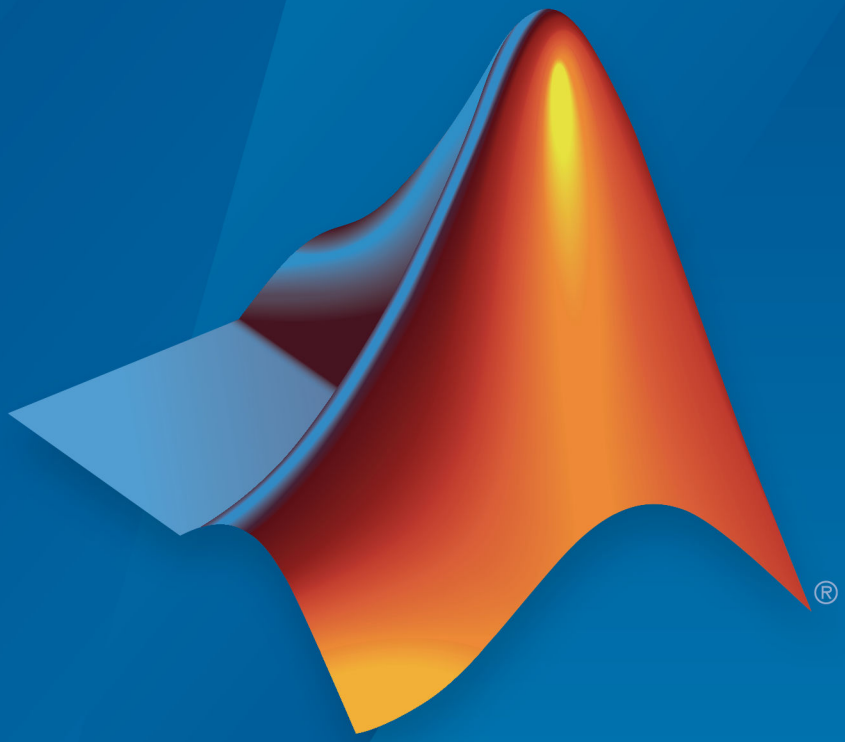
Simulink® Test™

Reference



MATLAB® & SIMULINK®

R2019b

MathWorks®

How to Contact MathWorks

Latest news: `www.mathworks.com`

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

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

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

Phone: 508-647-7000

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

Simulink® Test™ Reference

Revision History

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

Contents

Functions — Alphabetical List

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: [1×1 Simulink.SimulationData.BlockPath]
       Values: [1×1 timeseries]
       Result: Fail
```

Input Arguments

as — Assessment object

`sltest.Assessment` | `sltest.AssessmentSet`

Assessment object for which to display results.

Example: `as`

See Also

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

Introduced in R2016b

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,'-regexp','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,'-regexp','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

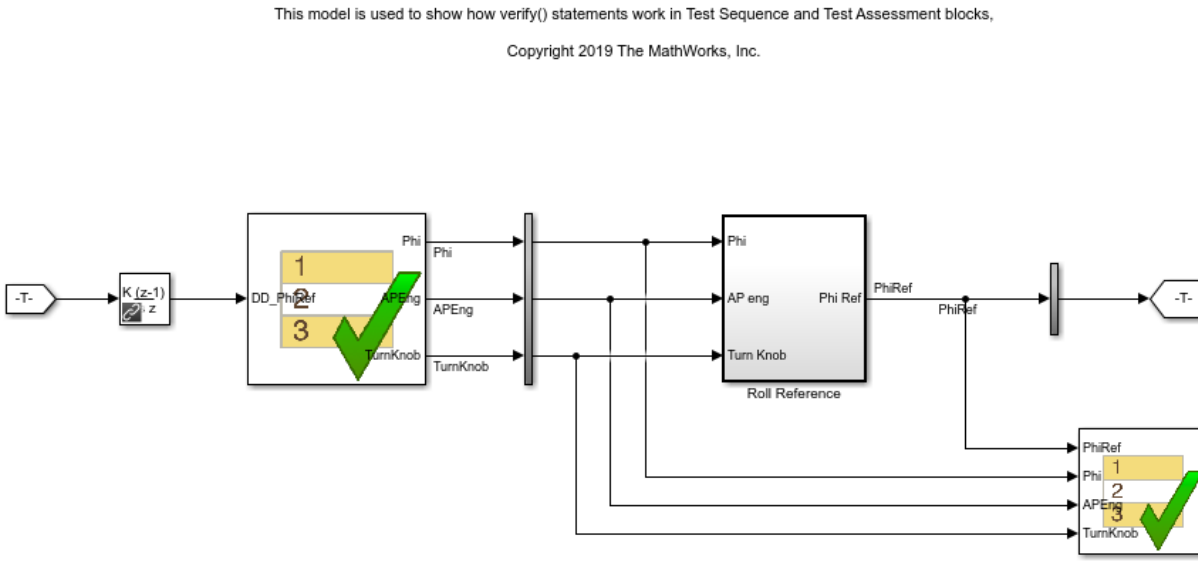
This example shows how to simulate a model with `verify` statements and obtain assessment results via the programmatic interface.

Get the Assessment Set and One Assessment Result

1. Open the model.

```
open_system(fullfile(matlabroot,'examples','simulinktest',...
    'sltestRollRefTestExample.slx'))

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



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: Untested
```

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):
    1 : Untested 'Simulink:verifyTKNormal'
    2 : Untested 'Simulink:verifyTKLow'
    3 : Untested 'Simulink:verify_high'

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

  Passed Assessments (first 10):
    5 : Pass 'Simulink:verify_low'
    6 : Pass 'Simulink:verify_normal'

  Failed Assessments (first 10):
    4 : 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.Assessment` | `sltest.AssessmentSet` | `sltest.getAssessments`

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

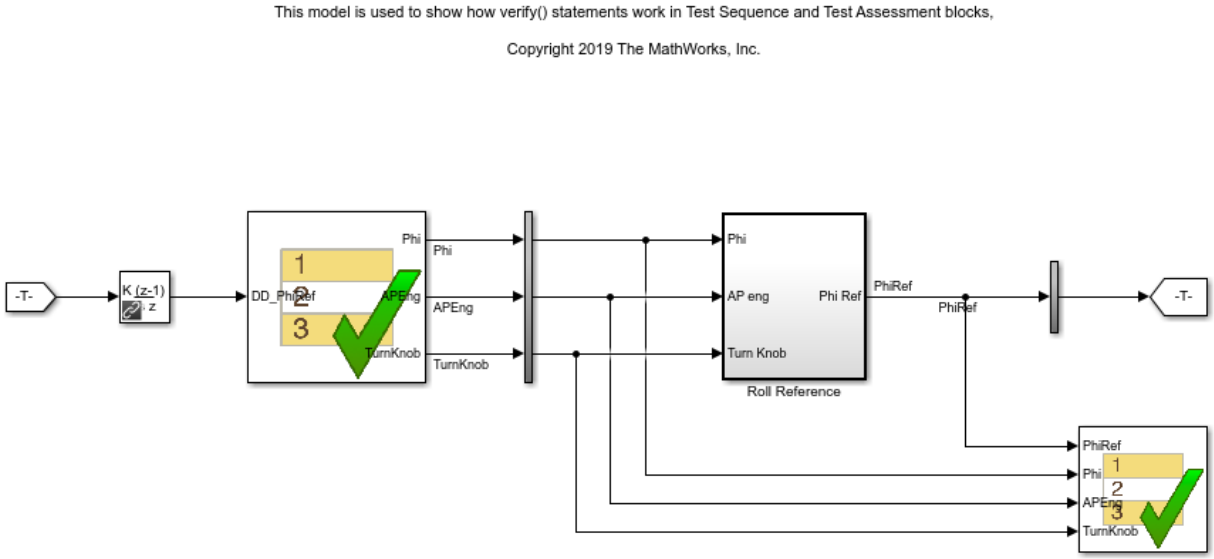
This example shows how to simulate a model with `verify` statements and obtain assessment results via the programmatic interface.

Get the Assessment Set and One Assessment Result

1. Open the model.

```
open_system(fullfile(matlabroot,'examples','simulinktest',...
    'sltestRollRefTestExample.slx'))

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



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: Untested
```

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):
    1 : Untested 'Simulink:verifyTKNormal'
    2 : Untested 'Simulink:verifyTKLow'
    3 : Untested 'Simulink:verify_high'

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

  Passed Assessments (first 10):
    5 : Pass 'Simulink:verify_low'
    6 : Pass 'Simulink:verify_normal'

  Failed Assessments (first 10):
    4 : 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.Assessment` | `sltest.AssessmentSet` | `sltest.getAssessments`

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

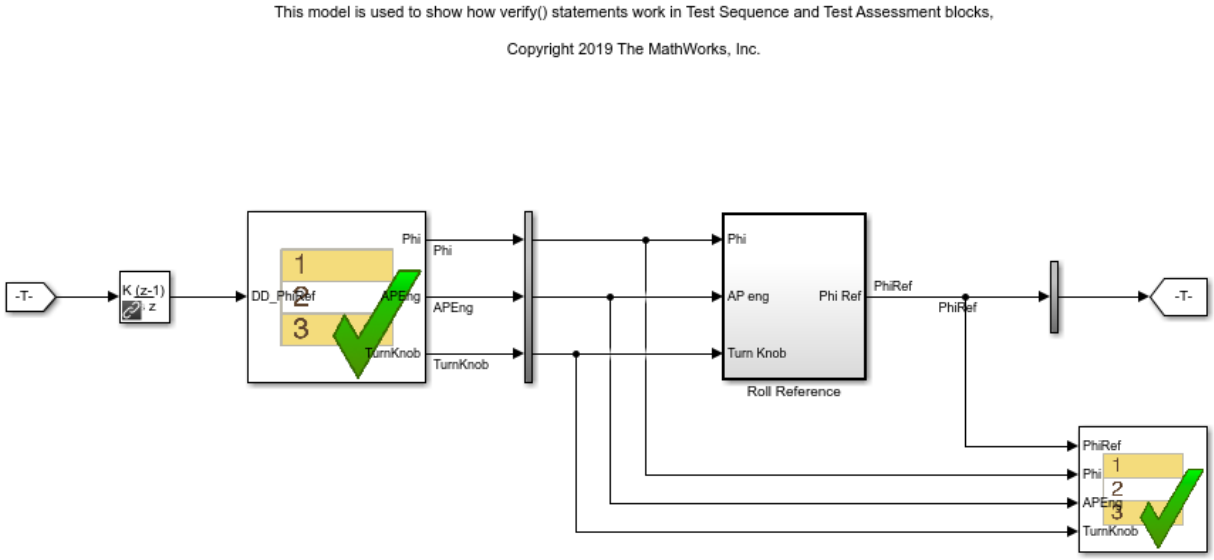
This example shows how to simulate a model with `verify` statements and obtain assessment results via the programmatic interface.

Get the Assessment Set and One Assessment Result

1. Open the model.

```
open_system(fullfile(matlabroot,'examples','simulinktest',...
    'sltestRollRefTestExample.slx'))

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



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: Untested
```

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):
    1 : Untested 'Simulink:verifyTKNormal'
    2 : Untested 'Simulink:verifyTKLow'
    3 : Untested 'Simulink:verify_high'

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

  Passed Assessments (first 10):
    5 : Pass 'Simulink:verify_low'
    6 : Pass 'Simulink:verify_normal'

  Failed Assessments (first 10):
    4 : 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.Assessment` | `sltest.AssessmentSet` | `sltest.getAssessments`

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

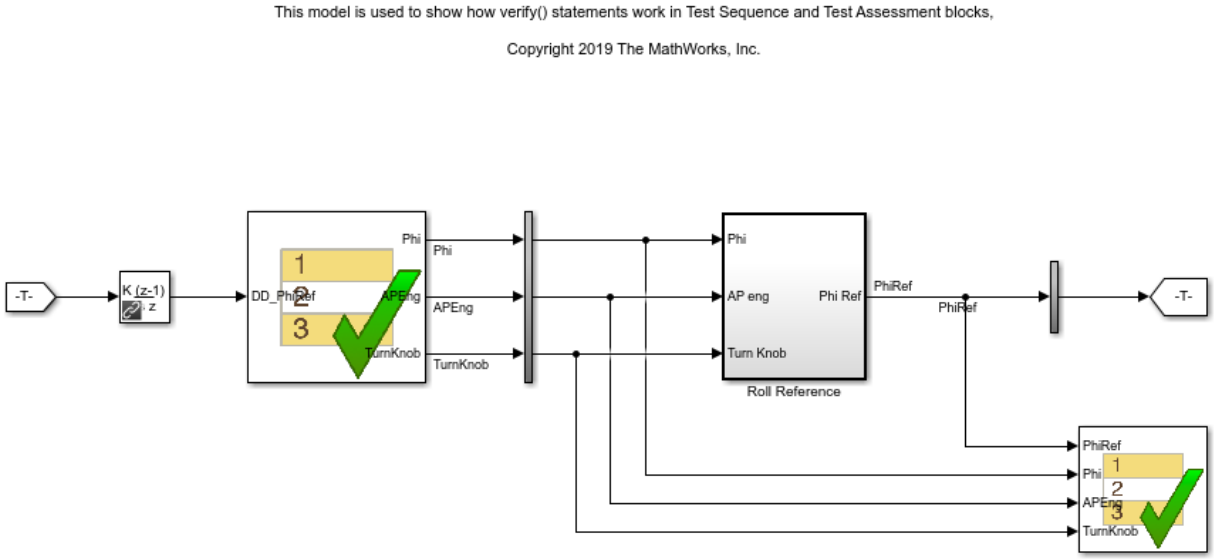
This example shows how to simulate a model with `verify` statements and obtain assessment results via the programmatic interface.

Get the Assessment Set and One Assessment Result

1. Open the model.

