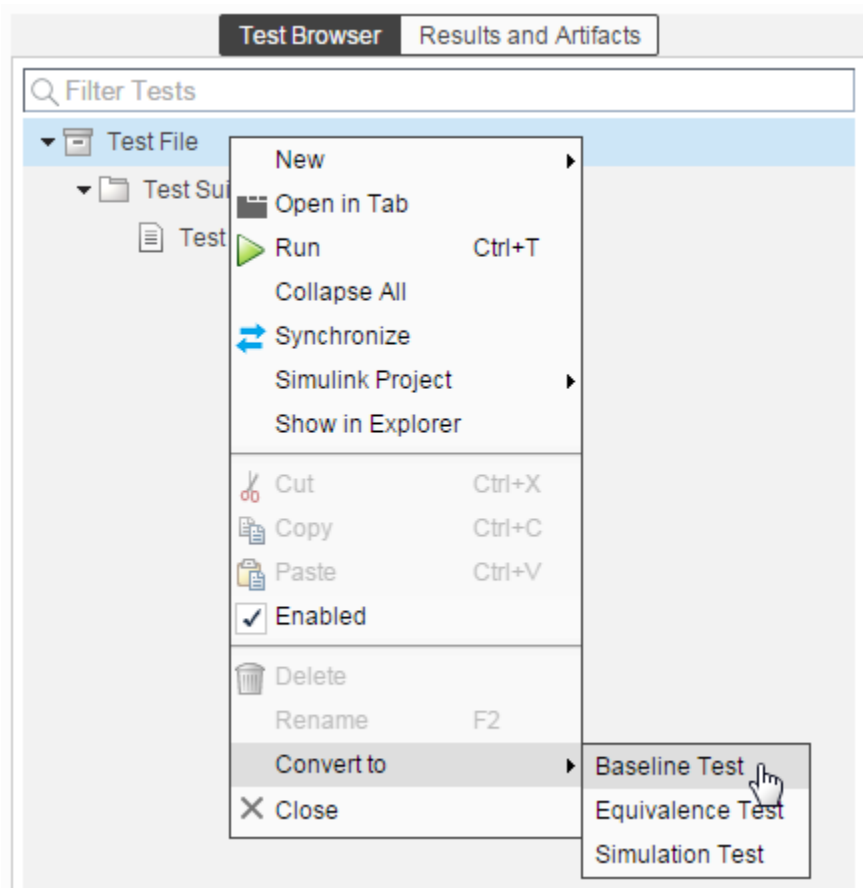
% Create new test case
tc2 = createTestCase(ts, 'baseline', 'API Test Case');

% Convert test cases to simulation type
convertTestType(tf, sltest.testmanager.TestCaseTypes.Simulation);

```

## Alternatives

You can also convert the test case type using the context menu in the **Test Browser** pane. Right-click the test file, select **Convert to**, and then select the test case type you want to convert the test cases to.



## See Also

`sltest.testmanager.TestCase`

## Topics

“Create and Run Test Cases with Scripts”

**Introduced in R2016b**

## convertTestType

**Class:** `sltest.testmanager.TestSuite`

**Package:** `sltest.testmanager`

Convert test from one type to another

### Syntax

```
convertTestType(ts, testType)
```

### Description

`convertTestType(ts, testType)` converts the test case type to a different type. The function converts all of the test cases contained in the test suite. If you want to convert a single test case, then use the `convertTestType (TestCase)` method.

If you convert certain test case types to another type, then you can lose information about the original test case:

- Baseline to simulation or equivalence — baseline criteria is lost
- Equivalence to simulation or baseline — equivalence criteria is lost for Simulation 1 and 2

### Input Arguments

#### **ts — Test suite**

`sltest.testmanager.TestSuite` object

Test suite that contains the test cases you want to convert to a different type, specified as a `sltest.testmanager.TestSuite` object.

#### **testType — Test case type**

`sltest.testmanager.TestCaseTypes.Baseline` |  
`sltest.testmanager.TestCaseTypes.Equivalence` |  
`sltest.testmanager.TestCaseTypes.Simulation`

Test case type that you want to convert to, specified as a `sltest.testmanager.TestCaseTypes` enumeration. Specify:

- `sltest.testmanager.TestCaseTypes.Baseline` to convert to a baseline test case
- `sltest.testmanager.TestCaseTypes.Equivalence` to convert to an equivalence test case
- `sltest.testmanager.TestCaseTypes.Simulation` to convert to a simulation test case

### Examples

#### **Change Baseline Test Cases to Simulation**

```
% Create new test file with test suite and default test case
tf = sltest.testmanager.TestFile('API Test File');
```

```

ts = getTestSuites(tf);
tc1 = getTestCases(ts);

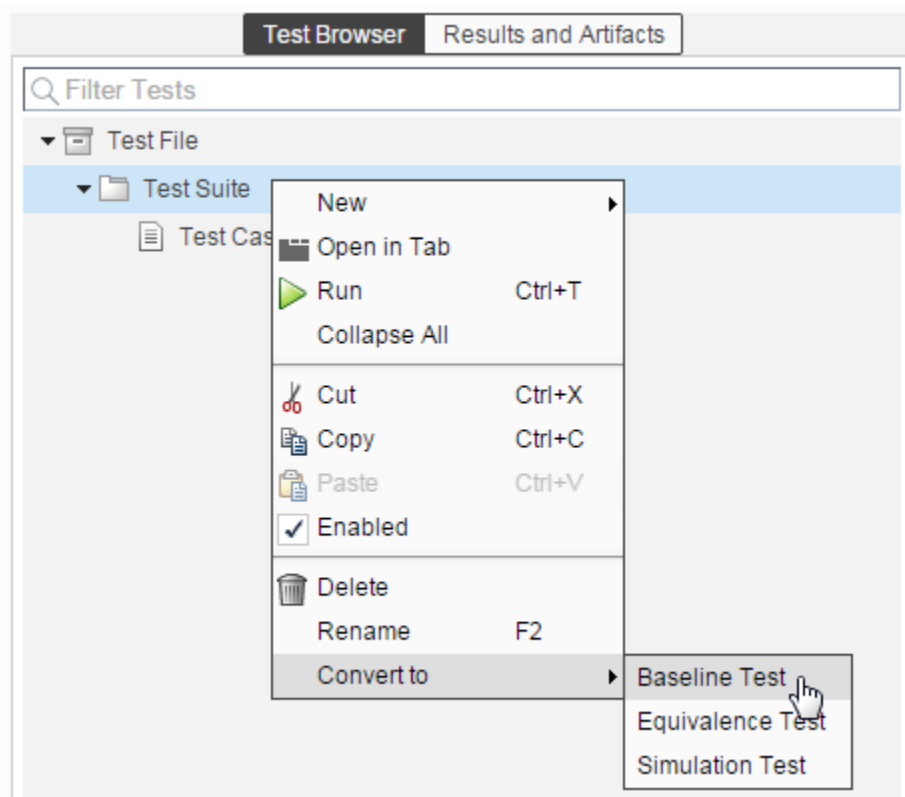
% Create new test case
tc2 = createTestCase(ts, 'baseline', 'API Test Case');

% Convert test cases to simulation type
convertTestType(ts, sltest.testmanager.TestCaseTypes.Simulation);

```

## Alternatives

You can also convert the test case type using the context menu in the **Test Browser** pane. Right-click the test suite, select **Convert to**, and then select the test case type you want to convert the test cases to.



## See Also

`sltest.testmanager.TestCase`

## Topics

“Create and Run Test Cases with Scripts”

**Introduced in R2016b**

## copySimulationSettings

**Class:** sltest.testmanager.TestCase

**Package:** sltest.testmanager

Copy simulation setting in equivalence test case

### Syntax

```
copySimulationSettings(tc, fromSimIndex, toSimIndex)
```

### Description

`copySimulationSettings(tc, 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

**tc — Equivalence test case**

sltest.testmanager.TestCase object

Equivalence test case that you want to copy simulation settings in, specified as a sltest.testmanager.TestCase object.

**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.

### Examples

**Copy Simulation Settings in Test Case**

```
% Open the model for this example
openExample('sldemo_absbrake');

% Create the test file, test suite, and test case structure
tf = sltest.testmanager.TestFile('API Test File');
ts = createTestSuite(tf, 'API Test Suite');
tc = createTestCase(ts, 'equivalence', 'Equivalence Test Case');

% Remove the default test suite
```

```
tsDel = getTestSuiteByName(tf, 'New Test Suite 1');
remove(tsDel);

% Assign the system under test to the test case
% for Simulation 1 and Simulation 2
setProperty(tc, 'Model', 'sldemo_absbrake', 'SimulationIndex', 1);
setProperty(tc, 'Model', 'sldemo_absbrake', 'SimulationIndex', 2);

% Change simulation stop time in Simulation 1
setProperty(tc, 'StopTime', 100, 'SimulationIndex', 1);

% Copy simulation setting to Simulation 2
copySimulationSettings(tc, 1, 2);
```

## See Also

### Topics

“Create and Run Test Cases with Scripts”

**Introduced in R2015b**

## createTestCase

**Class:** `sltest.testmanager.TestSuite`

**Package:** `sltest.testmanager`

Create test case

### Syntax

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

### Description

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

### Input Arguments

**ts — Test suite**

`sltest.testmanager.TestSuite` object

Test suite that you want to add a test case to, specified as an `sltest.testmanager.TestSuite` object.

**type — Test case type**

"baseline" (default) | "equivalence" | "simulation"

Test case type, specified as:

- "baseline"
- "equivalence"
- "simulation"

**name — Test case name**

string scalar | character vector

Test case name, specified as a string scalar or character vector. If this input argument is empty, then the Test Manager gives the test case a unique name.

**runOnTarget — Run simulation on target**

cell array of Booleans

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



## Output Arguments

### **tc** — Test case

sltest.testmanager.TestCase object

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

## Examples

### Create Test Case

```
% Create test file
tf = sltest.testmanager.TestFile("test_file");

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

% Create test case
tc = createTestCase(ts,"baseline","My Baseline Test")
```

## See Also

### Topics

“Create and Run Test Cases with Scripts”

**Introduced in R2015b**

## createTestForSubsystem

**Class:** `sltest.testmanager.TestFile`

**Package:** `sltest.testmanager`

(To be removed) Create test harness and test case for subsystem in test file

---

**Note** `sltest.testmanager.TestFile.createTestForSubsystem` will be removed in a future release. Use `sltest.testmanager.createTestForComponent` instead. For more information, see “Compatibility Considerations”.

---

### Syntax

```
tc = createTestForSubsystem(tf, 'Subsystem', subsystem)
tc = createTestForSubsystem(tf, 'Subsystem', subsystem, Name, Value)
```

### Description

`tc = createTestForSubsystem(tf, 'Subsystem', subsystem)` creates a harness on the specified subsystem, Model block, Stateflow chart, or another supported model component (see “Test Harness and Model Relationship”). It also creates a baseline test case and test suite in the specified Excel file. This function also simulates the model and adds the input and the output files to the test case, as MAT-files. For more information, see “Generate Tests and Test Harnesses for a Component or Model”.

`tc = createTestForSubsystem(tf, 'Subsystem', subsystem, Name, Value)` uses additional options specified by one or more `Name, Value` pair arguments. Use this syntax to use Microsoft Excel files as input and output files.

### Input Arguments

#### **tf** — Test file

`sltest.testmanager.TestFile` object

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

#### **subsystem** — Subsystem path

character vector | string array

Full path of the subsystem, specified as a character vector or string array. If the subsystem or component is in a Model block, you do not have to include the name of the block in the path. You can specify only the top-level model and system or the component under test.

Example: `'f14/Controller'`

#### **Name-Value Pair Arguments**

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

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

Example: `'CreateExcelFile',true,'Sheet','mysheet'`

### **TopModel — Model name at top of hierarchy**

character vector | string

Model name at the top of the hierarchy if the subsystem is in a referenced model, specified as a character vector or string.

Example: `'TopModel','Plant'`

### **TestType — Test case type**

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

Test case type to create, specified as 'baseline', 'equivalence', or 'simulation'.

Example: `'TestType','equivalence'`

### **UseSubsystemInputs — Option to simulate model to obtain inputs**

true (default) | false

Option to simulate the model to obtain subsystem inputs to use as inputs in the created test harness, specified as a logical. If this property is true, the test harness uses the subsystem inputs from the model simulation. If this property is false, the test harness does not use the subsystem inputs from the simulation.

Example: `'UseSubsystemInputs',false`

### **Pairs for Equivalence Testing**

#### **Simulation1Mode — Simulation mode for simulation 1**

"Normal" | "Accelerator"

Simulation mode for simulation 1 of an equivalence test, specified as either "Normal" or "Accelerator". If you do not specify a simulation mode, the mode of the system under test is used.

Example: `"Simulation1Mode","Normal"`

#### **Simulation2Mode — Simulation mode for simulation 2**

string | character vector

Simulation mode for simulation 2 of an equivalence test, specified as one of these values:

- "Normal"
- "Accelerator"
- "Rapid Accelerator"
- "Software-in-the-Loop (SIL)"
- "Processor-in-the-Loop (PIL)"

If you do not specify a simulation mode, the mode of the system under test is used. For information about simulation modes, see “Choosing a Simulation Mode”. If `TestType` is `equivalence` and `Simulation2Mode` is `Software-in-the-Loop (SIL)`, an extra test harness is created in addition to the test case and test harness.

Example: `"Simulation2Mode","Software-in-the-Loop (SIL)"`

**Pairs for MAT-Files****InputsLocation — Input file name and path for MAT-files**

character vector | string

Input file name and path for MAT-files, specified as a character vector or string array. Include the file extension `.mat`.

Example: `'InputsLocation','C:\MATLAB\inputs_data.mat'`

**BaselineLocation — Baseline file location**

character vector | string

File name and path to save baseline data to, specified as a character vector or string. Include the file extension `.mat`.

Example: `'BaselineLocation','C:\MATLAB\baseline_data.mat'`

**Pairs for Microsoft Excel Files****CreateExcelFile — Use Excel format for inputs and outputs**

false (default) | true

Option to use Excel format for inputs and, for baseline tests only, outputs, specified as `true` or `false`. If you use the `ExcelFileLocation` argument to specify the file name and location, you do not need to also use `CreateExcelFile`.

Example: `'CreateExcelFile',true`

**ExcelFileLocation — Name and location for Excel file**

character vector | string

File name and path to save the Excel file to, specified as a character vector or string. Include the extension `.xlsx`. If you specify a location, you do not need to also use the `'CreateExcelFile'` option.

---

**Note** If `SLDVTTestGeneration` is `true` and `HarnessSource` is "Signal Editor", you cannot save data to an Excel file.

---

Example: `'ExcelFileLocation','C:\MATLAB\baseline_data.xlsx'`

**Sheet — Name of Excel sheet to save data to**

character vector | string

Name of the sheet to save Excel data to, specified as a character vector or string.

Example: `'Sheet','MySubsysTest'`

**Pairs for Simulink Design Verifier****SLDVTTestGeneration — Whether to generate tests using Simulink Design Verifier**

false (default) | true

Whether to generate tests using Simulink Design Verifier, specified as a logical. If this property is `true`, Simulink Design Verifier generates the tests to include in the test file. An error occurs if this property is `true`, but Simulink Design Verifier is not installed.

---

**Note** To generate tests from Simulink Design Verifier, the system under test must be an atomic subsystem.

---

Example: 'SLDVTTestGeneration',true

### **HarnessSource — Input source block for the harness**

"Inport" (default) | "Signal Editor"

Input source block for the test harness, specified as "Inport" or "Signal Editor".

Example: "HarnessSource", "Signal Editor"

## **Output Arguments**

### **tc — Test case**

sltest.testmanager.testcase object

Test case, returned as an sltest.testmanager.testcase object.

## **Examples**

### **Create Test for a Subsystem**

Create a baseline test case and test harness for a subsystem in a model reference, then save the inputs in Excel format. Baseline tests are used primarily for debugging.

```
% Load the model
load_system('sltestBasicCruiseControl');

% Create a test file
tf = sltest.testmanager.TestFile('My Test File');

% Create test from subsystem
createTestForSubsystem(tf, 'Subsystem', ...
 'sltestBasicCruiseControl/Controller/PI_Controller', ...
 'TestType', 'baseline', 'CreateExcelFile', true);
```

## **Compatibility Considerations**

### **sltest.testmanager.TestFile.createTestForSubsystem will be removed**

*Not recommended starting in R2020b*

sltest.testmanager.TestFile.createTestForSubsystem will be removed in a future release. Use sltest.testmanager.createTestForComponent instead. This replacement function allows you to create test cases for models, in addition to subsystems. Four additional name-value pairs are included in sltest.testmanager.createTestForComponent that require updates to your code:

- **TestFile** — Required name-value pair to specify the test file for the created test case
- **Component** — Required name-value pair to specify the component or model to test
- **CreateTestFile** — Optional name-value pair for whether to create a new test file
- **CreateHarness** — Optional name-value pair for whether to create a test harness in addition to the test case

## **See Also**

`sltest.testmanager.createTestForComponent`

## **Topics**

“Create and Run Test Cases with Scripts”

“Generate Tests and Test Harnesses for a Component or Model”

“Create and Run a Back-to-Back Test”

**Introduced in R2016a**

# createTestForSubsystem

**Class:** `sltest.testmanager.TestSuite`

**Package:** `sltest.testmanager`

(To be removed) Create test harness and test case for subsystem in test suite

---

**Note** `sltest.testmanager.TestSuite.createTestForSubsystem` will be removed in a future release. Use `sltest.testmanager.createTestForComponent` instead. For more information, see “Compatibility Considerations”.

---

## Syntax

```
tc = createTestForSubsystem(ts, 'Subsystem', subsystem)
tc = createTestForSubsystem(ts, 'Subsystem', subsystem, Name, Value)
```

## Description

`tc = createTestForSubsystem(ts, 'Subsystem', subsystem)` creates a harness on the specified subsystem and a baseline test case in the specified test suite. This function also simulates the model and adds the input and the output files to the test case as MAT-files. files. For more information, see “Generate Tests and Test Harnesses for a Component or Model”.

`tc = createTestForSubsystem(ts, 'Subsystem', subsystem, Name, Value)` uses additional options specified by one or more `Name, Value` pair arguments. Use this syntax to use Microsoft Excel files as input and output files.

## Input Arguments

### **ts** — Test suite

`sltest.testmanager.TestSuite` object

Test suite in which to add the new test case, specified as an `sltest.testmanager.TestSuite` object.

### **subsystem** — Subsystem path

character vector | string array

Full path of the subsystem, specified as a character vector or string array. If the subsystem or component is in a Model block, you do not have to include the name of the block in the path. You can specify only the top-level model and system or the component under test.

Example: `'f14/Controller'`

### **Name-Value Pair Arguments**

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

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

Example: 'CreateExcelFile',true,'Sheet','mysheet'

### **TopModel — Model name at top of hierarchy**

character vector | string array

Model name at the top of the hierarchy if the subsystem is in a referenced model, specified as a character vector or string array.

Example: 'TopModel','Plant'

### **TestType — Test case type**

"baseline" (default) | "equivalence" | "simulation"

Test case type, specified as one of these strings: 'baseline', 'equivalence', or 'simulation'.

Example: 'TestType','equivalence'

### **UseSubsystemInputs — Option to simulate model to obtain inputs**

true (default) | false

Option to simulate the model to obtain subsystem inputs to use as inputs in the created test harness, specified as a logical. If this property is true, the test harness uses the subsystem inputs from the model simulation. If this property is false, the test harness does not use the subsystem inputs from the simulation.

Example: 'UseSubsystemInputs',false

### **Pairs for Equivalence Testing**

#### **Simulation1Mode — Simulation mode for simulation 1**

"Normal" | "Accelerator"

Simulation mode for simulation 1 of an equivalence test, specified as either "Normal" or "Accelerator". If you do not specify a simulation mode, the mode of the system under test is used.

Example: "Simulation1Mode","Normal"

#### **Simulation2Mode — Simulation mode for simulation 2**

string | character vector

Simulation mode for simulation 2 of an equivalence test, specified as one of these values:

- "Normal"
- "Accelerator"
- "Rapid Accelerator"
- "Software-in-the-Loop (SIL)"
- "Processor-in-the-Loop (PIL)"

If you do not specify a simulation mode, the mode of the system under test is used. If `TestType` is `equivalence` and `Simulation2Mode` is `Software-in-the-Loop (SIL)`, an extra test harness is created in addition to the test case and test harness.

Example: "Simulation2Mode","Software-in-the-Loop (SIL)"



**Pairs for MAT-Files****InputsLocation — Input file name and path for MAT-files**

character vector | string array

Input file name and location for MAT-files, specified as a character vector or string array. Include the file extension `.mat`.

Example: `'InputsLocation','C:\MATLAB\inputs_data.mat'`

**BaselineLocation — Baseline file location**

character vector | string array

File name and path to save baseline data to, specified as a character vector or string. Include the file extension `.mat`.

Example: `'BaselineLocation','C:\MATLAB\baseline_data.mat'`

**Pairs for Microsoft Excel Files****CreateExcelFile — Use Excel format for inputs and outputs**

false (default) | true

Option to use Excel format for inputs and, for baseline tests only, outputs, specified as `true` or `false`. If you use the `'ExcelFileLocation'` argument to specify the file name and location, you do not need to also use `'CreateExcelFile'`.

Example: `'CreateExcelFile',true`

**ExcelFileLocation — Name and location for Excel file**

character vector | string array

File name and path to save the Excel file to, specified as a character vector or string array. Include the extension `.xlsx`. If you specify a location, you do not need to also use the `'CreateExcelFile'` option.

---

**Note** If `SLDVTTestGeneration` is `true` and `HarnessSource` is "Signal Editor", you cannot save data to an Excel file.

---

Example: `'ExcelFileLocation','C:\MATLAB\baseline_data.xlsx'`

**Sheet — Name of Excel sheet to save data to**

character vector | string array

Name of the sheet to save Excel data to, specified as a character vector or string array.

Example: `'Sheet','MySubsysTest'`

**Pairs for Simulink Design Verifier****SLDVTTestGeneration — Whether to generate tests using Simulink Design Verifier**

false (default) | true

Whether to generate tests using Simulink Design Verifier, specified as a logical. If this property is `true`, Simulink Design Verifier generates the tests to include in the test suite.

---

**Note** To generate tests from Simulink Design Verifier, the system under test must be an atomic subsystem.

---

Example: 'SLDVTTestGeneration',true

### **HarnessSource — Input source block for the harness**

"Inport" (default) | "Signal Editor"

Input source block for the test harness, specified as "Inport" or "Signal Editor".

Example: "HarnessSource", "Signal Editor"

## **Output Arguments**

### **tc — Test case**

sltest.testmanager.testcase object

Test case, returned as an sltest.testmanager.testcase object.

## **Examples**

### **Create Test for a Subsystem**

Create a baseline test case and test harness for a subsystem, then save the inputs in Excel format.

```
% Load the model
load_system('rtwdemo_sil_block');

% Create a test file and get the test suite
tf = sltest.testmanager.TestFile('My B2B Test File');
ts = getTestSuites(tf);

% Create test from subsystem
createTestForSubsystem(ts, 'Subsystem', ...
 'rtwdemo_sil_block/Controller', ...
 'CreateExcelFile', true);
```

## **Compatibility Considerations**

### **sltest.testmanager.TestSuite.createTestForSubsystem Will Be Removed**

*Not recommended starting in R2020b*

sltest.testmanager.TestSuite.createTestForSubsystem will be removed in a future release. Use sltest.testmanager.createTestForComponent instead. This replacement function allows you to create test cases for models, in addition to subsystems. Four additional name-value pairs are included in sltest.testmanager.createTestForComponent that require updates to your code:

- **TestFile** — Required name-value pair to specify the test file for the created test case
- **Component** — Required name-value pair to specify the component or model to test
- **CreateTestFile** — Optional name-value pair for whether to create a new test file
- **CreateHarness** — Optional name-value pair for whether to create a test harness in addition to the test case

## **See Also**

`sltest.testmanager.createTestForComponent`

## **Topics**

“Create and Run Test Cases with Scripts”

“Generate Tests and Test Harnesses for a Component or Model”

“Create and Run a Back-to-Back Test”

**Introduced in R2016a**

## createTestSuite

**Class:** `sltest.testmanager.TestFile`

**Package:** `sltest.testmanager`

Create new test suite

### Syntax

```
ts = createTestSuite(tf,suiteName)
```

### Description

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

### Input Arguments

**tf — Test file**

`sltest.testmanager.TestFile` object

Test file that you want to create the test suite within, specified as a `sltest.testmanager.TestFile` object.

**suiteName — Test suite name**

string scalar | character vector

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

Example: "My Test Suite"

### Output Arguments

**ts — Test suite object**

`sltest.testmanager.TestSuite` object

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

### Examples

**Create a Test Suite**

```
% Create a test file
tf = sltest.testmanager.TestFile("My Test File");
```

```
% Create a test suite
ts = createTestSuite(tf, "My Test Suite");
```

## See Also

### Topics

“Create and Run Test Cases with Scripts”

**Introduced in R2015b**

## createTestSuite

**Class:** `sltest.testmanager.TestSuite`

**Package:** `sltest.testmanager`

Create test suite

### Syntax

```
tsOut = createTestSuite(ts,suiteName)
```

### Description

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

### Input Arguments

**ts — Test suite**

`sltest.testmanager.TestSuite` object

Test suite that want to add another test suite to, specified as an `sltest.testmanager.TestSuite` object.

**suiteName — Test suite name**

string scalar | character vector

Test suite name, specified as a string scalar or character vector. If this input argument is empty, then the Test Manager gives the test suite a unique name.

Example: "My Test Suite"

### Output Arguments

**tsOut — Test suite**

`sltest.testmanager.TestSuite` object

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

### Examples

**Create Test Suite**

```
% Create test file
tf = sltest.testmanager.TestFile("test_file");

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

```
% Create another new test suite using method
ts2 = createTestSuite(ts, "Baseline Tests")
```

## See Also

### Topics

“Create and Run Test Cases with Scripts”

**Introduced in R2015b**

## deleteIterations

**Class:** sltest.testmanager.TestCase

**Package:** sltest.testmanager

Delete test iterations that belong to test case

### Syntax

```
deleteIterations(tc,iter)
```

### Description

deleteIterations(tc,iter) deletes one or more test iterations from the test case.

### Input Arguments

**tc** — Test case to delete iteration from

sltest.testmanager.TestCase object

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

**iter** — Test iteration to delete

sltest.testmanager.TestIteration object array

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

### Examples

#### Delete Test Iteration from Test Case

```
% Open the model for this example
openExample('sldemo_autotrans');

% Create test file, test suite, and test case structure
tf = sltest.testmanager.TestFile('Iterations Test File');
ts = getTestSuites(tf);
tc = createTestCase(ts,'simulation','Simulation Iterations');

% Specify model as system under test
setProperty(tc,'Model','sldemo_autotrans');

% Set up table iteration
% Create iteration object
testItr1 = sltestiteration;
% Set iteration settings
setTestParam(testItr1,'SignalBuilderGroup','Passing Maneuver');
% Add the iteration to test case
addIteration(tc,testItr1);
```



```
% Set up another table iteration
% Create iteration object
testItr2 = sltestiteration;
% Set iteration settings
setTestParam(testItr2,'SignalBuilderGroup','Coasting');
% Add the iteration to test case
addIteration(tc,testItr2);

% Delete first iteration
deleteIterations(tc,testItr1);
```

## See Also

sltest.testmanager.TestIteration

## Topics

“Test Iterations”

“Create and Run Test Cases with Scripts”

**Introduced in R2016a**

## genBaselineInfoTable

**Class:** sltest.testmanager.TestResultReport

**Package:** sltest.testmanager

Generate baseline dataset information table

### Syntax

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

### Description

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

#### Baseline Information

Baseline Name: baseline1.mat

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

| Name    | Data Type | Units | Sample Time | Interp | Sync  | Link to Plot         |
|---------|-----------|-------|-------------|--------|-------|----------------------|
| yout.Ww | double    |       | Continuous  | linear | union | <a href="#">Link</a> |
| yout.Vs | double    |       | Continuous  | linear | union | <a href="#">Link</a> |
| yout.Sd | double    |       | Continuous  | linear | union | <a href="#">Link</a> |
| slp     | double    |       | Continuous  | linear | union | <a href="#">Link</a> |

### Input Arguments

#### obj — Test report object

sltest.testmanager.TestResultReport object

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

#### result — Result set

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

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

### Output Arguments

#### baselineTable — Table

mlreportgen.dom.FormalTable object

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

### **See Also**

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

### **Topics**

“Export Test Results”

“Create and Run Test Cases with Scripts”

**Introduced in R2016a**

## genCoverageTable

**Class:** `sltest.testmanager.TestResultReport`

**Package:** `sltest.testmanager`

Generate coverage collection table



### Syntax

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

### Description

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

#### Aggregated Coverage Results

| Model Name                                                                                                                   | C1   | TBL  | Execution |
|------------------------------------------------------------------------------------------------------------------------------|------|------|-----------|
|  <a href="#">sldemo_absbrake</a>            | --   | 100% | 100%      |
|  <a href="#">sldemo_wheelspeed_absbrake</a> | 100% | --   | 100%      |

### Input Arguments

#### **obj** — Test report object

`sltest.testmanager.TestResultReport` object

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

#### **resultObj** — Result set

`sltest.testmanager.TestResult` object

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

### Output Arguments

#### **groupObj** — Group object

`mlreportgen.dom.Group` object

The coverage section generated by the method, returned as an `mlreportgen.dom.Group` object.

### See Also

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

#### Topics

“Export Test Results”

“Create and Run Test Cases with Scripts”

**Introduced in R2016a**

## genHyperLinkToToC

**Class:** `sltest.testmanager.TestResultReport`

**Package:** `sltest.testmanager`

Generate link to table of contents

### Syntax

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

### Description

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

The link appears at the end of sections:

[Back to Report Summary](#)

### Input Arguments

**obj — Test report**

`sltest.testmanager.TestResultReport` object

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

**indent — Link indentation**

character vector

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

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

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

Example: `'5mm'`

## Output Arguments

### **para — Paragraph**

`mlreportgen.dom.Paragraph` object

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

## See Also

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

### **Topics**

“Export Test Results”

“Create and Run Test Cases with Scripts”

**Introduced in R2016a**

## genIterationSettingTable

**Class:** sltest.testmanager.TestResultReport

**Package:** sltest.testmanager

Generate iteration settings table

### Syntax

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

### Description

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

#### Iteration Settings

##### Test Overrides

| Parameter Name | Value           |
|----------------|-----------------|
| ParameterSet   | Parameter Set 1 |

### Input Arguments

#### obj — Test report object

sltest.testmanager.TestResultReport object

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

#### result — Result set

sltest.testmanager.ReportUtility.ReportResultData object

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

### Output Arguments

#### groupObj — Group object

mlreportgen.dom.Group object

The iteration settings section generated by the method, returned as an mlreportgen.dom.Group object.

### See Also

sltest.testmanager.TestResultReport | mlreportgen.dom.Group

#### Topics

“Export Test Results”

“Create and Run Test Cases with Scripts”



**Introduced in R2016a**

## genMetadataBlockForTestResult

**Class:** `sltest.testmanager.TestResultReport`

**Package:** `sltest.testmanager`

Generate result metadata section

### Syntax

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

### Description

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

If called from `genTestSuiteResultBlock`, then this method also calls:

- `sgenTableRowsForResultMetaInfo`
- `genRequirementLinksTable`

If called from `genTestCaseResultBlock`, then this method also calls:

- `genTableRowsForResultMetaInfo`
- `genRequirementLinksTable`
- `genIterationSettingTable`

### Input Arguments

#### **obj** — Test report object

`sltest.testmanager.TestResultReport` object

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

#### **result** — Result set

`sltest.testmanager.ReportUtility.ReportResultData` object

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

#### **isTestSuiteResult** — Test suite indicator

`true` | `false`

Flag to indicate whether the test result metadata block is for a test suite result or not, specified as a Boolean, `true` or `false`.