```
open_system(fullfile(matlabroot,'examples','simulinktest',...
    'sltestRollRefTestExample.slx'))

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



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: Untested
```

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):
    1 : Untested 'Simulink:verifyTKNormal'
    2 : Untested 'Simulink:verifyTKLow'
    3 : Untested 'Simulink:verify_high'

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

  Passed Assessments (first 10):
    5 : Pass 'Simulink:verify_low'
    6 : Pass 'Simulink:verify_normal'

  Failed Assessments (first 10):
    4 : 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.

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 mo
```

```
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 comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

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`

Introduced in R2016a

sltest.harness.create

Create test harness

Syntax

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

Description

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

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

Examples

Create Harness for a Model

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

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

Create Harness for a Subsystem

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

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

Create Default Harness for a Subsystem

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

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

Input Arguments

`harnessOwner` — Model or component

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 comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

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

`Name` — Harness name

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'`

`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'`

`Source` — Component under test input
`'Inport'` (default) | `'Signal Builder'` | `'From Workspace'` | `'From File'` | `'Test Sequence'` | `'Signal Editor'` | `'None'` | `'Custom'`

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

Example: `'Source','Signal Builder'`

`CustomSourcePath` — Path to library block for custom source
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'`

`Sink` — Harness output
`'Outport'` (default) | `'Scope'` | `'To Workspace'` | `'To File'` | `'None'` | `'Custom'`

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

Example: `'Sink','Scope'`

`CustomSinkPath` — Path to library block for custom sink
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/Terminator'`

`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`

`SynchronizationMode` — Specifies the synchronization behavior of the component under test
`'SyncOnOpenAndClose'` (default) | `'SyncOnOpen'` | `'SyncOnPushRebuildOnly'`

An option to specify when the component under test synchronizes between the main model and the test harness.

- `'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'`

`CreateWithoutCompile` — Option to create harness without compiling main model
`false` (default) | `true`

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

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

`true` creates the harness without model compilation.

Example: `'CreateWithoutCompile',false`

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

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

Example: `'VerificationMode','SIL'`

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

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

Example: `'RebuildOnOpen',true`

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

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

Example: `'RebuildModelData',true`

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

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

Example: `'SaveExternally',true`

`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'`

`PostCreateCallback` — Harness customization after creation
character vector

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

Example: `'PostCreateCallback','HarnessCustomization'`

`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'`

`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.

Example: `'ScheduleInitTermReset',true`

`SchedulerBlock` — Include scheduler block for periodic signals and function calls
`'Test Sequence'` | `'MATLAB Function'` | `'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'`

`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 Builder, From Workspace, or From File blocks.

Example: `'AutoShapeInputs',true`

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

Description

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

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

Examples

Use RegExp to Find Harnesses for a Model Component

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

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

harnessList =

1x2 struct array with fields:

    model
    name
```

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

Input Arguments

`harnessOwner` — Model or component

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 comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

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

`Name` — Harness name to search for

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'`

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.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.

```
exPath = fullfile(matlabroot,'examples','simulinktest');
mainModel = 'sltestBasicCruiseControl';
harnessModel = 'sltestBasicCruiseControlHarnessModel';
```

1. Load the main model.

```
load_system(fullfile(exPath,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=2×19 struct
    model
    name
    description
    type
    ownerHandle
    ownerFullPath
    ownerType
    isOpen
    canBeOpened
    lockMode
    verificationMode
    saveExternally
    rebuildOnOpen
    rebuildModelData
    postRebuildCallback
    graphical
    origSrc
    origSink
    synchronizationMode
      ⋮
```

```
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 comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

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'`

An option to specify when the component under test synchronizes between the main model and the test harness.

- `'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'`

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

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

Example: `'RebuildOnOpen',true`

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

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

Example: `'RebuildModelData',true`

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

Option to have each test harness saved as a separate SLX file, specified as the comma-separated pair consisting of `'SaveExternally'` and `true` or `false`. A model cannot

use both external and internal test harness storage. If a model already has test harnesses, a new test harness follows the storage type of the existing harnesses, which this option does not override. See “Manage Test Harnesses”.

Example: `'SaveExternally',true`

`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'`

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 comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes.

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

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`.

Examples

Push Parameters from Harness to Model

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

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

Input Arguments

harnessOwner — Model or component

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.

Examples

Rebuild Parameters from Harness to Model

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

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

Input Arguments

harnessOwner — Model or component

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 comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

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

`Name` — New harness name

character vector

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

Example: `'Name','new_harness_name'`

`Description` — New harness description
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'`

An option to specify when the component under test synchronizes between the main model and the test harness.

- `'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'`

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

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

Example: `'RebuildOnOpen',true`

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

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

Example: `'RebuildModelData',true`

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

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

Example: `'RebuildWithoutCompile',true`

`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'`

See Also

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

Introduced in R2015a

sltest.import.sldvData

Create test cases from Simulink Design Verifier results

Syntax

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

Description

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

`[owner,testHarness,testFile] = sltest.import.sldvData(
dataFile,'TestCase',testcase)` uses the specified test case for the import operation.

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

Examples

Create Test Cases for ShiftLogic Subsystem

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

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

Create the test case.

```
[component,harness,testfile] = sltest.import.sldvData...
('./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.

Example: `'ShiftLogic0/ShiftLogic0_sldvdata.mat'`

Example: `'Controller_sldvdata.mat'`

`testcase` — Test case
character vector | string scalar

Full path name of test case to use, specified as a character vector or string scalar.

Name-Value Pair Options

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

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

`CreateHarness` — Create a test harness for the model or subsystem
`true` (default) | `false`

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

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

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

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

Example: `'CreateHarness',false`

`TestHarnessName` — Harness name
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 Builder'`

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

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/'`

See Also

Topics

“Increase Coverage by Generating Test Inputs”

Introduced in R2015b

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.close` | `sltest.testmanager.view`

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

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

ans =

  TestCase with properties:

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

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.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, Signal Editor scenarios, and Signal Builder groups in the model and assigns them to a test file. If there are multiple Signal Builder groups, the test file includes iterations for each group instead of separate test cases.

Examples

Create Test Cases From a Model

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

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

```
testFile =

  TestFile with properties:

        Name: 'TestFile'
```

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

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: `'sldemo_autotrans'`

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

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 =

  1×2 struct array with fields:

    Name
    MATLABRoot
    IsDefault
    Selected
```

Get the `Name` value.

```
settings.Name
```

```
ans =

    '14a'


ans =
```

```
'R2017b'
```

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: `'Description'`

Example: `{'TestTag','Description'}`

Output Arguments

settings — Test Manager preferences

struct

Preference settings, returned as a struct.

See Also

`sltest.testmanager.setpref`

Topics

“Test Sections”

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.TestFile` | `sltest.testmanager.view`

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 Example Test File

In this example, the test file is from the “Override Model Parameters in a Test Case” example.

```
% Path to the example test file
exampleFile = fullfile(matlabroot,...
          'toolbox','simulinktest','simulinktestdemos',...
          'sltestParameterOverridesTestSuite.mldatx');

% Load the example test file
sltest.testmanager.load(exampleFile);
```

```
% View the test file in the test manager
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.

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

Topics

“Create and Run Test Cases with Scripts”

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),...
    'IncludeSimulitionSignalPlots',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` 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 comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'IncludeTestRequirement',true`

`Author` — Report author

empty 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`.

`IncludeErrorMessages` — 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
`true` (default) | `false`

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

See Also

`sltest.testmanager.ResultSet` | `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, returning a `sltest.testmanager.ResultSet` object `resultObj`.

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

Examples

Run a Test Case

You can run tests at a test-file, test-suite, or test-case level if they are loaded in the test manager.

```
% 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');

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

```
% Run the test case and return results data
ro = run(tc);
```

Input Arguments

Name-Value Pair Options

Example: `'Parallel',true,'Tags',{'safety'}`

`Parallel` — Run with parallel computing

false (default) | true

Specifies whether to run tests with Parallel Computing Toolbox™. Requires Parallel Computing Toolbox license.

Example: `'Parallel',true`

`Tags` — Run only tests with specified tags

cell array of character vectors

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

Example: `'Tags',{'safety'}`

Example: `'Tags',{'safety','regression'}`

Output Arguments

`resultObj` — Results set object

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`

Topics

“Create and Run Test Cases with Scripts”

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)
```

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.

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 ManagerMATLAB Release preferences.

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

```
r1 = struct('Name','12b',...
            'MATLABRoot','\\mycompany\R2012b\matlab',...
```

```
            'Selected',true);
r2 = struct('Name','14a',...
            'MATLABRoot','\\mycompany\R2014a\matlab',...
            'Selected',true);
r3 = struct('Name','15a',...
            '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','13a',...
            'MATLABRoot','\\mycompany\R2013a\matlab',...
            'Selected',true);
sltest.testmanager.setpref('MATLABReleases','13a',{r4});
```

Delete a release from the preferences.

```
sltest.testmanager.setpref('MATLABReleases','14a',{[]});
```

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

`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: `'Description'`

Example: `{'Description','TagText'}`

`value` — Preference value
Boolean | cell array of Boolean 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: `'11a'`

`releasePrefs` — Release information
struct | cell array of structs

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

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

Example: `struct('Name','14a','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','14a','MATLABRoot','\\mypath','Selected',true)`

Output Arguments

`settings` — Test Manager preferences
struct

Preference settings, returned as a struct.

See Also

`sltest.testmanager.getpref`

Topics

“Test Sections”
“Run Tests in Multiple Releases”

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 manager 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. All other properties default to `true` and thus, their information is included in the report.

```
testmgrFile = fullfile(matlabroot, ...
      'toolbox', 'simulinktest', 'simulinktestdemos', ...
      '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);
```

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 a Simulink Report Generator license. See “Templates” (MATLAB Report Generator) for more information.

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

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

zipTemplate('MyCustomTemplate.pdftx')

testmgrFile = fullfile(matlabroot, ...
      'toolbox', 'simulinktest', 'simulinktestdemos', ...
      '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.pdftx');
```

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 comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

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.

`IncludeTest Details` — 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.

`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 genearated 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 `pdftx`, `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 `pdftx`, `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.TestCase` | `sltest.testmanager.TestFile` | `sltest.testmanager.TestSuite` | `sltest.testmanager.getTestFiles`

Topics

“Generate a Test Specification Report” on page 1-114
“Generate a Customized Test Specification Report” on page 1-115

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

```
% 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.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

character vector

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

Example: `'SystemHeatingTest.InitializeHeating'`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

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`.

Set paths and open the model.

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

Open the test harness.

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

Create a new local variable `h`.

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

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

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

Input Arguments

`blockPath` — Test Sequence block path

character vector

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

Example: `'SystemHeatingTest.InitializeHeating'`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

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`.

Set paths and open the model.

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

Open the test harness.

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

Create a new local variable `h`.

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

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

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

Input Arguments

`blockPath` — Test Sequence block path

character vector

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

Example: `'SystemHeatingTest.InitializeHeating'`

Name-Value Pair Options

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

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.

Set paths and open the model.

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

Open the test harness.

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

Add a parameter.

```
sltest.testsequence.addSymbol('RollReference_Requirement1_3/Test Sequence',...
'theta','Data','Parameter')
```

Input Arguments

`blockPath` — Test Sequence block path

character vector

Path to a Test Sequence block, including the block name, specified as a 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.find` | `sltest.testsequence.findSymbol` | `sltest.testsequence.readSymbol`

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.

```
cd(matlabroot);
Model = 'sltestRollRefTestExample';
load_system(fullfile(matlabroot,'examples','simulinktest',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

character vector

Path to a Test Sequence block, including the block name, specified as a 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.find` | `sltest.testsequence.readTransition`

Topics

“Programmatically Create a Test Sequence”

Introduced in R2016a

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.

```
cd(matlabroot);
Model = 'sltestRollRefTestExample';
load_system(fullfile(matlabroot,'examples','simulinktest',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

character vector

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

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.

```
cd(matlabroot);
Model = 'sltestRollRefTestExample';
load_system(fullfile(matlabroot,'examples','simulinktest',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
character vector

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

```
cd(matlabroot);
Model = 'sltestRollRefTestExample';
load_system(fullfile(matlabroot,'examples','simulinktest',Model));
```

Delete the transition from the test step `SetKnobAndPhi` under the parent step `TurnKnobAndAttitude`.

```
sltest.testsequence.deleteTransition...
('sltestRollRefTestExample/Test Sequence',...
'TurnKnobAndAttitude.SetKnobAndPhi',1)
```

Close the model.

```
close_system(Model,0);
```

Input Arguments

`blockPath` — Test Sequence block path
character vector

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

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.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

character vector

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

Example: `'SystemHeatingTest.InitializeHeating'`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

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.

```
cd(matlabroot)
Model = 'sltestRollRefTestExample';
load_system(fullfile(matlabroot,'examples','simulinktest',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

character vector

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

Example: `'FanSpeedTestHarness/Test Sequence'`

`name` — Symbol name

character vector

Name of the symbol, specified as a character vector.

Example: `'theta'`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Find valid name-value pairs by applying `sltest.testsequence.readSymbol` to the existing symbol.

Example: `'DataType','single','Scope','Constant'`

See Also

`sltest.testsequence.addStep` | `sltest.testsequence.addStepAfter` | `sltest.testsequence.addStepBefore` | `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.

```
cd(matlabroot);
Model = 'sltestRollRefTestExample';
load_system(fullfile(matlabroot,'examples','simulinktest',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

character vector

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

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 comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

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.

```
cd(matlabroot)
Model = 'sltestRollRefTestExample';
load_system(fullfile(matlabroot,'examples','simulinktest',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 array
  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 comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

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.

```
cd(matlabroot)
Model = 'sltestRollRefTestExample';
load_system(fullfile(matlabroot,'examples','simulinktest',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

character vector

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

Example: `'FanSpeedTestHarness/Test Sequence'`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

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.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 struct `blockInfo` 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.

```
cd(matlabroot)
Model = 'sltestRollRefTestExample';
load_system(fullfile(matlabroot,'examples','simulinktest',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: ''
```

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. 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: 0
          ActiveStepDataSymbol: ''
```

5. Close the model.

```
close_system(Model,0)
```

Input Arguments

`blockPath` — Test Sequence block path

character vector

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

Example: `'FanSpeedTestHarness/Test Sequence'`

`propertyName` — Test Sequence block property name

`'Name'` | `'UpdateMethod'` | `'SampleTime'` | `'Description'` | `'Document'` | `'Tag'` | `'SupportVariableSizing'` | `'SaturateOnIntegerOverflow'` | `'InputFimath'` | `'FimathForFiConstructors'` | `'EnableActiveStepData'` | `'ActiveStepDataSymbol'`

Name of a particular Test Sequence block property to get a value for.

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.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: ''
```

3. Close the model.

```
close_system('tsb_model',0)
```

Input Arguments

`blockPath` — Test Sequence block path

character vector

Path to a Test Sequence block, including the block name, specified as a 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

Find Test Sequence blocks

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.

```
cd(matlabroot)
Model = 'sltestRollRefTestExample';
load_system(fullfile(matlabroot,'examples','simulinktest',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

character vector

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

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.

```
cd(matlabroot)
Model = 'sltestRollRefTestExample';
load_system(fullfile(matlabroot,'examples','simulinktest',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

character vector

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

```
cd(matlabroot)
Model = 'sltestRollRefTestExample';
load_system(fullfile(matlabroot,'examples','simulinktest',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
character vector

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

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`.

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.

```
cd(matlabroot)
Model = 'sltestRollRefTestExample';
load_system(fullfile(matlabroot,'examples','simulinktest',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: ''
```

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. 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: 0
           ActiveStepDataSymbol: ''
```

5. Close the model.

```
close_system(Model,0)
```

Input Arguments

`blockPath` — Test Sequence block path

character vector

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

Example: `'FanSpeedTestHarness/Test Sequence'`

Name-Value Pair Arguments

Example: `'Description','Temperature cycle','EnableActiveStepData',true`

Valid name-value pairs are block-specific. Obtain properties for the block using `sltest.testsequence.getProperty`. For example:

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: ''
```

3. Close the model.

```
close_system('tsb_model',0)
```

See Also

`sltest.testsequence.find` | `sltest.testsequence.getProperty` | `sltest.testsequence.newBlock`

Topics

“Programmatically Create a Test Sequence”

Introduced in R2017a

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 Builder Groups

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

```
%% Iterate Over All Signal Builder Groups

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

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

    % Set iteration settings
```

```
    setTestParam(testItr,'SignalBuilderGroup',sltest_signalBuilderGroups{k});

    % 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`

Topics

“Test Iterations”
“Create and Run Test Cases with Scripts”

Introduced in R2016a

Classes — Alphabetical List

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: [1×1 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

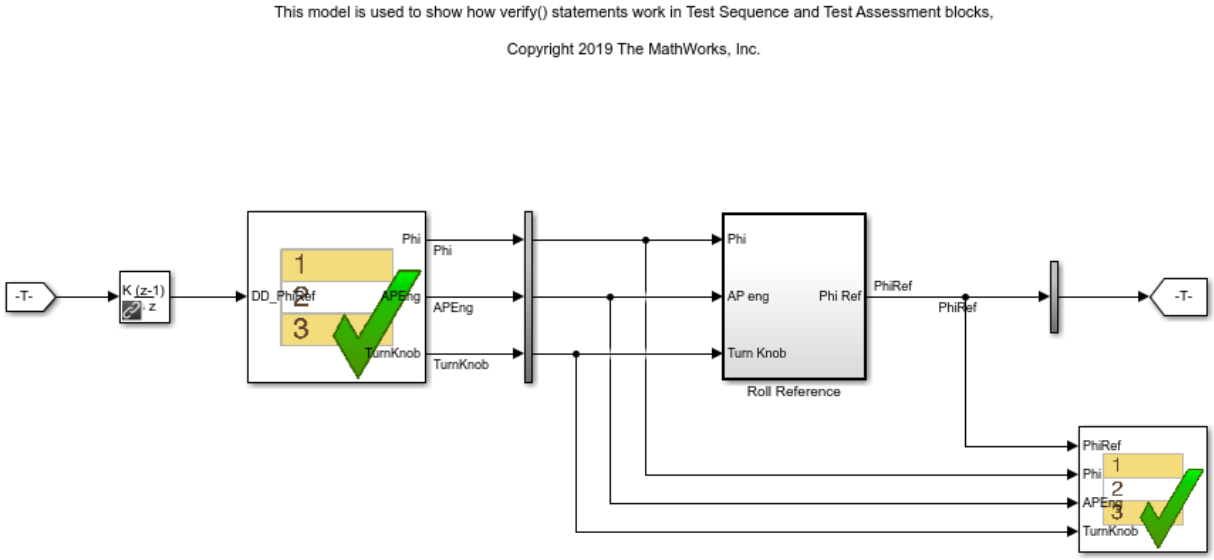
This example shows how to simulate a model with `verify` statements and obtain assessment results via the programmatic interface.

Get the Assessment Set and One Assessment Result

1. Open the model.

```
open_system(fullfile(matlabroot,'examples','simulinktest',...
    'sltestRollRefTestExample.slx'))

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



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: Untested
```

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):
    1 : Untested 'Simulink:verifyTKNormal'
    2 : Untested 'Simulink:verifyTKLow'
    3 : Untested 'Simulink:verify_high'

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

  Passed Assessments (first 10):
    5 : Pass 'Simulink:verify_low'
    6 : Pass 'Simulink:verify_normal'

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

Re-enable warnings

```
warning on Stateflow:Runtime:TestVerificationFailed
```

See Also

`sltest.AssessmentSet` | `sltest.getAssessments`

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

disp	Display results of sltest.AssessmentSet or sltest.Assessment
find	Find assessments in sltest.AssessmentSet or sltest.Assessment object
get	Get assessment of sltest.AssessmentSet
getSummary	Get summary of sltest.AssessmentSet

Examples

Get Assessments from a Simulation

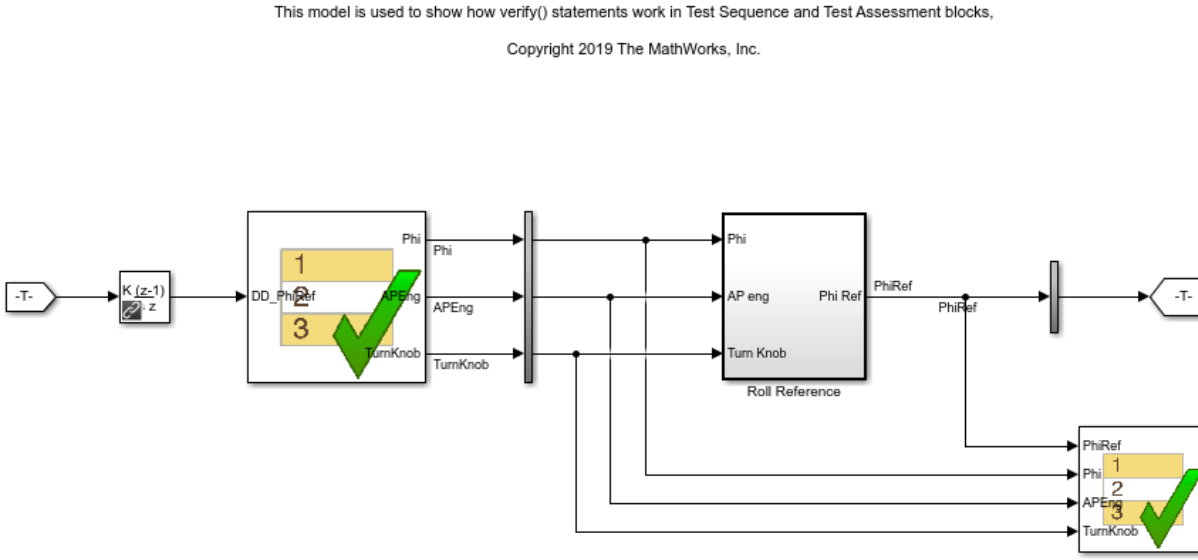
This example shows how to simulate a model with `verify` statements and obtain assessment results via the programmatic interface.

Get the Assessment Set and One Assessment Result

1. Open the model.

```
open_system(fullfile(matlabroot,'examples','simulinktest',...
    'sltestRollRefTestExample.slx'))

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



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: Untested
```

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):
    1 : Untested 'Simulink:verifyTKNormal'
```

```
    2 : Untested 'Simulink:verifyTKLow'
    3 : Untested 'Simulink:verify_high'

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

  Passed Assessments (first 10):
    5 : Pass 'Simulink:verify_low'
    6 : Pass 'Simulink:verify_normal'

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

Re-enable warnings

```
warning on Stateflow:Runtime:TestVerificationFailed
```

See Also

`sltest.Assessment` | `sltest.getAssessments`

Introduced in R2016b

sltest.plugins.ModelCoveragePlugin class

Package: `sltest.plugins`

Collect model coverage using the MATLAB Unit Test framework

Description

The `sltest.plugins.ModelCoveragePlugin` class creates a coverage collection object for running Simulink Test test cases with the MATLAB Unit Test framework. Add the `sltest.plugins.ModelCoveragePlugin` object to the test runner.

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 the object:

```
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.

Example: `'Collecting',covSettings`

Example: `'Collecting',CoverageMetrics('MCDC',true,'Decision',false,'Condition',false)`

Attributes:

SetAccess	public
GetAccess	public

`Producing` — Specify model coverage report options
`ModelCoverageReport` object

Property that specifies coverage report options with a `sltest.plugins.coverage.ModelCoverageReport` object.

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',false,'Condition',false,'MCDC',true);

covSettings = ModelCoveragePlugin('RecordModelReferenceCoverage',true,...
    'Collecting',mcdcMet);
```

3. Create a MATLAB® Unit Test test suite from the test file.

```
tf = sltest.testmanager.TestFile(fullfile(matlabroot,'toolbox','simulinktest',...
    'simulinktestdemos','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)
```

```
APResult =
  TestResult with properties:

          Name: 'AutopilotTestFile > Basic Design Test Cases/Requirement 1.3 Test'
        Passed: 0
        Failed: 1
    Incomplete: 1
      Duration: 0.0663
       Details: [1x1 struct]
```

```
Totals:
   0 Passed, 1 Failed, 1 Incomplete.
   0.066328 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`

Topics

“Test Models Using MATLAB Unit Test”

Introduced in R2018a

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 a plugin object `tmr` that directs 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 comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes.

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

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 = fullfile(matlabroot,'examples',...
    'simulinktest','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(S
        Passed: 0
        Failed: 1
    Incomplete: 0
      Duration: 1.5151
       Details: [1x1 struct]

Totals:
   0 Passed, 1 Failed, 0 Incomplete.
   1.5151 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 and Generate Test Results Reports”
“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.

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.

Example: `'Condition',true`

Attributes:

SetAccess	public
GetAccess	public

`MCDC` — MCDC coverage
false (default) | true

Enable or disable modified condition / decision coverage collection.

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',false,'Condition',false,'MCDC',true);

covSettings = ModelCoveragePlugin('RecordModelReferenceCoverage',true,...
    'Collecting',mcdcMet);
```

3. Create a MATLAB® Unit Test test suite from the test file.

```
tf = sltest.testmanager.TestFile(fullfile(matlabroot,'toolbox','simulinktest',...
    'simulinktestdemos','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)

APResult =
  TestResult with properties:

          Name: 'AutopilotTestFile > Basic Design Test Cases/Requirement 1.3 Test'
        Passed: 0
        Failed: 1
    Incomplete: 1
      Duration: 0.0663
       Details: [1x1 struct]

Totals:
   0 Passed, 1 Failed, 1 Incomplete.
   0.066328 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)

Introduced in R2018a

sltest.plugins.coverage.ModelCoverageReport class

Package: `sltest.plugins.coverage`

Specify model coverage report details for tests run with MATLAB Unit Test

Description

`mcr = sltest.plugins.coverage.ModelCoverageReport(path)` creates `mcr`, a `ModelCoverageReport` that specifies the report folder `path`. Use a `ModelCoverageReport` to specify report properties when you run Simulink Test test files run with the MATLAB Unit Test framework.

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.

Construction

`mcr = sltest.plugins.coverage.ModelCoverageReport(path)` creates `mcr`, a `ModelCoverageReport` that specifies the report folder `path`.

You can also import the class, then use the name:

```
import sltest.plugins.coverage.ModelCoverageReport
mcr = ModelCoverageReport(path)
```

Input Arguments

`path` — Report folder
character vector

The path of the folder in which the model coverage report is saved after the test is complete.

Example: `'results/reports/coverage/model'`

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: 1
       Duration: 0.1039
        Details: [1x1 struct]

Totals:
   0 Passed, 1 Failed, 1 Incomplete.
   0.10391 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.ModelCoveragePlugin` | `sltest.plugins.coverage.CoverageMetrics`

Topics

“Test Models Using MATLAB Unit Test”

Introduced in R2018b

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.

Construction

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

Properties

`AbsTol` — Absolute tolerance

scalar

Absolute tolerance for the baseline criteria set, specified as a 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.

`ExcelSpecifications` — Sheet and range information for Excel baseline file

1-by-N array

This property is read-only.

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'` Rectangular Range	Specify the range using the syntax `'Corner1:Corner2'`, where *`Corner1`* and *`Corner2`* 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). **Example:** `'Range','Corner1:Corner2'`
`''` Unspecified or Empty	If unspecified, the importing function automatically detects the used range. **Example:** `'Range',''` **Note:** *Used Range* 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'` Row Range	You can identify the range by specifying the beginning and ending rows using Excel row designators. Then `readtable` 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). **Example:** `'Range','1:7'`

Ways to specify Range	Description
`'Column1:Column2'` Column Range	You can identify the range by specifying the beginning and ending columns using Excel column designators. Then `readtable` 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). **Example:** `'Range','A:F'`
`'NamedRange'` Excel's 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. **Example:** `'Range','myTable'`

`FilePath` — File path

character vector

This property is read-only.

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

`LaggingTol` — Lagging time tolerance

scalar

Lagging time tolerance for the baseline criteria set, specified as a scalar.

`LeadingTol` — Leading tolerance

scalar

Leading time tolerance for the baseline criteria set, specified as a scalar.

`Name` — Name of baseline criteria

character vector

This property is read-only.

Name of the baseline criteria, returned as a character vector.

`RelTol` — Relative tolerance

scalar

Relative tolerance for the baseline criteria set, specified as a scalar.