### Output Arguments

#### **groupObj** — Group object

`mlreportgen.dom.Group` object

The result set section generated by the method, returned as an `mlreportgen.dom.Group` object.

## **See Also**

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

## **Topics**

“Export Test Results”

“Create and Run Test Cases with Scripts”

**Introduced in R2016a**

## genParameterOverridesTable

**Class:** `sltest.testmanager.TestResultReport`

**Package:** `sltest.testmanager`

Generate test case parameter overrides table

### Syntax

```
overridesTable = genParameterOverridesTable(obj,result,simIndex)
```

### Description

`overridesTable = genParameterOverridesTable(obj,result,simIndex)` generates a section for parameter overrides used in the test case.

#### Parameter Overrides

| Workspace Variable | Value | Source         | Model Element                                               |
|--------------------|-------|----------------|-------------------------------------------------------------|
| Parameter Set 1    |       |                |                                                             |
| Rr                 | 1.5   | base workspace | sldemo_absbrake/Rr, sldemo_absbrake/Vehicle speed (angular) |

### Input Arguments

#### **obj** — Test report object

`sltest.testmanager.TestResultReport` object

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

#### **result** — Result set

`sltest.testmanager.TestResult` object

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

#### **simIndex** — Simulation index

1 | 2

Simulation number that the test case parameter overrides table applies to, specified as an integer, 1 or 2. This setting applies to the simulation test case.

## Output Arguments

### **overridesTable** — Table

`mlreportgen.dom.FormalTable` object

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

## See Also

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

## Topics

“Export Test Results”

“Create and Run Test Cases with Scripts”

**Introduced in R2016a**

## genRequirementLinksTable

**Class:** sltest.testmanager.TestResultReport

**Package:** sltest.testmanager

Generate requirement links table

### Syntax

```
reqTable = genRequirementLinksTable(obj,resultObj,isTestSuiteResult)
```

### Description

reqTable = genRequirementLinksTable(obj,resultObj,isTestSuiteResult) generates a section for table of requirement links.

#### Test Case Requirements

Description: requirement

Document: <http://www.mathworks.com>

### Input Arguments

#### obj — Test report object

sltest.testmanager.TestResultReport object

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

#### resultObj — Result set

sltest.testmanager.TestResult object

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

#### isTestSuiteResult — Test suite indicator

true | false

Flag to indicate whether the requirement links table is for a test suite result or not, specified as a Boolean, true or false.

### Output Arguments

#### reqTable — Table

mlreportgen.dom.FormalTable object

The requirements links table generated by the method, returned as a mlreportgen.dom.FormalTable object.

### See Also

sltest.testmanager.TestResultReport | mlreportgen.dom.FormalTable

**Topics**

“Export Test Results”

“Create and Run Test Cases with Scripts”

**Introduced in R2016a**

## genResultSetBlock

**Class:** `sltest.testmanager.TestResultReport`

**Package:** `sltest.testmanager`

Generate results set section

### Syntax

```
groupObj = genResultSetBlock(obj,result)
```



### Description

`groupObj = genResultSetBlock(obj,result)` generates the results set section.

#### Results: 2015-Dec-01 11:46:00

Result Type:        Result Set  
 Parent:            None  
 Start Time:        2015-Dec-01 11:46:00  
 End Time:          2015-Dec-01 11:46:11  
 Outcome:           Total: 2, Passed: 2

#### Aggregated Coverage Results

| Model Name                                                                                                                     | C1   | TBL  | Execution |
|--------------------------------------------------------------------------------------------------------------------------------|------|------|-----------|
|  <a href="#">sldemo_absbrake</a>            | --   | 100% | 100%      |
|  <a href="#">sldemo_wheelspeed_absbrake</a> | 100% | --   | 100%      |

[Back to Report Summary](#)

This method also calls:

- `genTableRowsForResultMetaInfo`
- `genCoverageTable`
- `genHyperLinkToToC`

### Input Arguments

#### **obj** — Test report object

`sltest.testmanager.TestResultReport` object

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

#### **result** — Result set

`sltest.testmanager.ReportUtility.ReportResultData` object

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



## Output Arguments

### **groupObj** — Group object

`mlreportgen.dom.Group` object

The result set section generated by the method, returned as an `mlreportgen.dom.Group` object.

## See Also

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

## Topics

“Export Test Results”

“Create and Run Test Cases with Scripts”

**Introduced in R2016a**

## genRunBlockForTestCaseResult

**Class:** `sltest.testmanager.TestResultReport`

**Package:** `sltest.testmanager`

Generate test case configuration and results section

### Syntax

```
groupObj = genRunBlockForTestCaseResult(obj, run, runType, result, simIndex)
```

### Description

`groupObj = genRunBlockForTestCaseResult(obj, run, runType, result, simIndex)` generates a combined section for baseline data, simulation configuration, parameter overrides, simulation output, criteria comparison, and verify run data.

This method also calls:

- `genBaselineInfoTable`
- `genSimulationConfigurationTable`
- `genParameterOverridesTable`
- `genSignalSummaryTable`
- `plotOneSignalToFile`
- `genHyperLinkToToC`

### Input Arguments

**obj — Test report object**

`sltest.testmanager.TestResultReport` object

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

**run — Run data**

`Simulink.sdi.Run` object

Test case result run data, specified as a `Simulink.sdi.Run` object.

**runType — Run type**

`sltest.testmanager.RunTypes` object

The run type, specified as a `sltest.testmanager.RunTypes` object.

**result — Result**

`sltest.testmanager.ReportUtility.ReportResultData` object

Run result, specified as a `sltest.testmanager.ReportUtility.ReportResultData` object.

**simIndex — Simulation index**

1 | 2

Simulation number that the result applies to, specified as an integer, 1 or 2. This setting applies to the simulation test case.

## Output Arguments

### **groupObj** – Group object

`mlreportgen.dom.Group` object

The result set section generated by the method, returned as an `mlreportgen.dom.Group` object.

## See Also

`sltest.testmanager.TestResultReport` | `Simulink.sdi.Run` | `mlreportgen.dom.Group`

## Topics

“Export Test Results”

“Create and Run Test Cases with Scripts”

## Introduced in R2016a

## genSignalSummaryTable

**Class:** `sltest.testmanager.TestResultReport`

**Package:** `sltest.testmanager`

Generate signal output and comparison data

### Syntax

```
groupObj = genSignalSummaryTable(obj, signalList, isComparison, isSummaryTable)
```

### Description

`groupObj = genSignalSummaryTable(obj, signalList, isComparison, isSummaryTable)` generates a section for signal output and comparison data.

| Name    | Data Type | Units | Sample Time | Interp | Sync  | Link to Plot         |
|---------|-----------|-------|-------------|--------|-------|----------------------|
| yout.Vw | double    |       | Continuous  | linear | union | <a href="#">Link</a> |
| yout.Vs | double    |       | Continuous  | linear | union | <a href="#">Link</a> |
| yout.Sd | double    |       | Continuous  | linear | union | <a href="#">Link</a> |
| slp     | double    |       | Continuous  | linear | union | <a href="#">Link</a> |

### Input Arguments

#### **obj** — Test report object

`sltest.testmanager.TestResultReport` object

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

#### **signalList** — Signal list

`sltest.testmanager.ReportUtility.Signal` object

Signals in the summary table, specified as an array of `sltest.testmanager.ReportUtility.Signal` objects.

#### **isComparison** — Comparison indicator

`true` | `false`

Flag to indicate whether the signal is from a comparison run or not, specified as a Boolean, `true` or `false`.

#### **isSummaryTable** — Summary table indicator

`true` | `false`

Flag to indicate whether this is for a signal summary table in the report or an individual signal plot, specified as a Boolean. `true` for the signal summary table or `false` for an individual signal plot.

## Output Arguments

### **groupObj** — Group object

`mlreportgen.dom.Group` object

The signal summary section generated by the method, returned as an `mlreportgen.dom.Group` object.

## See Also

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

## Topics

“Export Test Results”

“Create and Run Test Cases with Scripts”

**Introduced in R2016a**

# genSimulationConfigurationTable

**Class:** `sltest.testmanager.TestResultReport`

**Package:** `sltest.testmanager`

Generate test case simulation configuration table

## Syntax

```
simCfgTable = genSimulationConfigurationTable(obj,result,simIndex)
```

## Description

`simCfgTable = genSimulationConfigurationTable(obj,result,simIndex)` generates a section for simulation configuration data used in the test case.

### Simulation

#### System Under Test Information

|                    |                                                |
|--------------------|------------------------------------------------|
| Model:             | sldemo_absbrake                                |
| Simulation Mode    | Normal                                         |
| Configuration Set: | Configuration                                  |
| Start Time:        | 0                                              |
| Stop Time:         | 14.635438874554231                             |
| Checksum:          | 1514901952 3523488043 2320696550<br>3824373991 |
| Simulink Version:  | 8.7                                            |

## Input Arguments

### **obj** — Test report object

`sltest.testmanager.TestResultReport` object

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

### **result** — Result set

`sltest.testmanager.ReportUtility.ReportResultData` object

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

### **simIndex** — Simulation index

1 | 2

Simulation number that the test case configuration table applies to, specified as an integer, 1 or 2. This setting applies to the simulation test case.

## Output Arguments

### **simCfgTable** — Table

mlreportgen.dom.FormalTable object

The test case simulation configuration table generated by the method, returned as a mlreportgen.dom.FormalTable object.

## See Also

sltest.testmanager.TestResultReport | mlreportgen.dom.FormalTable

## Topics

“Export Test Results”

“Create and Run Test Cases with Scripts”

**Introduced in R2016a**

## genTableRowsForResultMetaInfo

**Class:** sltest.testmanager.TestResultReport

**Package:** sltest.testmanager

Generate test result metadata table

### Syntax

```
rowList = genTableRowsForResultMetaInfo(obj, result)
```

### Description

`rowList = genTableRowsForResultMetaInfo(obj, result)` generates a section for test result metadata used in a result set, test file, test suite, test case, or test iteration.

#### Test Result Information

|              |                                               |
|--------------|-----------------------------------------------|
| Result Type: | Test Suite Result                             |
| Parent:      | <a href="#">test1</a>                         |
| Start Time:  | 2015-Dec-01 14:14:16                          |
| End Time:    | 2015-Dec-01 14:14:20                          |
| Outcome:     | Total: 2, <b>Passed: 1</b> , <b>Failed: 1</b> |

### Input Arguments

#### **obj** — Test report object

sltest.testmanager.TestResultReport object

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

#### **result** — Result set

sltest.testmanager.ReportUtility.ReportResultData object

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

### Output Arguments

#### **rowList** — Table row

mlreportgen.dom.TableRow object

The metadata information table row generated by the method, returned as a mlreportgen.dom.TableRow object.

### See Also

sltest.testmanager.TestResultReport | mlreportgen.dom.TableRow



**Topics**

“Export Test Results”

“Create and Run Test Cases with Scripts”

**Introduced in R2016a**

## genTestCaseResultBlock

**Class:** `sltest.testmanager.TestResultReport`

**Package:** `sltest.testmanager`

Generate test case result section

### Syntax

```
groupObj = genTestCaseResultBlock(obj,result)
```

### Description

`groupObj = genTestCaseResultBlock(obj,result)` generates a test case result section.

This method also calls:

- `genMetadataBlockForTestResult`
- `genCoverageTable`
- `genHyperLinkToToC`

### Input Arguments

**obj — Test report object**

`sltest.testmanager.TestResultReport` object

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

**result — Result set**

`sltest.testmanager.ReportUtility.ReportResultData` object

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

### Output Arguments

**groupObj — Group object**

`mlreportgen.dom.Group` object

The result set section generated by the method, returned as a `mlreportgen.dom.Group` object.

### See Also

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

### Topics

“Export Test Results”

“Create and Run Test Cases with Scripts”

**Introduced in R2016a**

# genTestSuiteResultBlock

**Class:** sltest.testmanager.TestResultReport

**Package:** sltest.testmanager

Generate test suite result section

## Syntax

```
groupObj = genTestSuiteResultBlock(obj, result)
```

## Description

groupObj = genTestSuiteResultBlock(obj, result) generates a test suite result section.

### New Test Suite 1

#### Test Result Information

|              |                            |
|--------------|----------------------------|
| Result Type: | Test Suite Result          |
| Parent:      | <a href="#">test1</a>      |
| Start Time:  | 2015-Dec-01 11:46:00       |
| End Time:    | 2015-Dec-01 11:46:11       |
| Outcome:     | Total: 2, <b>Passed: 2</b> |

#### Test Suite Information

|       |                  |
|-------|------------------|
| Name: | New Test Suite 1 |
|-------|------------------|

[Back to Report Summary](#)

This method also calls:

- genMetadataBlockForTestResult
- genCoverageTable
- genHyperLinkToToC

## Input Arguments

### obj — Test report object

sltest.testmanager.TestResultReport object

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

### result — Result set

sltest.testmanager.ReportUtility.ReportResultData object

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

## Output Arguments

### **groupObj** — Group

`mlreportgen.dom.Group` object

The result set section generated by the method, returned as a `mlreportgen.dom.Group` object.

## See Also

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

## Topics

“Export Test Results”

“Create and Run Test Cases with Scripts”

**Introduced in R2016a**

# getAssessmentsCallback

**Class:** `sltest.testmanager.TestCase`

**Package:** `sltest.testmanager`

Get test case assessments callback

## Syntax

```
callback = getAssessmentsCallback(tc)
```

## Description

`callback = getAssessmentsCallback(tc)` returns the assessments callback for the specified test case.

## Input Arguments

### **tc** — Test case

`sltest.testmanager.TestCase` object

Test case to which the assessment callback is associated, specified as an `sltest.testmanager.TestCase` object.

Data Types: `string`

## Output Arguments

### **callback** — Assessment callback

`string`

Assessments callback, returned as a string.

Data Types: `string`

## Examples

### Get the Callback for a Test Case

- 1 Specify the test file and get the test suite and test case object in the file. Assume there is only one test suite and one test case in the file. Then, use `getAssessmentsCallback` to get the callback code.

```
tf = sltest.testmanager.TestFile('myTestFile.mldatx');
ts = tf.getTestSuites();
tc = ts.getTestCases();
```

- 2 Get the assessment callback for the test case object.

```
tc.getAssessmentsCallback()
```

**See Also**

setAssessmentsCallback

**Topics**

“Logical and Temporal Assessment Syntax”

**Introduced in R2020b**

# getBaselineCriteria

**Class:** sltest.testmanager.TestCase

**Package:** sltest.testmanager

Get baseline criteria

## Syntax

```
baselines = getBaselineCriteria(tc)
```

## Description

`baselines = getBaselineCriteria(tc)` gets the baseline criteria sets in a test case and returns them as an array of baseline criteria objects, `sltest.testmanager.BaselineCriteria`.

## Input Arguments

**tc** — Baseline test case

`sltest.testmanager.TestCase` object

Baseline test case that you want to get baseline criteria from, specified as a `sltest.testmanager.TestCase` object.

## Output Arguments

**baselines** — Baseline criteria object

`sltest.testmanager.BaselineCriteria` object array

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

## Examples

### Get Baseline Criteria

```
% Open the model for this example
openExample('sldemo_absbrake');

% Create the test file, test suite, and test case structure
tf = sltest.testmanager.TestFile('API Test File');
ts = createTestSuite(tf,'API Test Suite');
tc = createTestCase(ts,'baseline','Baseline API Test Case');

% Remove the default test suite
tsDel = getTestSuiteByName(tf,'New Test Suite 1');
remove(tsDel);

% Assign the system under test to the test case
setProperty(tc,'Model','sldemo_absbrake');
```

```
% Capture the baseline criteria
baseline = captureBaselineCriteria(tc,'baseline_API.mat',true);

% Set the baseline criteria tolerance for one signal
sc = getSignalCriteria(baseline);
sc(1).AbsTol = 9;

% Get baseline criteria
baselineOut = getBaselineCriteria(tc);
```

## **See Also**

### **Topics**

“Create and Run Test Cases with Scripts”

**Introduced in R2015b**



# getBaselineRun

**Class:** sltest.testmanager.TestCaseResult

**Package:** sltest.testmanager

Get test case baseline dataset

## Syntax

```
baseline = getBaselineRun(result)
```

## Description

`baseline = getBaselineRun(result)` gets the baseline dataset used in a test case, which belongs to the test case results object. The baseline dataset is saved with the test case result only if the **Save baseline data in test result** check box is selected in the test case under the **Baseline Criteria** section.

To record the baseline data in the test case result, you must set the `SaveBaselineRunInTestResult` test case property to `true`:

```
setProperty(testcase, 'SaveBaselineRunInTestResult', true);
```

## Input Arguments

**result** — Test case result

sltest.testmanager.TestCaseResult object

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

## Output Arguments

**baseline** — Baseline dataset

Simulink.sdi.Run object

Test case baseline dataset, returned as a Simulink.sdi.Run object. If the **Save baseline data in test result** check box is not selected in the test case, then the function returns an empty array.

## Examples

### Get Baseline Data From Test Case

```
% Open the model for this example
openExample('sldemo_absbrake');

% Create the test file, test suite, and test case structure
tf = sltest.testmanager.TestFile('API Test File');
ts = createTestSuite(tf, 'API Test Suite');
```

```
tc = createTestCase(ts,'baseline','Baseline API Test Case');

% Remove the default test suite
tsDel = getTestSuiteByName(tf,'New Test Suite 1');
remove(tsDel);

% Assign the system under test to the test case
setProperty(tc,'Model','sldemo_absbrake');

% Capture the baseline criteria and record baseline
baseline = captureBaselineCriteria(tc,'baseline_API.mat',true);
setProperty(tc,'SaveBaselineRunInTestResult',true);

% Test a new model parameter by overriding it in the test case
% parameter set
ps = addParameterSet(tc,'Name','API Parameter Set');
po = addParameterOverride(ps,'m',55);

% Set the baseline criteria tolerance for one signal
sc = getSignalCriteria(baseline);
sc(1).AbsTol = 9;

% Run the test case and return an object with results data
resultsObj = run(tc);

% Get test case result
tcr = getTestCaseResults(resultsObj);

% Get the baseline run dataset
baselineOut = getBaselineRun(tcr);
```

## See Also

Simulink.sdi.Run

## Topics

“Create and Run Test Cases with Scripts”

**Introduced in R2016a**

# getTestCase

**Class:** `sltest.testmanager.TestCaseResult`

**Package:** `sltest.testmanager`

Get test case that produced result

## Syntax

```
tc = getTestCase(tcresult)
```

## Description

`tc = getTestCase(tcresult)` returns the test case that produced the test case results, `tc`.

## Input Arguments

**tcresult** — Test case result

`sltest.testmanager.TestCaseResult` object

Test case result from a test case run, specified as an `sltest.testmanager.TestCaseResult` object.

## Output Arguments

**tc** — Test case

`sltest.testmanager.TestCase` object

Test case that produced the test case results, returned as an `sltest.testmanager.TestCase` object.

## Examples

### Obtain Test Case That Produced Result

This example shows how to create a test file, test suite, and simulation test case programmatically. The test case runs on the `HeatPumpScenario` model and uses `getTestCase` to obtain the test case that produced the test results.

Clear existing test files from the Test Manager.

```
sltest.testmanager.clear;
```

Create test file, test suite, and test case.

```
tf = sltest.testmanager.TestFile('Test File 1');
ts = createTestSuite(tf, 'Test Suite 1');
tc = createTestCase(ts, 'simulation', 'Test Case 1');
```

Remove default test suite so that only the created test suite is used.

```
tsDel = tf.getTestSuites();
remove(tsDel(1));
```

Assign the system under test to the test case, and run the test.

```
setProperty(tc, 'Model', 'HeatPumpScenario');
tresult = run(tc);
```

**Obtain the test case results.**

```
tresultobj = getTestCaseResults(tc);
```

Obtain the test case that produced the results.

```
tobj = getTestCase(tresultobj);
```

**See Also**

`getTestCaseResults`

**Introduced in R2019b**

# getBaselineRun

**Class:** sltest.testmanager.TestIterationResult

**Package:** sltest.testmanager

Get test iteration baseline dataset

## Syntax

```
baseline = getBaselineRun(resultObj)
```

## Description

`baseline = getBaselineRun(resultObj)` gets the baseline dataset used in a test iteration, which belongs to the test iteration results object. The baseline dataset is saved with the test iteration result only if the **Save baseline data in test result** check box is selected in the parent test case under the **Baseline Criteria** section.

## Input Arguments

**resultObj** — Test iteration result object

Test iteration results object to get baseline dataset from, specified as a `sltest.testmanager.TestIterationResult` object.

## Output Arguments

**baseline** — Baseline dataset object

Test iteration baseline dataset, returned as a `Simulink.sdi.Run` object. If the **Save baseline data in test result** check box is not selected in the parent test case, then the function returns an empty array.

## See Also

`Simulink.sdi.Run`

## Topics

“Create and Run Test Cases with Scripts”

**Introduced in R2016a**

## getCleanupPlots

**Class:** `sltest.testmanager.TestFileResult`

**Package:** `sltest.testmanager`

Get plots from cleanup callbacks

### Syntax

```
figs = getCleanupPlots(result)
```

### Description

`figs = getCleanupPlots(result)` returns figure handles of plots generated from the cleanup callbacks of the test file associated with the results. Figures returned using this method are not visible. To see the plots, set the figure handle `Visible` property to `'on'`.

### Input Arguments

**result** — Test file results

`sltest.testmanager.TestFileResult` object

Test file results to get cleanup plot figure handles from, specified as a `sltest.testmanager.TestFileResult` object.

### Output Arguments

**figs** — Figures from test file cleanup callbacks

array of figure handles

Figures from test file cleanup callbacks, returned as an array of figure handles.

### Examples

#### Get Figure Handles from Test File Results

```
% Open the model for this example
openExample('sldemo_absbrake');

Create the test file, suite, and case
tf = sltest.testmanager.TestFile('Cleanup Plots');
ts = createTestSuite(tf,'Cleanup Plots Test Suite');
tc = createTestCase(ts,'baseline','Cleanup Plots Test Case');

% Remove the default test suite
tsDel = getTestSuiteByName(tf,'New Test Suite 1');
remove(tsDel);

% Create a plot in the test file cleanup callback
setProperty(tf,'CleanupCallback','a = [1,2,3]; f = figure; plot(a);');
```

```
% Set option to save figures
opt = getOptions(tf);
opt.SaveFigures = true;

% Assign the system under test to the test case
setProperty(tc, 'Model', 'sldemo_absbrake');

% Run the test and capture results
resultset = run(tf);
tfr = getTestFileResults(resultset);

% Get the cleanup plot figure handles
figs = tfr.getCleanupPlots;
```

## See Also

[sltest.testmanager.TestFile](#) | [sltest.testmanager.TestFileResult](#) | [setProperty](#) | [sltest.testmanager.Options](#)

## Topics

“Create, Store, and Open MATLAB Figures”  
“Export Test Results”

## Introduced in R2017a

## getCleanupPlots

**Class:** `sltest.testmanager.TestSuiteResult`

**Package:** `sltest.testmanager`

Get plots from cleanup callbacks of test suite

### Syntax

```
figs = getCleanupPlots(result)
```

### Description

`figs = getCleanupPlots(result)` returns figure handles of plots from the cleanup callbacks of the test suite associated with the results. Figures returned using this method are not visible. To see the plots, set the figure handle `Visible` property to `'on'`.

### Input Arguments

**result** — Test suite results

`sltest.testmanager.TestSuiteResult` object

Test suite results to get cleanup plot figure handles from, specified as an `sltest.testmanager.TestSuiteResult` object.

### Output Arguments

**figs** — Figures from test suite cleanup callbacks

array of figure handles

Figures from test suite cleanup callbacks, returned as an array of figure handles.

## Examples

### Get Figure Handles from Test Suite Results

```
% Open the model for this example
openExample('sldemo_absbrake');

% Create the test file, suite, and case
tf = sltest.testmanager.TestFile('Test Suite Cleanup Plots');
ts = createTestSuite(tf,'Cleanup Plots Test Suite');
tc = createTestCase(ts,'baseline','Cleanup Plots Test Case');

% Remove the default test suite
tsDel = getTestSuiteByName(tf,'New Test Suite 1');
remove(tsDel);

% Create a plot in the test suite cleanup callback
setProperty(ts,'CleanupCallback','a = [1,2,3]; f = figure; plot(a);');
```



```
% Set option to save figures
opt = getOptions(tf);
opt.SaveFigures = true;

% Assign the system under test to the test case
setProperty(tc, 'Model', 'sldemo_absbrake');

% Run the test and capture results
resultset = run(tf);
tfr = getTestFileResults(resultset);
tsr = getTestSuiteResults(tfr);

% Get the cleanup plot figure handles
figs = tsr.getCleanupPlots;
```

## See Also

[sltest.testmanager.TestSuite](#) | [sltest.testmanager.TestSuiteResult](#) | [setProperty \(Test Suite\)](#) | [sltest.testmanager.Options](#)

## Topics

“Create, Store, and Open MATLAB Figures”

“Export Test Results”

**Introduced in R2017a**

## getComparisonResult

**Class:** `sltest.testmanager.TestCaseResult`

**Package:** `sltest.testmanager`

Get test data comparison result

### Syntax

```
cr = getComparisonResult(tcr)
```

### Description

`cr = getComparisonResult(tcr)` returns baseline or equivalence data comparison results `cr` from the `sltest.testmanager.TestCaseResult` object `tcr`.

### Input Arguments

**tcr — Test case result**

`sltest.testmanager.TestCaseResult` object

Test case result to get baseline or equivalence data results from, specified as a `sltest.testmanager.TestCaseResult` object.

### Output Arguments

**cr — Data comparison result**

`sltest.testmanager.ComparisonRunResult` object

Result of the baseline or equivalence data comparison, specified as a `sltest.testmanager.ComparisonRunResult` object.

### Examples

#### Get Comparison Results of a Baseline Test

This example shows how to programmatically get the comparison results of the second iteration of a baseline test case.

1. Get the path to the test file, then run the test file.

```
extf = 'sltestTestCaseRealTimeReuseExample.mldatx';
tf = sltest.testmanager.TestFile(extf);
ro = run(tf);
```

2. Get the test iteration results.

```
tfr = getTestFileResults(ro);
tsr = getTestSuiteResults(tfr);
tcr = getTestCaseResults(tsr);
tir = getIterationResults(tcr);
```

3. Get the comparison run result of iteration 2.

```
cr2 = getComparisonResult(tir(2))
```

```
cr2 =
```

```
ComparisonRunResult with properties:
```

```
Outcome: Passed
```

4. Get the comparison signal result of the run result.

```
cr2sig = getComparisonSignalResults(cr2)
```

```
cr2sig =
```

```
1×2 ComparisonSignalResult array with properties:
```

```
Outcome
Baseline
ComparedTo
Difference
```

5. Clear the results and the Test Manager.

```
sltest.testmanager.clearResults;
sltest.testmanager.clear;
sltest.testmanager.close;
```

## See Also

### Topics

“Compare Model Output to Baseline Data”  
“Test Iterations”

**Introduced in R2017b**

## getComparisonResult

**Class:** `sltest.testmanager.TestIterationResult`

**Package:** `sltest.testmanager`

Get test data comparison result

### Syntax

```
cr = getComparisonResult(tir)
```

### Description

`cr = getComparisonResult(tir)` returns baseline or equivalence data comparison results `cr` from the `sltest.testmanager.TestIterationResult` object `tir`.

### Input Arguments

**tir — Test iteration result**

`sltest.testmanager.TestIterationResult` object

Test iteration result to get baseline or equivalence data results from, specified as an `sltest.testmanager.TestIterationResult` object.

### Output Arguments

**cr — Data comparison result**

`sltest.testmanager.TestIterationResult` object | object

Result of the baseline or equivalence data comparison, specified as an `sltest.testmanager.ComparisonRunResult` object.

### Examples

#### Get Comparison Results of a Baseline Test

This example shows how to programmatically get the comparison results of the second iteration of a baseline test case.

1. Get the path to the test file, then run the test file.

```
extf = 'sltestTestCaseRealTimeReuseExample.mldatx';
tf = sltest.testmanager.TestFile(extf);
ro = run(tf);
```

2. Get the test iteration results.

```
tfr = getTestFileResults(ro);
tsr = getTestSuiteResults(tfr);
tcr = getTestCaseResults(tsr);
tir = getIterationResults(tcr);
```

3. Get the comparison run result of iteration 2.

```
cr2 = getComparisonResult(tir(2))
```

```
cr2 =
```

```
ComparisonRunResult with properties:
```

```
Outcome: Passed
```

4. Get the comparison signal result of the run result.

```
cr2sig = getComparisonSignalResults(cr2)
```

```
cr2sig =
```

```
1×2 ComparisonSignalResult array with properties:
```

```
Outcome
Baseline
ComparedTo
Difference
```

5. Clear the results and the Test Manager.

```
sltest.testmanager.clearResults;
sltest.testmanager.clear;
sltest.testmanager.close;
```

## See Also

```
sltest.testmanager.ComparisonRunResult
```

**Introduced in R2017b**

## getTestIteration

**Class:** `sltest.testmanager.TestIterationResult`

**Package:** `sltest.testmanager`

Get test iteration that produced result

### Syntax

```
ti = getTestIteration(ti_result)
```

### Description

`ti = getTestIteration(ti_result)` returns the test iteration that produced the test iteration results, `ti_result`.

### Input Arguments

**ti\_result — Test iteration result**

`sltest.testmanager.TestIterationResult` object

Test iteration result for a single iteration, specified as an `sltest.testmanager.TestIterationResult` object.

### Output Arguments

**ti — Test iteration**

`sltest.testmanager.TestIteration` object

Test iteration that produced the test iteration results, returned as an `sltest.testmanager.TestIteration` object.

### See Also

`getIterationResults`

**Introduced in R2019b**

# getComparisonRun

**Class:** sltest.testmanager.TestCaseResult

**Package:** sltest.testmanager

Get test case signal comparison results

## Syntax

```
run = getComparisonRun(result)
```

## Description

`run = getComparisonRun(result)` gets the test case comparison results that belong to the test case results object. The results are output to a `Simulink.sdi.Run` object, which contains signal data for each comparison, tolerance, and difference result.

## Input Arguments

**result — Test case result**

`sltest.testmanager.TestCaseResult` object

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

## Output Arguments

**run — Comparison results**

`Simulink.sdi.Run` object

Test case comparison signal results, returned as a `Simulink.sdi.Run` object.