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`.

```
% 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.SignalCriteria` | `sltest.testmanager.TestCase`

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

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 = fullfile(matlabroot,'examples','simulinktest',...
    '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))
```

4. Get the comparison signal result of the run result.

```
cr2sig = getComparisonSignalResults(cr2)
```

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

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 = fullfile(matlabroot,'examples','simulinktest',...
    '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))
```

4. Get the comparison signal result of the run result.

```
cr2sig = getComparisonSignalResults(cr2)
```

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 setting under the **Coverage Settings** section in a test file, test suite, or test case.

Construction

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

Properties

`RecordCoverage` — Enable coverage collection

`false` (default) | `true`

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

Coverage collection is enabled or disabled in the Test Manager **Coverage Settings**. The corresponding UI option is **Record coverage for system under test**.

`MdlRefCoverage` — 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**.

`MetricSettings` — Coverage metric setting selection

character vector

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

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

Example: `'dw'`

`CoverageFilterFilename` — Coverage filter to use for coverage analysis

character vector

Absolute path or relative path of a coverage filter to use during coverage analysis, or the file name of a coverage filter on the path, specified as a character vector. The default setting, `'[Model Settings]'`, honors the model configuration parameter 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, or from test suites to test cases.

Specify the coverage filter filename in the Test Manager, in the **Coverage filter filename** field of the **Coverage Settings** section.

Example: `'[Model Settings]'` (default)

Example: `'covfilter.cvf'`

Example: `''`

Examples

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

```
% Specify a coverage filter
cov.CoverageFilterFilename = 'covfilter.cvf';
```

See Also

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

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.

Construction

`obj = getCustomCriteria(tc)` creates an `sltest.testmanager.CustomCriteria` object for a test case object `tc`.

`output_args = function(input_args,Name,Value)`.

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

`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),'.']);

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.

Construction

`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

`DiagnosticRecord` — Criteria script
`sltest.testmanager.DiagnosticRecord` object.

Diagnostic record of the custom criteria result, returned as an `sltest.testmanager.DiagnosticRecord` object.

Example: `DiagnosticRecord`

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 a verification during custom criteria analysis.

Construction

`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

`TestDiagnosticResult` — Record test diagnostic results

cell array

Diagnostic record of the custom criteria result, returned as a cell array.

Example: `'Final: 0.'`

`FrameworkDiagnosticResult` — Record framework diagnostic results
cell array

Framework diagnostic record of the custom criteria result, returned as a cell array.

Example: `'verifyEqual passed.…'`

`Event` — Record event name
character vector

Name of the recorded event, returned as a character vector.

Example: `VerificationPassed`

`Report` — Record diagnostic information
character vector

Report of the diagnostic result, returned as a character vector.

`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`

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.

Construction

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

Properties

`Enabled` — Enabled indicator
`0` | `1`

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

Methods

getSignalCriteria	Get signal criteria
remove	Remove equivalence criteria

Examples

Add Equivalence Criteria to Test Case

```
% 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
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` | `addDataStoreSignal` | `addLoggedSignal` | `getLoggedSignals` | `sltest.testmanager.LoggedSignalSet`

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 comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

`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

addDataStoreSignal	Add a data store or Simulink.Signal object to a set
addLoggedSignal	Add a logged signal to a set
getLoggedSignals	Return logged signals contained in a set
remove	Remove a logged signal set

Examples

Add a Signal Set to a Test Case

Open a model and create a signal set.

```
% Open model
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

`addLoggedSignalSet` | `getLoggedSignalSets` | `sltest.testmanager.EquivalenceCriteria` | `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 these options. View options using:

- `getOptions (TestCase)`
- `getOptions (TestSuite)`
- `getOptions (TestFile)`

Construction

`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.

`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.

`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".

`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`.

`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`.

`IncludeCoverageResult` — Option to include coverage results in reports
`false` (default) | `true`

Option to include coverage results in reports, specified as `true` or `false`.

`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`.

`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.

`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`.

`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`.

`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`.

`IncludeTestRequirement` — Option to include test requirements in reports
`true` (default) | `false`

Option to include test requirements in reports, specified as `true` or `false`.

`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`

`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`

`ReportPath` — Path name of file to save report to
character vector

Path name of file to save report to, specified as a character vector.

`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`.

`Title` — Title of report
character vector

Title of the report, specified as a character vector.

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 (TestCase)` | `getOptions (TestFile)` | `getOptions (TestSuite)`

Topics

“Create, Store, and Open MATLAB Figures”
“Export Test Results and Generate Test Results Reports”

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.

Construction

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

Properties

`Name` — Parameter override name
character vector

Name of the parameter override, specified as a character vector.

`Value` — Override value
scalar | vector

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

`Enabled` — Enabled indicator
`0` | `1`

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

`Source` — Parameter override source
character vector

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

Methods

remove	Remove parameter override

Examples

Add Parameter Override to Test Case

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

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.

Construction

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

Properties

`Name` — Parameter set name
character vector

Name of the parameter set, specified as a character vector.

`FilePath` — File path
character vector

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

`Enable` — Enabled indicator
`0` | `1`

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

Methods

addParameterOverride	Add parameter override to set
getParameterOverrides	Get parameter overrides
remove	Remove parameter set

Examples

Add Parameter Override to Test Case

```
% 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.

Construction

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

Properties

`CoverageResults` — Coverage analysis results
row vector of `cvdata` objects

This property is read-only.

Coverage analysis results, returned as a row vector of `cvdata` objects. For information on `cvdata` objects, see cv.cvdatagroup.

`Duration` — Length of time the test ran, in seconds
duration

This property is read-only.

Length of time the test ran, in seconds, returned as a duration.

`Release` — Release in which the test was run
character vector

This property is read-only.

Release in which the test was run, returned as a character vector.

`NumDisabled` — Number of disabled tests
integer

This property is read-only.

Number of test cases that were disabled in the results set.

`NumFailed` — Number of failed tests
integer

This property is read-only.

Number of failed tests contained in the results set.

`NumPassed` — Number of passed tests
integer

This property is read-only.

Number of passed tests contained in the results set.

`NumTestCaseResults` — Number of test case result children
integer

This property is read-only.

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

`NumTestSuiteResults` — Number of test suite result children
integer

This property is read-only.

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

`NumTestFileResults` — Number of test file result children
integer

This property is read-only.

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

`NumTotal` — Total number of tests
integer

This property is read-only.

Total number of tests in the results set.

`Outcome` — Test outcome
`Passed` | `Failed`

This property is read-only.

Test outcome, returned as `Passed` or `Failed`.

`StartTime` — Time the test began to run
datetime

This property is read-only.

Time the test began to run, returned as a datetime.

`StopTime` — Time the test completed
datetime

This property is read-only.

Time the test completed, returned as a datetime.

Methods

getCoverageResults	Get coverage results
remove	Remove result set
getTestCaseResults	Get test case results object
getTestFileResults	Get test suite results object
getTestSuiteResults	Get test suite results object

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

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.

Construction

`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.

`BlockPath` — Signal block path
character vector

Signal block path, returned as a character vector. This property is read-only.

`DataSource` — Signal data source
character vector

Signal data source, returned as a character vector. This property is read-only.

`Enabled` — Enabled indicator
`0` | `1`

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

`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.

`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.

`LeadingTol` — Leading 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.

`Name` — Signal name
character vector

Signal name, returned as a character vector. This property is read-only.

`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.

`SID` — Signal identifier
character vector

Signal identifier, returned as a character vector. This property is read-only.

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.

Methods

remove	Remove signal criteria

Examples

Set Absolute Tolerance in Baseline Criteria

```
% 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.BaselineCriteria` | `sltest.testmanager.EquivalenceCriteria` | `sltest.testmanager.TestCase`

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`.

Construction

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

Input Arguments

`parent` — Parent test suite

`sltest.testmanager.TestSuite` object

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

`type` — Test case type

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

Test case type, specified as `'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.

`name` — Test case name
character vector

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

Example: `'Test Case 5'`

`runOnTarget` — Run simulation on target
cell array of Booleans

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

Properties

`Description` — Test case description
character vector

Test case description text, specified as a character vector.

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

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

`Name` — Test case name
character vector

Name of the test case, returned as a character vector.

`Parent` — Parent test suite
`sltest.testmanager.TestSuite` object

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

`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`.

`Releases` — Releases available for testing
string array

This property is read-only.

Releases available for testing, returned as a string array. Add releases using `sltest.testmanager.setpref`.

`Requirements` — Test file requirements
structure array

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

`RunOnTarget` — Target indicator
cell array

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

`Tags` — Tags to categorize by
character vector | string array

Tags to use for categorizing, specified as a character vector or string array.

`TestType` — Test case type
character vector

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

`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.

`TestPath` — Test hierarchy
character vector

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

Methods

addBaselineCriteria	Add baseline criteria to test case
addInput	Add input file to test case
addIteration	Add test iteration to test case
getTestCaseResults	Get test case results history
addLoggedSignalSet	Add logged signal set to a test case
addParameterSet	Add parameter set
createInputDataFile	Create file as basis for test case input signal data
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
deleteIterations	Delete test iterations that belong to test case
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
remove	Remove test case
run	Run test case
setProperty	Set test case property

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”

Introduced in R2015b

sltest.testmanager.TestCaseResult class

Package: `sltest.testmanager`

Access test case results data

Description

An `sltest.testmanager.TestCaseResult` enables you to access results from executing test cases or test files.

Construction

`tcr = getTestCaseResults(ResultSet)` returns `tcr` a test case result from a `ResultSet`.

Properties

`Duration` — Length of time the test case ran, in seconds
duration

This property is read-only.

Length of time the test case ran, in seconds, returned as a duration.

`NumDisabledIterations` — Number of disabled tests
integer

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

`NumFailedIterations` — Number of failed tests
integer

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

`NumIncompleteIterations` — Number of incomplete tests
integer

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

`NumPassedIterations` — Number of passed tests
integer

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

`NumTotalIterations` — Total number of tests
integer

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

`Outcome` — Outcome of test case result
0 | 1 | 2 | 3

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

`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.

`Release` — Release in which the test was run
character vector

This property is read-only.

Release in which the test was run, returned as a character vector.

`RunOnTarget` — Target indicator
cell array

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

`StartTime` — Time the test case began to run
datetime

This property is read-only.

Time the test case began to run, returned as a datetime.

`StopTime` — Time the test case completed
datetime

This property is read-only.

Time the test case completed, returned as a datetime.

`Tags` — Tags to filter test file results
string array

This property is read-only.

Tags to filter the test file results. Use tags to view a subset of the test results. See “Tags” for more information.

`TestCasePath` — Result hierarchy path
character vector

The hierarchy path in the parent result set.

`TestCaseType` — Type of test case
`'Simulation'` | `'Baseline'` | `'Equivalence'`

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

`TestFilePath` — Test file path
character vector

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

`ErrorMessages` — Error messages
array of strings

Error messages produced by the test case, returned as an array of strings.

`LogMessages` — Log messages
array of strings

Log messages produced by the test case, returned as an array of strings.

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 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);

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

See Also

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

Topics

“Create and Run Test Cases with Scripts”

Introduced in R2015a

sltest.testmanager.TestFile class

Package: `sltest.testmanager`

Create or modify test file

Description

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

Construction

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

Input Arguments

`filePath` — File name and path
character vector

The file name and path of the test file, specified as a character vector. The specified file name determines the name of the test file as seen in the Test Manager.

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

`mode` — Override existing test files
`false` (default) | `true`

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

Properties

`Description` — Test file description
character vector

Test file description text, specified as a character vector.

`Dirty` — Unsaved changes indicator
`0` | `1`

Indicates if the test file has unsaved changes, `0` if there are no unsaved changes, and `1` if there are unsaved changes.

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

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

`FilePath` — File path and name
character vector

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

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

`Name` — Test file name
character vector

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

`Releases` — Releases available for testing
string array

This property is read-only.

Releases available for testing, returned as a string array. Add releases using `sltest.testmanager.setpref`.

`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`.

`Requirements` — Test file requirements
structure array

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

`Tags` — Tags to categorize by
character vector | string array

Tags to use for categorizing, specified as a character vector or string array.

Methods

close	Close test file in Test Manager
convertTestType	Convert test from one type to another
createTestForSubsystem	Create test harness and test case for subsystem in test file
createTestSuite	Create new test suite
getCoverageSettings	Get coverage settings
getOptions	Get and set test file options
getProperty	Get test file property
getTestSuiteByName	Get test suite object by name
getTestSuites	Get test suites at first level of test file
run	Run test cases in test file
saveToFile	Save test file
setProperty	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.TestCase` | `sltest.testmanager.TestSuite`

Topics

“Create and Run Test Cases with Scripts”

Introduced in R2015b

sltest.testmanager.TestFileResult class

Package: `sltest.testmanager`

Access test file results data

Description

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

Construction

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

Properties

`Duration` — Length of time the test ran, in seconds
duration

This property is read-only.

Length of time the test ran, in seconds, returned as a duration.

`NumDisabled` — Number of disabled tests
integer

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

`NumFailed` — Number of failed tests
integer

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

`NumIncomplete` — Number of incomplete tests
integer

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

`NumPassed` — Number of passed tests
integer

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

`NumTestCaseResults` — Number of test case result children
integer

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

`NumTestSuiteResults` — Number of test suite result children
integer

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

`NumTotal` — Total number of tests
integer

The total number of tests in the test file result.

`Parent` — Parent of the result object
object

Parent of the result. The parent of a test file result is a result set object.

`Release` — Release in which the test was run
character vector

This property is read-only.

Release in which the test was run, returned as a character vector.

`StartTime` — Time the test began to run
datetime

This property is read-only.

Time the test began to run, returned as a datetime.

`StopTime` — Time the test completed

datetime

This property is read-only.

Time the test completed, returned as a datetime.

`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.

`TestFileResultPath` — Result hierarchy path

character vector

The hierarchy path in the result set.

`TestFilePath` — Test file path

character vector

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

Methods

getCleanupPlots	Get plots from cleanup callbacks
getCoverageResults	Get coverage results
getSetupPlots	Get plots from setup callbacks
getTestSuiteResults	Get test suite results object

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.TestCaseResult` | `sltest.testmanager.TestSuiteResult`

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.

Construction

`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.

`ExcelSpecifications` — Sheet and range information for Excel baseline file

1-by-N array

This property is read-only.

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'` Rectangular Range	Specify the range using the syntax `'Corner1:Corner2'`, where *`Corner1`* and *`Corner2`* 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). **Example:** `'Range','Corner1:Corner2'`
`''` Unspecified or Empty	If unspecified, the importing function automatically detects the used range. **Example:** `'Range',''` **Note:** *Used Range* 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'` Row Range	You can identify the range by specifying the beginning and ending rows using Excel row designators. Then `readtable` 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). **Example:** `'Range','1:7'`

Ways to specify Range	Description
'*Column1:Column2*' Column Range	You can identify the range by specifying the beginning and ending columns using Excel column designators. Then `readtable` 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). **Example:** `'Range','A:F'`
'*NamedRange*' Excel's 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. **Example:** `'Range','myTable'`

`FilePath` — File path
character vector

File path of the test input, returned as a character vector.

Example: `'C:\MATLAB\sltestExampleInputs.xlsx'`

`InputString` — Input
character vector

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)'`

`Name` — Test input name
character vector

Name of the test input, returned as a character vector.

Example: `'sltestExampleInputs.xlsx'`

`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” (Simulink).

Example: `'Successfully mapped inputs.'`

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

You can add data from a Microsoft Excel spreadsheet. The spreadsheet used in this example is located in the example folder.

```
% Load example model
open_system('sltestExcelExample');

% Create new test file
tf = sltest.testmanager.TestFile('C:\MATLAB\input_test_file.mldatx');
% Get test suite object
ts = getTestSuites(tf);
% Get test case object
tc = getTestCases(ts);

% Add the example model as the system under test
setProperty(tc,'Model','sltestExcelExample');

% Add Excel data to Inputs section
% Specify two sheets to add: Acceleration and Braking
input_path = fullfile(matlabroot,'toolbox','simulinktest',...
             'simulinktestdemos','sltestExampleInputs.xlsx');
input = addInput(tc,input_path,'Sheets',["Acceleration","Braking"]);

% Map the input signal for the sheets by block name
```

```
map(input(1),0);
map(input(2),0);
```

See Also

`addInput` | `sltest.testmanager.TestCase`

Topics

“Create and Run Test Cases with Scripts”

Introduced in R2015b

sltest.testmanager.TestIteration class

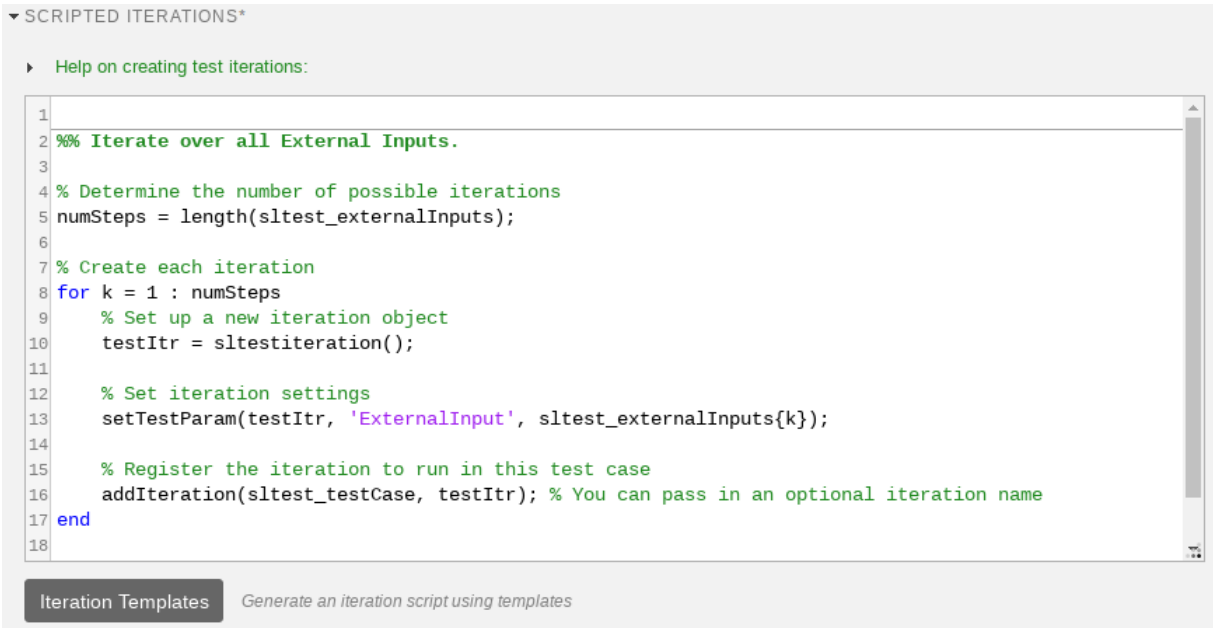
Package: `sltest.testmanager`

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.



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”.

Construction

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

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

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

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

Properties

`Name` — Iteration name

empty (default) | 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'`

`ModelParams` — Model parameter overrides

empty (default) | cell array

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

`TestParams` — Test parameter settings

empty (default) | cell array

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

`Variables` — Model variable overrides
empty (default) | cell array

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

`Enabled` — Run iteration with test case
`false` (default) | `true`

Option to run the iteration with the test case, specified as a logical.

Methods

getIterationResults	Get test iteration results history
setModelParam	Set model parameter for iteration
setTestParam	Set test case 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.

```
% 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.

Construction

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

Properties

`Outcome` — Outcome of test iteration result
0 | 1 | 2 | 3

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

`Duration` — Length of time the test iteration ran, in seconds
duration

This property is read-only.

Length of time the test iteration ran, in seconds, returned as a duration.

`StartTime` — Time the test iteration began to run
datetime

This property is read-only.

Time the test iteration began to run, returned as a datetime.

`StopTime` — Time the test iteration completed
datetime

This property is read-only.

Time the test completed, returned as a datetime.

`TestFilePath` — Test file path
character vector

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

`TestCasePath` — Result hierarchy path
character vector

The hierarchy path in the parent result set.

`TestCaseType` — Type of test case
`'Simulation'` | `'Baseline'` | `'Equivalence'`

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

`RunOnTarget` — Target indicator
cell array

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

`Parent` — Parent of the result object
object

Parent of the result. The parent of a test iteration result is a test case result object.

`ErrorMessages` — Error messages
array of strings

Error messages produced by the iteration, returned as a array of strings.

`LogMessages` — Log messages
array of strings

Log messages produced by the iteration, returned as a array of strings.

Methods

getBaselineRun	Get test iteration baseline dataset
getComparisonResult	Get test data comparison result
getTestIteration	Get test iteration that produced result
getComparisonRun	Get test iteration signal comparison results
getCoverageResults	Get coverage results
getCustomCriteriaPlots	Get plots from custom criteria
getCustomCriteriaResult	Get custom criteria results from test iteration
getInputRuns	Get inputs from simulations captured with the test result
getOutputRuns	Get test iteration simulation output results
getSimulationPlots	Get plots from callbacks
getVerifyRuns	Get test iteration verify statement

Examples

Get Test Iteration Results

```
% 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”
“Create and Run Test Cases with Scripts”

Introduced in R2016a

sltest.testmanager.TestResultReport class

Package: `sltest.testmanager`

Customize generated results report

Description

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

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

Construction

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

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

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

Input Arguments

`resultObjects` — Results set object
object

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

`reportFilePath` — File name and path of the generated report
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)

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

Example: `'Test Engineer'`

`BodyFontColor` — Body paragraph font color
`'Black'` (default) | character vector

Body paragraph text font color, specified as a character vector.

Example: `'Red'`

`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'`

`BodyFontSize` — Body paragraph font size
`'12pt'` (default) | character vector

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

Example: `'14pt'`

`ChapterIndent` — First level indentation width
`'3mm'` (default) | character vector

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

Example: `'5mm'`

`ChapterIndentL2` — Second level indentation width
`'6mm'` (default) | character vector

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

Example: `'8mm'`

`ChapterIndentL3` — Third level indentation width
`'8mm'` (default) | character vector

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

Example: `'10mm'`

`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'`

`HeadingFontColor` — Section heading font color
`'Black'` (default) | character vector

Section heading text font color, specified as a character vector.

Example: `'Blue'`

`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'`

`HeadingFontSize` — Section heading font size
`'14pt'` (default) | character vector

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

Example: `'16pt'`

`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'`

`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'`

`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'`

`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'`

`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'`

`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'`

`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'`

`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'`

`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'`

`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'`

`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'`

`IncludeComparisonSignalPlots` — Include comparison signal plots
`false` (default) | `true`

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

`IncludeCoverageResult` — Include coverage results
`false` (default) | `true`

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

`IncludeErrorMessages` — Include error messages
`true` (default) | `false`

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

`IncludeMWVersion` — Include MATLAB version
`true` (default) | `false`

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

`IncludeSimulationMetaData` — Include simulation metadata
`false` (default) | `true`

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

`IncludeSimulationSignalPlots` — Include simulation signal plots
`false` (default) | `true`

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

`IncludeTestRequirement` — Include test requirement
`true` (default) | `false`

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

`IncludeTestResults` — Include all or subset of test results
`2` (default) | `0` | `1`

Include all or a subset of test results in the report. You can select all 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`.

`LaunchReport` — Open report at completion
`true` (default) | `false`

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

`ReportTitle` — Report title
empty character vector (default)

Title of the report, specified as a character vector

Example: `'Test Case Report'`

`SectionSpacing` — Spacing between sections
`'2mm'` (default) | character vector

Spacing between sections, specified in millimeters as a character vector.

Example: `'5mm'`

`SignalPlotHeight` — Plot height
`'600px'` (default) | character vector

Plot height, specified in pixels as a character vector.

Example: `'500px'`

`SignalPlotWidth` — Plot width
`'500px'` (default) | character vector

Plot width, specified in pixels as a character vector.

Example: `'400px'`

`TableFontColor` — Table font color
`'Black'` (default) | character vector

Table font color, specified as a character vector.

Example: `'Blue'`

`TableFontName` — Table font style name
`'Arial'` (default) | character vector

Table font-style name, specified as a character vector.

Example: `'Times New Roman'`

`TableFontSize` — Table font size
`'7.5pt'` (default) | character vector

Table font size, specified in points as a character vector.

Example: `'10pt'`

`TitleFontColor` — Title font color
`'Black'` (default) | character vector

Title font color, specified as a character vector.

Example: `'Blue'`

`TitleFontName` — Title font style name
`'Arial'` (default) | character vector

Title font-style name, specified as a character vector.

Example: `'Times New Roman'`

`TitleFontSize` — Title font size
`'16pt'` (default) | character vector

Title font size, specified in points as a character vector.

Example: `'20pt'`

Methods

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

    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.ResultSet` | `sltest.testmanager.report`

Topics

“Customize Test Results Reports”

“Export Test Results and Generate Test Results Reports”
“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.

Construction

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

Input Arguments

`parent` — Parent test file or test suite

object

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

`name` — Test suite name

character vector

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

Example: `'Coverage Test Suite'`

Properties

`Description` — Test suite description
character vector

Test suite description text, specified as a character vector.

`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`.

`Name` — Test suite name
character vector

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

`Parent` — Parent test file or test suite
object

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

`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`.

`Releases` — Releases available for testing
string array

This property is read-only.

Releases available for testing, returned as a string array. Add releases using `sltest.testmanager.setpref`.

`Requirements` — Test suite requirements
structure array

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

Tags — Tags to categorize by
character vector | string array

Tags to use for categorizing, specified as a character vector or string array.

`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.

`TestPath` — Test hierarchy
character vector

Test file and test suite hierarchy, returned as a character vector.

Methods

convertTestType	Convert test from one type to another
createTestCase	Create test case
createTestForSubsystem	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.TestCase` | `sltest.testmanager.TestFile`

Topics

“Create and Run Test Cases with Scripts”

Introduced in R2015b

sltest.testmanager.TestSuiteResult class

Package: `sltest.testmanager`

Access test suite results data

Description

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

Construction

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

Properties

`Duration` — Length of time the test suite ran, in seconds
duration

This property is read-only.

Length of time the test suite ran, in seconds, returned as a duration.

`NumDisabled` — Number of disabled tests
integer

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

`NumFailed` — Number of failed tests
integer

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

`NumPassed` — Number of passed tests
integer

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

`NumTestCaseResults` — Number of test case result children
integer

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

`NumTestSuiteResults` — Number of test suite result children
integer

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

`NumTotal` — Total number of tests
integer

The total number of tests in the test suite result.

`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.

`Release` — Release in which the test was run
character vector

This property is read-only.

Release in which the test was run, returned as a character vector.

`StartTime` — Time the test suite began to run
datetime

This property is read-only.

Time the test sutie began to run, returned as a datetime.

`StopTime` — Time the test suite completed
datetime

This property is read-only.

Time the test suite completed, returned as a datetime.

Tags — Tags to filter test file results

string array

This property is read-only.

Tags to filter the test file results. Use tags to view a subset of the test results. See “Tags” for more information.

TestFilePath — Test file path

character vector

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

TestSuitePath — Result hierarchy path

character vector

The hierarchy path in the parent result set.

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.TestCaseResult` | `sltest.testmanager.TestFileResult`

Topics

“Create and Run Test Cases with Scripts”

Introduced in R2015a

Methods — Alphabetical List

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 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 or MAT-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 or a Microsoft Excel file.

Example: `'C:\MATLAB\baseline_API.mat'`, `'C:\MATLAB\baseline.xlsx'`

`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 comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

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

```
% 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.

```
% 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.

```
% 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.

```
% 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 | character vector

`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
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;
```

Select the `sldemo_mdlref_dsm_bot2` model reference and enter `gcb`. Open `sldemo_mdlref_dsm_bot2` and select the data store write block and enter `gcb`. Use the returned paths to create a `Simulink.BlockPath` object for the global data store.

```
% Add signal to set
bPath = Simulink.BlockPath({'sldemo_mdlref_dsm/A1',...
                            'sldemo_mdlref_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.

Select the `sldemo_mdlref_dsm_bot` model reference and enter `gcb`. Open `sldemo_mdlref_dsm_bot`, select the `PositiveSS` subsystem and enter `gcb`. Open the subsystem, select the Data Store Write block and enter `gcb`. Use the returned paths to create a `Simulink.BlockPath` object for the local data store.