## Examples

### Get Comparison Data From Test Case

```
% Open the model for this example
openExample('sldemo_absbrake');

% Create the test file, test suite, and test case structure
tf = sltest.testmanager.TestFile('API Test File');
ts = createTestSuite(tf,'API Test Suite');
tc = createTestCase(ts,'baseline','Baseline API Test Case');

% Remove the default test suite
tsDel = getTestSuiteByName(tf,'New Test Suite 1');
remove(tsDel);

% Assign the system under test to the test case
setProperty(tc,'Model','sldemo_absbrake');
```

```
% Capture the baseline criteria and record baseline
baseline = captureBaselineCriteria(tc,'baseline_API.mat',true);

% Test a new model parameter by overriding it in the test case
% parameter set
ps = addParameterSet(tc,'Name','API Parameter Set');
po = addParameterOverride(ps,'m',55);

% Set the baseline criteria tolerance for one signal
sc = getSignalCriteria(baseline);
sc(1).AbsTol = 9;

% Run the test case and return an object with results data
resultsObj = run(tc);

% Get test case result
tcr = getTestCaseResults(resultsObj);

% Get the baseline run dataset
compOut = getComparisonRun(tcr);
```

## See Also

Simulink.sdi.Run

## Topics

“Create and Run Test Cases with Scripts”

**Introduced in R2015a**



# getComparisonRun

**Class:** `sltest.testmanager.TestIterationResult`

**Package:** `sltest.testmanager`

Get test iteration signal comparison results

## Syntax

```
run = getComparisonRun(result)
```

## Description

`run = getComparisonRun(result)` gets the test iteration comparison results that belong to the test iteration results object. The results are output to a `Simulink.sdi.Run` object, which contains signal data for each comparison, tolerance, and difference result.

## Input Arguments

**result — Test iteration result**

`sltest.testmanager.TestIterationResult` object

Test iteration results to get results from, specified as a `sltest.testmanager.TestIterationResult` object.

## Output Arguments

**run — Comparison results**

`Simulink.sdi.Run` object

Test iteration comparison results, returned as a `Simulink.sdi.Run` object.

## See Also

`sltest.testmanager.TestIterationResult`

## Topics

“Create and Run Test Cases with Scripts”

**Introduced in R2016a**

## getComparisonSignalResults

**Class:** `sltest.testmanager.ComparisonRunResult`

**Package:** `sltest.testmanager`

Get test signal comparison result from comparison run result

### Syntax

```
csr = getComparisonSignalResults(cr)
```

### Description

`csr = getComparisonSignalResults(cr)` returns baseline or equivalence signal comparison results `csr` as a `sltest.testmanager.ComparisonSignalResult` object from the `sltest.testmanager.ComparisonRunResult` object `cr`.

### Input Arguments

**cr — Overall data comparison result**

`sltest.testmanager.ComparisonRunResult` object

Overall result of a baseline or equivalence data comparison, specified as a `sltest.testmanager.ComparisonRunResult` object. You get signal comparison results from the overall data comparison result.

### Output Arguments

**csr — Signal data comparison result**

`sltest.testmanager.ComparisonSignalResult` object

Result of the baseline or equivalence data comparison, specified as a `sltest.testmanager.ComparisonSignalResult` object.

### Examples

#### Get Comparison Results of a Baseline Test

This example shows how to programmatically get the comparison results of the second iteration of a baseline test case.

1. Get the path to the test file, then run the test file.

```
extf = 'sltestTestCaseRealTimeReuseExample.mldatx';
tf = sltest.testmanager.TestFile(extf);
ro = run(tf);
```

2. Get the test iteration results.

```
tfr = getTestFileResults(ro);
tsr = getTestSuiteResults(tfr);
```

```
tcr = getTestCaseResults(tsr);
tir = getIterationResults(tcr);
```

3. Get the comparison run result of iteration 2.

```
cr2 = getComparisonResult(tir(2))
```

```
cr2 =
```

```
ComparisonRunResult with properties:
```

```
Outcome: Passed
```

4. Get the comparison signal result of the run result.

```
cr2sig = getComparisonSignalResults(cr2)
```

```
cr2sig =
```

```
1x2 ComparisonSignalResult array with properties:
```

```
Outcome
Baseline
ComparedTo
Difference
```

5. Clear the results and the Test Manager.

```
sltest.testmanager.clearResults;
sltest.testmanager.clear;
sltest.testmanager.close;
```

## See Also

**Introduced in R2017b**

## clearSimulationData

**Class:** `sltest.testmanager.ResultSet`

**Package:** `sltest.testmanager`

Delete specified simulation test results in result set objects

### Syntax

```
bytes = clearSimulationData(rs, results)
```

### Description

`bytes = clearSimulationData(rs, results)` removes the simulation test results specified by `results` from the result set object array, `rs`.

### Input Arguments

**rs — Results sets**

`sltest.testmanager.ResultSet` object array

Results sets, specified as a `sltest.testmanager.ResultSet` object array.

**results — Test results to remove**

"Baseline" | "Comparison" | "InputRun" | ...

Test results to remove from the results set, specified by one of the following options:

- "Baseline": Baseline run data. See "SaveBaselineRunInTestResult" on page 3-0 .
- "Comparison": Passed baseline or equivalence criteria comparison results. See "Compare Model Output to Baseline Data".
- "InputRun": External inputs. See "Use External Excel or MAT-File Data in Test Cases".
- "Simulation": Simulation output results. See "Capture Simulation Data in a Test Case".
- "VerifyResults": Passed or untested verify statements. See "Assess Model Simulation Using verify Statements".

### Output Arguments

**bytes — Number of bytes**

positive integer

Number of bytes deleted, returned as a positive integer.

Data Types: `uint64`

### Examples

## Remove Baseline Criteria Comparison Results

Open the model for this example.

```
openExample('sldemo_absbrake');
```

Create the test file, test suite, and test case structure.

```
tf = sltest.testmanager.TestFile("API Test File");
ts = createTestSuite(tf, "API Test Suite");
tc = createTestCase(ts, "baseline", "Baseline API Test Case");
```

Remove the default test suite.

```
tsDel = getTestSuiteByName(tf, "New Test Suite 1");
remove(tsDel);
```

Assign the system under test to the test case.

```
setProperty(tc, "Model", "sldemo_absbrake");
```

Capture the baseline criteria and set the tolerance.

```
baseline = captureBaselineCriteria(tc, "baseline_API.mat", true);
baseline.AbsTol = 15;
```

Change the model. Set the value of the Constant block, Desired relative slip, to 0.22.

```
set_param("sldemo_absbrake/Desired relative slip", Value="0.22");
```

Run the test case and return a `sltest.testmanager.ResultSet` object with results data.

```
resultsObj = run(tc);
```

Delete the passed baseline criteria comparison results.

```
clearSimulationData(resultsObj, "Comparison");
```

## See Also

`remove`

## Topics

“Specify Test Properties in the Test Manager”

## Introduced in R2022a

## getCoverageResults

**Class:** `sltest.testmanager.ResultSet`

**Package:** `sltest.testmanager`

Get coverage results

### Syntax

```
covResult = getCoverageResults(result)
covResult = getCoverageResults(result,model)
```

### Description

`covResult = getCoverageResults(result)` gets the coverage results that belong to the result set object.

`covResult = getCoverageResults(result,model)` gets the coverage results that belong to the result set object and the specified model.

### Input Arguments

**result — Result set**

`sltest.testmanager.ResultSet` object

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

**model — Model name**

character vector

Name of a model within the set of coverage results, specified as a character vector.

### Output Arguments

**covResult — Coverage results**

object array

Coverage results contained in the result set, returned as an array of `cvdata` objects. For more information on `cvdata` objects, see `cvdata`.

### Examples

**Get Result Set Coverage Results**

```
% Open the model for this example
openExample('sldemo_autotrans');

% Create the test file, test suite, and test case structure
```

```
tf = sltest.testmanager.TestFile('API Test File');
ts = createTestSuite(tf, 'API Test Suite');
tc = createTestCase(ts, 'simulation', 'Coverage Test Case');

% Remove the default test suite
tsDel = getTestSuiteByName(tf, 'New Test Suite 1');
remove(tsDel);

% Assign the system under test to the test case
setProperty(tc, 'Model', 'sldemo_autotrans');

% Turn on coverage settings at test-file level
cov = getCoverageSettings(tf);
cov.RecordCoverage = true;

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

% Run the test case and return an object with results data
ro = run(tf);

% Get the coverage results
cr = getCoverageResults(ro);
```

## See Also

`sltest.testmanager.CoverageSettings` | `cvdata`

## Topics

“Create and Run Test Cases with Scripts”

“Collect Coverage in Tests”

**Introduced in R2016a**

## remove

**Class:** `sltest.testmanager.ResultSet`

**Package:** `sltest.testmanager`

Remove result set

### Syntax

```
remove(result)
```

### Description

`remove(result)` removes `result`, a `sltest.testmanager.ResultSet` object. If the Test Manager is visible, the corresponding results are removed from the **Results and Artifacts** pane.

### Input Arguments

**result — Results set**

`sltest.testmanager.ResultSet` object

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

### Examples

#### Remove a Test Result

This example shows how to run a test and remove the result.

#### Create a Test File

```
tf = sltest.testmanager.TestFile('f14ParameterSweepTest.mldatx');
```

#### Run the Test

```
result = run(tf)
```

```
result =
```

```
 ResultSet with properties:
```

```
 Name: 'Results: 2022-Feb-27 02:53:58'
 Outcome: Failed
 NumPassed: 0
 NumFailed: 25
 NumDisabled: 0
 NumIncomplete: 0
 NumTotal: 25
 NumTestCaseResults: 0
 NumTestSuiteResults: 0
 NumTestFileResults: 1
 StartTime: 27-Feb-2022 02:54:05
```



```
StopTime: 27-Feb-2022 02:54:07
Duration: 1.759 sec
Description: []
CoverageResults: []
UserData: []
```

### **Remove the Result**

```
remove(result)
```

### **See Also**

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

### **Introduced in R2019a**

## getCoverageResults

**Class:** `sltest.testmanager.TestCaseResult`

**Package:** `sltest.testmanager`

Get coverage results

### Syntax

```
covResult = getCoverageResults(result)
covResult = getCoverageResults(result,model)
```

### Description

`covResult = getCoverageResults(result)` gets the coverage results that belong to the test case results object.

`covResult = getCoverageResults(result,model)` gets the coverage results that belong to the test case results object and the specified model.

### Input Arguments

#### **result** — Test case result

`sltest.testmanager.TestCaseResult` object

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

#### **model** — Model name

character vector

Name of a model within the set of coverage results, specified as a character vector.

### Output Arguments

#### **covResult** — Coverage results

object array

Coverage results contained in the test case result, returned as an array of `cvdata` objects. For more information on `cvdata` objects, see `cvdata`.

### Examples

#### **Get Test Suite Coverage Results**

```
% Open the model for this example
openExample('sldemo_autotrans');

% Create the test file, test suite, and test case structure
```

```
tf = sltest.testmanager.TestFile('API Test File');
ts = createTestSuite(tf, 'API Test Suite');
tc = createTestCase(ts, 'simulation', 'Coverage Test Case');

% Remove the default test suite
tsDel = getTestSuiteByName(tf, 'New Test Suite 1');
remove(tsDel);

% Assign the system under test to the test case
setProperty(tc, 'Model', 'sldemo_autotrans');

% Turn on coverage settings at test-file level
cov = getCoverageSettings(tf);
cov.RecordCoverage = true;

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

% Run the test case and return an object with results data
rs = run(tf);

% Get the coverage results
cr = getCoverageResults(rs);
```

## See Also

`sltest.testmanager.CoverageSettings` | `cvdata`

## Topics

“Create and Run Test Cases with Scripts”  
“Collect Coverage in Tests”

**Introduced in R2016a**

## getCoverageResults

**Class:** `sltest.testmanager.TestFileResult`

**Package:** `sltest.testmanager`

Get coverage results

### Syntax

```
covResult = getCoverageResults(result)
covResult = getCoverageResults(result,model)
```

### Description

`covResult = getCoverageResults(result)` gets the coverage results that belong to the test file results object.

`covResult = getCoverageResults(result,model)` gets the coverage results that belong to the test file results object and the specified model.

### Input Arguments

#### **result** — Test file result

`sltest.testmanager.ResultSet` object

Test file results to get coverage results from, specified as a `sltest.testmanager.TestFileResult` object.

#### **model** — Model name

character vector

Name of a model within the set of coverage results, specified as a character vector.

### Output Arguments

#### **covResult** — Coverage results

object array

Coverage results contained in the test file result, returned as an array of `cvdata` objects. For more information on `cvdata` objects, see `cvdata`.

### Examples

#### **Get Test File Coverage Results**

```
% Open the model for this example
openExample('sldemo_autotrans');

% Create the test file, test suite, and test case structure
```

```
tf = sltest.testmanager.TestFile('API Test File');
ts = createTestSuite(tf, 'API Test Suite');
tc = createTestCase(ts, 'simulation', 'Coverage Test Case');

% Remove the default test suite
tsDel = getTestSuiteByName(tf, 'New Test Suite 1');
remove(tsDel);

% Assign the system under test to the test case
setProperty(tc, 'Model', 'sldemo_autotrans');

% Turn on coverage settings at test-file level
cov = getCoverageSettings(tf);
cov.RecordCoverage = true;

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

% Run the test case and return an object with results data
rs = run(tf);

% Get the coverage results
cr = getCoverageResults(rs);
```

## See Also

`sltest.testmanager.CoverageSettings` | `cvdata`

## Topics

“Create and Run Test Cases with Scripts”  
“Collect Coverage in Tests”

**Introduced in R2016a**

## getCoverageResults

**Class:** `sltest.testmanager.TestIterationResult`

**Package:** `sltest.testmanager`

Get coverage results

### Syntax

```
covResult = getCoverageResults(resultObj)
covResult = getCoverageResults(resultObj,model)
```

### Description

`covResult = getCoverageResults(resultObj)` gets the coverage results that belong to the test iteration results object.

`covResult = getCoverageResults(resultObj,model)` gets the coverage results that belong to the test iteration results object and the specified model.

### Input Arguments

**resultObj — Test iteration result**

`sltest.testmanager.TestIterationResult` object

Test iteration results object to get coverage results from, specified as a `sltest.testmanager.TestIterationResult` object.

**model — Model name**

character vector

Name of a model within the set of coverage results, specified as a character vector.

Example: `'myModel'`

### Output Arguments

**covResult — Coverage results**

object array

Coverage results contained in the test iteration result, returned as an array of `cvdata` objects. For more information on `cvdata` objects, see `cvdata`.

### See Also

`sltest.testmanager.CoverageSettings` | `cvdata`

### Topics

“Create and Run Test Cases with Scripts”

“Collect Coverage in Tests”

**Introduced in R2016a**

## getCoverageResults

**Class:** `sltest.testmanager.TestSuiteResult`

**Package:** `sltest.testmanager`

Get coverage results

### Syntax

```
covResult = getCoverageResults(result)
covResult = getCoverageResults(result,model)
```

### Description

`covResult = getCoverageResults(result)` gets the coverage results that belong to the test suite results object.

`covResult = getCoverageResults(result,model)` gets the coverage results that belong to the test suite results object and the specified model.

### Input Arguments

#### **result** — Test suite result

`sltest.testmanager.TestSuiteResult` object

Test suite results to get coverage results from, specified as a `sltest.testmanager.TestSuiteResult` object.

#### **model** — Model name

character vector

Name of a model within the set of coverage results, specified as a character vector.

### Output Arguments

#### **covResult** — Coverage results

object array

Coverage results contained in the test suite result, returned as an array of `cvdata` objects. For more information on `cvdata` objects, see `cvdata`.

### Examples

#### **Get Test Suite Coverage Results**

```
% Open the model for this example
openExample('sldemo_autotrans');
```

```
% Create the test file, test suite, and test case structure
```



```
tf = sltest.testmanager.TestFile('API Test File');
ts = createTestSuite(tf, 'API Test Suite');
tc = createTestCase(ts, 'simulation', 'Coverage Test Case');

% Remove the default test suite
tsDel = getTestSuiteByName(tf, 'New Test Suite 1');
remove(tsDel);

% Assign the system under test to the test case
setProperty(tc, 'Model', 'sldemo_autotrans');

% Turn on coverage settings at test-file level
cov = getCoverageSettings(tf);
cov.RecordCoverage = true;

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

% Run the test case and return an object with results data
rs = run(tf);

% Get the coverage results
cr = getCoverageResults(rs);
```

## See Also

`sltest.testmanager.CoverageSettings` | `cvdata`

## Topics

“Create and Run Test Cases with Scripts”

“Collect Coverage in Tests”

**Introduced in R2016a**

## getCoverageSettings

**Class:** `sltest.testmanager.TestCase`

**Package:** `sltest.testmanager`

Get coverage settings

### Syntax

```
covSettings = getCoverageSettings(tc)
```

### Description

`covSettings = getCoverageSettings(tc)` gets the coverage settings for a test case and returns an `sltest.testmanager.CoverageSettings` object.

### Input Arguments

**tc** — Test case

`sltest.testmanager.TestCase` object

Test case to get coverage settings from, specified as an `sltest.testmanager.TestCase` object.

### Output Arguments

**covSettings** — Coverage settings

`sltest.testmanager.CoverageSettings` object

Coverage settings for the test case, returned as an `sltest.testmanager.CoverageSettings` object. For information on coverage metrics, see `sltest.plugins.coverage.CoverageMetrics`.

### Examples

#### Get Coverage Settings

```
% Open the model for this example
openExample('sldemo_autotrans');

% Create the test file, test suite, and test case structure
tf = sltest.testmanager.TestFile('API Test File');
ts = createTestSuite(tf, 'API Test Suite');
tc = createTestCase(ts, 'simulation', 'Coverage Test Case');

% Remove the default test suite
tsDel = getTestSuiteByName(tf, 'New Test Suite 1');
remove(tsDel);

% Assign the system under test to the test case
setProperty(tc, 'Model', 'sldemo_autotrans');
```

```
% Turn on coverage settings at test-file level
cov = getCoverageSettings(tf);
cov.RecordCoverage = true;

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

% Get and check coverage settings
covSettings = getCoverageSettings(tc);
```

## See Also

`sltest.testmanager.CoverageSettings` | `sltest.plugins.coverage.CoverageMetrics`

## Topics

“Create and Run Test Cases with Scripts”

## Introduced in R2016a

## getCoverageSettings

**Class:** `sltest.testmanager.TestFile`

**Package:** `sltest.testmanager`

Get coverage settings

### Syntax

```
covSettings = getCoverageSettings(tf)
```

### Description

`covSettings = getCoverageSettings(tf)` gets the coverage settings for a test file and returns an `sltest.testmanager.CoverageSettings` object.

### Input Arguments

#### **tf** — Test file

`sltest.testmanager.TestFile` object

Test file object to get coverage settings from, specified as an `sltest.testmanager.TestFile` object.

### Output Arguments

#### **covSettings** — Coverage settings

object

Coverage settings for the test file, returned as an `sltest.testmanager.CoverageSettings` object. For information on coverage metrics, see `sltest.plugins.coverage.CoverageMetrics`.

### Examples

#### Turn On Coverage For a Test File

```
% Open the model for this example
openExample('sldemo_autotrans');

% Create the test file, test suite, and test case structure
tf = sltest.testmanager.TestFile('API Test File');
ts = createTestSuite(tf,'API Test Suite');
tc = createTestCase(ts,'simulation','Coverage Test Case');

% Remove the default test suite
tsDel = getTestSuiteByName(tf,'New Test Suite 1');
remove(tsDel);

% Assign the system under test to the test case
setProperty(tc,'Model','sldemo_autotrans');
```

```
% Turn on coverage settings at test-file level
cov = getCoverageSettings(tf);
cov.RecordCoverage = true;
```

## See Also

`sltest.testmanager.CoverageSettings` | `sltest.plugins.coverage.CoverageMetrics`

## Topics

“Create and Run Test Cases with Scripts”

## Introduced in R2016a

## getCoverageSettings

**Class:** sltest.testmanager.TestSuite

**Package:** sltest.testmanager

Get coverage settings

### Syntax

```
covSettings = getCoverageSettings(ts)
```

### Description

`covSettings = getCoverageSettings(ts)` gets coverage settings for a test suite and returns an `sltest.testmanager.CoverageSettings` object.

### Input Arguments

**ts — Test suite**

`sltest.testmanager.TestSuite` object

Test suite object to get coverage settings from, specified as an `sltest.testmanager.TestSuite` object.

### Output Arguments

**covSettings — Coverage settings**

object

Coverage settings for the test suite, returned as an `sltest.testmanager.CoverageSettings` object. For information on coverage metrics, see `sltest.plugins.coverage.CoverageMetrics`.

### Examples

#### Turn Off Coverage For a Test Suite

```
% Open the model for this example
openExample('sldemo_autotrans');

% Create the test file, test suite, and test case structure
tf = sltest.testmanager.TestFile('API Test File');
ts = createTestSuite(tf,'API Test Suite');
tc = createTestCase(ts,'simulation','Coverage Test Case');

% Remove the default test suite
tsDel = getTestSuiteByName(tf,'New Test Suite 1');
remove(tsDel);

% Assign the system under test to the test case
setProperty(tc,'Model','sldemo_autotrans');
```

```
% Turn on coverage settings at test-file level
cov = getCoverageSettings(tf);
cov.RecordCoverage = true;

% Turn off coverage at test-suite level
cov = getCoverageSettings(ts);
cov.RecordCoverage = false;
```

## See Also

`sltest.testmanager.CoverageSettings` | `sltest.plugins.coverage.CoverageMetrics`

## Topics

“Create and Run Test Cases with Scripts”

**Introduced in R2016a**

## getCustomCriteria

**Class:** `sltest.testmanager.TestCase`

**Package:** `sltest.testmanager`

Get custom criteria that belong to test case

### Syntax

```
customCriteria = getCustomCriteria(tc)
```

### Description

`customCriteria = getCustomCriteria(tc)` creates the custom criteria object `customCriteria` from the test case `tc`.

### Input Arguments

**tc — Test case**

`sltest.testmanager.TestCase` object

Test case to get the custom criteria from, specified as a `sltest.testmanager.TestCase` object.

### Output Arguments

**customCriteria — Test case custom criteria**

`sltest.testmanager.CustomCriteria` object

Custom criteria of the test case, returned as an `sltest.testmanager.CustomCriteria` object.

### Examples

#### Get Custom Criteria in Test Case

Create a test case object from the test suite `ts`.

```
tc = ts.getTestCaseByName('Requirement 1.3 Test');
```

Get the custom criteria from the test case `tc`.

```
tcCriteria = getCustomCriteria(tc);
```

### See Also

`sltest.testmanager.TestIteration`

#### Topics

“Process Test Results with Custom Scripts”

“Custom Criteria Programmatic Interface Example”

“Create and Run Test Cases with Scripts”



**Introduced in R2016b**

## getCustomCriteriaPlots

**Class:** `sltest.testmanager.TestCaseResult`

**Package:** `sltest.testmanager`

Get plots from test case custom criteria

### Syntax

```
figs = getCustomCriteriaPlots(result)
```

### Description

`figs = getCustomCriteriaPlots(result)` returns figure handles of plots generated from the custom criteria of the test case associated with the results. Figures returned using this method are not visible. To see the plots, set the figure handle `Visible` property to `'on'`.

### Input Arguments

**result** — Test case result

`sltest.testmanager.TestCaseResult` object

Test case result to get custom criteria figure handles from, specified as a `sltest.testmanager.TestCaseResult` object.

### Output Arguments

**figs** — Figures from test case custom criteria

array of figure handles

Figures from test case custom criteria, returned as an array of figure handles.

### Examples

#### Get Figure Handles from Test Case Results

```
% Open the model for this example
openExample('sldemo_absbrake');

% Create the test file, suite, and case
tf = sltest.testmanager.TestFile('Test Case Custom Criteria');
ts = createTestSuite(tf,'CC Test Suite');
tc = createTestCase(ts,'baseline','CC Test Case');

% Remove the default test suite
tsDel = getTestSuiteByName(tf,'New Test Suite 1');
remove(tsDel);

% Create a plot in custom criteria and enable custom criteria
tcCriteria = getCustomCriteria(tc);
```

```
tcCriteria.Callback = 'a = [1,2,3]; f= figure; plot(a);';
tcCriteria.Enabled = true;

% Set option to save figures
opt = getOptions(tf);
opt.SaveFigures = true;

% Assign the system under test to the test case
setProperty(tc, 'Model', 'sldemo_absbrake');

% Run the test and capture results
resultset = run(tf);
tfr = getTestFileResults(resultset);
tsr = getTestSuiteResults(tfr);
tcr = getTestCaseResults(tsr);

% Get the custom criteria plot figure handles
figs = tcr.getCustomCriteriaPlots;
```

## See Also

sltest.testmanager.TestCaseResult | sltest.testmanager.CustomCriteria |  
sltest.testmanager.Options

## Topics

“Create, Store, and Open MATLAB Figures”  
“Export Test Results”

## Introduced in R2017a

## getCustomCriteriaPlots

**Class:** sltest.testmanager.TestIterationResult

**Package:** sltest.testmanager

Get plots from custom criteria

### Syntax

```
figs = getCustomCriteriaPlots(result)
```

### Description

`figs = getCustomCriteriaPlots(result)` returns figure handles of plots from the custom criteria of the test case associated with the iteration results. Figures returned using this method are not visible. To see the plots, set the figure handle `Visible` property to 'on'.

### Input Arguments

**result** — Test iteration result

sltest.testmanager.TestIterationResult object

Test iteration result to get custom criteria plots from, specified as an sltest.testmanager.TestIterationResult object.

### Output Arguments

**figs** — Figures from test case custom criteria

array of figure handles

Figures from test case custom criteria for the specified iteration, returned as an array of figure handles.

### Examples

#### Get Figure Handles from Test Iteration Results

```
% Open the model for this example
openExample('sldemo_absbrake');

% Create the test file, suite, and case
tf = sltest.testmanager.TestFile...
 ('Iteration Custom Criteria Plots');
ts = createTestSuite(tf, 'CC Test Suite');
tc = createTestCase(ts, 'baseline', 'CC Test Case');

% Remove the default test suite
tsDel = getTestSuiteByName(tf, 'New Test Suite 1');
remove(tsDel);
```

```

% Create a plot in custom criteria and
% enable custom criteria
tcCriteria = getCustomCriteria(tc);
tcCriteria.Callback = 'a = [1,2,3];
f= figure; plot(a);';
tcCriteria.Enabled = true;

% Set option to save figures
opt = getOptions(tf);
opt.SaveFigures = true;

% Assign the system under test to the test case
setProperty(tc, 'Model', 'sldemo_absbrake');

% Specify iterations
vars = 32 : 0.5 : 34;

for k = 1 : length(vars)

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

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

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

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

% Run the test and capture results
resultset = run(tf);
tfr = getTestFileResults(resultset);
tsr = getTestSuiteResults(tfr);
tcr = getTestCaseResults(tsr);
tir = getIterationResults(tcr);

% Get the custom criteria plot figure handles from first iteration
figs = tir(1).getCustomCriteriaPlots;

```

## See Also

sltest.testmanager.TestIterationResult | sltest.testmanager.CustomCriteria |  
sltest.testmanager.Options | getIterationResults

## Topics

“Create, Store, and Open MATLAB Figures”  
“Export Test Results”

**Introduced in R2017a**

## getCustomCriteriaResult

**Class:** `sltest.testmanager.TestCaseResult`

**Package:** `sltest.testmanager`

Get custom criteria results from test case result

### Syntax

```
ccResult = getCustomCriteriaResult(tcResult)
```

### Description

`ccResult = getCustomCriteriaResult(tcResult)` creates the custom criteria result object `ccResult` from test case result `tcResult`.

### Input Arguments

**tcResult — Test case result**

`sltest.testmanager.TestCaseResult` object

Test case result to get the custom criteria result from, specified as a `TestCaseResult` object.

### Output Arguments

**ccResult — Custom criteria result**

`sltest.testmanager.CustomCriteriaResult` object

Custom criteria result of the test case result, returned as an `CustomCriteriaResult` object.

### Examples

#### Get Custom Criteria Result from Test Case Result

Create a test case result object from the test case result set `tcResultSet`.

```
tcResult = getTestCaseResults(tcResultSet);
```

Get the custom criteria from the test case result `tcResult`.

```
ccResult = getCustomCriteriaResult(tcResult);
```

### See Also

`sltest.testmanager.TestIteration`

#### Topics

“Process Test Results with Custom Scripts”

“Custom Criteria Programmatic Interface Example”

“Create and Run Test Cases with Scripts”

**Introduced in R2016b**

## getCustomCriteriaResult

**Class:** `sltest.testmanager.TestIterationResult`

**Package:** `sltest.testmanager`

Get custom criteria results from test iteration

### Syntax

```
ccResult = getCustomCriteriaResult(tiResult)
```

### Description

`ccResult = getCustomCriteriaResult(tiResult)` creates the custom criteria result object `ccResult` from test iteration result `tiResult`.

### Input Arguments

**tiResult — Test iteration result**

`sltest.testmanager.TestIterationResult` object

Test iteration result to get the custom criteria result from, specified as a `TestIterationResult` object.

### Output Arguments

**ccResult — Custom criteria result**

`sltest.testmanager.CustomCriteriaResult` object

Custom criteria result of the test case result, returned as an `CustomCriteriaResult` object.

### Examples

**Get Custom Criteria Result from Test Case Result**

Create a test case result object from the test case result set `tcResultSet`.

```
tcResult = getTestCaseResults(tcResultSet);
```

Get the iteration result from the test case result `tcResult`.

```
iterResult = getIterationResults(tcResult);
```

Get the custom criteria from the test case result `tcResult`.

```
ccResult = getCustomCriteriaResult(iterResult);
```

### See Also

`sltest.testmanager.TestIteration`



**Topics**

“Process Test Results with Custom Scripts”

“Custom Criteria Programmatic Interface Example”

“Create and Run Test Cases with Scripts”

**Introduced in R2016b**

## getEquivalenceCriteria

**Class:** `sltest.testmanager.TestCase`

**Package:** `sltest.testmanager`

Get equivalence criteria from test case

### Syntax

```
eq = getEquivalenceCriteria(tc)
```

### Description

`eq = getEquivalenceCriteria(tc)` gets the equivalence criteria set from the test case. The function returns an equivalence criteria object, `sltest.testmanager.EquivalenceCriteria`. This function can be used only if the test type is an equivalence test case.

### Input Arguments

**tc** — Test case

`sltest.testmanager.TestCase` object

Test case to get equivalence criteria from, specified as an `sltest.testmanager.TestCase` object.

### Output Arguments

**eq** — Equivalence criteria

`sltest.testmanager.EquivalenceCriteria` object

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

### Examples

#### Get Equivalence Criteria

```
% Open the model for this example
openExample('sldemo_absbrake');

% Create the test file, test suite, and test case structure
tf = sltest.testmanager.TestFile('API Test File');
ts = createTestSuite(tf,'API Test Suite');
tc = createTestCase(ts,'equivalence','Equivalence Test Case');

% Remove the default test suite
tsDel = getTestSuiteByName(tf,'New Test Suite 1');
remove(tsDel);

% Assign the system under test to the test case
% for Simulation 1 and Simulation 2
```

```
setProperty(tc, 'Model', 'sldemo_absbrake', 'SimulationIndex', 1);
setProperty(tc, 'Model', 'sldemo_absbrake', 'SimulationIndex', 2);

% Capture equivalence criteria
eq = captureEquivalenceCriteria(tc);

% Set the equivalence criteria tolerance for one signal
sc = getSignalCriteria(eq);
sc(1).AbsTol = 2.2;

% Get and check the equivalence criteria
eq = getEquivalenceCriteria(tc);
```

## See Also

### Topics

“Create and Run Test Cases with Scripts”

**Introduced in R2015b**

## getInputs

**Class:** `sltest.testmanager.TestCase`

**Package:** `sltest.testmanager`

Get test case inputs

### Syntax

```
inputs = getInputs(tc)
inputs = getInputs(tc,simulationIndex)
```

### Description

`inputs = getInputs(tc)` gets the input sets in a test case and returns them as an array of test input objects, `sltest.testmanager.TestInput`.

`inputs = getInputs(tc,simulationIndex)` gets the input sets in a test case and returns them as an array of test input objects, `sltest.testmanager.TestInput`. If the test case is an equivalence test case, then specify the simulation index.

### Input Arguments

#### **tc — Test case**

`sltest.testmanager.TestCase` object

Test case to get test inputs from, specified as an `sltest.testmanager.TestCase` object.

#### **simulationIndex — Test case simulation number**

1 | 2

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

### Output Arguments

#### **inputs — Test input**

`sltest.testmanager.TestInput` object array

Test inputs that belong to the test case, returned as an array of `sltest.testmanager.TestInput` objects.

### Examples

#### **Get Test Inputs**

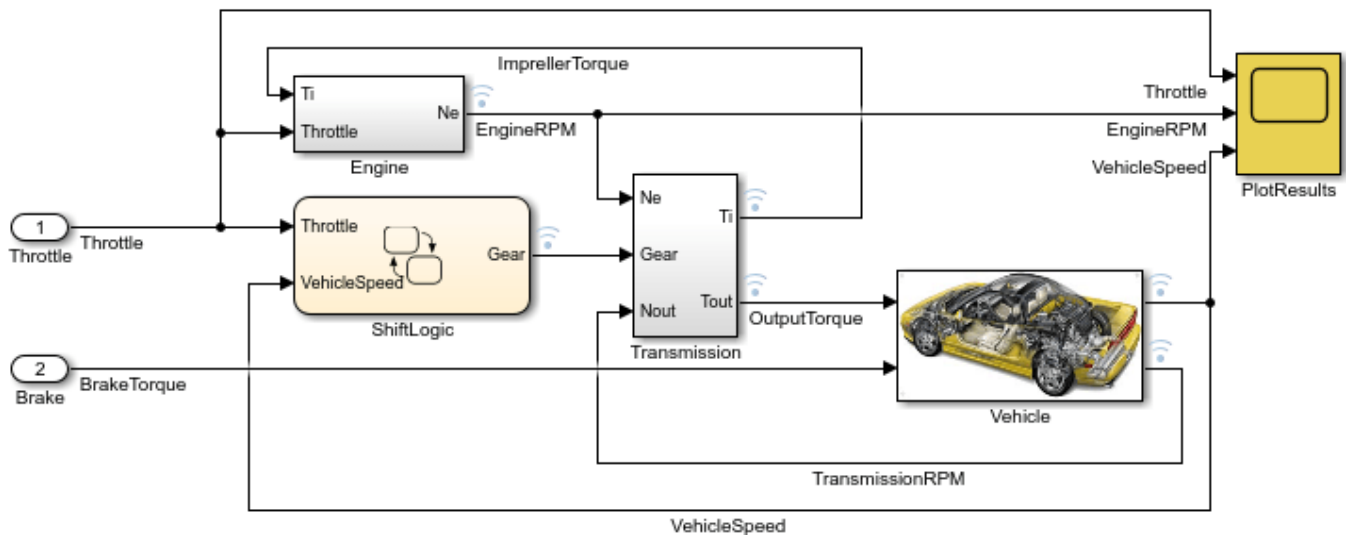
This example shows how to get the test inputs to a test case.

## Load the Example Model and Create a New Test File

```
open_system('sltestExcelExample');
tf = sltest.testmanager.TestFile('input_test_file.mldatx');
```

## Importing Microsoft Excel Data

This model is used to demonstrate the use of loading input data from Microsoft® Excel® file in test manager. To see the demo, execute `showdemo sltestUsingExcel` in MATLAB(R).



Copyright 2015 The MathWorks, Inc.

## Get the Test Suite and Test Case Objects

```
ts = getTestSuites(tf);
tc = getTestCases(ts);
```

## Add the Model as the System Under Test

```
setProperty(tc, 'Model', 'sltestExcelExample');
```

## Add Excel Data to Inputs Section and Specify Sheets to Add

```
excelfile = 'sltestExampleInputs.xlsx';
input = addInput(tc, excelfile, 'Sheets', ["Acceleration", "Braking"]);
```

## Map the Input Signal for the Sheets by Block Name

```
map(input(1), 0);
map(input(2), 0);
```

## Get and Check the Test Inputs

```
inputsOut = getInputs(tc);
inputsOut.ExcelSpecifications
```

```
ans =
ExcelSpecifications with properties:
 Sheet: 'Acceleration'
 Range: ''
```

```
ans =
ExcelSpecifications with properties:
 Sheet: 'Braking'
 Range: ''
```

## **See Also**

### **Topics**

“Create and Run Test Cases with Scripts”

**Introduced in R2015b**

# getIterationResults

**Class:** sltest.testmanager.TestCaseResult

**Package:** sltest.testmanager

Get iteration results

## Syntax

```
iterArray = getIterationResults(result)
```

## Description

`iterArray = getIterationResults(result)` returns the test iteration results that are children of the test case result.

## Input Arguments

### result — Test case result

sltest.testmanager.TestCaseResult object

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

## Output Arguments

### iterArray — Iteration result

sltest.testmanager.TestIterationResult object array

Iteration result set, returned as an array of sltest.testmanager.TestIterationResult objects.

## Examples

### Get Test Iteration Results

```
% Open the model for this example
openExample('sldemo_autotrans');

% Create test file, test suite, and test case structure
tf = sltest.testmanager.TestFile('Iterations Test File');
ts = getTestSuites(tf);
tc = createTestCase(ts, 'simulation', 'Simulation Iterations');

% Specify model as system under test
setProperty(tc, 'Model', 'sldemo_autotrans');

% Set up table iteration
% Create iteration object
testItr1 = sltestiteration;
% Set iteration settings
```

```
setTestParam(testItr1,'SignalBuilderGroup','Passing Maneuver');
% Add the iteration to test case
addIteration(tc,testItr1);

% Set up another table iteration
% Create iteration object
testItr2 = sltestiteration;
% Set iteration settings
setTestParam(testItr2,'SignalBuilderGroup','Coasting');
% Add the iteration to test case
addIteration(tc,testItr2);

% Run test case that contains iterations
results = run(tc);

% Get iteration results
tcResults = getTestCaseResults(results);
iterResults = getIterationResults(tcResults);
```

## See Also

`sltest.testmanager.TestIterationResult`

## Topics

“Create and Run Test Cases with Scripts”

**Introduced in R2016a**



# getIterations

**Class:** `sltest.testmanager.TestCase`

**Package:** `sltest.testmanager`

Get test iterations that belong to test case

## Syntax

```
iterArray = getIterations(tc)
iterArray = getIterations(tc,iterName)
```

## Description

`iterArray = getIterations(tc)` get one or more test iteration objects that belong to the test case.

`iterArray = getIterations(tc,iterName)` get one or more test iteration objects with the specified name that belong to the test case.

## Input Arguments

**tc — Test case to get iteration from**

`sltest.testmanager.TestCase` object

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

**iterName — Test iteration name**

character vector

Test iteration name, specified as a character vector. This is an optional argument.

Example: 'Test Iteration 5'

## Output Arguments

**iterArray — Test iterations**

`sltest.testmanager.TestIteration` object array

Test iterations that belong to the test case, returned as an array of `sltest.testmanager.TestIteration` objects.

## Examples

**Get Test Iterations in Test Case**

```
% Open the model for this example
openExample('sldemo_autotrans');
```

```
% Create test file, test suite, and test case structure
tf = sltest.testmanager.TestFile('Iterations Test File');
ts = getTestSuites(tf);
tc = createTestCase(ts,'simulation','Simulation Iterations');

% Specify model as system under test
setProperty(tc,'Model','sldemo_autotrans');

% Set up table iteration
% Create iteration object
testItr1 = sltestiteration;
% Set iteration settings
setTestParam(testItr1,'SignalBuilderGroup','Passing Maneuver');
% Add the iteration to test case
addIteration(tc,testItr1);

% Set up another table iteration
% Create iteration object
testItr2 = sltestiteration;
% Set iteration settings
setTestParam(testItr2,'SignalBuilderGroup','Coasting');
% Add the iteration to test case
addIteration(tc,testItr2);

% Get iterations
iters = getIterations(tc);
```

## See Also

sltest.testmanager.TestIteration

## Topics

“Test Iterations”

“Create and Run Test Cases with Scripts”

**Introduced in R2016a**

# getLoggedSignals

**Class:** `sltest.testmanager.LoggedSignalSet`

**Package:** `sltest.testmanager`

Return logged signals contained in a set

## Syntax

```
objs = getLoggedSignals(lgset)
```

## Description

`objs = getLoggedSignals(lgset)` returns a vector of the `sltest.testmanager.LoggedSignal` objects contained in an `sltest.testmanager.LoggedSignalSet` object.

## Input Arguments

### **lgset** — Logged signal set

`sltest.testmanager.LoggedSignalSet` object

Logged signal set object contained in a test case.

## Examples

### Remove a Signal from a Signal Set

Open a model and create a signal set.

```
openExample('sldemo_absbrake')

% Create test case
tf = sltest.testmanager.TestFile(strcat(pwd, '\mytf.mldatx'));
ts = sltest.testmanager.TestSuite(tf, 'myts');
tc = sltest.testmanager.TestCase(ts, 'baseline', 'mytc');
```

```
% Create signal set
mylgset = tc.addLoggedSignalSet;
```

Select the `Vehicle Speed` block and enter `gcb`. Use the returned path to create a `Simulink.BlockPath` object.

```
% Add signals to set
bPath = Simulink.BlockPath('sldemo_absbrake/Vehicle speed');
sig1 = mylgset.addLoggedSignal(bPath,1);
sig2 = mylgset.addLoggedSignal(bPath,2);
```

```
setProperty(tc, 'Model', 'sldemo_absbrake');
```

```
% Remove signal
```

```
remove(sig2);
```

```
% Check that signal is removed
sigs = mylgset.getLoggedSignals
```

## **See Also**

gcb

## **Topics**

“Create and Run Test Cases with Scripts”

“Capture Simulation Data in a Test Case”

**Introduced in R2019a**

# getLoggedSignalSets

**Class:** sltest.testmanager.TestCase

**Package:** sltest.testmanager

Get logged signal set from a test case

## Syntax

```
objs = getLoggedSignalSets(tc)
objs = getLoggedSignalSets(tc, 'SimulationIndex', Value)
```

## Description

`objs = getLoggedSignalSets(tc)` creates and returns a vector of the `sltest.testmanager.LoggedSignalSet` objects that are stored in a test case object.

`objs = getLoggedSignalSets(tc, 'SimulationIndex', Value)` creates and returns a vector of the `sltest.testmanager.LoggedSignalSet` objects from a specific simulation in an equivalence test.

## Input Arguments

### tc — Test case

`sltest.testmanager.TestCase` object

Test case object.

### Value — Value of simulation index for equivalence criteria

1 (default) | 2

When the test case is an equivalence test, this index specifies the simulation that contains the signal set.

Example: `obj = getLoggedSignalSets(tc_equiv, 'SimulationIndex', 2);`

## Examples

### Get Signal Sets from a Test Case

Open a model and create a test case.

```
openExample('sldemo_absbrake');

% Create test case
tf = sltest.testmanager.TestFile(strcat(pwd, '\mytf.mldatx'));
ts = sltest.testmanager.TestSuite(tf, 'myts');
tc = sltest.testmanager.TestCase(ts, 'baseline', 'mytc');

% Create signal set
```

```
lgset = tc.addLoggedSignalSet;
lgset2 = tc.addLoggedSignalSet;

% Get signal sets from test case
mysets = getLoggedSignalSets(tc)
```

**See Also**

`sltest.testmanager.EquivalenceCriteria` | `sltest.testmanager.LoggedSignalSet`

**Topics**

“Create and Run Test Cases with Scripts”

“Capture Simulation Data in a Test Case”

**Introduced in R2019a**

# getOptions

**Class:** `sltest.testmanager.TestCase`

**Package:** `sltest.testmanager`

Get test file options

## Syntax

```
opt = getOptions(tc)
```

## Description

`opt = getOptions(tc)` returns the test file options object `sltest.testmanager.Options` associated with the test case `tc`.

## Input Arguments

### **tc** — Test case

`sltest.testmanager.TestCase` object

Test case to get test file options from, specified as an `sltest.testmanager.TestCase` object.

## Output Arguments

### **opt** — Test file options

`sltest.testmanager.Options` object

Test file options, returned as an `sltest.testmanager.Options` object.

## Examples

### Get Test Case Test File Options

```
% Create the test file, test suite, and test case structure
tf = sltest.testmanager.TestFile('API Test File');
ts = createTestSuite(tf,'API Test Suite');
tc = createTestCase(ts,'baseline','Baseline API Test Case');

% Get the test file options
opt = getOptions(tc);
```

## See Also

`sltest.testmanager.Options` | `getOptions (TestSuite)` | `getOptions (TestFile)`

## Topics

“Create, Store, and Open MATLAB Figures”

“Export Test Results”

**Introduced in R2017a**



# getOptions

**Class:** `sltest.testmanager.TestFile`

**Package:** `sltest.testmanager`

Get and set test file options

## Syntax

```
opt = getOptions(tf)
```

## Description

`opt = getOptions(tf)` returns the test file options object `sltest.testmanager.Options` associated with the test file `tf`.

## Input Arguments

### **tf** — Test file

`sltest.testmanager.TestFile` object

Test file whose options to get, specified as an `sltest.testmanager.TestFile` object.

## Output Arguments

### **opt** — Test file options

`sltest.testmanager.Options` object

Test file options, returned as an `sltest.testmanager.Options` object.

## Examples

### Get and Set Test File Options

```
% Create the test file, test suite, and test case structure
tf = sltest.testmanager.TestFile('API Test File');
ts = createTestSuite(tf, 'API Test Suite');
tc = createTestCase(ts, 'baseline', 'Baseline API Test Case');

% Get the test file options
opt = getOptions(tf);

% Set the title for the report and specify to save figures
opt.Title = 'ABC Co. Test Results';
opt.SaveFigures = true;
```

## See Also

`sltest.testmanager.Options` | `sltest.testmanager.TestFile`

**Topics**

“Create, Store, and Open MATLAB Figures”

“Export Test Results”

**Introduced in R2017a**

# getOptions

**Class:** `sltest.testmanager.TestSuite`

**Package:** `sltest.testmanager`

Get test file options

## Syntax

```
opt = getOptions(ts)
```

## Description

`opt = getOptions(ts)` returns the test file options object `sltest.testmanager.Options` associated with the test suite `ts`.

## Input Arguments

### **ts** — Test suite

`sltest.testmanager.TestSuite` object

Test suite to get test file options from, specified as an `sltest.testmanager.TestSuite` object.

## Output Arguments

### **opt** — Test file options

`sltest.testmanager.Options` object

Test file options, returned as an `sltest.testmanager.Options` object.

## Examples

### Get Test File Options

```
% Create the test file, test suite, and test case structure
tf = sltest.testmanager.TestFile('API Test File');
ts = createTestSuite(tf, 'API Test Suite');
tc = createTestCase(ts, 'baseline', 'Baseline API Test Case');

% Get the test file options
opt = getOptions(ts);
```

## See Also

`sltest.testmanager.Options` | `sltest.testmanager.TestSuite`

## Topics

“Create, Store, and Open MATLAB Figures”

“Export Test Results”

**Introduced in R2017a**

# getInputRuns

**Class:** `sltest.testmanager.TestCaseResult`, `sltest.testmanager.TestIterationResult`

**Package:** `sltest.testmanager`

Get inputs from simulations captured with the test result

## Syntax

```
runArray = getInputRuns(result)
```

## Description

`runArray = getInputRuns(result)` gets the simulation inputs from the test result.

## Input Arguments

### result — Result

`sltest.testmanager.TestCaseResult` object | `sltest.testmanager.TestiterationResult` object

Test results to get simulation input results from, specified as a `sltest.testmanager.TestCaseResult` or `sltest.testmanager.TestIterationResult` object.

## Output Arguments

### runArray — Simulation run input results

`Simulink.sdi.Run` object

Simulation run input results, returned as a `Simulink.sdi.Run` object array.

## Examples

### Get Simulation Run Inputs from Test Case

#### Load the Example Model and Create a New Test File

```
open_system('sltestExcelExample');
tf = sltest.testmanager.TestFile('Input Run Test File');
```

#### Get the Test Suite and Test Case Objects

```
ts = getTestSuites(tf);
tc = getTestCases(ts);
```

#### Add the Model as the System Under Test and Save the Input Run

```
setProperty(tc, 'Model', 'sltestExcelExample', 'SaveInputRunInTestResult', true);
```

**Add Excel® Data to Inputs Section and Specify Sheets to Add**

```
excelfile = 'sltestExampleInputs.xlsx';
input = addInput(tc,excelfile,'Sheets',["Acceleration","Braking"]);
```

**Map the Input Signal for the Sheets by Block Name**

```
map(input(1),0);
map(input(2),0);
```

**Capture the baseline criteria**

```
baseline = captureBaselineCriteria(tc,'baseline_API.mat',true);
```

**Run the Test Case and Get the Test Case and Iteration Results**

```
resultsObj = run(tc);
tcr = getTestCaseResults(resultsObj);
tir = tcr.getIterationResults;
```

**Get the Inputs From the Simulation Run**

```
inrun = tir(1).getInputRuns;
```

**See Also**

[Simulink.sdi.Run | getOutputRuns](#)

**Topics**

“Create and Run Test Cases with Scripts”

**Introduced in R2017a**

# getOutputRuns

**Class:** `sltest.testmanager.TestCaseResult`

**Package:** `sltest.testmanager`

Get test case simulation output results

## Syntax

```
runArray = getOutputRuns(result)
```

## Description

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

## Input Arguments

**result — Test case result**

`sltest.testmanager.TestCaseResult` object

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

## Output Arguments

**runArray — Simulation output results**

`Simulink.sdi.Run` object

Test case simulation output results, returned as a `Simulink.sdi.Run` object array.

## Examples

### Get and Export Simulation Output From Test Case

```
% Open the model for this example
openExample('sldemo_absbrake');

% Create the test file, test suite, and test case structure
tf = sltest.testmanager.TestFile('API Test File');
ts = createTestSuite(tf, 'API Test Suite');
tc = createTestCase(ts, 'baseline', 'Baseline API Test Case');

% Remove the default test suite
tsDel = getTestSuiteByName(tf, 'New Test Suite 1');
remove(tsDel);

% Assign the system under test to the test case
setProperty(tc, 'Model', 'sldemo_absbrake');
```

```
% Capture the baseline criteria
baseline = captureBaselineCriteria(tc,'baseline_API.mat',true);

% Test a new model parameter by overriding it in the test case
% parameter set
ps = addParameterSet(tc,'Name','API Parameter Set');
po = addParameterOverride(ps,'m',55);

% Set the baseline criteria tolerance for one signal
sc = getSignalCriteria(baseline);
sc(1).AbsTol = 9;

% Run the test case and return an object with results data
resultsObj = run(tc);

% Get test case result
tcr = getTestCaseResults(resultsObj);

% Get the Sim Output run dataset
runDataset = getOutputRuns(tcr);

% Export the Sim Output run dataset, which is a
% Simulink.sdi.run object
dataset = export(runDataset);
```

## See Also

Simulink.sdi.Run

## Topics

“Create and Run Test Cases with Scripts”

**Introduced in R2015a**



# getOutputRuns

**Class:** `sltest.testmanager.TestIterationResult`

**Package:** `sltest.testmanager`

Get test iteration simulation output results

## Syntax

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

## Description

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

## Input Arguments

**resultObj** — Test iteration result

object

Test iteration results object to get results from, specified as a `sltest.testmanager.TestIterationResult` object.

## Output Arguments

**runArray** — Simulation output results

object

Test iteration simulation output results, returned as a `Simulink.sdi.Run` object array.

## See Also

`sltest.testmanager.TestIterationResult`

## Topics

“Create and Run Test Cases with Scripts”

**Introduced in R2016a**

## getParameterOverrides

**Class:** sltest.testmanager.ParameterSet

**Package:** sltest.testmanager

Get parameter overrides

### Syntax

```
ovrs = getParameterOverrides(ps)
```

### Description

`ovrs = getParameterOverrides(ps)` gets the parameter overrides in a parameter set and returns them as an array of parameter override objects, `sltest.testmanager.ParameterOverride`.

### Input Arguments

**ps — Parameter set**

`sltest.testmanager.ParameterSet` object

Parameter set that you want to get the override from, specified as a `sltest.testmanager.ParameterSet` object.

### Output Arguments

**ovrs — Parameter overrides**

`sltest.testmanager.ParameterOverride` object array

Parameter overrides that are in the parameter set object, returned as an array of `sltest.testmanager.ParameterOverride` objects.

### Examples

#### Add Parameter Override to Parameter Set

```
% Open the model for this example
openExample('sldemo_absbrake');

% Create the test file, test suite, and test case structure
tf = sltest.testmanager.TestFile('API Test File');
ts = createTestSuite(tf, 'API Test Suite');
tc = createTestCase(ts, 'baseline', 'Baseline API Test Case');

% Remove the default test suite
tsDel = getTestSuiteByName(tf, 'New Test Suite 1');
remove(tsDel);
```

```
% Assign the system under test to the test case
setProperty(tc, 'Model', 'sldemo_absbrake');

% Capture the baseline criteria
baseline = captureBaselineCriteria(tc, 'baseline_API.mat', true);

% Test a new model parameter by overriding it in the test case
% parameter set
ps = addParameterSet(tc, 'Name', 'API Parameter Set');
po = addParameterOverride(ps, 'm', 55);

% Check that the parameter override is applied
ovr = getParameterOverrides(ps);
```

## See Also

### Topics

“Create and Run Test Cases with Scripts”

**Introduced in R2015b**

## getParameterSets

**Class:** `sltest.testmanager.TestCase`

**Package:** `sltest.testmanager`

Get test case parameter sets

### Syntax

```
psets = getParameterSets(tc)
psets = getParameterSets(tc,simulationIndex)
```

### Description

`psets = getParameterSets(tc)` gets the parameter sets in a test case and returns them as an array of parameter set objects, `sltest.testmanager.ParameterSet`.

`psets = getParameterSets(tc,simulationIndex)` gets the parameter sets in a test case and returns them as an array of parameter set objects, `sltest.testmanager.ParameterSet`. If the test case is an equivalence test case, then specify the simulation index.

### Input Arguments

#### **tc** — Test case

`sltest.testmanager.TestCase` object

Test case to get test inputs from, specified as an `sltest.testmanager.TestCase` object.

#### **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 default simulation index is 1.

### Output Arguments

#### **psets** — Parameter set

`sltest.testmanager.ParameterSet` object array

Parameter sets that belong to the test case, returned as an array of `sltest.testmanager.ParameterSet` objects.

### Examples

#### Get Parameter Sets

```
% Open the model for this example
openExample('sldemo_absbrake');
```

```
% Create the test file, test suite, and test case structure
tf = sltest.testmanager.TestFile('API Test File');
ts = createTestSuite(tf, 'API Test Suite');
tc = createTestCase(ts, 'baseline', 'Baseline API Test Case');

% Assign the system under test to the test case
setProperty(tc, 'Model', 'sldemo_absbrake');

% Test a new model parameter by overriding it in the test case
% parameter set
ps = addParameterSet(tc, 'Name', 'API Parameter Set');
po = addParameterOverride(ps, 'm', 55);

% Get and check the parameter set
psets = getParameterSets(tc);
```

## See Also

### Topics

“Create and Run Test Cases with Scripts”

### Introduced in R2015b

## getProperty

**Class:** `sltest.testmanager.TestCase`

**Package:** `sltest.testmanager`

Get test case property

### Syntax

```
val = getProperty(tc,propertyName)
val = getProperty(____,simulationIndex)
```

### Description

`val = getProperty(tc,propertyName)` gets a test case property.

`val = getProperty( ____,simulationIndex)` gets a test case property. If the test case is an equivalence test case, then specify the simulation index.

### Input Arguments

#### **tc — Test case**

`sltest.testmanager.TestCase` object

Test case to get test setting property from, specified as an `sltest.testmanager.TestCase` object.

#### **propertyName — Test case property**

character vector

Test suite property names, specified as a character vector. The properties are set using `thesetProperty` method. The available properties are:

- 'Model' — name of the model to be tested
- 'SimulationMode' — simulation mode of the model during the test
- 'OverrideSILPILMode' — override SIL or PIL simulation mode of block to Normal mode
- 'HarnessName' — harness name used for the test
- 'HarnessOwner' — harness owner name
- 'OverrideStartTime' — override start time
- 'StartTime' — start time override value of simulation
- 'OverrideStopTime' — override stop time
- 'StopTime' — stop time override value of simulation
- 'OverrideInitialState' — override initial state
- 'InitialState' — character vector evaluated to specify the initial state of the system under test
- 'PreloadCallback' — character vector evaluated before the model loads and before model callbacks

- 'PostloadCallback' — character vector evaluated after the system under test loads and PostLoadFcn callback completes
- 'PreStartRealTimeApplicationCallback' — character vector evaluated before the real-time application is started on target computer
- 'CleanupCallback' — character vector evaluated after simulation completes and model callbacks execute
- 'UseSignalEditorScenarios' — use signal editor scenarios for test input
- 'SignalEditorScenario' — signal editor scenario name to use
- 'IsTestDataReferenced' — create test case using test data from an external file, such as an Excel or MAT file
- 'TestDataPath' — character vector path to the external file being referenced for creating the test case
- 'OverrideModelOutputSettings' — override model output settings
- 'SaveOutput' — save simulation output
- 'SaveState' — save model states during simulation
- 'SaveFinalState' — save final state of simulation
- 'SignalLogging' — log signals
- 'DSMLogging' — log data store
- 'ConfigsetOverrideSetting' — value to determine override of configuration set
- 'ConfigsetName' — configuration set override name
- 'ConfigsetFileLocation' — path to a MAT-file that contains a configuration set object
- 'ConfigsetVarName' — name of the variable in ConfigsetFileLocation that is a configuration set
- 'IterationScript' — character vector evaluated for test case iteration script
- 'SimulationIndex' — determines which simulation a property applies to, applicable to the equivalence test case type
- 'FastRestart' — indicates if test iterations run using fast restart mode
- 'SaveBaselineRunInTestResult' — enable saving the baseline run used in the test case, saved in the test result
- 'LoadAppFrom' — location to load real-time application from
- 'TargetComputer' — target computer name
- 'TargetApplication' — target application name

### **simulationIndex — Test case simulation number**

1 | 2

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

## **Output Arguments**

### **val — Property content**

character vector | logical | scalar

The content of the test case property, returned as a character vector, logical, or scalar value.

## Examples

### Get Test Case Property

```
% Open the model for this example
openExample('sldemo_absbrake');

% Create the test file, test suite, and test case structure
tf = sltest.testmanager.TestFile('API Test File');
ts = createTestSuite(tf, 'API Test Suite');
tc = createTestCase(ts, 'baseline', 'Baseline API Test Case');

% Assign the system under test to the test case
setProperty(tc, 'Model', 'sldemo_absbrake');

% Get and check the system under test model
getProperty(tc, 'Model');
```

### See Also

setProperty

### Topics

“Create and Run Test Cases with Scripts”

**Introduced in R2015b**



# getProperty

**Class:** `sltest.testmanager.TestFile`

**Package:** `sltest.testmanager`

Get test file property

## Syntax

```
val = getProperty(tf,propertyName)
```

## Description

`val = getProperty(tf,propertyName)` gets a test file property.

## Input Arguments

### **tf — Test file**

`sltest.testmanager.TestFile` object

Test file to get the property from, specified as an `sltest.testmanager.TestFile` object.

### **propertyName — Test file property**

'CleanupCallback' | 'SetupCallback' |

Test file property names, specified as:

- 'SetupCallback' to get the content of the test file setup callback
- 'CleanupCallback' to get the content of the test file cleanup callback

## Output Arguments

### **val — Property content**

character vector

The content of the test suite property, returned as a character vector.

## Examples

### **Get Test File Property**

```
% Create a test file and test suite
tf = sltest.testmanager.TestFile('API Test File');
ts = createTestSuite(tf, 'API Test Suite');
```

```
% Get the setup callback property
propName = getProperty(tf, 'SetupCallback');
```

**See Also**

setProperty

**Topics**

“Create and Run Test Cases with Scripts”

**Introduced in R2015b**

# getProperty

**Class:** `sltest.testmanager.TestSuite`

**Package:** `sltest.testmanager`

Get test suite property

## Syntax

```
val = getProperty(ts,propertyName)
```

## Description

`val = getProperty(ts,propertyName)` gets a test suite property.

## Input Arguments

### **ts — Test suite**

`sltest.testmanager.TestSuite` object

Test suite object to get the property from, specified as an `sltest.testmanager.TestSuite` object.

### **propertyName — Test suite property**

character vector

Test suite property names, specified as a character vector. The available properties are 'SetupCallback' and 'CleanupCallback'.

## Output Arguments

### **val — Property content**

character vector

The content of the test suite property, returned as a character vector.

## Examples

### **Get Test Suite Property**

```
% Create a test file and new test suite
tf = sltest.testmanager.TestFile('API Test File');
ts = createTestSuite(tf, 'API Test Suite');
```

```
% Get the setup callback property
propName = getProperty(ts, 'SetupCallback');
```

## **See Also**

### **Topics**

“Create and Run Test Cases with Scripts”

**Introduced in R2015b**

# getSetupPlots

**Class:** `sltest.testmanager.TestFileResult`

**Package:** `sltest.testmanager`

Get plots from setup callbacks

## Syntax

```
figs = getSetupPlots(result)
```

## Description

`figs = getSetupPlots(result)` returns figure handles of plots from the setup callbacks of the test file associated with the results. Figures returned using this method are not visible. To see the plots, set the figure handle `Visible` property to `'on'`.

## Input Arguments

**result** — Test file result

`sltest.testmanager.TestFileResult` object

Test file results to get setup plot figure handles from, specified as a `sltest.testmanager.TestFileResult` object

## Output Arguments

**figs** — Figures from test file setup callbacks

array of figure handles

Figures from test file setup callbacks, returned as an array of figure handles.

## Examples

### Get Test File Setup Plots

```
% Open the model for this example
openExample('sldemo_absbrake');

% Create the test file, suite, and case
tf = sltest.testmanager.TestFile('Get Test File Setup Plots');
ts = createTestSuite(tf,'Setup Plots Test Suite');
tc = createTestCase(ts,'baseline','Setup Plots Test Case');

% Remove the default test suite
tsDel = getTestSuiteByName(tf,'New Test Suite 1');
remove(tsDel);

% Create a plot in the test file setup callback
setProperty(tf,'SetupCallback','a = [1,2,3]; f = figure; plot(a);');
```

```
% Set option to save figures
opt = getOptions(tf);
opt.SaveFigures = true;

% Assign the system under test to the test case
setProperty(tc, 'Model', 'sldemo_absbrake');

% Run the test and capture results
resultset = run(tf);
tfr = getTestFileResults(resultset);

% Get the setup plot figure handles
figs = tfr.getSetupPlots;
```

### **See Also**

[sltest.testmanager.TestFile](#) | [sltest.testmanager.TestFileResult](#) | [setProperty](#) | [sltest.testmanager.Options](#)

### **Topics**

“Create, Store, and Open MATLAB Figures”  
“Export Test Results”

**Introduced in R2017a**

# getSetupPlots

**Class:** `sltest.testmanager.TestSuiteResult`

**Package:** `sltest.testmanager`

Plots from setup callbacks

## Syntax

```
figs = getSetupPlots(result)
```

## Description

`figs = getSetupPlots(result)` returns figure handles of plots generated from the setup callbacks of the test suite associated with the results. Figures returned using this method are not visible. To see the plots, set the figure handle `Visible` property to `'on'`.