```
% Add signal to set
bPath = Simulink.BlockPath({'sldemo_mdlref_dsm/A',...
    'sldemo_mdlref_dsm_bot/PositiveSS',...
    'sldemo_mdlref_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 `X2Out` to the first set.

```
base(1).addExcelSpecification('Sheet','X2Out');
```

Show the contents of the `Sheet` property of the Excel specifications for each baseline criteria. The first set now includes the `X2Out` sheet.

```
base(1).ExcelSpecifications(:).Sheet

ans =

    'Optics'


ans =

    'X2Out'

base(2).ExcelSpecifications(:).Sheet

ans =

    'Converter'
```

Input Arguments

`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'` Rectangular Range	Specify the range using the syntax `'Corner1:Corner2'`, where *Corner1* and *Corner2* 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). **Example:** `'Range','Corner1:Corner2'`
`''` Unspecified or Empty	If unspecified, the importing function automatically detects the used range. **Example:** `'Range',''` **Note:** *Used Range* 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'` Row Range	You can identify the range by specifying the beginning and ending rows using Excel row designators. Then `readtable` 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). **Example:** `'Range','1:7'`

Ways to specify Range	Description
'*Column1:Column2*' Column Range	You can identify the range by specifying the beginning and ending columns using Excel column designators. Then `readtable` 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). **Example:** `'Range','A:F'`
'*NamedRange*' Excel's 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. **Example:** `'Range','myTable'`

Example: `'A1:C20'`

See Also

`sltest.testmanager.BaselineCriteria` | `sltest.testmanager.TestInput`

Topics

"Baseline Criteria"
"Inputs"
"Run Tests Using External Data"

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 comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

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*' Rectangular Range	Specify the range using the syntax '*Corner1:Corner2*', where *Corner1* and *Corner2* 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). **Example:** 'Range','Corner1:Corner2'
'' Unspecified or Empty	If unspecified, the importing function automatically detects the used range. **Example:** 'Range','' **Note:** *Used Range* 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*' Row Range	You can identify the range by specifying the beginning and ending rows using Excel row designators. Then readtable 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). **Example:** 'Range','1:7'

Ways to specify Range	Description
`'Column1:Column2'` Column Range	You can identify the range by specifying the beginning and ending columns using Excel column designators. Then `readtable` 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). **Example:** `'Range','A:F'`
`'NamedRange'` Excel's 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. **Example:** `'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

You can add data from a Microsoft Excel spreadsheet. The spreadsheet used in this example is located in the example folder. Add only the two sheets that have data as input.

```
% Load example model
open_system('sltestExcelExample');

% Create new test file
tf = sltest.testmanager.TestFile('C:\MATLAB\input_test_file.mldatx');
% Get test suite object
ts = getTestSuites(tf);
% Get test case object
tc = getTestCases(ts);

% Add the example model as the system under test
setProperty(tc,'Model','sltestExcelExample');

% Add Excel data to Inputs section
% Specify two sheets to add: Acceleration and Braking
input_path = fullfile(matlabroot,'toolbox','simulinktest',...
             'simulinktestdemos','sltestExampleInputs.xlsx');
input = addInput(tc,input_path,'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”
“Format Test Case Data in Excel”
“Inputs”

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

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. The test case runs on the `sldemo_autotrans` model and uses `getTestCaseResults` to obtain the results.

Clear existing test files from the Test Manager.

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

Create new test file, test suite, and test case.

```
tf = sltest.testmanager.TestFile('TestFile1');
ts = createTestSuite(tf,'TestSuite1');
tc = createTestCase(ts,'simulation','TestCase1');
```

Remove default test suite so that only the created test suite is used.

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

Assign the system under test to the test case. Run the test.

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

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.

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.

```
% Open model
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 comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

`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.

```
% Open model
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 set

Syntax

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

Description

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

Input Arguments

`ps` — Parameter set
`sltest.testmanager.ParameterSet` object

Parameter set that you want to add the override to, specified as a `sltest.testmanager.ParameterSet` object.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'Value',5.2`

`Name` — Variable name
character vector

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

`Value` — Variable value
numeric

Value of the variable you want to override.

`MaskedBlockPath` — Masked block path
character vector

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

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

```
% 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

Topics

“Create and Run Test Cases with Scripts”

Introduced in R2015b

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 comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'SimulationIndex',2`

`Name` — Parameter set name
auto-generated unique name (default) | 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 that the parameter set applies to, specified as an integer, 1 or 2. This parameter applies to the simulation test case where there are two simulations. For baseline and simulation test cases, the simulation index is 1.

Output Arguments

`pset` — Parameter set
`sltest.testmanager.ParameterSet` object

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

Examples

Add Parameter Set to Test Case

```
% 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.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'Sheet','mysheet','Range','C1:F10','CreateIterations',false`

Pairs for MAT-Files andMicrosoft 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”
“Run Tests Using External Data”
“Test Case Input Data Files”

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 and Generate Test Results Reports”

“Create and Run Test Cases with Scripts”

Introduced in R2016a

addReportTOC

Class: `sltest.testmanager.TestResultReport`
Package: `sltest.testmanager`

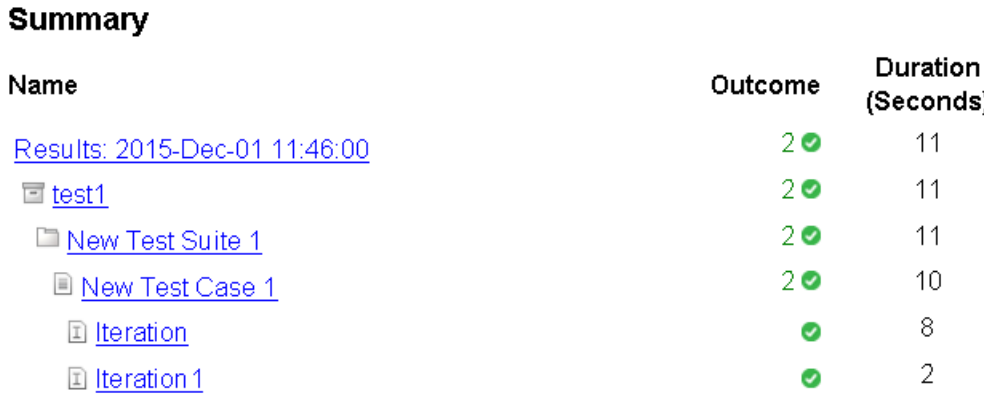
Add report table of contents

Syntax

```
addReportTOC(obj)
```

Description

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



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 and Generate Test Results Reports”
“Create and Run Test Cases with Scripts”

Introduced in R2016a

addTitlePage

Class: `sltest.testmanager.TestResultReport`
Package: `sltest.testmanager`

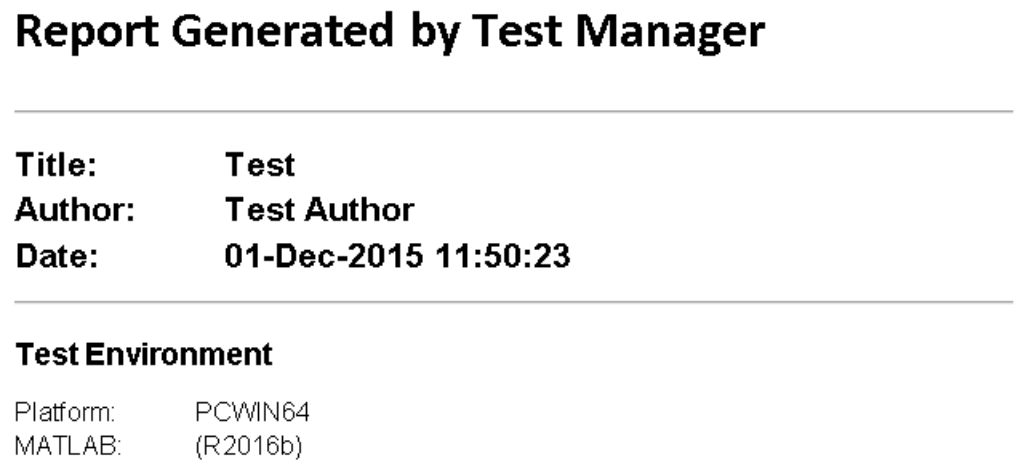
Add report title page

Syntax

```
addTitlePage(obj)
```

Description

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



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 and Generate Test Results Reports”
“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 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` — Input file name and path

character vector

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

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 comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'Sheet','mysheet','Range','C1:F10'`

Pairs for MAT-Files 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

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

Output Arguments

eq — Equivalence criteria object

object

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

Examples

Add Equivalence Criteria to Test Case

```
% 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

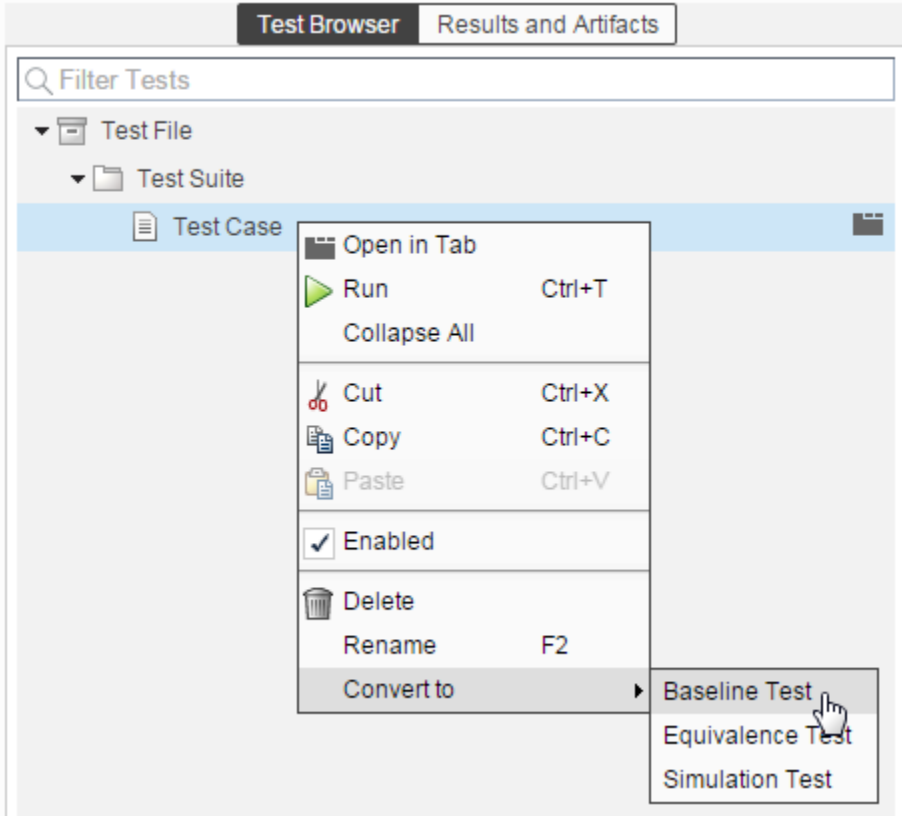
```
% Create new test file with test suite and default test case
tf = sltest.testmanager.TestFile('API Test File');
ts = getTestSuites(tf);
tc = getTestCases(ts);

% 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

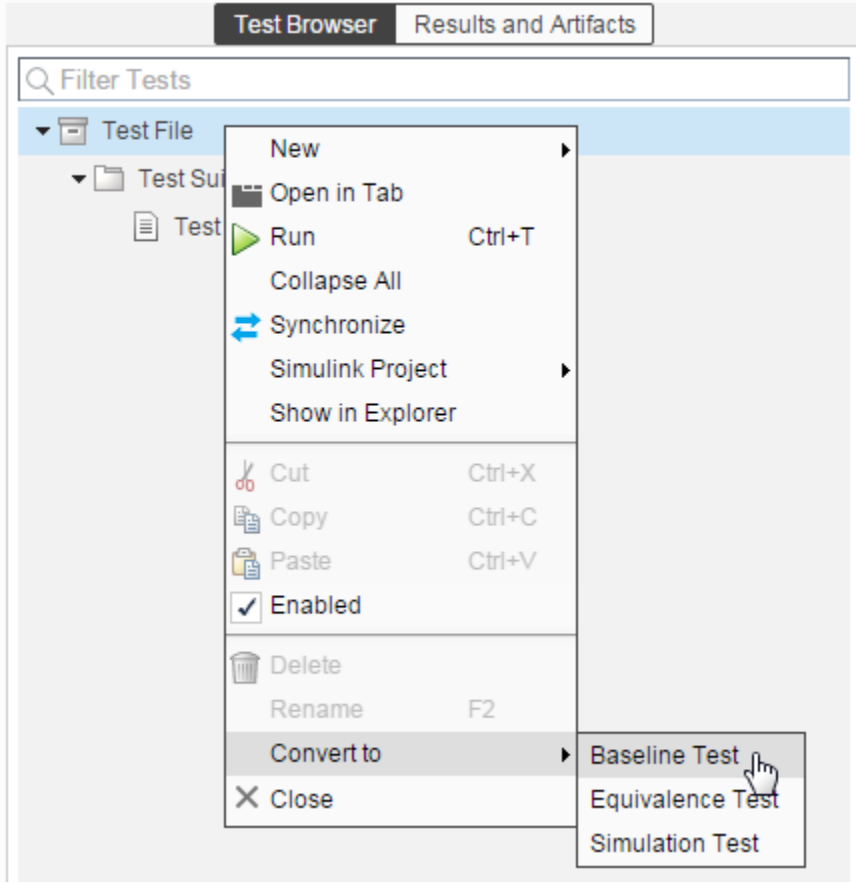
```
% Create new test file with test suite and default test case
tf = sltest.testmanager.TestFile('API Test File');
ts = getTestSuites(tf);
tc1 = getTestCases(ts);