## Input Arguments

**result** — Test suite results

`sltest.testmanager.TestSuiteResult` object

Test suite results to get setup plot figure handles from, specified as a `sltest.testmanager.TestSuiteResult` object.

## Output Arguments

**figs** — Figures from test suite setup callbacks

array of figure handles

Figures from test suite setup callbacks, returned as an array of figure handles.

## Examples

### Get Test Suite Setup Plots

```
% Open the model for this example
openExample('sldemo_absbrake');

% Create the test file, suite, and case
tf = sltest.testmanager.TestFile('Test Suite Setup Plots');
ts = createTestSuite(tf,'Setup Plots Test Suite');
tc = createTestCase(ts,'baseline','Setup Plots Test Case');

% Remove the default test suite
tsDel = getTestSuiteByName(tf,'New Test Suite 1');
remove(tsDel);

% Create a plot in the test suite setup callback
setProperty(ts,'SetupCallback','a = [1,2,3]; f = figure; plot(a);');
```

```
% Set option to save figures
opt = getOptions(tf);
opt.SaveFigures = true;

% Assign the system under test to the test case
setProperty(tc, 'Model', 'sldemo_absbrake');

% Run the test and capture results
resultset = run(tf);
tfr = getTestFileResults(resultset);
tsr = getTestSuiteResults(tfr);

% Get the setup plot figure handles
figs = tsr.getSetupPlots;
```

### **See Also**

`sltest.testmanager.TestSuite` | `sltest.testmanager.TestSuiteResult` | `setProperty` (Test Suite) | `sltest.testmanager.Options`

### **Topics**

“Create, Store, and Open MATLAB Figures”  
“Export Test Results”

**Introduced in R2017a**



# getSignalCriteria

**Class:** sltest.testmanager.BaselineCriteria

**Package:** sltest.testmanager

Get signal criteria

## Syntax

```
sigCriteria = getSignalCriteria(bc)
```

## Description

`sigCriteria = getSignalCriteria(bc)` gets the list of the signal criteria in a baseline criteria set and returns them as an array of signal criteria objects, `sltest.testmanager.SignalCriteria`.

## Input Arguments

### bc — Baseline criteria

`sltest.testmanager.BaselineCriteria` object

Baseline criteria that you want to get signal criteria from, specified as a `sltest.testmanager.BaselineCriteria` object.

## Output Arguments

### sigCriteria — Signal criteria object

object array

Signal criteria that are in the baseline criteria object, returned as an array of `sltest.testmanager.SignalCriteria` objects.

## Examples

### Add Baseline Criteria and Change Tolerance

In this example, a signal data set is capture for the baseline criteria, and the absolute tolerance is changed from 0 to 9.

```
% Open the model for this example
openExample('sldemo_absbrake');

% Create the test file, test suite, and test case structure
tf = sltest.testmanager.TestFile('API Test File');
ts = createTestSuite(tf,'API Test Suite');
tc = createTestCase(ts,'baseline','Baseline API Test Case');

% Remove the default test suite
```

```
tsDel = getTestSuiteByName(tf, 'New Test Suite 1');
remove(tsDel);

% Assign the system under test to the test case
setProperty(tc, 'Model', 'sldemo_absbrake');

% Capture the baseline criteria
baseline = captureBaselineCriteria(tc, 'baseline_API.mat', true);

% Set the baseline criteria tolerance for a signal
sc = getSignalCriteria(baseline);
sc(1).AbsTol = 9;
```

## See Also

### Topics

“Create and Run Test Cases with Scripts”

**Introduced in R2015b**

# getSignalCriteria

**Class:** sltest.testmanager.EquivalenceCriteria

**Package:** sltest.testmanager

Get signal criteria

## Syntax

```
sigCriteria = getSignalCriteria(eq)
```

## Description

`sigCriteria = getSignalCriteria(eq)` gets the list of the signal criteria in an equivalence criteria set and returns them as an array of signal criteria objects, `sltest.testmanager.SignalCriteria`.

## Input Arguments

### eq — Equivalence criteria

`sltest.testmanager.EquivalenceCriteria` object

Equivalence criteria that you want to get criteria from, specified as a `sltest.testmanager.EquivalenceCriteria` object.

## Output Arguments

### sigCriteria — Signal criteria object

object array

Signal criteria that are in the equivalence criteria object, returned as an array of `sltest.testmanager.SignalCriteria` objects.

## Examples

### Remove Equivalence Criteria

```
% Open the model for this example
openExample('sldemo_absbrake');

% Create the test file, test suite, and test case structure
tf = sltest.testmanager.TestFile('API Test File');
ts = createTestSuite(tf, 'API Test Suite');
tc = createTestCase(ts, 'equivalence', 'Equivalence Test Case');

% Remove the default test suite
tsDel = getTestSuiteByName(tf, 'New Test Suite 1');
remove(tsDel);
```

```
% Assign the system under test to the test case
% for Simulation 1 and Simulation 2
setProperty(tc, 'Model', 'sldemo_absbrake', 'SimulationIndex', 1);
setProperty(tc, 'Model', 'sldemo_absbrake', 'SimulationIndex', 2);

% Add a parameter override to Simulation 1 and 2
ps1 = addParameterSet(tc, 'Name', 'Parameter Set 1', 'SimulationIndex', 1);
po1 = addParameterOverride(ps1, 'Rr', 1.20);

ps2 = addParameterSet(tc, 'Name', 'Parameter Set 2', 'SimulationIndex', 2);
po2 = addParameterOverride(ps2, 'Rr', 1.24);

% Capture equivalence criteria
eq = captureEquivalenceCriteria(tc);

% Set the equivalence criteria tolerance for one signal
sc = getSignalCriteria(eq);
sc(1).AbsTol = 2.2;

% Check that signal criteria was added
sigCrit = getSignalCriteria(eq);
```

## See Also

### Topics

“Create and Run Test Cases with Scripts”

**Introduced in R2015b**

# getSimulationPlots

**Class:** `sltest.testmanager.TestCaseResult`

**Package:** `sltest.testmanager`

Get plots from test case callbacks

## Syntax

```
figs = getSimulationPlots(result)
figs = getSimulationPlots(result,index)
```

## Description

`figs = getSimulationPlots(result)` returns figure handles of plots generated from the callbacks of the test case associated with the results. Figures returned using this method are not visible. To see the plots, set the figure handle `Visible` property to `'on'`.

`figs = getSimulationPlots(result,index)` returns the figure handles from the simulation specified by `index`.

## Input Arguments

### **result** — Test case result

`sltest.testmanager.TestCaseResult` object

Test case results to get simulation plot figure handles from, specified as an `sltest.testmanager.TestCaseResult` object.

### **index** — Simulation index

1 (default) | 2

Simulation index, specified as 1 or 2.

## Output Arguments

### **figs** — Figures from test case callbacks

array of figure handles

Figures from test case callbacks, returned as an array of figure handles.

## Examples

### Get Figure Handles from Test Case Results

```
% Open the model for this example
openExample('sldemo_absbrake');

% Create the test file, suite, and case
```

```
tf = sltest.testmanager.TestFile('Simulation Plots Test Case');
ts = createTestSuite(tf, 'Sim Plots Test Suite');
tc = createTestCase(ts, 'baseline', 'Sim Plots Test Case');

% Remove the default test suite
tsDel = getTestSuiteByName(tf, 'New Test Suite 1');
remove(tsDel);

% Create a plot in a test case callback
setProperty(tc, 'PostloadCallback', 'a = [1,2,3]; f = figure; plot(a);');

% Set option to save figures
opt = getOptions(tf);
opt.SaveFigures = true;

% Assign the system under test to the test case
setProperty(tc, 'Model', 'sldemo_absbrake');

% Run the test and capture results
resultset = run(tf);
tfr = getTestFileResults(resultset);
tsr = getTestSuiteResults(tfr);
tcr = getTestCaseResults(tsr);

% Get the test case callback plots figure handles
figs = tcr.getSimulationPlots;
```

## See Also

[sltest.testmanager.TestCaseResult](#) | [sltest.testmanager.Options](#) | [setProperty](#)

## Topics

“Create, Store, and Open MATLAB Figures”

“Export Test Results”

## Introduced in R2017a

# getTestCaseByName

**Class:** `sltest.testmanager.TestFile`

**Package:** `sltest.testmanager`

Get test case object by name

## Syntax

```
tc = getTestCaseByName(tf, name)
```

## Description

`tc = getTestCaseByName(tf, name)` returns a test case object with the specified name.

## Input Arguments

### **tf** — Test file

`sltest.testmanager.TestFile` object

Test file containing the test cases to get, specified as an `sltest.testmanager.TestFile` object.

### **name** — Test case name

character vector

Name of the test case with the test file object, specified as a character vector. If the name does not match a test case, then the function returns an empty test case object.

## Output Arguments

### **tc** — Test case

`sltest.testmanager.TestCase` 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.

## Examples

### Get Test Cases

```
tf = sltest.testmanager.TestFile('My Test File');
tcArray = getTestCaseByName(tf, 'testOne');
```

## See Also

`sltest.testmanager.TestFile` | `getTestCases`

**Introduced in R2020b**

## getTestCaseByName

**Class:** `sltest.testmanager.TestSuite`

**Package:** `sltest.testmanager`

Get test case object by name

### Syntax

```
tc = getTestCaseByName(ts, name)
```

### Description

`tc = getTestCaseByName(ts, name)` returns a test case with the specified name.

### Input Arguments

**ts — Test suite**

`sltest.testmanager.TestSuite` object

Test suite with the test case you want to get, specified as an `sltest.testmanager.TestSuite` object.

**name — Test suite name**

character vector

The name of the test case within a test suite object, specified as a character vector. If the name does not match a test case, then the function returns an empty test case object.

### Output Arguments

**tc — Test case**

`sltest.testmanager.TestCase` 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.

### Examples

**Get Test Case Object**

```
% Create a test file with default test suite and case
tf = sltest.testmanager.TestFile('My Test File');

% Get test suite
ts = getTestSuites(tf);
```



```
% Get the test case object by test case name
tc = getTestCaseByName(ts, 'New Test Case 1');
```

## See Also

### Topics

“Create and Run Test Cases with Scripts”

**Introduced in R2015b**

## getIterationResults

**Class:** `sltest.testmanager.TestIteration`

**Package:** `sltest.testmanager`

Get test iteration results history

### Syntax

```
ti_result = getTestIterationResults(ti)
```

### Description

`ti_result = getTestIterationResults(ti)` returns the test iteration results history for the specified test iteration, `ti`. The test iteration history includes the results for all runs of the test iteration in the Test Manager.

### Input Arguments

**ti – Test iteration**

`sltest.testmanager.TestIteration` object

Test iteration for which to obtain results, specified as an `sltest.testmanager.TestIteration` object.

### Output Arguments

**ti\_result – Test iteration result**

array of `sltest.testmanager.TestIterationResult` objects

Test iteration result, returned as an array of `sltest.testmanager.TestIterationResult` objects. Each object in the array contains the results for a single test iteration run.

### See Also

`getTestIteration`

**Introduced in R2019b**

# getSimulationPlots

**Class:** `sltest.testmanager.TestIterationResult`

**Package:** `sltest.testmanager`

Get plots from callbacks

## Syntax

```
figs = getSimulationPlots(result)
figs = getSimulationPlots(result,index)
```

## Description

`figs = getSimulationPlots(result)` returns figure handles of plots generated from the callbacks of the test iteration associated with the results. Figures returned using this method are not visible. To see the plots, set the figure handle `Visible` property to `'on'`.

`figs = getSimulationPlots(result,index)` returns the figure handles from the simulation specified by `index`.

## Input Arguments

### **result** — Test iteration result

`sltest.testmanager.TestIterationResult` object

Test case iteration result to get callback figure handles from, specified as an `sltest.testmanager.TestIterationResult` object or a simulation index of the result.

### **index** — Simulation index

1 (default) | 2

Simulation index, specified as 1 or 2.

## Output Arguments

### **figs** — Figures from test case callbacks

array of figure handles

Figures from test case callbacks for the specified iteration, returned as an array of figure handles.

## Examples

### Get Figure Handles from Test Iteration Results

```
% Open the model for this example
openExample('sldemo_absbrake');

% Create the test file, suite, and case
```

```

tf = sltest.testmanager.TestFile...
 ('Simulation Plots for Test Iterations');
ts = createTestSuite(tf,'Sim Plots Test Suite');
tc = createTestCase(ts,'baseline','Sim Plots Test Case');

% Remove the default test suite
tsDel = getTestSuiteByName(tf,'New Test Suite 1');
remove(tsDel);

% Create a plot in a callback
setProperty(tc,'PostloadCallback',...
 'a = [1,2,3]; f = figure; plot(a);');

% Set option to save figures
opt = getOptions(tf);
opt.SaveFigures = true;

% Assign the system under test to the test case
setProperty(tc,'Model','sldemo_absbrake');

% Define iterations
vars = 32 : 0.5 : 34;

for k = 1 : length(vars)

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

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

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

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

% Run the test and capture results
resultset = run(tf);
tfr = getTestFileResults(resultset);
tsr = getTestSuiteResults(tfr);
tcr = getTestCaseResults(tsr);
tir = getIterationResults(tcr);

% Get the callback plot figure handles from the first iteration
figs = tir(1).getSimulationPlots;

```

## See Also

sltest.testmanager.TestIterationResult | setProperty (TestCase) |  
getIterationResults | sltest.testmanager.Options

## Topics

“Create, Store, and Open MATLAB Figures”  
“Export Test Results”

**Introduced in R2017a**

## getTestCaseResults

**Class:** `sltest.testmanager.ResultSet`

**Package:** `sltest.testmanager`

Get test case results object

### Syntax

```
testCaseResultArray = getTestCaseResults(result)
```

### Description

`testCaseResultArray = getTestCaseResults(result)` gets the test case results that are direct children of the results set object.

### Input Arguments

**result — Results set**

`sltest.testmanager.ResultSet` object

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

### Output Arguments

**testCaseResultArray — Test case results**

`sltest.testmanager.TestCaseResult` object array

Test case results, returned as an array of `sltest.testmanager.TestCaseResult` objects. The function returns objects that are direct children of the results set object.

### Examples

#### Get Test Case Result Data

Use the function `sltest.testmanager.run` to return a result set that contains test file, test suite, and test case results.

```
% Run tests in the Test Manager
result = sltest.testmanager.run;
```

```
% Get the test file results
testFileResultArray = getTestFileResults(result);
```

```
% Get the test suite results
testSuiteResultArray = getTestSuiteResults(result);
```

```
% Get the test case results
testCaseResultArray = getTestCaseResults(result);
```

## See Also

### Topics

“Create and Run Test Cases with Scripts”

**Introduced in R2015a**

## getTestCaseResults

**Class:** `sltest.testmanager.TestSuiteResult`

**Package:** `sltest.testmanager`

Get test case results object

### Syntax

```
testCaseResultArray = getTestCaseResults(result)
```

### Description

`testCaseResultArray = getTestCaseResults(result)` gets the test case results that are direct children of the test suite results object.

### Input Arguments

**result — Test suite result**

`sltest.testmanager.TestSuiteResult` object

Test suite results to get test case results from, specified as a `sltest.testmanager.TestSuiteResult` object.

### Output Arguments

**testCaseResultArray — Test case results**

`sltest.testmanager.TestCaseResult` object array

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

### Examples

#### Get Test Case Result Data

Use the function `sltest.testmanager.run` to return a result set that contains test file, test suite, and test case results.

```
% Run tests in the Test Manager
result = sltest.testmanager.run;

% Get the test file results
testFileResultArray = getTestFileResults(result);

% Get the test suite results
testSuiteResultArray = getTestSuiteResults(testFileResultArray);
```



```
% Get the test case results
testCaseResultArray = getTestCaseResults(testSuiteResultArray);
```

## See Also

### Topics

“Create and Run Test Cases with Scripts”

**Introduced in R2015a**

## getTestCases

**Class:** `sltest.testmanager.TestFile`

**Package:** `sltest.testmanager`

Get test cases in a test file

### Syntax

```
tcArray = getTestCases(tf)
```

### Description

`tcArray = getTestCases(tf)` returns an array of test case objects in a test file. Only test cases that are direct children of the test file (that is, not within a test suite) are returned.

### Input Arguments

**tf — Test file**

`sltest.testmanager.TestFile` object

Test file containing the test cases to get, specified as an `sltest.testmanager.TestFile` object.

### Output Arguments

**tcArray — Test case array**

`sltest.testmanager.TestCase` objects

Test cases at the top level of the test file, returned as an array of `sltest.testmanager.TestCase` objects.

### Examples

**Get Test Cases**

```
tf = sltest.testmanager.TestFile('My Test File');
tcArray = getTestCases(tf);
```

### See Also

`sltest.testmanager.TestFile` | `getTestCaseByName` | `getAllTestCases`

**Introduced in R2020b**

# getAllTestCases

**Class:** `sltest.testmanager.TestFile`

**Package:** `sltest.testmanager`

Get all test cases in a test file

## Syntax

```
tcArray = getAllTestCases(tf)
```

## Description

`tcArray = getAllTestCases(tf)` returns a flat array of all test case objects in a test file.

## Input Arguments

**tf — Test file**

`sltest.testmanager.TestFile` object

Test file containing the test cases to get, specified as an `sltest.testmanager.TestFile` object.

## Output Arguments

**tcArray — Test case array**

`sltest.testmanager.TestCase` objects

Test cases in the test file, returned as a flat array of `sltest.testmanager.TestCase` objects.

## Examples

### Get All Test Cases

```
tf = sltest.testmanager.TestFile('My Test File');
tcArray = getAllTestCases(tf);
```

## See Also

`sltest.testmanager.TestFile` | `getTestCases` | `getTestCaseByName`

**Introduced in R2021b**

## getTestCases

**Class:** `sltest.testmanager.TestSuite`

**Package:** `sltest.testmanager`

Get test cases at first level of test suite

### Syntax

```
tcArray = getTestCases(ts)
```

### Description

`tcArray = getTestCases(ts)` returns an array of test case objects that are at the first level of the specified test suite.

### Input Arguments

**ts — Test suite**

`sltest.testmanager.TestSuite` object

Test suite with the test cases you want to get, specified as an `sltest.testmanager.TestSuite` object.

### 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.

### Examples

#### Get Test Case Object

```
% Create a test file with default test suite and case
tf = sltest.testmanager.TestFile('My Test File');
```

```
% Get test suite
ts = getTestSuites(tf);
```

```
% Get the test case object
tc = getTestCases(ts);
```

### See Also

#### Topics

“Create and Run Test Cases with Scripts”

**Introduced in R2015b**

## getTestSuiteByName

**Class:** `sltest.testmanager.TestFile`

**Package:** `sltest.testmanager`

Get test suite object by name

### Syntax

```
ts = getTestSuiteByName(tf,name)
```

### Description

`ts = getTestSuiteByName(tf,name)` returns a test suite with the specified name.

### Input Arguments

**tf — Test file**

`sltest.testmanager.TestFile` object

Test file that contains the test suite, specified as a `sltest.testmanager.TestFile` object.

**name — Test suite name**

character vector

The name of the test suite within the test file, specified as a character vector. If the name does not match a test suite, then the function returns an empty test suite object.

Example: 'Test Suite 5'

### 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.

### Examples

**Get Test Suite Object**

```
% Create a test file with default test suite
tf = sltest.testmanager.TestFile('My Test File');
```

```
% Get the test suite object by test suite name
ts = getTestSuiteByName(tf, 'New Test Suite 1');
```

## See Also

### Topics

“Create and Run Test Cases with Scripts”

**Introduced in R2015b**

## getTestFileResults

**Class:** `sltest.testmanager.ResultSet`

**Package:** `sltest.testmanager`

Get test suite results object

### Syntax

```
testFileResultArray = getTestFileResults(result)
```

### Description

`testFileResultArray = getTestFileResults(result)` gets the test file results that are direct children of the results set object.

### Input Arguments

**result — Results set**

`sltest.testmanager.ResultSet` object

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

### Output Arguments

**testFileResultArray — Test file results**

`sltest.testmanager.TestFileResult` object array

Test file results, returned as an array of `sltest.testmanager.TestFileResult` objects. The function returns objects that are direct children of the results set input object.

### Examples

#### Get Test File Result Data

Use the function `sltest.testmanager.run` to return a result set that contains test file, test suite, and test case results.

```
% Run tests in the Test Manager
result = sltest.testmanager.run;
```

```
% Get the test file results
testSFileResultArray = getTestFileResults(result);
```

```
% Get the test suite results
testSuiteResultArray = getTestSuiteResults(result);
```



```
% Get the test case results
testCaseResultArray = getTestCaseResults(result);
```

## See Also

sltest.testmanager.ResultSet | sltest.testmanager.TestSuiteResult |  
sltest.testmanager.TestCaseResult

## Topics

“Create and Run Test Cases with Scripts”

**Introduced in R2016a**

## getTestSuiteByName

**Class:** `sltest.testmanager.TestSuite`

**Package:** `sltest.testmanager`

Get test suite object by name

### Syntax

```
tsOut = getTestSuiteByName(tsIn,name)
```

### Description

`tsOut = getTestSuiteByName(tsIn,name)` returns a test suite with the specified name.

### Input Arguments

**tsIn — Test suite**

`sltest.testmanager.TestSuite` object

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

**name — Test suite name**

character vector

The name of the test suite within a test suite object, specified as a character vector. If the name does not match a test suite, then the function returns an empty test suite object.

### Output Arguments

**tsOut — Test suite**

`sltest.testmanager.TestSuite` 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.

### Examples

**Get Test Suite Object**

```
% Create a test file and new test suite
tf = sltest.testmanager.TestFile('API Test File');
ts = getTestSuiteByName(tf,'New Test Suite 1');

% Create new test suite
createTestSuite(ts,'API Test Suite');
```

```
% Get test suite by name
ts2 = getTestSuiteByName(ts, 'API Test Suite');
```

## See Also

### Topics

“Create and Run Test Cases with Scripts”

**Introduced in R2015b**

## getTestSuites

**Class:** `sltest.testmanager.TestFile`

**Package:** `sltest.testmanager`

Get test suites at first level of test file

### Syntax

```
tsArray = getTestSuites(tf)
```

### Description

`tsArray = getTestSuites(tf)` returns an array of test suite objects that are at the first level of the test file.

### Input Arguments

**tf — Test file**

`sltest.testmanager.TestFile` object

Test file that contains the test suites, specified as a `sltest.testmanager.TestFile` object.

### 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.

### Examples

#### Get Test Suite Object

```
% Create a test file with default test suite
tf = sltest.testmanager.TestFile('My Test File');
```

```
% Get the test suite object from the test file
ts = getTestSuites(tf);
```

### See Also

#### Topics

“Create and Run Test Cases with Scripts”

**Introduced in R2015b**

# getTestSuites

**Class:** `sltest.testmanager.TestSuite`

**Package:** `sltest.testmanager`

Get test suites at first level of test suite

## Syntax

```
tsArray = getTestSuites(ts)
```

## Description

`tsArray = getTestSuites(ts)` returns an array of test suite objects that are at the first level of the specified test suite.

## Input Arguments

### **ts** — Test suite

`sltest.testmanager.TestSuite` object

Test suite that contains the test suite you want to get, specified as an `sltest.testmanager.TestSuite` object.

## 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.

## Examples

### Get Test Suite Object

```
% Create a test file with default test suite
tf = sltest.testmanager.TestFile('My Test File');
```

```
% Get the default test suite
ts1 = getTestSuites(tf);
```

```
% Add a test suite to default test suite
ts2 = createTestSuite(ts1, 'New Test Suite 2');
```

```
% Get the test suite object from the test file
```

```
% as a new object
tsNew = getTestSuites(tsl);
```

## **See Also**

### **Topics**

“Create and Run Test Cases with Scripts”

**Introduced in R2015b**

# getTestSuiteResults

**Class:** `sltest.testmanager.ResultSet`

**Package:** `sltest.testmanager`

Get test suite results object

## Syntax

```
testSuiteResultArray = getTestSuiteResults(result)
```

## Description

`testSuiteResultArray = getTestSuiteResults(result)` gets the test suite results that are direct children of the results set object.

## Input Arguments

**result — Results set**

`sltest.testmanager.ResultSet` object

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

## Output Arguments

**testSuiteResultArray — Test suite results**

`sltest.testmanager.TestSuiteResult` object array

Test suite results, returned as an array of `sltest.testmanager.TestSuiteResult` objects. The function returns objects that are direct children of the results set object.

## Examples

### Get Test Suite Result Data

Use the functions `sltest.testmanager.run` to return a result set that contains test file, test suite, and test case results.

```
% Run tests in the Test Manager
result = sltest.testmanager.run;
```

```
% Get the test file results
testSFileResultArray = getTestFileResults(result);
```

```
% Get the test suite results
testSuiteResultArray = getTestSuiteResults(result);
```

```
% Get the test case results
testCaseResultArray = getTestCaseResults(result);
```

## **See Also**

### **Topics**

“Create and Run Test Cases with Scripts”

**Introduced in R2015a**



# getTestSuiteResults

**Class:** `sltest.testmanager.TestFileResult`

**Package:** `sltest.testmanager`

Get test suite results object

## Syntax

```
testSuiteResultArray = getTestSuiteResults(result)
```

## Description

`testSuiteResultArray = getTestSuiteResults(result)` gets the test suite results that are direct children of the test file results object.

## Input Arguments

### **result — Test file results**

`sltest.testmanager.TestFileResult` object

Test file results to get test suite results from, specified as a `sltest.testmanager.TestFileResult` object.

## Output Arguments

### **testSuiteResultArray — Test suite results**

`sltest.testmanager.TestSuiteResult` object array

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

## Examples

### **Get Test Suite Result Data**

Use the function `sltest.testmanager.run` to return a result set that contains test file, test suite, and test case results.

```
% Run tests in the Test Manager
result = sltest.testmanager.run;

% Get the test file results
testFileResultArray = getTestFileResults(result);
```

```
% Get the test suite results
testSuiteResultArray = getTestSuiteResults(testFileResultArray);
```

## **See Also**

### **Topics**

“Create and Run Test Cases with Scripts”

**Introduced in R2016a**

# getTestSuiteResults

**Class:** `sltest.testmanager.TestSuiteResult`

**Package:** `sltest.testmanager`

Get test suite results object

## Syntax

```
testSuiteResultArray = getTestSuiteResults(result)
```

## Description

`testSuiteResultArray = getTestSuiteResults(result)` gets the test suite results that are direct children of the test suite results object.

## Input Arguments

**resultObj — Test suite results**

`sltest.testmanager.TestSuiteResult` object

Test suite results to get test suite results from, specified as a `sltest.testmanager.TestSuiteResult` object.

## Output Arguments

**testSuiteResultArray — Test suite results**

`sltest.testmanager.TestSuiteResult` object array

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

## Examples

### Get Test Suite Result Data

Use the functions `sltest.testmanager.run` to return a result set that contains test file, test suite, and test case results.

```
% Run tests in the Test Manager
result = sltest.testmanager.run;

% Get the test file results
testFileResultArray = getTestFileResults(result);

% Get the test suite results
testSuiteResultArray = getTestSuiteResults(testFileResultArray);
```

```
% Get the next level of test suite results
testSuite2ResultArray = getTestSuiteResults(testSuiteResultArray);
```

## **See Also**

### **Topics**

“Create and Run Test Cases with Scripts”

**Introduced in R2015a**

# getVerifyRuns

**Class:** `sltest.testmanager.TestCaseResult`

**Package:** `sltest.testmanager`

Get test case verify statement

## Syntax

```
dataset = getVerifyRuns(result)
```

## Description

`dataset = getVerifyRuns(result)` gets the verify statement dataset from a test case result. Verify statements are constructed in the Test Sequence or Test Assessment blocks in the system under test.

## Input Arguments

### **result** — Test case result

`sltest.testmanager.TestCaseResult` object

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

## Output Arguments

### **dataset** — Verify statement dataset

`Simulink.sdi.Run` object array

Test case verify statement dataset, returned as an array of `Simulink.sdi.Run` objects.

## Examples

### Get Verify Output From Test Case

This example shows how to obtain the verify statement dataset from a test case result. You use verify statements in the Test Sequence or Test Assessment blocks in the system under test. You can use the output, which is an array of objects, with `Simulink.sdi.Run`.

```
rollModel = 'TransmissionDownshiftTestSeqVerify';
testHarness = 'TransmissionDownshiftTestSeqVerify_Harness1';
testFile = 'TransmissionDownshiftTestSeqVerify.mldatx';

open_system(rollModel);
testFile = sltest.testmanager.load(testFile);
sltest.harness.open([rollModel, '/shift_controller'], testHarness);

open_system([testHarness, '/Test Sequence'])
```

```
open_system([testHarness, '/Test Assessment Block'])

ro = run(testFile);
tfr = getTestFileResults(ro);
tsr = getTestSuiteResults(tfr);
tcr = getTestCaseResults(tsr);

verifyOut = getVerifyRuns(tcr);

close_system('TransmissionDownshiftTestSeqVerify_Harness1')
close_system('TransmissionDownshiftTestSeqVerify')
sltest.testmanager.clear
sltest.testmanager.clearResults
sltest.testmanager.close
```

## See Also

Simulink.sdi.Run

## Topics

“Create and Run Test Cases with Scripts”

**Introduced in R2016a**

# getVerifyRuns

**Class:** `sltest.testmanager.TestIterationResult`

**Package:** `sltest.testmanager`

Get test iteration verify statement

## Syntax

```
dataset = getVerifyRuns(result)
```

## Description

`dataset = getVerifyRuns(result)` gets the verify statement dataset from a test iteration result. Verify statements are made in the Test Sequence or Test Assessment blocks in the system under test.

## Input Arguments

**result — Test iteration result**

`sltest.testmanager.TestIterationResult` object

Test iteration results to get verify statement dataset from, specified as a `sltest.testmanager.TestIterationResult` object.

## Output Arguments

**dataset — Verify statement dataset**

`Simulink.sdi.Run` object

Test iteration verify statement dataset, returned as a `Simulink.sdi.Run` object.

## See Also

`Simulink.sdi.Run` | `verify`

## Topics

“Create and Run Test Cases with Scripts”

**Introduced in R2016a**

## layoutReport

**Class:** sltest.testmanager.TestResultReport

**Package:** sltest.testmanager

Incorporates parts of report into one document

### Syntax

```
layoutReport(obj)
```

### Description

layoutReport(obj) incorporates the report parts into one document. The report is divided into three main parts: title page, table of contents, and the main body.

This method also calls:

- addTitlePage
- addReportTOC
- addReportBody

### Input Arguments

**obj — Test report object**

sltest.testmanager.TestResultReport object

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

### See Also

sltest.testmanager.TestResultReport

### Topics

“Export Test Results”

“Create and Run Test Cases with Scripts”

**Introduced in R2016a**



# map

**Class:** `sltest.testmanager.TestInput`

**Package:** `sltest.testmanager`

Map test input to system under test

## Syntax

`map(input,Name,Value)`

## Description

`map(input,Name,Value)` maps the test input data `input` to the system under test.

## Input Arguments

### **input** — Test input

`sltest.testmanager.TestInput` object

The test input to map, specified as a `sltest.testmanager.TestInput` object.