% 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

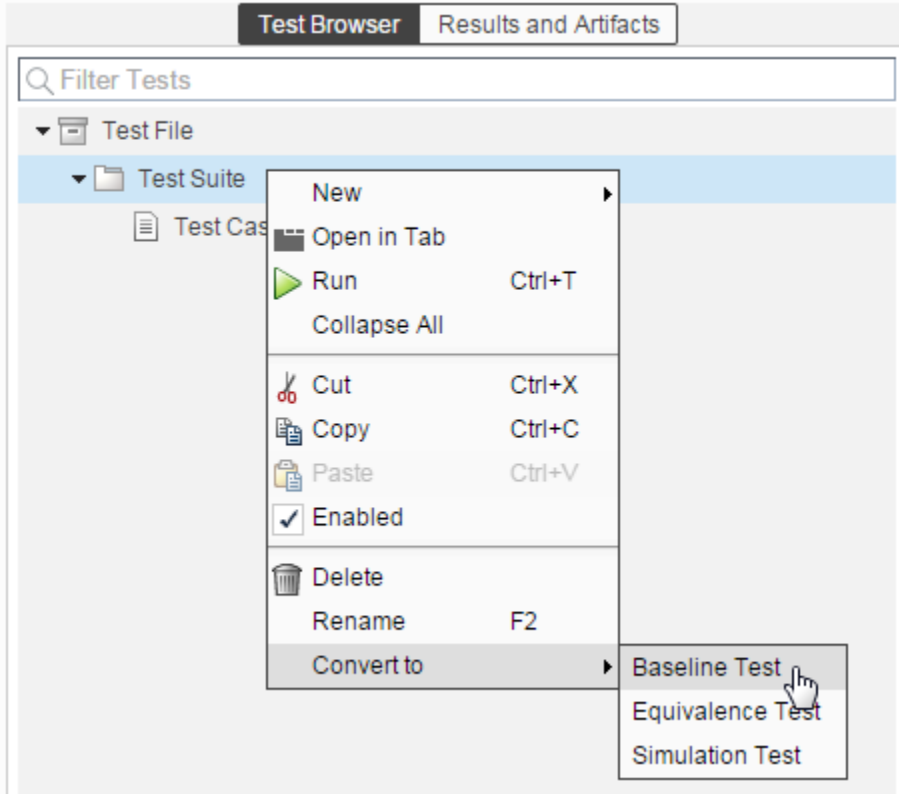
```
% Create new test file with test suite and default test case
tf = sltest.testmanager.TestFile('API Test File');
ts = getTestSuites(tf);
tc1 = getTestCases(ts);

% 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

```
% 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 a character vector.

`name` — Test case name

character vector

Test case name, specified as a 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('C:\MATLAB\test_file.mldatx');

% 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`

Create test harness and test case for subsystem in test file

Syntax

```
harn_tc = createTestForSubsystem(tf,'Subsystem',subsystem)
l = createTestForSubsystem(tf,'Subsystem',subsystem,Name,Value)
```

Description

`harn_tc = createTestForSubsystem(tf,'Subsystem',subsystem)` creates a harness on the specified subsystem, model reference 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 test 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 for a Component”.

`l = 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 comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

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'`

`UseSubsytemInputs` — 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: `'UseSubstyemInputs',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” (Simulink).

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 `SLDVTestGeneration` 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

`SLDVTestGeneration` — 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: `'SLDVTestGeneration',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

`harn_tc` — Harness and test case creation result

logical

Returns 0 if the harness and test case are not created, or 1 if the harness and test case are created.

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);
```

Create Back-to-Back Test

Create an equivalence 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
tf = sltest.testmanager.TestFile('My B2B Test File');

% Create test from subsystem
createTestForSubsystem(tf,'Subsystem',...
    'rtwdemo_sil_block/Controller','TestType',...
    'equivalence','Simulation1Mode','Normal',...
    'Simulation2Mode','Software-in-the-Loop (SIL)',...
    'CreateExcelFile',true);
```

See Also

`sltest.testmanager.TestFile`

Topics

“Create and Run Test Cases with Scripts”
“Generate Tests for a Component”
“Create and Run a Back-to-Back Test”

Introduced in R2016a

createTestForSubsystem

Class: `sltest.testmanager.TestSuite`
Package: `sltest.testmanager`

Create test harness and test case for subsystem in test suite

Syntax

```
harn_tc = createTestForSubsystem(ts,'Subsystem',subsystem)
harn_tc = createTestForSubsystem(ts,'Subsystem',subsystem,
Name,Value)
```

Description

`harn_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 for a Component”.

`harn_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, 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 comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

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'`

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.

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 `SLDVTestGeneration` 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

`SLDVTestGeneration` — 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: `'SLDVTestGeneration',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

`harn_tc` — Harness and test case creation result

logical

Returns 0 if the harness and test case are not created, or 1 if the harness and test case are created.

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);
```

Create Back-to-Back Test for a Subsystem

Create an equivalence 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','TestType',...
    'equivalence','Simulation1Mode','Normal',...
```

```
    'Simulation2Mode','Software-in-the-Loop (SIL)',...
    'CreateExcelFile',true);
```

See Also

`sltest.testmanager.TestSuite`

Topics

“Create and Run Test Cases with Scripts”
“Generate Tests for a Component”
“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.

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

character vector

Name of the test suite, specified as a character vector.

Example: `'My Test Suite'`

Output Arguments

`ts` — Test suite object

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

character vector

Test suite name, specified as a character vector. If this input argument is empty, then the Test Manager gives the test suite a unique name.

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('C:\MATLAB\test_file.mldatx');

% 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

```
% 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	Link
yout.Vs	double		Continuous	linear	union	Link
yout.Sd	double		Continuous	linear	union	Link
slp	double		Continuous	linear	union	Link

Input Arguments

`obj` — Test report object

`sltest.testmanager.TestResultReport` object

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

`result` — Result set
`sltest.testmanager.TestCaseResult` object | `sltest.testmananger.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

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

Topics

“Export Test Results and Generate Test Results Reports”
“Create and Run Test Cases with Scripts”

Introduced in R2016a

genCoverageTable

Class: `sltest.testmanager.TestResultReport`
Package: `sltest.testmanager`

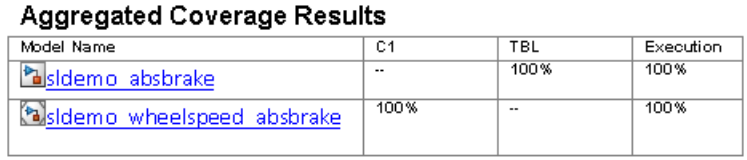
Generate coverage collection table

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
sldemo_absbrake	--	100%	100%
sldemo_wheelspeed_absbrake	100%	--	100%

Input Arguments

`obj` — Test report object

`sltest.testmanager.TestResultReport` object

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

`resultObj` — Result set

`sltest.testmanager.TestResult` object

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

Output Arguments

`groupObj` — Group object
`mlreportgen.dom.Group` object

The coverage section generated by the method, returned as an `mlreportgen.dom.Group` object.

See Also

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

Topics

“Export Test Results and Generate Test Results Reports”
“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

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

Topics

“Export Test Results and Generate Test Results Reports”
“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

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

Topics

“Export Test Results and Generate Test Results Reports”
“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

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

Topics

“Export Test Results and Generate Test Results Reports”
“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

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

Topics

“Export Test Results and Generate Test Results Reports”
“Create and Run Test Cases with Scripts”

Introduced in R2016a

genRequirementLinksTable

Class: `sltest.testmanager.TestResultReport`
Package: `sltest.testmanager`

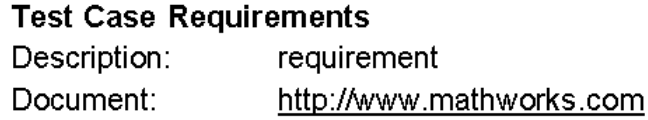
Generate requirement links table

Syntax

```
reqTable = genRequirementLinksTable(obj,resultObj,isTestSuiteResult)
```

Description

`reqTable = genRequirementLinksTable(obj,resultObj,isTestSuiteResult)` generates a section for table of requirement links.



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

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

Topics

“Export Test Results and Generate Test Results Reports”
“Create and Run Test Cases with Scripts”

Introduced in R2016a

genResultSetBlock

Class: `sltest.testmanager.TestResultReport`
Package: `sltest.testmanager`

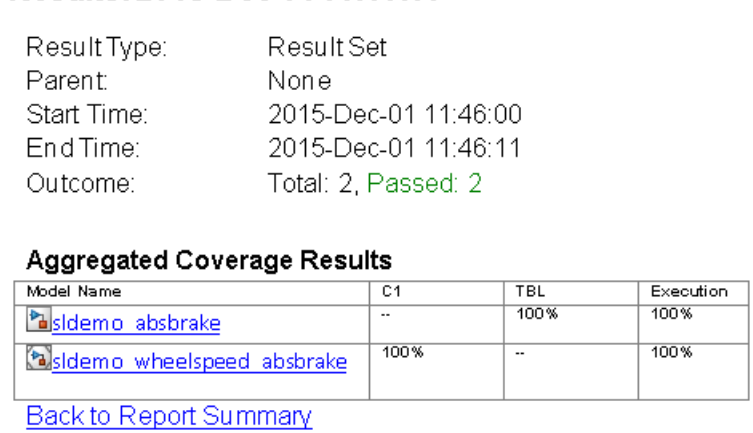
Generate results set section

Syntax

```
groupObj = genResultSetBlock(obj,result)
```

Description

`groupObj = genResultSetBlock(obj,result)` generates the results set section.



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

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

Topics

“Export Test Results and Generate Test Results Reports”
“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

`Simulink.sdi.Run` | `mlreportgen.dom.Group` | `sltest.testmanager.TestResultReport`

Topics

“Export Test Results and Generate Test Results Reports”
“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.Ww	double		Continuous	linear	union	Link
yout.Vs	double		Continuous	linear	union	Link
yout.Sd	double		Continuous	linear	union	Link
slp	double		Continuous	linear	union	Link

Input Arguments

`obj` — Test report object

`sltest.testmanager.TestResultReport` object

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

`signalList` — Signal list

`sltest.testmanager.ReportUtility.Signal` object

Signals in the summary table, specified as an array of `sltest.testmanager.ReportUtility.Signal` objects.

`isComparison` — Comparison indicator
`true` | `false`

Flag to indicate whether the signal is from a comparison run or not, specified as a Boolean, `true` or `false`.

`isSummaryTable` — Summary table indicator
`true` | `false`

Flag to indicate whether this is for a signal summary table in the report or an individual signal plot, specified as a Boolean. `true` for the signal summary table or `false` for an individual signal plot.

Output Arguments

`groupObj` — Group object
`mlreportgen.dom.Group` object

The signal summary section generated by the method, returned as an `mlreportgen.dom.Group` object.

See Also

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

Topics

“Export Test Results and Generate Test Results Reports”
“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 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

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

Topics

“Export Test Results and Generate Test Results Reports”
“Create and Run Test Cases with Scripts”

Introduced in R2016a

genTableRowsForResultMetaInfo

Class: `sltest.testmanager.TestResultReport`
Package: `sltest.testmanager`

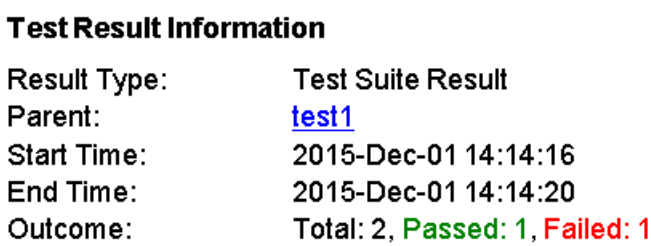
Generate test result metadata table

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.



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

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

Topics

“Export Test Results and Generate Test Results Reports”
“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

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

Topics

“Export Test Results and Generate Test Results Reports”
“Create and Run Test Cases with Scripts”

Introduced in R2016a

genTestSuiteResultBlock

Class: `sltest.testmanager.TestResultReport`
Package: `sltest.testmanager`

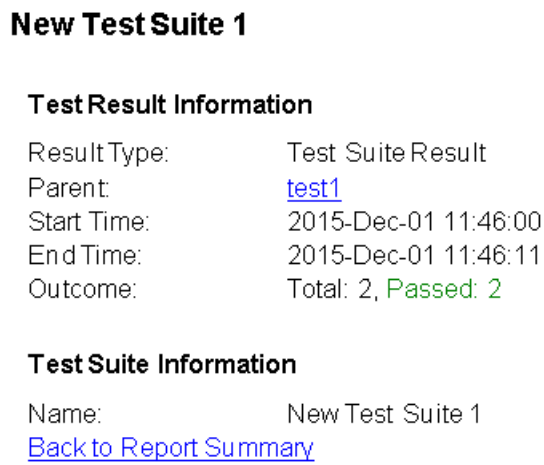
Generate test suite result section

Syntax

```
groupObj = genTestSuiteResultBlock(obj,result)
```

Description

`groupObj = genTestSuiteResultBlock(obj,result)` generates a test suite result section.



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

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

Topics

“Export Test Results and Generate Test Results Reports”
“Create and Run Test Cases with Scripts”

Introduced in R2016a

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

```
% 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

```
% 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 `sldemo_autotrans` 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 = getTestSuiteByName(tf,'New Test Suite 1');
remove(tsDel);
```

Assign the system under test to the test case. Run the test.

```
setProperty(tc,'Model','sldemo_autotrans');
tcresult = run(tc);
```

Obtain the test case results.

```
tcresultobj = getTestCaseResults(tc);
```

Obtain the test case that produced the results.

```
tcobj = getTestCase(tcresultobj);
```

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

```
% 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

`setProperty` | `sltest.testmanager.Options` | `sltest.testmanager.TestFile` | `sltest.testmanager.TestFileResult`

Topics

“Create, Store, and Open MATLAB Figures”
“Export Test Results and Generate Test Results Reports”

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

```
% 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

`setProperty (Test Suite)` | `sltest.testmanager.Options` | `sltest.testmanager.TestSuite` | `sltest.testmanager.TestSuiteResult`

Topics

“Create, Store, and Open MATLAB Figures”
“Export Test Results and Generate Test Results Reports”

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 = fullfile(matlabroot,'examples','simulinktest',...
    '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))
```

4. Get the comparison signal result of the run result.