### **Name-Value Pair Arguments**

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

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

Example: `'Mode',4,'CustomFunction','mapfcn'`

### **Mode** — Mapping mode

`0 | 1 | 2 | 3 | 4`

Mapping mode, specified as the comma-separated pair consisting of `'mode'` and an integer corresponding to the desired mapping mode:

- `0` — Block name
- `1` — Block path
- `2` — Signal name
- `3` — Port order (index)
- `4` — Custom

For more information on mapping modes, see “Map Root Inport Signal Data”.

Example: `'Mode',2`

### **CustomFunction** — Custom mapping function name

character vector

Name of function used for custom mapping, specified as the comma-separated pair consisting of 'customFunction' and a character vector. This argument is optional and valid only when mode is set to 4.

Example: 'CustomFunction','mapfcn'

### **CompileModel — Model compilation for mapping**

true (default) | false

Option to compile or not compile the model when performing input mapping, specified as the comma-separated pair consisting of 'CompileModel' and false or true.

Example: 'CompileModel',false

## **Examples**

### **Add Microsoft® Excel® Data as Input**

This example shows how to add data from a Microsoft® Excel® spreadsheet and map it to a test case. Only the two sheets that have data are added and mapped.

#### **Load the Example Model**

```
open_system('sltestExcelExample');
```

#### **Create a New Test File**

```
tf = sltest.testmanager.TestFile('input_test_file.mldatx');
```

#### **Get the Test Suite and Test Case Objects**

```
ts = getTestSuites(tf);
tc = getTestCases(ts);
```

#### **Add the Example Model as the System Under Test**

```
setProperty(tc, 'Model', 'sltestExcelExample');
```

#### **Add Excel® Data to Inputs Section and Specify Sheets to Add**

```
excelfile = 'sltestExampleInputs.xlsx';
input = addInput(tc, excelfile, 'Sheets', ["Acceleration", "Braking"]);
```

#### **Map the Input Signal for the Sheets by Block Name**

```
map(input(1), 0);
map(input(2), 0);
```

## **See Also**

### **Topics**

“Create and Run Test Cases with Scripts”

### **Introduced in R2015b**

# plot

**Package:** Simulink.SimulationData

Plot simulation output data in the Simulation Data Inspector

## Syntax

```
plot(simOutObj)
runObj = plot(simOutObj)
```

## Description

`plot(simOutObj)` plots the simulation output data in the simulation output object, `simOutObj`, in the Simulation Data Inspector and opens the Simulation Data Inspector so you can view the plotted simulation output data. You can use the `plot` function to plot simulation results stored in these simulation output objects:

- `Simulink.SimulationOutput`
- `Simulink.SimulationData.DataStoreMemory`
- `Simulink.SimulationData.Parameter`
- `Simulink.SimulationData.Signal`
- `Simulink.SimulationData.State`
- `Stateflow.SimulationData.Data`
- `Stateflow.SimulationData.State`
- `sltest.Assessment`

These simulation output objects also have plot functions that plot the data in and then open the Simulation Data Inspector:

- `Simulink.SimulationData.Dataset`
- `Simulink.SimulationData.DatasetRef`

When a simulation output object contains fewer than eight signals to plot, the Simulation Data Inspector layout changes to 1-by- $n$ , where  $n$  is the number of signals to plot, and plots one signal on each subplot. When the simulation output object contains more than eight signals to plot, the Simulation Data Inspector layout changes to 1-by-1 and plots the first signal in the simulation output object.

When some or all of the data in a `Simulink.SimulationOutput` object is in a Simulation Data Inspector run, the `plot` function opens the Simulation Data Inspector and plots all the signals in the run. When the data does not correspond to a run in the Simulation Data Inspector, the `plot` function imports the data to a new run. When you use the `plot` function to plot a single signal, the `plot` function always imports the data for the signal to a new run.

`runObj = plot(simOutObj)` returns the `Simulink.sdi.Run` object corresponding to the plotted data.

## Examples

### Access and Plot Simulation Output Data

The `ex_vdp_simout_plot` model used in this example is configured to log signals, outputs, and states and return all logged data in a single simulation output. The single simulation output is a `Simulink.SimulationOutput` object that contains one or more types of other simulation output objects, depending on the kinds of data you log. This example shows how to access each type of logged data and use the `plot` function to plot the data in the Simulation Data Inspector. To generate the `SimulationOutput` object containing all the logged data, simulate the model.

```
open_system('ex_vdp_simout_plot.slx')
out = sim('ex_vdp_simout_plot.slx');
```

### Plot Data in a SimulationOutput Object

You can pass the single simulation output, stored in a `Simulink.SimulationOutput` object, to the `plot` function to view the data in the Simulation Data Inspector. When you plot data in a `SimulationOutput` object that corresponds to a run in the Simulation Data Inspector, data in the object that also logs to the Simulation Data Inspector is plotted. The model logs data using the `Dataset` format, so all logged data streams to the Simulation Data Inspector.

When you use the `plot` function to plot the data, the Simulation Data Inspector updates to a 1-by-8 layout and plots one signal on each subplot.

```
plot(out)
```

### Plot Data for a Single Signal

When you plot the data for a single signal, the Simulation Data Inspector always imports the data for the signal to a new run. Use the `get` function for the `SimulationOutput` object to access the signal logging `Dataset` object, `logout`.

```
logout = get(out, 'logout');
```

Then, use the `get` function for the `Dataset` object to access the data for the first element.

```
sig1 = get(logout, 1);
```

When you plot the data for the signal, the Simulation Data Inspector imports the signal to a new run, updates the layout to 1-by-1, and plots the signal.

```
plot(sig1)
```

## Input Arguments

### **simOutObj** — Simulation output object containing simulation data to plot

`Simulink.SimulationOutput` | `Simulink.SimulationData.DataStoreMemory` |  
`Simulink.SimulationData.Parameter` | `Simulink.SimulationData.Signal` |  
`Simulink.SimulationData.State` | `sltest.Assessment` |  
`Stateflow.SimulationData.Data` | `Stateflow.SimulationData.State`

Simulation output object containing data you want to plot and view in the Simulation Data Inspector. This `plot` function supports these simulation output objects:

- `Simulink.SimulationOutput`
- `Simulink.SimulationData.DataStoreMemory`
- `Simulink.SimulationData.Parameter`
- `Simulink.SimulationData.Signal`
- `Simulink.SimulationData.State`
- `Stateflow.SimulationData.Data`
- `Stateflow.SimulationData.State`
- `sltest.Assessment`

Example: `plot(out)`

## Output Arguments

### **runObj** — Run that corresponds to plotted data

`Simulink.sdi.Run` object

Run that corresponds to the plotted data, returned as a `Simulink.sdi.Run` object.

## See Also

`plot`

## Topics

“View Data in the Simulation Data Inspector”

## Introduced in R2019b

## plotOneSignalToFile

**Class:** `sltest.testmanager.TestResultReport`

**Package:** `sltest.testmanager`

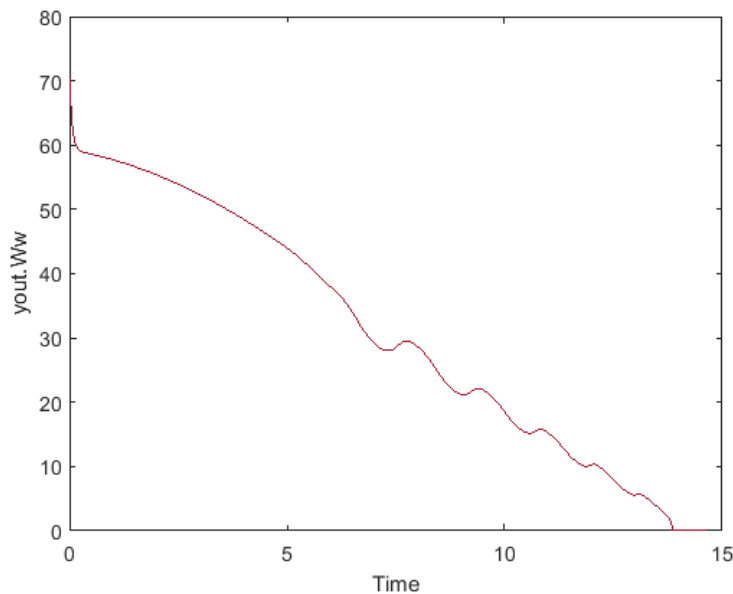
Save signal plot to file

### Syntax

```
plotOneSignalToFile(obj, filePath, onesig, isComparison)
```

### Description

`plotOneSignalToFile(obj, filePath, onesig, isComparison)` saves a signal plot to a PNG image file. If the signal plot is a comparison, then the baseline signal, the difference, and the tolerance are plotted in the same plot.



### Input Arguments

**obj** — Test report object

`sltest.testmanager.TestResultReport` object

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

**filePath** — Image file path

character vector

File path and name of the image you want to save, specified as a character vector.

**onesig — Result signal**`sltest.testmanager.ReportUtility.Signal` object

The result signal, specified as a `sltest.testmanager.ReportUtility.Signal` object.

**isComparison — Comparison indicator**`true | false`

Flag to indicate whether the signal is from a comparison run or not, specified as a Boolean, `true` or `false`.

**See Also**`sltest.testmanager.TestResultReport`**Topics**

“Export Test Results”

“Create and Run Test Cases with Scripts”

**Introduced in R2016a**

## remove

**Class:** `sltest.testmanager.BaselineCriteria`

**Package:** `sltest.testmanager`

Remove baseline criteria

### Syntax

```
remove(bc)
```

### Description

`remove(bc)` removes the baseline criteria from a test case. The baseline criteria object is empty after a call to this function.

### Input Arguments

**bc — Baseline criteria**

`sltest.testmanager.BaselineCriteria` object

Baseline criteria that you want to remove from a test case, specified as a `sltest.testmanager.BaselineCriteria` object.

### Examples

#### Remove Baseline Criteria

```
% Open the model for this example
openExample('sldemo_absbrake');

% Create the test file, test suite, and test case structure
tf = sltest.testmanager.TestFile('API Test File');
ts = createTestSuite(tf, 'API Test Suite');
tc = createTestCase(ts, 'baseline', 'Baseline API Test Case');

% Remove the default test suite
tsDel = getTestSuiteByName(tf, 'New Test Suite 1');
remove(tsDel);

% Assign the system under test to the test case
setProperty(tc, 'Model', 'sldemo_absbrake');

% Capture the baseline criteria
baseline = captureBaselineCriteria(tc, 'baseline_API.mat', true);
```



```
% Remove baseline criteria;
remove(baseline);
```

## **See Also**

### **Topics**

“Create and Run Test Cases with Scripts”

**Introduced in R2015b**

## remove

**Class:** `sltest.testmanager.EquivalenceCriteria`

**Package:** `sltest.testmanager`

Remove equivalence criteria

### Syntax

```
remove(eq)
```

### Description

`remove(eq)` removes the equivalence criteria from a test case. The equivalence criteria object is empty after a call to this function.

### Input Arguments

**eq** — **Equivalence criteria**

`sltest.testmanager.EquivalenceCriteria` object

Equivalence criteria that you want to remove from a test case, specified as a `sltest.testmanager.EquivalenceCriteria` object.

### Examples

#### Remove Equivalence Criteria

```
% Open the model for this example
openExample('sldemo_absbrake');

% Create the test file, test suite, and test case structure
tf = sltest.testmanager.TestFile('API Test File');
ts = createTestSuite(tf,'API Test Suite');
tc = createTestCase(ts,'equivalence','Equivalence Test Case');

% Remove the default test suite
tsDel = getTestSuiteByName(tf,'New Test Suite 1');
remove(tsDel);

% Assign the system under test to the test case
% for Simulation 1 and Simulation 2
setProperty(tc,'Model','sldemo_absbrake','SimulationIndex',1);
setProperty(tc,'Model','sldemo_absbrake','SimulationIndex',2);

% Add a parameter override to Simulation 1 and 2
ps1 = addParameterSet(tc,'Name','Parameter Set 1','SimulationIndex',1);
po1 = addParameterOverride(ps1,'Rr',1.20);

ps2 = addParameterSet(tc,'Name','Parameter Set 2','SimulationIndex',2);
po2 = addParameterOverride(ps2,'Rr',1.24);
```

---

```
% Capture equivalence criteria
eq = captureEquivalenceCriteria(tc);

% Set the equivalence criteria tolerance for one signal
sc = getSignalCriteria(eq);
sc(1).AbsTol = 2.2;

% Remove second signal criteria from baseline
remove(eq);
```

## See Also

### Topics

“Create and Run Test Cases with Scripts”

**Introduced in R2015b**

## remove

**Class:** `sltest.testmanager.LoggedSignal`

**Package:** `sltest.testmanager`

Remove a logged signal

### Syntax

```
remove(obj)
```

### Description

`remove(obj)` removes an `sltest.testmanager.LoggedSignal` object from an `sltest.testmanager.LoggedSignalSet` object and invalidates the `LoggedSignal` object.

### Input Arguments

**obj** — **Logged signal**

`sltest.testmanager.LoggedSignal` object

Logged signal object contained in a logged signal set.

### Examples

#### Remove a Signal from a Signal Set

Open a model and create a signal set.

```
openExample('sldemo_absbrake')

% Create test case
tf = sltest.testmanager.TestFile(strcat(pwd, '\mytf.mldatx'));
ts = sltest.testmanager.TestSuite(tf, 'myts');
tc = sltest.testmanager.TestCase(ts, 'baseline', 'mytc');

% Create signal set
lgset = tc.addLoggedSignalSet;

Select the Vehicle Speed block and enter gcb. Use the returned path to create a
Simulink.BlockPath object.

% Add signals to set
bPath = Simulink.BlockPath('sldemo_absbrake/Vehicle speed');
sig1 = lgset.addLoggedSignal(bPath,1);
sig2 = lgset.addLoggedSignal(bPath,2);

setProperty(tc, 'Model', 'sldemo_absbrake');

% Remove signal
remove(sig2);
```

```
% Check that signal is removed
sigs = lgset.getLoggedSignals
```

## See Also

gcb

## Topics

“Create and Run Test Cases with Scripts”

“Capture Simulation Data in a Test Case”

**Introduced in R2019a**

## remove

**Class:** `sltest.testmanager.LoggedSignalSet`

**Package:** `sltest.testmanager`

Remove a logged signal set

### Syntax

```
remove(lgset)
```

### Description

`remove(lgset)` removes an `sltest.testmanager.LoggedSignalSet` object from an `sltest.testmanager.TestCase` object and invalidates its child signal objects.

### Input Arguments

**lgset — Logged signal set**

`sltest.testmanager.LoggedSignalSet` object

Logged signal set object contained in a test case.

### Examples

#### Remove a Signal Set from a Test Case

Open a model and create a test case.

```
openExample('sldemo_absbrake')

% Create test case
tf = sltest.testmanager.TestFile(strcat(pwd, '\mytf.mldatx'));
ts = sltest.testmanager.TestSuite(tf, 'myts');
tc = sltest.testmanager.TestCase(ts, 'baseline', 'mytc');

% Create signal set
mylgset = tc.addLoggedSignalSet;

% Remove the signal set
remove(mylgset)
```

### See Also

`sltest.testmanager.TestCase`

#### Topics

“Create and Run Test Cases with Scripts”

“Capture Simulation Data in a Test Case”

**Introduced in R2019a**

## remove

**Class:** `sltest.testmanager.ParameterOverride`

**Package:** `sltest.testmanager`

Remove parameter override

### Syntax

```
remove(po)
```

### Description

`remove(po)` removes the parameter override from the parameter set. The parameter override object is empty after a call to this function.

### Input Arguments

**po — Parameter override**

`sltest.testmanager.ParameterOverride` object

Parameter override that you want to remove from a parameter set, specified as a `sltest.testmanager.ParameterOverride` object.

### Examples

#### Remove Parameter Override from Parameter Set

```
% Open the model for this example
openExample('sldemo_absbrake');

% Create the test file, test suite, and test case structure
tf = sltest.testmanager.TestFile('API Test File');
ts = createTestSuite(tf,'API Test Suite');
tc = createTestCase(ts,'baseline','Baseline API Test Case');

% Remove the default test suite
tsDel = getTestSuiteByName(tf,'New Test Suite 1');
remove(tsDel);

% Assign the system under test to the test case
setProperty(tc,'Model','sldemo_absbrake');

% Capture the baseline criteria
baseline = captureBaselineCriteria(tc,'baseline_API.mat',true);

% Test a new model parameter by overriding it in the test case
% parameter set
ps = addParameterSet(tc,'Name','API Parameter Set');
po = addParameterOverride(ps,'m',55);
```



```
% Remove parameter override from parameter set
remove(po);
```

## **See Also**

### **Topics**

“Create and Run Test Cases with Scripts”

**Introduced in R2015b**

## remove

**Class:** sltest.testmanager.ParameterSet

**Package:** sltest.testmanager

Remove parameter set

### Syntax

```
remove(ps)
```

### Description

`remove(ps)` removes the parameter set from a test case. The parameter set object is empty after a call to this function.

### Input Arguments

**ps — Parameter set**

sltest.testmanager.ParameterSet object

Parameter set that you want to remove from a test case, specified as a sltest.testmanager.ParameterSet object.

### Examples

#### Remove Parameter Set from Test Case

```
% Open the model for this example
openExample('sldemo_absbrake');

% Create the test file, test suite, and test case structure
tf = sltest.testmanager.TestFile('API Test File');
ts = createTestSuite(tf,'API Test Suite');
tc = createTestCase(ts,'baseline','Baseline API Test Case');

% Remove the default test suite
tsDel = getTestSuiteByName(tf,'New Test Suite 1');
remove(tsDel);

% Assign the system under test to the test case
setProperty(tc,'Model','sldemo_absbrake');

% Capture the baseline criteria
baseline = captureBaselineCriteria(tc,'baseline_API.mat',true);

% Test a new model parameter by overriding it in the test case
% parameter set
ps = addParameterSet(tc,'Name','API Parameter Set');
po = addParameterOverride(ps,'m',55);
```

```
% Remove parameter set from test case
remove(ps);
```

## **See Also**

### **Topics**

“Create and Run Test Cases with Scripts”

**Introduced in R2015b**

## remove

**Class:** sltest.testmanager.SignalCriteria

**Package:** sltest.testmanager

Remove signal criteria

### Syntax

```
remove(sc)
```

### Description

remove(sc) removes signal criteria from the baseline or equivalence criteria set. The signal criteria object is empty after a call to this function.

### Input Arguments

**sc — Signal criteria**

sltest.testmanager.SignalCriteria object

Signal criteria that you want to remove from a baseline or equivalence criteria set, specified as a sltest.testmanager.SignalCriteria object.

### Examples

#### Remove Signal from Baseline Criteria Set

```
% Open the model for this example
openExample('sldemo_absbrake');

% Create the test file, test suite, and test case structure
tf = sltest.testmanager.TestFile('API Test File');
ts = createTestSuite(tf,'API Test Suite');
tc = createTestCase(ts,'baseline','Baseline API Test Case');

% Remove the default test suite
tsDel = getTestSuiteByName(tf,'New Test Suite 1');
remove(tsDel);

% Assign the system under test to the test case
setProperty(tc,'Model','sldemo_absbrake');

% Capture the baseline criteria
baseline = captureBaselineCriteria(tc,'baseline_API.mat',true);

% Test a new model parameter by overriding it in the test case
% parameter set
ps = addParameterSet(tc,'Name','API Parameter Set');
po = addParameterOverride(ps,'m',55);
```

```
% Set the baseline criteria tolerance for one signal
sc = getSignalCriteria(baseline);
sc(1).AbsTol = 9;

% Remove second signal criteria from baseline
remove(sc(2));
```

## See Also

### Topics

“Create and Run Test Cases with Scripts”

**Introduced in R2015b**

## remove

**Class:** `sltest.testmanager.TestCase`

**Package:** `sltest.testmanager`

Remove test case

### Syntax

```
remove(tc)
```

### Description

`remove(tc)` removes the test case. The test case object is empty after a call to this function. Parameter overrides, baseline criteria, or equivalence criteria associated with the test case become invalid.

### Input Arguments

**tc — Test case**

`sltest.testmanager.TestCase` object

Test case to remove, specified as an `sltest.testmanager.TestCase` object.

### 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
tc = sltest.testmanager.TestCase(ts, 'equivalence', ...
 'Eq Test Case');

% Remove the test case
remove(tc);
```

### See Also

#### Topics

“Create and Run Test Cases with Scripts”

**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.

### Input Arguments

**input** — Test input

`sltest.testmanager.TestInput` object

The test input that you want to remove, specified as a `sltest.testmanager.TestInput` object.

### Examples

#### Remove Test Input Data

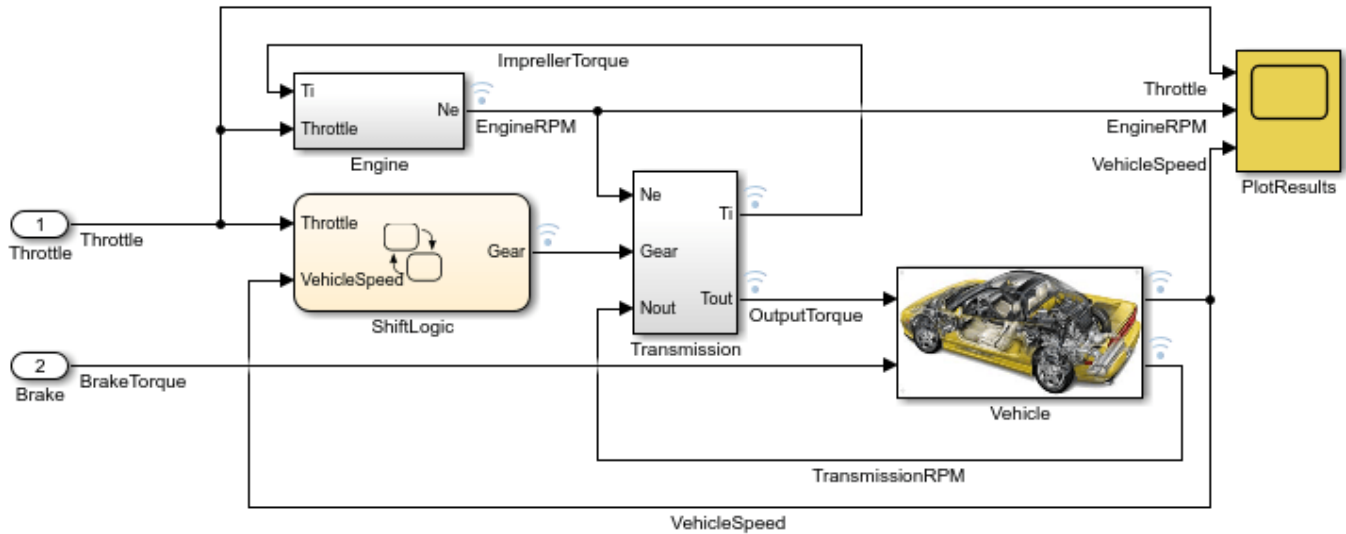
Remove an Excel test input sheet

#### Load Example Model, Create New Test File, and Get Suites and Cases

```
open_system('sltestExcelExample');
tf = sltest.testmanager.TestFile('input_test_file.mldatx');
ts = getTestSuites(tf);
tc = getTestCases(ts);
```

## Importing Microsoft Excel Data

This model is used to demonstrate the use of loading input data from Microsoft® Excel® file in test manager. To see the demo, execute `showdemo sltestUsingExcel` in MATLAB(R).



Copyright 2015 The MathWorks, Inc.

### Add the Model as the System Under Test

```
setProperty(tc, 'Model', 'sltestExcelExample');
```

### Add Excel Data to Inputs Section

```
excelfile = 'sltestExampleInputs.xlsx';
input = addInput(tc, excelfile);
```

### Map the First Input Signal By Block Name

```
map(input(1), 0);
```

### Remove the Second Input

```
remove(input(2));
```

## See Also

### Topics

“Create and Run Test Cases with Scripts”

### Introduced in R2015b



## remove

**Class:** `sltest.testmanager.TestSuite`

**Package:** `sltest.testmanager`

Remove test suite

### Syntax

```
remove(ts)
```

### Description

`remove(ts)` removes the test suite. The test suite object is empty after a call to this function.

### Input Arguments

**ts** — Test suite

`sltest.testmanager.TestSuite` object

Test suite that you want to remove, specified as an `sltest.testmanager.TestSuite` object.

### 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
remove(ts);
```

### See Also

#### Topics

“Create and Run Test Cases with Scripts”

**Introduced in R2015b**

## run

**Class:** `sltest.testmanager.TestCase`

**Package:** `sltest.testmanager`

Run test case

### Syntax

```
resultObj = run(tc)
resultObj = run(tc,"Parallel",setting)
```

### Description

`resultObj = run(tc)` runs the test case.

`resultObj = run(tc,"Parallel",setting)` runs test case with or without Parallel Computing Toolbox or MATLAB Parallel Server.

### Input Arguments

#### **tc** — Test case

`sltest.testmanager.TestCase` object

Test case you want to run, specified as an `sltest.testmanager.TestCase` object.

#### **setting** — Whether to run with parallel computing

`false` (default) | `true`

Whether to run the test case with Parallel Computing Toolbox or MATLAB Parallel Server, specified as either `false` or `true`. Requires Parallel Computing Toolbox or MATLAB Parallel Server license, respectively.

Example: `run(tc,"Parallel",true)`

Data Types: `logical`

### Output Arguments

#### **resultObj** — Results set object

object

Test results, returned as a results set object, `sltest.testmanager.ResultSet`.

### Examples

#### **Run a Test Case**

Open the model for this example.

```
openExample('sldemo_autotrans');
```

Create the test file, test suite, and test case structure.

```
tf = sltest.testmanager.TestFile("API Test File");
ts = createTestSuite(tf, "API Test Suite");
tc = createTestCase(ts, "simulation", "Simulation Test Case");
```

Remove the default test suite.

```
tsDel = getTestSuiteByName(tf, "New Test Suite 1");
remove(tsDel);
```

Assign the system under test to the test case.

```
setProperty(tc, "Model", "sldemo_autotrans");
```

Run the test case and return the results data.

```
ro = run(tc);
```

## Run the Test Case with Parallel Computing Toolbox

Open the model for this example.

```
openExample('sldemo_autotrans');
```

Create the test file, test suite, and test case structure.

```
tf = sltest.testmanager.TestFile("API Test File");
ts = createTestSuite(tf, "API Test Suite");
tc = createTestCase(ts, "simulation", "Simulation Test Case");
```

Remove the default test suite.

```
tsDel = getTestSuiteByName(tf, "New Test Suite 1");
remove(tsDel);
```

Assign the system under test to the test case.

```
setProperty(tc, "Model", "sldemo_autotrans");
```

Run the test case with Parallel Computing Toolbox and return the results data.

```
ro = run(tc, "Parallel", true);
```

## See Also

### Topics

“Create and Run Test Cases with Scripts”

**Introduced in R2015b**

## run

**Class:** `sltest.testmanager.TestFile`

**Package:** `sltest.testmanager`

Run test cases in test file

### Syntax

```
resultObj = run(tf)
resultObj = run(tf,Name,Value)
```

### Description

`resultObj = run(tf)` runs the enabled test cases in the test file.

`resultObj = run(tf,Name,Value)` uses additional options specified by one or more name-value arguments.

### Input Arguments

#### **tf** — Test file

`sltest.testmanager.TestFile` object

Test file with the test cases you want to run, specified as an `sltest.testmanager.TestFile` object.

#### **Name-Value Pair Arguments**

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

Example: `Parallel=true,Tags=["safety","regression"]`

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

Example: `"Parallel",true,"Tags",["safety","regression"]`

#### **Parallel** — Whether to run with parallel computing

`false` (default) | `true`

Whether to run tests with Parallel Computing Toolbox or MATLAB Parallel Server, specified as either `false` or `true`. Requires Parallel Computing Toolbox or MATLAB Parallel Server license, respectively.

Example: `run(tf,"Parallel",true)`

Data Types: `logical`

#### **Tags** — Run only tests with specified tags

string array | cell array of character vectors

Specifies test tags for execution. For more information, see “Tags”.

Example: `run(tf, "Tags", ["safety"])`

Example: `run(tf, "Tags", ["safety", "regression"])`

Data Types: `cell` | `char` | `string`

## Output Arguments

### **resultObj** — Results set object

`sltest.testmanager.ResultSet` object

Test results, returned as a results set object, `sltest.testmanager.ResultSet`.

## Examples

### Run a Test File

Open the model for this example.

```
openExample('sldemo_autotrans');
```

Create the test file, test suite, and test case structure.

```
tf = sltest.testmanager.TestFile("API Test File");
ts = createTestSuite(tf, "API Test Suite");
tc = createTestCase(ts, "simulation", "Simulation Test Case");
```

Remove the default test suite.

```
tsDel = getTestSuiteByName(tf, "New Test Suite 1");
remove(tsDel);
```

Assign the system under test to the test case.

```
setProperty(tc, "Model", "sldemo_autotrans");
```

Run the test file and return the results data.

```
ro = run(tf);
```

### Run the Test File with Parallel Computing Toolbox

Open the model for this example.

```
openExample('sldemo_autotrans');
```

Create the test file, test suite, and test case structure.

```
tf = sltest.testmanager.TestFile("API Test File");
ts = createTestSuite(tf, "API Test Suite");
tc = createTestCase(ts, "simulation", "Simulation Test Case");
```

Remove the default test suite.

```
tsDel = getTestSuiteByName(tf, "New Test Suite 1");
remove(tsDel);
```

Assign the system under test to the test case.

```
setProperty(tc, "Model", "sldemo_autotrans");
```

Run the test file with Parallel Computing Toolbox and return the results data.

```
ro = run(tf, "Parallel", true);
```

## **See Also**

### **Topics**

“Create and Run Test Cases with Scripts”

**Introduced in R2015b**

## run

**Class:** `sltest.testmanager.TestSuite`

**Package:** `sltest.testmanager`

Run test cases in test suite

### Syntax

```
resultObj = run(ts)
resultObj = run(ts,Name,Value)
```

### Description

`resultObj = run(ts)` runs the enabled test cases in the test suite.

`resultObj = run(ts,Name,Value)` uses additional options specified by one or more name-value arguments.

### Input Arguments

#### **ts** — Test suite

`sltest.testmanager.TestSuite` object

Test suite with the test cases you want to run, specified as an `sltest.testmanager.TestSuite` object.

#### **Name-Value Pair Arguments**

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

Example: `Parallel=true,Tags=["safety","regression"]`

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

Example: `"Parallel",true,"Tags",["safety","regression"]`

#### **Parallel** — Whether to run with parallel computing

`false` (default) | `true`

Whether to run tests with Parallel Computing Toolbox or MATLAB Parallel Server, specified as either `false` or `true`. Requires Parallel Computing Toolbox or MATLAB Parallel Server license, respectively.

Example: `run(ts,"Parallel",true)`

Data Types: `logical`

#### **Tags** — Run only tests with specified tags

string array | cell array of character vectors

Specifies test tags for execution. For more information, see “Tags”.

Example: `run(ts, "Tags", ["safety"])`

Example: `run(ts, "Tags", ["safety", "regression"])`

Data Types: `cell` | `char` | `string`

## Output Arguments

### **resultObj** — Result set object

`sltest.testmanager.ResultSet` object

Test results, returned as a `sltest.testmanager.ResultSet` results set object.

## Examples

### Run a Test Suite

Open the model for this example.

```
openExample('sldemo_autotrans');
```

Create the test file, test suite, and test case structure.

```
tf = sltest.testmanager.TestFile("API Test File");
ts = createTestSuite(tf, "API Test Suite");
tc = createTestCase(ts, "simulation", "Simulation Test Case");
```

Remove the default test suite.

```
tsDel = getTestSuiteByName(tf, "New Test Suite 1");
remove(tsDel);
```

Assign the system under test to the test case.

```
setProperty(tc, "Model", "sldemo_autotrans");
```

Run the test suite and return the results data.

```
ro = run(ts);
```

### Run the Test Suite with Parallel Computing Toolbox

Open the model for this example.

```
openExample('sldemo_autotrans');
```

Create the test file, test suite, and test case structure.

```
tf = sltest.testmanager.TestFile("API Test File");
ts = createTestSuite(tf, "API Test Suite");
tc = createTestCase(ts, "simulation", "Simulation Test Case");
```

Remove the default test suite.



---

```
tsDel = getTestSuiteByName(tf, "New Test Suite 1");
remove(tsDel);
```

Assign the system under test to the test case.

```
setProperty(tc, "Model", "sldemo_autotrans");
```

Run the test suite with Parallel Computing Toolbox and return the results data.

```
ro = run(ts, "Parallel", true);
```

## See Also

### Topics

“Create and Run Test Cases with Scripts”

**Introduced in R2015b**

## saveToFile

**Class:** `sltest.testmanager.TestFile`

**Package:** `sltest.testmanager`

Save test file

### Syntax

```
saveToFile(tf)
saveToFile(tf, filePath)
```

### Description

`saveToFile(tf)` saves the changes to the test file.

`saveToFile(tf, filePath)` saves the test file to the specified file path.

### Input Arguments

#### **tf** — Test file

`sltest.testmanager.TestFile` object

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

#### **filePath** — File path

character vector

The file path and name to save the test file at, specified as a character vector.

Example: `'C:\MATLAB\New Test File.mldatx'`

### Examples

#### Save Test File With Changes

```
% Open the model for this example
openExample('sldemo_autotrans');

% Create the test file, test suite, and test case structure
tf = sltest.testmanager.TestFile('C:\MATLAB\My Test File.mldatx');
ts = createTestSuite(tf, 'My Test Suite');
tc = createTestCase(ts, 'simulation', 'Simulation Test Case');

% Remove the default test suite
tsDel = getTestSuiteByName(tf, 'New Test Suite 1');
remove(tsDel);

% Assign the system under test to the test case
setProperty(tc, 'Model', 'sldemo_autotrans');
```

```
% Save the test file
saveToFile(tf);

% Save test file object as another test file
saveToFile(tf, 'C:\MATLAB\New Test File.mldatx');
```

## See Also

### Topics

“Create and Run Test Cases with Scripts”

**Introduced in R2015b**

## setAssessmentsCallback

**Class:** `sltest.testmanager.TestCase`

**Package:** `sltest.testmanager`

Set test case assessment callback

### Syntax

```
setAssessmentsCallback(tc, callback)
```

### Description

`setAssessmentsCallback(tc, callback)` specifies the assessment callback to assign to the specified test case. You can use an assessment callback to define variables to use in logical and temporal assessment conditions and expressions. The callback you specify appears in the **Assessment Callback** section of the test case in the Test Manager.

### Input Arguments

#### **tc** — Test case

`sltest.testmanager.TestCase` object

Test case to which the assessment callback is associated, specified as an `sltest.testmanager.TestCase` object.

Data Types: `string`

#### **callback** — Assessment callback

`string`

Assessment callback, specified as a string. For a list of predefined variables you can use in a callback, see “Logical and Temporal Assessment Syntax”.

Example: `"test_value = 20; increment_value = 5;"`

Data Types: `string`

### Examples

#### Use a Callback to Set Assessment Variables

- 1 Specify the test file and get the test suite and test case object in the file. Assume there is only one test suite and one test case in the file. If a test file contains more than one test suite, an array of test suites is returned from the `getTestSuites` method. If the file or suite contains more than one test case, an array of test cases is returned.

```
tf = sltest.testmanager.TestFile('myTestFile.mldatx');
ts = tf.getTestSuites();
tc = ts.getTestCases();
```

- 2 Specify the callback for the test case object.

```
tc.setAssessmentsCallback("test_value = 20; end_value = 65;");
```

## See Also

[getAssessmentsCallback](#)

## Topics

[“Logical and Temporal Assessment Syntax”](#)

**Introduced in R2020b**

## setModelParam

**Class:** `sltest.testmanager.TestIteration`

**Package:** `sltest.testmanager`

Set model parameter for iteration

### Syntax

```
setModelParam(obj,modelObject,paramName,value)
setModelParam(obj,modelObject,paramName,value,'SimulationIndex',sim)
```

### Description

`setModelParam(obj,modelObject,paramName,value)` sets a model parameter for the test iteration object.

`setModelParam(obj,modelObject,paramName,value,'SimulationIndex',sim)` sets a model parameter for the specified simulation in an equivalence test.

### Input Arguments

**obj** — Test iteration to set model parameter for

`sltest.testmanager.TestIteration` object

Test iteration that you want to set the model parameter for, specified as a `sltest.testmanager.TestIteration` object.

**modelObject** — Name of model or block

character vector | string

Name of a model or block, specified as a character vector or string.

Example: `'vdp/Mu'`

**paramName** — Model or block parameter name

character vector

Model or block parameter name, specified as the comma-separated pair consisting of the parameter name, specified as a character vector, and the value, specified in the format determined by the parameter type. Case is ignored for parameter names. Value character vectors are case sensitive. Values are often character vectors, but they can also be numeric, arrays, and other types. Many block parameter values are specified as character vectors, but two exceptions are these parameters: **Position**, specified as a value vector, and **UserData**, which can be any data type.

For more information on parameter name and value pairs, see `set_param`.

Example: `'Solver','ode15s'`

Data Types: `char`

**sim** — Simulation to set model parameters for

1 | 2

Simulation in an equivalence test to set model parameters for, specified as 1 or 2.

## Examples

### Override Gain Value

```
setModelParam(obj,[sltest_bdroot '/Mu'],'Gain','1000')
```

### See Also

`sltest.testmanager.TestIteration` | `set_param`

### Topics

“Test Iterations”

“Create and Run Test Cases with Scripts”

### Introduced in R2016a

## setProperty

**Class:** `sltest.testmanager.TestCase`

**Package:** `sltest.testmanager`

Set test case property

### Syntax

```
setProperty(tc,Name,Value)
```

### Description

`setProperty(tc,Name,Value)` sets a test case property.

### Input Arguments

#### **tc — Test case**

`sltest.testmanager.TestCase` object

Test case to set property, specified as an `sltest.testmanager.TestCase` object.

#### **Name-Value Pair Arguments**

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

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

Example: `'StopTime',100`

#### **Model — System Under Test model name**

empty character vector (default)

The model name in the System Under Test section, specified as a character vector.

Example: `'sldemo_absbrake'`

#### **SimulationMode — Simulation mode**

empty character vector (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 character vector. To return to the default model settings, specify an empty character vector, `''`.

Example: `'SimulationMode','Rapid Accelerator'`

#### **OverrideSILPILMode — Override SIL or PIL mode**

false or 0 (default) | true or 1



Override SIL/PIL simulation mode of model blocks to Normal simulation mode, specified as a numeric or logical 1 (true) or 0 (false). If this property is true or 1, the associated check box in the Simulation Settings Overrides section of the Test Manager is selected.

**OverrideStartTime — Override model start time**

false or 0 (default) | true or 1

Indicate if the test case overrides the model start time, specified as a numeric or logical 1 (true) or 0 (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 or 0 (default) | true or 1

Indicate if the test case overrides the model stop time, specified as a numeric or logical 1 (true) or 0 (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 or 0 (default) | true or 1

Indicate if the test case overrides the model initial state, specified as a numeric or logical 1 (true) or 0 (false).

**InitialState — Model initial state**

empty character vector (default)

Model initial state from a workspace variable, specified as a character vector.

**HarnessName — Test harness name**

empty character vector (default)

Name of a test harness to use in the System Under Test section, specified as a character vector.

**HarnessOwner — Test harness owner name**

empty character vector (default)

Name of the test harness owner, specified as a character vector.

**UseSignalEditorScenario — Override Signal Editor scenario**

false or 0 (default) | true or 1

Indicate if the test case overrides the model and uses a different Signal Editor scenario in the Inputs section, specified as a numeric or logical 1 (true) or 0 (false).

**SignalEditorScenario — Signal Editor scenario name**

empty character vector (default)

Signal Editor scenario name, specified as a character vector. To return to the default model settings, specify an empty character vector, ''.

**IsTestDataReferenced — Use external file to create test case**

false or 0 (default) | true or 1

Whether to use test data from an external file, such as an Excel or MAT file, to create the test case, specified as a numeric or logical 1 (true) or 0 (false). If `isTestDataReferenced` is true or 1, use `TestDataPath` to specify the path to file.

**TestDataPath — Path to external file for test case**

character vector

Path to the external file being referenced for creating the test case, specified as a character vector.

**TestSequenceBlock — Test Sequence block path**

character vector

Test Sequence block path, specified as a character vector. The specified block contains the test sequence scenarios to use in the test case. Specify `TestSequenceBlock` and `TestSequenceScenario` to run a scenario other than the active scenario. If you do not specify a value for `TestSequenceScenario`, the test runs the active scenario in the Test Sequence block. If your Test Sequence block does not have scenarios, the test runs the single test sequence in the block.

**TestSequenceScenario — Test Sequence scenario name**

character vector

Test Sequence scenario name, specified as a character vector. The specified test sequence scenario runs instead of the active scenario or, if you are using iterations, the scenario runs as the default scenario for all iterations. Use `setTestParam` to assign a different scenario to an iteration. You must specify `TestSequenceBlock` to use `TestSequenceScenario`.

**Release — Release on which to run the test case**

Current (default) | string | character vector | cell array | string array

Release on which to run the test case, specified as a string, character vector, or cell array. For equivalence tests, you can specify only one release for each simulation index, for example, ('Release', releaseNames{1}, 'SimulationIndex', 1). For other test types, you can specify multiple releases as a cell or string array, for example, ('Release', releaseNames), where `releaseNames` is a cell array, such as {'Current', 'R2020a'}.

**OverrideModelOutputSettings — Override model output settings**

false or 0 (default) | true or 1

Indicate if the test case overrides the model settings under the Outputs section, specified as a numeric or logical 1 (true) or 0 (false).

**SaveOutput — Override saving output**

false or 0 (default) | true or 1

Indicate if the test case overrides saving model output, specified as a numeric or logical 1 (true) or 0 (false).

**SaveState — Save output state values**

false or 0 (default) | true or 1

Indicate if the test case is set to save output state values, specified as a numeric or logical 1 (true) or 0 (false).

**SignalLogging — Log signals**

true or 1 (default) | false or 0

Indicate if the test case is set to log signals marked for logging in the model, specified as a numeric or logical 1 (true) or 0 (false).

**DSMLogging — Log Data Store variables**

true or 1 (default) | false or 0

Indicate if the test case is set to log Data Store variables, specified as a numeric or logical 1 (true) or 0 (false).

**SaveFinalState — Save final state**

false or 0 (default) | true or 1

Indicate if the test case is set to store final state values, specified as a numeric or logical 1 (true) or 0 (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 character vector (default)

Name of the configuration setting in a model, specified as a character vector.

**ConfigSetVarName — Configuration set variable name**

empty character vector (default)

Variable name in a configuration set file, specified as a character vector.

**ConfigSetFileLocation — Configuration set file path**

empty character vector (default)

File name and path of the configuration set, specified as a character vector.

**PreLoadCallback — Pre-load callback script**

character vector

Pre-load callback script, specified as a character vector.

**PostloadCallback — Post-load callback script**

character vector

Post-load callback script, specified as a character vector.

**CleanupCallback — Cleanup callback script**

character vector

Test-case level cleanup callback script, specified as a character vector. The function deletes any existing callback script and replaces it with the specified character vector.

Example: 'clear a % clear value from workspace'

**PreStartRealTimeApplicationCallback — Real-time pre-start callback**

character vector

Character vector evaluated before the real-time application is started on the target computer, specified as a character vector. For more information on real-time testing, see “Test Models in Real Time”.

**IterationScript — Iteration script**

character vector

Iteration script evaluated to create test case iterations, specified as a character vector. For more information about test iteration scripts, see “Test Iterations”.

**FastRestart — Run iterations using fast restart**

false or 0 (default) | true or 1

Indicate if the test iterations run using fast restart mode, specified as a numeric or logical 1 (true) or 0 (false).

**SaveBaselineRunInTestResult — Save baseline in test result**

false (default) | true

Indicate if the test case saves the baseline used in the test result after test execution, specified as a numeric or logical 1 (true) or 0 (false).

**SaveInputRunInTestResult — Save input in test result**

false or 0 (default) | true or 1

Enable saving external input run used in test result, specified as a numeric or logical 1 (true) or 0 (false).

**StopSimAtLastTimePoint — Stop simulation at last input time**

false or 0 (default) | true or 1

Enable stopping the simulation at the final time value of the input, specified as a numeric or logical 1 (true) or 0 (false).

**LoadAppFrom — Application location**

1 (default) | 2 | 3

Location from which to load the application, specified as an integer. This property is available only in real-time test cases.

- 1 — Model
- 2 — Target application
- 3 — Target computer

For more information on real-time testing, see “Test Models in Real Time”.

**TargetComputer — Target computer name**

character vector

Name of the target computer, specified as a character vector. This property is available only in real-time test cases. For more information on real-time testing, see “Test Models in Real Time”.

**TargetApplication — Target application name and path**

character vector

Name and path of the target application, specified as a character vector. This property is available only in real-time test cases. For more information on real-time testing, see “Test Models in Real Time”.

**Examples****Set Model as System Under Test**

```
% Create the test file, test suite, and test case structure
tf = sltest.testmanager.TestFile('API Test File');
ts = createTestSuite(tf, 'API Test Suite');
tc = createTestCase(ts, 'baseline', 'Baseline API Test Case');

% Remove the default test suite
tsDel = getTestSuiteByName(tf, 'New Test Suite 1');
remove(tsDel);

% Assign the system under test to the test case
setProperty(tc, 'Model', 'sldemo_absbrake');
```

**See Also****Topics**

“Create and Run Test Cases with Scripts”

“Programmatically Create and Run Test Sequence Scenarios”

**Introduced in R2015b**

## setProperty

**Class:** `sltest.testmanager.TestFile`

**Package:** `sltest.testmanager`

Set test file property

### Syntax

```
setProperty(tf,Name,Value)
```

### Description

`setProperty(tf,Name,Value)` sets a test file property.

### Input Arguments

#### **tf** — Test file

`sltest.testmanager.TestFile` object

Test file object whose property to set, specified as an `sltest.testmanager.TestFile` object.

#### **Name-Value Pair Arguments**

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

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

Example: `'SetupCallback','a = 300; % set nominal value'`

#### **SetupCallback** — Setup callback script

empty (default) | character vector

Test-file level setup callback script, specified as a character vector. The function replaces any existing callback script with this value.

Example: `'a = 300; % set nominal value'`

#### **CleanupCallback** — Cleanup callback script

empty (default) | character vector

Test-file level cleanup callback script, specified as a character vector. The function replaces any existing callback script with this value.

Example: `'clear a % clear value from workspace'`

### Examples

### Set Test File Property

```
% Create a test file
tf = sltest.testmanager.TestFile('API Test File');

% Set the setup callback property
setProperty(tf, 'CleanupCallback', 'clearvars % Clear variables');
```

### See Also

getProperty

### Topics

“Create and Run Test Cases with Scripts”

**Introduced in R2015b**

## setProperty

**Class:** `sltest.testmanager.TestSuite`

**Package:** `sltest.testmanager`

Set test suite property

### Syntax

```
setProperty(ts, Name, Value)
```

### Description

`setProperty(ts, Name, Value)` sets a test suite property.

### Input Arguments

**ts — Test suite**

`sltest.testmanager.TestSuite` object

Test suite object to set the property, specified as an `sltest.testmanager.TestSuite` object.

**Name-Value Pair Arguments**

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

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

Example: `'SetupCallback', 'a = 300; % set nominal value'`

**SetupCallback — Setup callback script**

empty (default) | character vector

Test-suite level setup callback script, specified as a character vector. The function deletes any existing callback script and replaces it with the specified character vector.

Example: `'a = 300; % set nominal value'`

**CleanupCallback — Cleanup callback script**

empty (default) | character vector

Test-suite level cleanup callback script, specified as a character vector. The function deletes any existing callback script and replaces it with the specified character vector.

Example: `'clear a % clear value from workspace'`

### Examples



## Set Test Suite Property

```
% Create a test file and new test suite
tf = sltest.testmanager.TestFile('API Test File');
ts = createTestSuite(tf, 'API Test Suite');

% Set the setup callback property
setProperty(ts, 'CleanupCallback', 'clearvars % Clear variables');
```

## See Also

### Topics

“Create and Run Test Cases with Scripts”

**Introduced in R2015b**

## setTestParam

**Class:** sltest.testmanager.TestIteration

**Package:** sltest.testmanager

Set test case iteration parameter

### Syntax

```
setTestParam(obj,parameter,parameterValue)
setTestParam(obj,parameter,parameterValue,'SimulationIndex',indexValue)
```

### Description

setTestParam(obj,parameter,parameterValue) sets the specified test iteration parameter.

setTestParam(obj,parameter,parameterValue,'SimulationIndex',indexValue) sets the specified test iteration parameter and the SimulationIndex name-value pair. The simulation index is the index of an equivalence test case, specified as an integer.

### Input Arguments

#### obj — Test iteration object

object

Test iteration object that you want to apply the test parameter to, specified as a sltest.testmanager.TestIteration object.

#### ParameterSet — Parameter set

current test case setting (default) | character vector

Parameter set name, specified as the comma-separated pair consisting of 'ParameterSet' and a character vector. Parameter set overrides are set up in the **Parameter Overrides** section of the test case.

Example: 'ParameterSet','Param Set 1'

#### Baseline — Baseline criteria dataset

current test case setting (default) | character vector

Baseline criteria dataset name, specified as the comma-separated pair consisting of 'Baseline' and a character vector. Baseline criteria datasets are set up in the **Baseline Criteria** section of the test case. It is available only for baseline test cases.

Example: 'Baseline','BaselineSet\_High'

#### LoggedSignalSet — Logged signal set

current test case setting (default) | character vector

Logged signal set name, specified as the comma-separated pair consisting of 'LoggedSignalSet' and a character vector. Logged signal sets are set up in the **Simulation Outputs** section of the test case.

Example: 'LoggedSignalSet','Logged signal set 1'

### **ExternalInput — External input**

current test case setting (default) | character vector

External input name, specified as the comma-separated pair consisting of 'ExternalInput' and a character vector. External input overrides are set up in the **Inputs** section of the test case. You can specify a simulation index as the ExternalInput, for example, `ti.setTestParam('ExternalInput','input','SimulationIndex',2)`.

Example: 'ExternalInput','Run1'

### **ConfigSet — Configuration setting**

current test case setting (default) | character vector

Configuration setting name, specified as the comma-separated pair consisting of 'ConfigSet' and a character vector. Configuration setting overrides are set up in the **Configuration Settings Overrides** section of the test case.

Example: 'ConfigSet','Solver 3'

### **TestSequenceScenario — Test Sequence scenario name**

character vector

Test Sequence scenario name, specified as a character vector. The specified scenario, which is defined in a Test Sequence block, is used in the iteration. You specify the Test Sequence block by using `setProperty`. The test sequence scenario for an iteration is set up in the test case **Inputs** section of the Test Manager.

### **SignalEditorScenario — Signal Editor scenario**

current test case setting (default) | character vector

Signal Editor scenario input name, specified as the comma-separated pair consisting of 'SignalEditorScenario' and a character vector. Signal Editor scenario overrides are set up in the test case **Inputs** section.

Example: 'SignalEditorScenario','Acceleration'

### **PreLoadFcn — Pre-load callback script**

current test case setting (default) | character vector

Pre-load callback script, specified as the comma-separated pair consisting of 'PreLoadFcn' and a character vector. The pre-load callback script is set up in the **Callbacks** section of the test case.

### **PostLoadFcn — Post-load callback script**

current test case setting (default) | character vector

Post-load callback script, specified as the comma-separated pair consisting of 'PostLoadFcn' and a character vector. The post-load callback script is set up in the **Callbacks** section of the test case.

### **PreStartRealTimeApplicationFcn — Pre-start real-time application callback script**

current test case setting (default) | character vector

Pre-start real-time application callback script, specified as the comma-separated pair consisting of 'PreStartRealTimeApplicationFcn' and a character vector. The pre-start real-time application callback script is set up in the **Callbacks** section of the test case.

**CleanupFcn — Cleanup callback script**

current test case setting (default) | character vector

Cleanup callback script, specified as the comma-separated pair consisting of 'CleanupFcn' and a character vector. The cleanup callback script is set up in the **Callbacks** section of the test case.

**Description — Description**

current test case setting (default) | character vector

Test description text, specified as the comma-separated pair consisting of 'Description' and a character vector. The Description is set up in the **Description** section of the test case.

Example: 'Description', 'Test the autopilot controller for wind gusts'

**Examples****Set Test Parameter**

```
setTestParam(obj, 'Description', ...
 'Test the autopilot controller for wind gusts');
```

**See Also**

sltest.testmanager.TestIteration

**Topics**

“Test Iterations”

“Create and Run Test Cases with Scripts”

“Programmatically Create and Run Test Sequence Scenarios”

**Introduced in R2016a**

# setVariable

**Class:** sltest.testmanager.TestIteration

**Package:** sltest.testmanager

Set model variable override

## Syntax

```
setVariable(obj, 'Name', varName, 'Source', srcName, 'Value', value,
'SimulationIndex', simIndex)
```

## Description

setVariable(obj, 'Name', varName, 'Source', srcName, 'Value', value, 'SimulationIndex', simIndex) sets a model variable override for the test iteration. Specify the sltest.testmanager.TestIteration object, and then specify the variable name, source, override value, and optionally, the simulation index. The method overrides the variable in the test iteration and does not permanently change the model variable.

## Input Arguments

### obj — Test iteration object

object

The test iteration you want to apply the override to, specified as a sltest.testmanager.TestIteration object.

### varName — Variable name

character vector

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

### srcName — Variable source

'base workspace' | 'model workspace' | 'mask workspace' | data dictionary name | model element paths | workspace name

The source of the variable to override, specified as a character vector. For non-real-time test cases, the possible sources can be 'base workspace', 'model workspace', 'mask workspace', or the name of a data dictionary, such as 'data.slidd' or model workspace name.

For real-time test cases, the possible sources are model element paths, which, if the source is a model parameter, might be an empty character vector.

### value — Override value

numeric | logical | enum | struct

Value of the variable to override.

### simIndex — Equivalence test simulation index

1 (default) | 2

Simulation index of an equivalence test case, specified as 1 or 2, where 1 is Simulation 1 and 2 is Simulation 2.

Example: 'SimulationIndex',2

## Examples

### Set Variable in Base Workspace

```
setVariable(obj, 'Name', 'g', 'Source', 'base workspace', 'Value', 33);
```

### See Also

`sltest.testmanager.TestIteration`

### Topics

“Test Iterations”

“Create and Run Test Cases with Scripts”

### Introduced in R2016a

# addModelCoverage

Enable collect model coverage collection for Simulink tests

## Syntax

```
addModelCoverage(runner)
addModelCoverage(runner,Name,Value)
```

## Description

`addModelCoverage(runner)` enables the specified test runner to add model coverage to MATLAB-based Simulink tests and generates a coverage report. This method adds `sltest.plugins.ModelCoveragePlugin` to the test runner.

---

**Note** For MATLAB-based Simulink tests, to include model coverage results in the Simulink Test Manager, you must also use `addSimulinkTestResults`.

---

`addModelCoverage(runner,Name,Value)` enables model coverage collection with additional options specified by one or more `Name,Value` pair arguments.

## Input Arguments

### runner — Test runner

`matlab.unittest.TestRunner` object

Test runner, specified as a `matlab.unittest.TestRunner` object.

### Name-Value Pair Arguments

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

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

Example: `'CollectMetrics', ['MCDC', 'Decision']`

### html-report — Name of folder to export HTML coverage report

string | character vector

Name of folder to export HTML coverage report, specified as a string or character vector.

Example: `'html-report', 'mytests/coverage/'`

### cobertura — Path to export Cobertura coverage report

string | character vector

Path to export Cobertura coverage report, specified as a string or character vector. The report is saved to the specified file location and file name.

Example: `'cobertura', 'mytests/coverage/test1_coverage.xml'`

### **IncludeModelReference — Collect coverage for model references**

`false` (default) | `true`

Collect coverage for model references, specified as a logical `true` or `false`. If `IncludeModelReference` is `true`, the runner collects coverage for the main model and all referenced models.

Example: `'IncludeModelReference', true`

### **CollectMetrics — Collect coverage for metrics**

array of strings | array of character vectors

Collect coverage for metrics, specified as an array of strings or an array of character vectors. For supported coverage metrics, see the `sltest.plugins.coverage.CoverageMetrics` properties.

Example: `'CollectMetrics', ['MCDC', 'Decision']`

## **Examples**

### **Enable Model Coverage Collection**

This example shows how to enable collecting model coverage and specify metrics when using the test runner.

Before running this example, create a test file named `myTests.m`. The example creates a test suite for the file, and then creates a test runner with the `'textoutput'` option to output status and diagnostics to the command line. Add `addSimulinkTestResults` to the test runner to enable pushing the results in the Test Manager and exporting the test results to a file named `testmgr_results.mldatx`

Before running this example, create a test file named `myTests.m`. The example creates a test suite for the file, and then creates a test runner with the `'textoutput'` option to output status and diagnostics to the command line. Add `addModelCoverage` to the test runner to enable collecting model coverage that uses MCDC and condition metrics, specify the folder name for the HTML report, and include referenced models. Add `addSimulinkTestResults` to the test runner to enable pushing the test results to the Test Manager.

```
suite = testsuite('myTests');
runner = testrunner('textoutput');
addModelCoverage(runner, ...
 "CollectMetrics", ["MCDC", "Condition"], ...
 "html-report", "tests/cov/",
 "IncludeModelReference", true);
addSimulinkTestResults(runner);
run(runner, suite);
```

### **See Also**

`matlab.unittest.TestRunner` | `sltest.plugins.coverage.CoverageMetrics` | `addSimulinkTestResults`

### **Topics**

“Collect Coverage Using MATLAB-Based Simulink Tests”



“Test Models Using MATLAB-Based Simulink Tests”  
“Using MATLAB-Based Simulink Tests in the Test Manager”

**Introduced in R2021a**

## addSimulinkTestResults

Enable pushing test results to Simulink Test Manager

### Syntax

```
addSimulinkTestResults(runner)
addSimulinkTestResults(runner, 'ExportToFile', filename)
```

### Description

`addSimulinkTestResults(runner)` enables pushing test results from MATLAB to integrate them with Simulink Test Manager. Use `addSimulinkTestResults` for tests that you run using the test runner, including MATLAB-based Simulink tests and

`addSimulinkTestResults` adds a set of plugins to a `matlab.unittest.TestRunner` object. The added plugins are:

- `sltest.plugins.MATLABTestCaseIntegrationPlugin` — Adds simulation and test results to the Simulink Test Manager for MATLAB-based Simulink tests.
- `sltest.plugins.TestManagerResultsPlugin` — Captures Simulink Test Manager results and MATLAB test Results, and allows exporting test results from the Test Manager.
- `matlab.unittest.DiagnosticsOutputPlugin` — Adds diagnostics output to the **Logs** section of the Test Manager. To include log information, see `sltest.plugins.ToTestManagerLog`.

`addSimulinkTestResults(runner, 'ExportToFile', filename)` enables exporting test results to an MLDATX file with the specified `filename`. Load this file at the MATLAB command line to view the test results.

---

**Note** `addSimulinkTestResults` is not supported for parallel workflows.

---

### Input Arguments

#### **runner** — Test runner

`matlab.unittest.TestRunner` object

Test runner, specified as a `matlab.unittest.TestRunner` object.

#### **filename** — Name of test results file

MLDATX file

Name of test results file, specified as an MLDATX file.

### Examples

## Enable Simulink Test Results

This example shows how to enable a test runner object to push test results to the Simulink Test Manager for a MATLAB-based Simulink test, and then export the results to a data file.

Before running this example, create a test file named `myTests.m`. The example creates a test suite for the file, and then creates a test runner with the `'textoutput'` option to output status and diagnostics to the command line. Add `addSimulinkTestResults` to the test runner to enable pushing the results in the Test Manager and exporting the test results to a file named `testmgr_results.mldatx`.

```
suite = testsuite('myTests');
runner = testrunner('textoutput');
addSimulinkTestResults(runner,"ExportToFile",...
 "testmgr_results.mldatx");
run(runner,suite);
```

## See Also

`matlab.unittest.TestRunner` | `testrunner` |  
`sltest.plugins.MATLABTestCaseIntegrationPlugin` |  
`sltest.plugins.TestManagerResultsPlugin` |  
`matlab.unittest.plugins.DiagnosticsOutputPlugin` |  
`sltest.plugins.ToTestManagerLog`

## Topics

“Test Models Using MATLAB-Based Simulink Tests”  
“Using MATLAB-Based Simulink Tests in the Test Manager”  
“Collect Coverage Using MATLAB-Based Simulink Tests”

## Introduced in R2021a

## createSandbox

Create sandbox for C code unit testing

### Syntax

```
success = createSandbox
createSandbox("Overwrite", overwrite_option)
```

### Description

`success = createSandbox` creates a sandbox folder and subfolders, and returns true if the method successfully creates a sandbox. The folder name is `<LibraryFileName>_sandbox`, where `LibraryFileName` is the `LibraryFileName` property of the `sltest.CodeImporter` object. This method applies only if the `TestType` property of the `sltest.CodeImporter` object is `UnitTest`.

The created folder subdirectories are:

- `autostub` — Contains the `auto_stub.h` and `auto_stub.c` files, which are generated only if the imported code has undefined symbols.
- `manualstub` — Contains the `man_stub.h` and `man_stub.c` files, which you can use to manually stub symbols. The `man_stub.h` header file includes the aggregated header if the `Mode` property setting of `sltest.CodeImporter.SandboxSettings` is `GenerateAggregatedHeader`.
- `include` — Contains the header files required by the sandbox. This folder also contains a generated `aggregatedHeader.h` or `interfaceHeader.h` file. An `aggregatedHeader.h` file is generated if the `Mode` property setting of `sltest.CodeImporter.SandboxSettings` is `GenerateAggregatedHeader`. Otherwise, the folder contains an `interfaceHeader.h` file.
- `src` — Contains copies of the code source files if the `CopySource` property of the `sltest.CodeImporter.SandboxSettings` object is true.

`createSandbox("Overwrite", overwrite_option)` overwrites the existing sandbox. If `overwrite_option` is "on", the method deletes the existing sandbox directory and creates a new sandbox directory. If `overwrite_option` is "off", the method deletes all folders except `manualstub` and generates a new sandbox. The method does not change the `manualstub` folder or its contents. The default is "off".

### Output Arguments

**success** — Whether sandbox creation is successful

true | false

Whether sandbox creation is successful, returned as a logical value.

### Examples

#### Create Sandbox for Testing Code

This example assumes you have existing C code files to test.

- 1 Create an `sltest.CodeImporter` object and specify `myCodeTest` as the Simulink library file name.

```
codeimport_obj = sltest.CodeImporter("myCodeTest");
```

- 2 Assign the source files to the `SourceFiles` property.

```
codeimport_obj.CustomCode.SourceFiles = {"myCode1.c", "myCode2.c"};
```

- 3 Create the sandbox. To check that the sandbox exists, confirm that `success` is `true`.

```
success = codeimport_obj.createSandbox
```

## See Also

`sltest.CodeImporter` | `sltest.CodeImporter.SandboxSettings`

## Topics

“Conduct Unit Testing on Imported Custom Code by Using the Wizardfilename”

“Import Custom Code for Unit Testing Using API Commands”

**Introduced in R2021a**



# Blocks

---

## Observer Port

Wirelessly link signals to use with verification

**Library:** Simulink Test



### Description

Include the Observer Port block within an Observer model to map signals from a system model to an Observer model. The Observer Port block can only be used within a model that is linked to an Observer Reference block.

### Ports

#### Output

##### Port\_1 — Mapped signal

scalar | vector

Mapped signal that flows through the Observer Port into the Observer model.

Data Types: double | single | Boolean | base integer | fixed point | enumerated | non-virtual bus

### See Also

#### Topics

“Access Model Data Wirelessly by Using Observers”  
Observer Reference

**Introduced in R2019a**



# Observer Reference

Create and contain an Observer model

**Library:** Simulink Test



## Description

An Observer Reference block references a separate model called the Observer model. The Observer model and Observer Reference block enable wireless monitoring of the signals in the main model. When you click or select **Add Observer Reference** from the **Test** tab or the context menu, respectively, an Observer Reference block is automatically added to the top level of your main Simulink model, and an Observer model is created. You can add Observer Reference blocks only at the top level of the model.

The Observer model contains an Observer Port block and can contain blocks such as scopes or a verification subsystem to monitor or verify the observed signals, respectively. Signals are mapped to the Observer model through the Observer Port block. The Observer Reference block does not have inports or outports. For more information and an example of using Observers, see “Access Model Data Wirelessly by Using Observers”.

To stop observing signals, delete all the Observer Reference blocks from the main model.

## Parameters

### Observer Model name — File name of Observer model

' ' (default) | character vector

Path to the Observer model. The file name must be a valid MATLAB identifier. The extension, for example, `.slx`, is optional. You enter the Observer model name in the Observer Reference block parameters dialog box, which you access by right-clicking on the block and selecting **Block Parameters**. You can also open the model you specify from the dialog box.

### Programmatic Use

**Parameter:** `ObserverModelName`

**Type:** character vector

**Value:** ' ' | '<observer model name>'

**Default:** ' '

## See Also

Observer Port

## Topics

“Access Model Data Wirelessly by Using Observers”

**Introduced in R2019a**

## Test Assessment

Assess simulation testing scenarios, function calls, and assessments

**Library:** Simulink Test



### Description

Define test assessments in a tabular series of steps. Like the Test Sequence block, the Test Assessment block uses MATLAB as the action language.

Double-click the Test Assessment block to open the Test Sequence Editor. By default, the Test Assessment block contains a test step Run, configured as a When decomposition sequence. To change the transition type to a standard transition, right-click the top-level step and clear **When decomposition**. For more information, see “Test Sequence Editor”.

### Ports

Ports correspond to inputs and outputs defined in the Test Sequence Editor **Symbols** pane.

### Parameters

For a description of block parameters, see Subsystem.

### Extended Capabilities

#### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

### See Also

“Test Sequence Editor” | “Test Sequence and Assessment Syntax” |  
`sltest.testmanager.Assessment` | `sltest.testmanager.AssessmentSymbol` |  
`addAssessment` | `getAssessments` | `addAssessmentSymbol` | `getAssessmentSymbols`

### Topics

“Assess Model Simulation Using verify Statements”  
 “Transitions, Temporal Operators, and Messages in Test Sequence Blocks”  
 “Generate Test Signals”

**Introduced in R2016a**

# Test Sequence

Create simulation testing scenarios, function calls, and assessments

**Library:** Simulink Test



## Description

Define a test sequence using a tabular series of steps. Like the Test Assessment block, the Test Sequence block uses MATLAB as the action language.

## Ports

Ports correspond to inputs and outputs defined in the Test Sequence Editor **Symbols** pane.

## Parameters

For a description of block parameters, see Subsystem.

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

## See Also

“Test Sequence Editor” | “Test Sequence and Assessment Syntax”

## Topics

“Assess Model Simulation Using verify Statements”

“Transitions, Temporal Operators, and Messages in Test Sequence Blocks”

“Generate Test Signals”

**Introduced in R2015a**

## Sequence Viewer

Display messages, events, states, transitions, and functions between blocks during simulation

**Library:** Simulink / Messages & Events  
 Simulink Test  
 SimEvents  
 Stateflow



### Description

The Sequence Viewer block displays messages, events, states, transitions, and functions between certain blocks during simulation. The blocks that you can display are called lifeline blocks and include:

- Subsystems
- Referenced models
- Blocks that contain messages, such as Stateflow charts
- Blocks that call functions or generate events, such as Function Caller, Function-Call Generator, and MATLAB Function blocks
- Blocks that contain functions, such as Function-Call Subsystem and Simulink Function blocks

To see states, transitions, and events for lifeline blocks in a referenced model, you must have a Sequence Viewer block in the referenced model. Without a Sequence Viewer block in the referenced model, you can see only messages and functions for lifeline blocks in the referenced model.

---

**Note** The Sequence Viewer block does not display function calls generated by MATLAB Function blocks and S-functions.

---

### Parameters

#### Time Precision for Variable Step — Digits for time increment precision

3 (default) | scalar

Number of digits for time increment precision. When using a variable step solver, change this parameter to adjust the time precision for the sequence viewer. By default the block supports 3 digits of precision.

Suppose the block displays two events that occur at times 0.1215 and 0.1219. Displaying these two events precisely requires 4 digits of precision. If the precision is 3, then the block displays two events at time 0.121.

#### Programmatic Use

**Block Parameter:** VariableStepTimePrecision

**Type:** string scalar or character vector

**Values:** "3" | scalar

**Default:** "3"

### **History — Maximum number of previous events to display**

5000 (default) | scalar

Total number of events before the last event to display.

For example, if **History** is 5 and there are 10 events in your simulation, then the block displays 6 events, including the last event and the five events prior the last event. Earlier events are not displayed. The time ruler is greyed to indicate the time between the beginning of the simulation and the time of the first displayed event.

Each send, receive, drop, or function call event is counted as one event, even if they occur at the same simulation time.

#### **Programmatic Use**

**Block Parameter:** History

**Type:** string scalar or character vector

**Values:** "1000" | scalar

**Default:** "1000"

## **Block Characteristics**

|                                 |                                                                      |
|---------------------------------|----------------------------------------------------------------------|
| <b>Data Types</b>               | Boolean   bus   double   enumerated   fixed point   integer   single |
| <b>Direct Feedthrough</b>       | no                                                                   |
| <b>Multidimensional Signals</b> | yes                                                                  |
| <b>Variable-Size Signals</b>    | no                                                                   |
| <b>Zero-Crossing Detection</b>  | no                                                                   |

## **Extended Capabilities**

### **C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

This block can be used for visualizing message transitions during simulation, but is not included in the generated code.

### **HDL Code Generation**

Generate Verilog and VHDL code for FPGA and ASIC designs using HDL Coder™.

This block displays messages during simulation when used in subsystems that generate HDL code, but it is not included in the hardware implementation.

## **See Also**

"Use the Sequence Viewer to Visualize Messages, Events, and Entities" (SimEvents)

**Introduced in R2015b**

# Language Operators

---

## after

Elapsed time since beginning of test step

### Syntax

```
after(n,timeunits)
```

### Description

`after(n,timeunits)` returns true if the amount of time (n) in `timeunits` has elapsed since the beginning of the current test step. You must include the `timeunits`.

Valid time units are:

- `sec` — seconds
- `msec` — milliseconds
- `usec` — microseconds

### Examples

#### Specify Time Elapsed After Specified Time

Specify 20 seconds as the time that has elapsed since the beginning of the test step. This statement returns `true` only after 20 seconds has passed.

```
after(20,sec)
```

### Tips

`after` statements can be used in the Test Sequence and Test Assessment blocks and in Stateflow charts.

### See Also

t | et | before | duration

#### Topics

“Test Sequence and Assessment Syntax”

“Test Sequence Basics”

“Programmatically Create a Test Sequence”

“Generate Test Signals”

**Introduced in R2015a**



# assert

Evaluate logical expression and stop simulation if false

## Syntax

```
assert(expression)
assert(expression,errmsg)
```

## Description

`assert(expression)` evaluates a logical expression. Logical expressions evaluate to `true` or `false`. If the `assert` statement evaluates to `false`, simulation stops and returns an error.

`assert(expression,errmsg)` returns the specified error message string (`errmsg`) for the failed `assert` statement. If you run the test in the Test Manager, the error message appears in the simulation log. If you run the test outside the Test Manager, the message appears in the Diagnostic Viewer.

## Examples

### Evaluate a Logical Expression

If either `h` or `k` are not 0, this `assert` statement fails and simulation stops.

```
assert(h==0 && k==0)
```

## Tips

- `assert` statements can be used in the Test Sequence and Test Assessment blocks and in Stateflow charts. `assert` in Model blocks works in Normal mode, but not in Rapid Accelerator mode simulations.
- When comparing floating-point data in `assert` statements, consider the precision limitations associated with floating-point numbers. If you need to use floating-point data, define a tolerance for the assessment. See “Floating-Point Numbers”. For example, instead of `assert(x == 5)`, `assert x` within a tolerance of 0.001:

```
assert(abs(x-5) < 0.001)
```

## See Also

`verify`

### Topics

“Test Sequence and Assessment Syntax”

“Test Sequence Basics”

“Programmatically Create a Test Sequence”

“Generate Test Signals”

**Introduced in R2015a**

# before

Elapsed time until specified time in test step

## Syntax

```
before(n,timeunits)
```

## Description

`before(n,timeunits)` returns true until the amount of time (n) in `timeunits` has elapsed since the beginning of the current test step. You must include the `timeunits`.

Valid time units are:

- `sec` — seconds
- `msec` — milliseconds
- `usec` — microseconds

## Examples

### Specify Time Elapsed Until Specified Time

Specify 20 seconds as the time that has elapsed since the beginning of the test step. This statement returns `true` until 20 seconds has passed.

```
before(20,sec)
```

## Tips

`before` statements can be used in the Test Sequence and Test Assessment blocks and in Stateflow charts.

## See Also

`t` | `et` | `after` | `duration`

## Topics

“Test Sequence and Assessment Syntax”

“Test Sequence Basics”

“Programmatically Create a Test Sequence”

“Generate Test Signals”

**Introduced in R2015a**

## duration

Elapsed time since beginning of test step

### Syntax

```
elapsed_time = duration(condition)
elapsed_time = duration(condition,timeunits)
```

### Description

`elapsed_time = duration(condition)` returns the elapsed time in seconds for which the condition has been true. The condition is a logical expression. The `elapsed_time` is reset when the test step is re-entered or when the condition is no longer true.

`elapsed_time = duration(condition,timeunits)` returns the elapsed time in `timeunits`.

Valid time units are:

- sec — seconds
- msec — milliseconds
- usec — microseconds

### Examples

#### Specify Time Elapsed For a Duration

Specify the duration after 100 msec from the time when `x` greater than 50 last became true.

```
duration(x > 50,msec) > 100
```

### Tips

- `duration` statements can be used in the Test Sequence and Test Assessment blocks and in Stateflow charts.
- In Stateflow charts, `duration` statements can be associated only with a single state. You cannot use a `duration` on a transition that has more than one source state connected using a junction.

### See Also

t | et | after | before

### Topics

“Test Sequence and Assessment Syntax”  
“Test Sequence Basics”  
“Programmatically Create a Test Sequence”  
“Generate Test Signals”

**Introduced in R2015a**

# et

Elapsed time of test step

## Syntax

```
et(timeunits)
```

## Description

`et(timeunits)` returns the elapsed time of the test sequence step in `timeunits`. If you omit `timeunits`, the elapsed time defaults to seconds. `et` is an alias for `elapsed` and both are valid.

Valid time units are:

- `sec` — seconds
- `msec` — milliseconds
- `usec` — microseconds

## Examples

### Specify Elapsed Step Time in Milliseconds

```
et(msec)
```

## Tips

`et` statements can be used in the Test Sequence and Test Assessment blocks and in Stateflow charts.

## See Also

`t` | `after` | `before` | `duration`

## Topics

“Test Sequence and Assessment Syntax”

“Test Sequence Basics”

“Programmatically Create a Test Sequence”

“Generate Test Signals”

**Introduced in R2017a**

## hasChanged

Detect data change in test step

### Syntax

```
hasChanged(u)
```

### Description

`hasChanged(u)` returns `true` if the value of `u` changes since the beginning of the test step. Otherwise, `hasChanged` returns `false`.

### Examples

#### Detect Value Change

Detect if `x` has changed value. This statement returns `true` if `x` has changed.

```
hasChanged(x)
```

#### Detect Value Change in a Bus

For a bus with two signals, `signal_1` and `signal_2`, and `data` as the name of the input to the test sequence, detect when the value of `signal_2` has changed.

```
hasChanged(data.signal2)
```

### Tips

`hasChanged` statements can be used in the Test Sequence and Test Assessment blocks and in Stateflow charts.

### See Also

`hasChangedFrom` | `hasChangedTo`

### Topics

“Test Sequence and Assessment Syntax”

“Test Sequence Basics”

“Programmatically Create a Test Sequence”

“Generate Test Signals”

### Introduced in R2015a

# hasChangedFrom

Detect data change from specified value in test step

## Syntax

```
hasChangedFrom(u,A)
```

## Description

`hasChangedFrom(u,A)` returns `true` if the value of `u` changes from the value `A`. Otherwise, `hasChangedFrom` returns `false`.

## Examples

### Detect Value Change From Specified Value

Detect if `x` has changed from a value of 15. This statement returns `true` when `x` had a value of 15 in the previous time step and has a value not equal to 15 in the current time step.

```
hasChangedFrom(x,15)
```

## Tips

`hasChangedFrom` statements can be used in the Test Sequence and Test Assessment blocks and in Stateflow charts.

## See Also

`hasChanged` | `hasChangedTo`

## Topics

“Test Sequence and Assessment Syntax”

“Test Sequence Basics”

“Programmatically Create a Test Sequence”

“Generate Test Signals”

## Introduced in R2015a

## hasChangedTo

Detect data change to specified value in test step

### Syntax

```
hasChangedTo(u, B)
```

### Description

`hasChangedTo(u, B)` returns `true` if the value of `u` changes to the value `B`. Otherwise, `hasChangedTo` returns `false`.

### Examples

#### Detect Value Change To Specified Value

Detect if `x` has changed to a value of 10. This statement returns `true` when `x` did not have value of 10 in the previous time step and has a value of 10 in the current time step.

```
hasChangedTo(x, 10)
```

### Tips

`hasChangedTo` statements can be used in the Test Sequence and Test Assessment blocks and in Stateflow charts.

### See Also

`hasChanged` | `hasChangedFrom`

### Topics

“Test Sequence and Assessment Syntax”

“Test Sequence Basics”

“Programmatically Create a Test Sequence”

“Generate Test Signals”

**Introduced in R2015a**



# heaviside

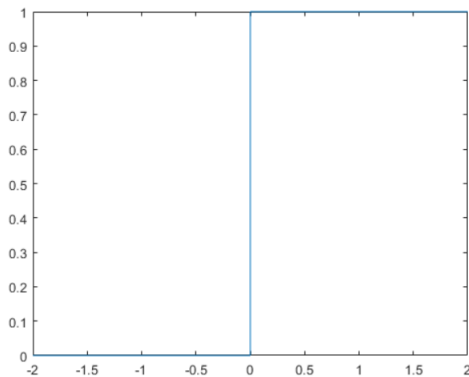
Heaviside step test signal

## Syntax

```
heaviside(x)
```

## Description

`heaviside(x)` creates a Heaviside step signal. Heaviside signals are also called unit step signals and are discontinuous functions. They return 0 for  $x < 0$  and 1 for  $x > 0$ .



To specify when to generate a Heaviside step signal within a test step, use this operator with the elapsed time (`et`) operator.

## Examples

### Create a Heaviside Signal

Create a Heaviside step signal after 10 seconds.

```
heaviside(et-10)
```

## Tips

- `heaviside` signal generation statements can be used in the Test Sequence and Test Assessment blocks.
- `heaviside` signal generation statements cannot be used in Stateflow charts.

## See Also

[ramp](#) | [sawtooth](#) | [square](#) | [triangle](#)

## Topics

“Test Sequence and Assessment Syntax”

“Test Sequence Basics”  
“Programmatically Create a Test Sequence”  
“Generate Test Signals”

**Introduced in R2015a**

# latch

Save value of an expression

## Syntax

```
latch(x)
```

## Description

`latch(x)` saves the value of `x` when the test step is entered and returns that value of `x`. The `latch(x)` statement reevaluates the next time the step is entered.

## Examples

### Use latch to Save a Value

Save the value of the torque in the variable `latch_val`.

```
latch_val = latch(torque)
```

## Tips

- `latch` signal generation statements can be used in the Test Sequence and Test Assessment blocks.
- `latch` signal generation statements cannot be used in Stateflow charts.

## See Also

### Topics

“Test Sequence and Assessment Syntax”

“Test Sequence Basics”

“Programmatically Create a Test Sequence”

**Introduced in R2015a**

## ramp

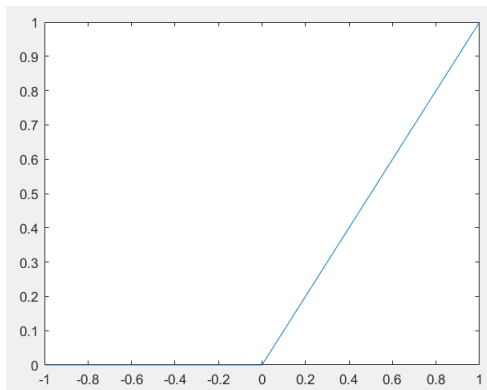
Ramp signal for test signal generation

### Syntax

```
ramp(x)
```

### Description

`ramp(x)` creates a ramp signal wave with a slope of 1 and returns the value of the ramp at time  $x$ .



To specify when to generate a ramp signal within a test step, use this operator with the elapsed time (`et`) operator.

---

**Note** `ramp(et)` returns the elapsed time of the test step and is the same as `et`.

---

### Examples

#### Create a Ramp Signal

Create a ramp signal that increases one unit every 3 seconds in the test step.

```
ramp(et/3)
```

### Tips

- ramp signal generation statements can be used in the Test Sequence and Test Assessment blocks.
- ramp signal generation statements cannot be used in Stateflow charts.

### See Also

heaviside | sawtooth | square | triangle

### Topics

“Test Sequence and Assessment Syntax”

“Test Sequence Basics”  
“Programmatically Create a Test Sequence”  
“Generate Test Signals”

**Introduced in R2015a**

## sawtooth

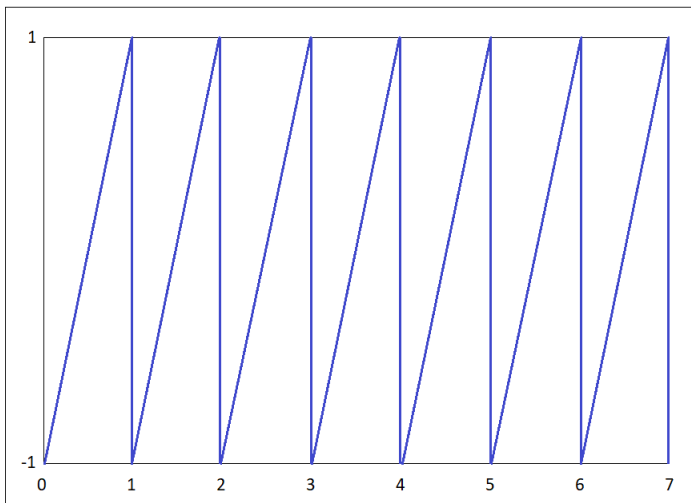
Sawtooth wave test signal

### Syntax

`sawtooth(x)`

### Description

`sawtooth(x)` creates a sawtooth wave with a period of 1 and range -1 to 1. In the interval  $0 \leq x < 1$ , `square(x)` increases. The difference between triangle waves and sawtooth waves is that a triangle wave has equal rise and fall times.



To specify the number of sawtooth wave cycles within a test step, use this operator with the elapsed time (`et`) operator.

### Examples

#### Create a Sawtooth Wave

Create a sawtooth wave with a period of 10.

```
sawtooth(et/10)
```

### Tips

- sawtooth signal generation statements can be used in the Test Sequence and Test Assessment blocks.
- sawtooth signal generation statements cannot be used in Stateflow charts.

## **See Also**

heaviside | ramp | square | triangle

## **Topics**

“Test Sequence and Assessment Syntax”

“Test Sequence Basics”

“Programmatically Create a Test Sequence”

“Generate Test Signals”

**Introduced in R2015a**

## square

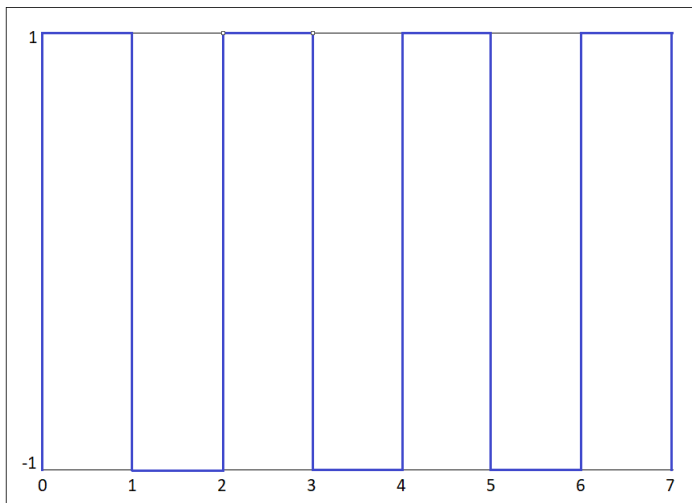
Square wave test signal

### Syntax

`square(x)`

### Description

`square(x)` creates a square wave with a period of 1 and range -1 to 1. In the interval  $0 \leq x < 1$ , `square(x)` returns 1 for  $0 \leq x < 0.5$  and -1 for  $0.5 \leq x < 1$ .



To specify the number of square wave cycles within a test step, use this operator with the elapsed time (`et`) operator.

### Examples

#### Create a Square Wave

Create a square wave with a period of 10.

```
square(et/10)
```

### Tips

- `square` signal generation statements can be used in the Test Sequence and Test Assessment blocks.
- `square` signal generation statements cannot be used in Stateflow charts.



## **See Also**

heaviside | ramp | sawtooth | triangle

## **Topics**

“Test Sequence and Assessment Syntax”

“Test Sequence Basics”

“Programmatically Create a Test Sequence”

“Generate Test Signals”

**Introduced in R2015a**

## **t**

Elapsed time of simulation

### **Syntax**

```
t(timeunits)
```

### **Description**

`t(timeunits)` returns the simulation time in `timeunits`. If you omit `timeunits`, the elapsed time defaults to seconds. `t` is an alias for `getSimulationTime` and both are valid.

Valid time units are:

- `sec` — seconds
- `msec` — milliseconds
- `usec` — microseconds

### **Examples**

#### **Obtain Simulation Time in Milliseconds**

```
t(msec)
```

### **Tips**

Elapsed simulation time ( `t` ) statements can be used in the Test Sequence and Test Assessment blocks and in Stateflow charts. However, you cannot specify the `timeunits` input in Stateflow charts.

### **See Also**

`et` | `before` | `after` | `duration`

### **Topics**

“Test Sequence and Assessment Syntax”

“Test Sequence Basics”

“Programmatically Create a Test Sequence”

“Generate Test Signals”

### **Introduced in R2015a**

# triangle

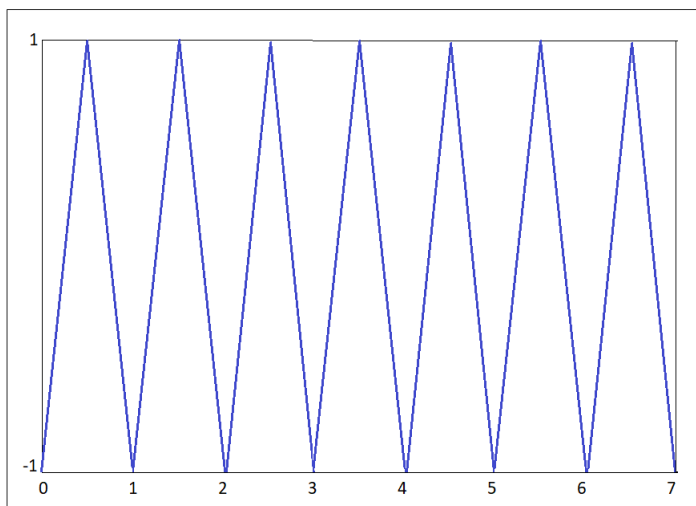
Triangle wave test signal

## Syntax

```
triangle(x)
```

## Description

`triangle(x)` creates a triangle wave with a period of 1 and range -1 to 1. In the interval  $0 \leq x < 0.5$ , `triangle(x)` increases. The difference between triangle waves and sawtooth waves is that a triangle wave has equal rise and fall times.



To specify the number of triangle wave cycles within a test step, use this operator with the elapsed time (`et`) operator.

## Examples

### Create a Triangle Wave

Create a triangle wave with a period of 10.

```
triangle(et/10)
```

## Tips

- `triangle` signal generation statements can be used in the Test Sequence and Test Assessment blocks.
- `triangle` signal generation statements cannot be used in Stateflow charts.

## **See Also**

heaviside | ramp | sawtooth | square

## **Topics**

“Test Sequence and Assessment Syntax”

“Test Sequence Basics”

“Programmatically Create a Test Sequence”

“Generate Test Signals”

**Introduced in R2015a**

# verify

Assess logical expression and automatically log result

## Syntax

```
verify(expression)
verify(expression,errorMessage)
verify(expression,identifier,errorMessage)
```

## Description

`verify(expression)` evaluates a scalar logical expression to true or false.

`verify(expression,errorMessage)` returns the specified error message for the failed `verify` statement. If you run the test in the Test Manager, the error message appears in the simulation log. If you run the test outside the Test Manager, the message appears in the Diagnostic Viewer. To format the error message, you can use any `sprintf` format, except strings and chars in Stateflow charts.

---

**Note** In a real-time environment, `verify` statement failures do not produce warnings. However, if you run a real-time test case in the Test Manager, the failures are shown in the **Verify Statements** section of the test case results. You can also access information about the `verify` runs using `getVerifyRuns`

---

`verify(expression,identifier,errorMessage)` uses the `identifier` as a label for the test results. The `identifier` is used as the signal label in the Test Manager. If you run the test outside the Test Manager, the label appears in the Simulation Data Inspector or, for a failure, in the Diagnostic Viewer. The identifier is a string that has at least two colon-separated MATLAB identifiers.

## Examples

### Verify a Logical Expression

```
verify(x > y && z > 10)
```

### Specify Error Message Text for verify Result

If this `verify` statement fails, it returns an error message that lists the values of `x`, `y`, and `z`.

```
verify(x > y && z > 10, 'x, y, and z are %d,%d,%d',x,y,z)
```

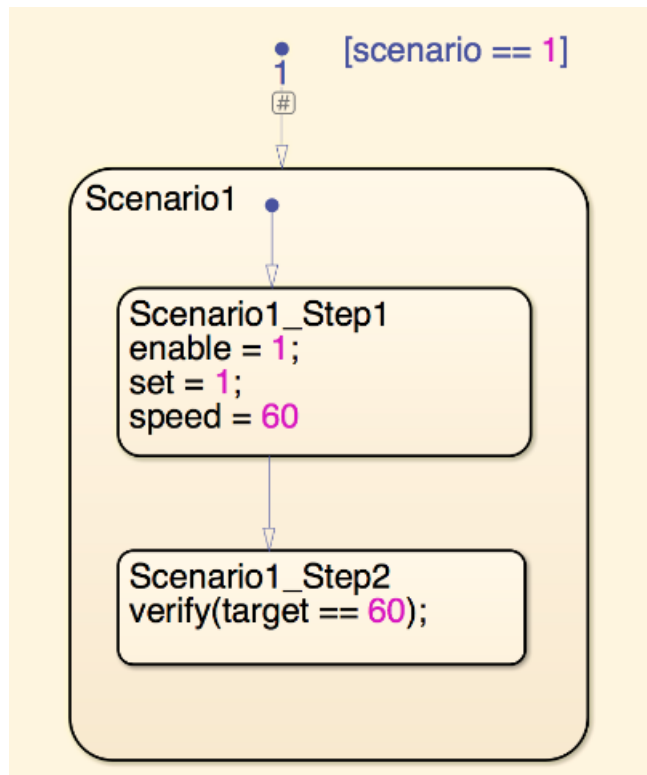
### Specify a Label for verify Result

The result of this `verify` statement is prefaced by the label, `TestReq1:bothGreater` and, if the test fails, the error message.

```
verify(x > y && z > 10, 'TestReq1:bothGreater', ...
 'x, y, and z are %d,%d,%d', x, y, z)
```

### Using verify in a Stateflow Chart

The second step in the Scenario1 state of this Stateflow chart verifies that the target equals 60.



### Tips

- You can use `verify` statements in Test Sequence and Test Assessment blocks and in Stateflow charts. A Stateflow license is required to use a chart. `verify` statements in charts are supported in the same locations, execution modes, and for the same code generation targets as the Test Sequence block.
- You cannot use `verify` statements in:
  - Test Sequence blocks that use continuous-time updating. Test Sequence block data can depend on factors such as the solver step time. Continuous-time updating can cause differences in when block data and `verify` statements update, which can lead to unexpected `verify` statement results. If your model uses continuous time and you use `verify` statements in a Test Sequence or Test Assessment block, consider explicitly setting a discrete block sample time.
  - Moore, Mealy, Discrete Event, or continuous charts
  - Charts that use C as the action language
  - Bind actions in a chart

- Transition or condition actions in a chart
  - MATLAB functions, graphical functions, or truth tables in a chart
  - MATLAB Function or Truth Table blocks
  - Rapid Accelerator mode simulations
  - Code generation targets other than Simulink Real-Time and HDL Verifier™
  - Standalone Stateflow charts
- If you use parallel test execution to run your tests, then you cannot use the **Highlight in Model** button for in the Test Manager to `verify` results.
  - You cannot use `verify` as a condition immediately after `when` in a When decomposition because `verify` statements do not produce output. You can, however, use `verify` statements as actions in When decomposition steps. See “Assess a Model by Using When Decomposition”.
  - When comparing floating-point data in `verify` statements, consider the precision limitations associated with floating-point numbers. If you need to use floating-point data, define a tolerance for the verification. For example, instead of `verify(x == 5)`, verify `x` within a tolerance of 0.001:

```
verify(abs(x-5) < 0.001)
```

For more information, see “Floating-Point Numbers”.

## See Also

`assert` | `getVerifyRuns`

### Topics

“Assess Model Simulation Using `verify` Statements”

“Verify Multiple Conditions at a Time”

“Test Sequence and Assessment Syntax”

“Test Sequence Basics”

“Programmatically Create a Test Sequence”

“Generate Test Signals”

### Introduced in R2016a