```
cr2sig = getComparisonSignalResults(cr2)
```

5. Clear the results and the Test Manager.

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

See Also

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 a `sltest.testmanager.TestIterationResult` 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 = fullfile(matlabroot,'examples','simulinktest',...
    '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))
```

4. Get the comparison signal result of the run result.

```
cr2sig = getComparisonSignalResults(cr2)
```

5. Clear the results and the Test Manager.

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

See Also

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

```
% 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 equivalience 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 = fullfile(matlabroot,'examples','simulinktest',...
    '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))
```

4. Get the comparison signal result of the run result.

```
cr2sig = getComparisonSignalResults(cr2)
```

5. Clear the results and the Test Manager.

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

See Also

Introduced in R2017b

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 cv.cvdatagroup.

Examples

Get Result Set Coverage Results

```
% 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`

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

Run a Test and Remove the Result

```
% Create a test file
tf = sltest.testmanager.TestFile('ControllerTests.mldatx');

% Run the test
result = run(tf);
```

```
% Remove the result
remove(result)
```

See Also

sltest.testmanager.ResultSet | sltest.testmanager.TestCaseResult | sltest.testmanager.TestSuiteResult

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 cv.cvdatagroup.

Examples

Get Test Suite Coverage Results

```
% 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
tfr = getTestFileResults(ro);
tsr = getTestSuiteResults(tfr);
```

```
tcs = getTestCaseResults(tsr);
cr = getCoverageResults(tcs);
```

See Also

`sltest.testmanager.CoverageSettings`

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 cv.cvdatagroup.

Examples

Get Test File Coverage Results

```
% 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
```

```
tfr = getTestFileResults(ro);
cr = getCoverageResults(tfr);
```

See Also

`sltest.testmanager.CoverageSettings`

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: `'sldemo_absbrake'`

Output Arguments

`covResult` — Coverage results
object array

Coverage results contained in the test iteration result, returned as an array of `cvdata` objects. For more information on `cvdata` objects, see cv.cvdatagroup.

See Also

`sltest.testmanager.CoverageSettings`

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 cv.cvdatagroup.

Examples

Get Test Suite Coverage Results

```
% 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
tfr = getTestFileResults(ro);
```

```
tsr = getTestSuiteResults(tfr);
cr = getCoverageResults(tsr);
```

See Also

`sltest.testmanager.CoverageSettings`

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.

Examples

Get Coverage Settings

```
% 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`

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.

Examples

Turn On Coverage For a Test File

```
% 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`

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.

Examples

Turn Off Coverage For a Test Suite

```
% 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`

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

```
% 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.CustomCriteria` | `sltest.testmanager.Options` | `sltest.testmanager.TestCaseResult`

Topics

“Create, Store, and Open MATLAB Figures”
“Export Test Results and Generate Test Results Reports”

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

```
% 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

`getIterationResults` | `sltest.testmanager.CustomCriteria` | `sltest.testmanager.Options` | `sltest.testmanager.TestIterationResult`

Topics

“Create, Store, and Open MATLAB Figures”
“Export Test Results and Generate Test Results Reports”

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

```
% 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

```
% Load example model
open_system('sltestExcelExample');

% Create new test file
tf = sltest.testmanager.TestFile('C:\MATLAB\input_test_file.mldatx');
% Get test suite object
ts = getTestSuites(tf);
% Get test case object
tc = getTestCases(ts);

% Add the example model as the system under test
setProperty(tc,'Model','sltestExcelExample');


% Add Excel data to Inputs section
% Specify two sheets to add: Acceleration and Braking
input_path = fullfile(matlabroot,'toolbox','simulinktest',...
             'simulinktestdemos','sltestExampleInputs.xlsx');
input = addInput(tc,input_path,'Sheets',["Acceleration","Braking"]);


% Map the input signal for the sheets by block name
% The third sheet is empty and cannot be mapped
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

```
% 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

```
% 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.

```
% Open model
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.

```
% Open model
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

`getOptions (TestFile)` | `getOptions (TestSuite)` | `sltest.testmanager.Options`

Topics

“Create, Store, and Open MATLAB Figures”
“Export Test Results and Generate Test Results Reports”

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 and Generate Test Results Reports”

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 and Generate Test Results Reports”

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

```
% Open the model
open_system('sltestExcelExample');

% Create test file and get test suite and test case objects
tf = sltest.testmanager.TestFile('Input Run Test File');
ts = getTestSuites(tf);
tc = getTestCases(ts);

% Add the model as the system under test, specify to save input runs
setProperty(tc,'Model','sltestExcelExample',...
    'SaveInputRunInTestResult',true);

% Add Excel data to Inputs section
% Specify two sheets to add: Acceleration and Braking
input_path = fullfile(matlabroot,'toolbox','simulinktest',...
             'simulinktestdemos','sltestExampleInputs.xlsx');
input = addInput(tc,input_path,'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 return the results data
resultsObj = run(tc);

% Get test case result and iteration result
tcr = getTestCaseResults(resultsObj);
tir = tcr.getIterationResults;

% Get inputs from 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 Simulation Output From Test Case

```
% 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 baseline run dataset
runArray = getOutputRuns(tcr);
```

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

```
% 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

```
% 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 the`setProperty` 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
- `'UseSignalBuilderGroups'` — use signal builder groups for test input
- `'SignalBuilderGroup'` — signal builder group name to use
- `'OverrideModelOutputSettings'` — override model output settings
- `'SaveOutput'` — save simulation output
- `'SaveState'` — save model states during simulation
- `'SaveFinalState'` — save final state of simulation
- `'SignalLogging'` — log signals
- `'DSMLogging'` — log data store
- `'ConfigsetOverrideSetting'` — value to determine override of configuration set
- `'ConfigsetName'` — configuration set override name
- `'ConfigsetFileLocation'` — path to a MAT-file that contains a configuration set object
- `'ConfigsetVarName'` — name of the variable in `ConfigsetFileLocation` that is a configuration set

- `'IterationScript'` — 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

```
% 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

```
% 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

`setProperty` | `sltest.testmanager.Options` | `sltest.testmanager.TestFile` | `sltest.testmanager.TestFileResult`

Topics

“Create, Store, and Open MATLAB Figures”
“Export Test Results and Generate Test Results Reports”

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

```
% 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

`setProperty (Test Suite)` | `sltest.testmanager.Options` | `sltest.testmanager.TestSuite` | `sltest.testmanager.TestSuiteResult`

Topics

“Create, Store, and Open MATLAB Figures”
“Export Test Results and Generate Test Results Reports”

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`.

```
% 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

```
% 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

```
% 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

`setProperty` | `sltest.testmanager.Options` | `sltest.testmanager.TestCaseResult`

Topics

“Create, Store, and Open MATLAB Figures”
“Export Test Results and Generate Test Results Reports”

Introduced in R2017a

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

```
% 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

`getIterationResults` | `setProperty (TestCase)` | `sltest.testmanager.Options` | `sltest.testmanager.TestIterationResult`

Topics

“Create, Store, and Open MATLAB Figures”
“Export Test Results and Generate Test Results Reports”

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

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(ts1);
```

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 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

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 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 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

```
% File paths and model names
filePath = fullfile(matlabroot,'toolbox','simulinktest','simulinktestdemos');
topModel = 'TestAndVerificationAutopilotExample';
reqDoc = 'RollAutopilotRequirements.txt';
rollModel = 'RollAutopilotMdlRef';
testHarness = 'RollReference_Requirement1_3';
testFile = 'AutopilotTestFile.mldatx';
harnessLink = ['http://localhost:31415/matlab/feval/rmiobjnavigate?arguments='...
    '[%22RollAutopilotMdlRef:urn:uuid:523e5d2d-bb86-43b2-a187-43c52a2bc174.'...
    'slx%22,%22GIDa_3fe26a28_ee1e_4aff_b1cd_3303ca12539c%22]'];

% Open the main model
open_system(fullfile(filePath,rollModel));

% Open the test file in the test manager
open(fullfile(filePath,testFile));

% Open the test harness
web(harnessLink)

% Open harness and highlight requirements links
sltest.harness.open([rollModel '/Roll Reference'],testHarness)
rmi('highlightModel','RollReference_Requirement1_3')

% Open test sequence and test assessment blocks
open_system('RollReference_Requirement1_3/Test Sequence')
open_system('RollReference_Requirement1_3/Test Assessment')

% Run the test file
ro = sltest.testmanager.run;

% Gett the test results
tfr = getTestFileResults(ro);
tsr = getTestSuiteResults(tfr);
tcr = getTestCaseResults(tsr);
```

```
% Get the verify output
verifyOut = getVerifyRuns(tcr);
```

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`

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 and Generate Test Results Reports”
“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 comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

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” (Simulink).

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 maps data from a Microsoft Excel spreadsheet to a test case.

```
% Load example model
open_system('sltestExcelExample');

% Create new test file
tf = sltest.testmanager.TestFile('C:\MATLAB\input_test_file.mldatx');
% Get test suite object
ts = getTestSuites(tf);
```

```
% Get test case object
tc = getTestCases(ts);

% Add the example model as the system under test
setProperty(tc,'Model','sltestExcelExample');

% Add Excel data to Inputs section
% Specify two sheets to add: Acceleration and Braking
input_path = fullfile(matlabroot,'toolbox','simulinktest',...
             'simulinktestdemos','sltestExampleInputs.xlsx');
input = addInput(tc,input_path,'Sheets',["Acceleration","Braking"]);

% Map the input signal for the sheets by block name
map(input(1),'Mode',0);
map(input(2),'Mode',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`
- `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 you do not select the **Record logged workspace data in Simulation Data Inspector** option, logged states data does not appear in the Simulation Data Inspector and is not plotted. 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

When you create a new model in R2019a or later, the default model configuration saves simulation outputs in a single 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 uses a model configured to save a single simulation output and shows how to access each type of logged data and use the `plot` function to plot the data in the Simulation Data Inspector and then open the Simulation Data Inspector so you can view the 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. 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 plot and 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 four signals, data for the two Outport blocks, and states. Signal and output data always log to the Simulation Data Inspector, and states data only appears in the Simulation Data Inspector when you select the **Record logged workspace data in Simulation Data Inspector** option, which is not selected for this model. When you use the `plot` function to plot the data, the Simulation Data Inspector layout updates to a 1-by-6 layout and plots one signal on each subplot.

```
plot(out)
```

Plot Logged States Data

When you do not log states data to the Simulation Data Inspector, you can use the `plot` function to import and plot the states data. The model logs data for two states, `x1` and `x2`. The states data is saved in a `Simulink.SimulationData.Dataset` object, `xout`, with one element corresponding to each `Simulink.SimulationData.State` object. You can use the `plot` function to plot the data for both signals by plotting `xout`, or you can plot data for a single state.

Access the `Dataset` object, `xout`, using the `get` function for the `SimulationOutput` object. You can also access the logged output and signal data using the `get` function.

```
xout = get(out,'xout');
```

When you plot the data for both states in the `Dataset` object, the Simulation Data Inspector layout changes to 1-by-2 and plots the data for each state on one subplot.

```
plot(xout)
```

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, `logsout`.

```
logsout = get(out,'logsout');
```

Then, use the `get` function for the `Dataset` object to access the data for the first element.

```
sig1 = get(logsout,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`

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`
- `sltest.Assessment`

Example: `plot(out)`

Output Arguments

`runObj` — `Simulink.sdi.Run` object corresponding to plotted data
`Simulink.sdi.Run` object

`Simulink.sdi.Run` object corresponding to the plotted data.

See Also

`plot`

Topics

“View Data with the Simulation Data Inspector” (Simulink)

Introduced in R2019b

plotOneSignalToFile

Class: `sltest.testmanager.TestResultReport`
Package: `sltest.testmanager`

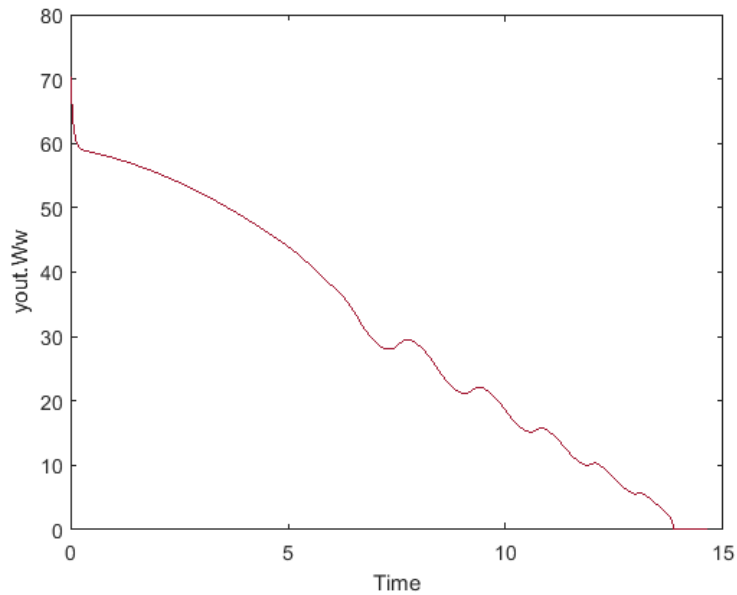
Save signal plot to file

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 and Generate Test Results Reports”
“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

```
% 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

```
% 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.

```
% Open model
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.

```
% Open model
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

```
% 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

```
% 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

```
% 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

```
% Load example model
open_system('sltestExcelExample');

% Create new test file
tf = sltest.testmanager.TestFile('C:\MATLAB\input_test_file.mldatx');
% Get test suite object
```

```
ts = getTestSuites(tf);
% Get test case object
tc = getTestCases(ts);

% Add the example model as the system under test
setProperty(tc,'Model','sltestExcelExample');

% Add Excel data to Inputs section
% Excel file has three sheets, creating three inputs
input_path = fullfile(matlabroot,'toolbox','simulinktest',...
             'simulinktestdemos','sltestExampleInputs.xlsx');
input = addInput(tc,input_path);

% Map the input signals by block name for the first two inputs
map(input(1),0);
map(input(2),0);

% Remove the third input, a blank sheet
remove(input(3));
```

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)
```

Description

`resultObj = run(tc)` runs the test case and returns a results set object.

Input Arguments

`tc` — Test case

`sltest.testmanager.TestCase` object

Test case you want to run, specified as an `sltest.testmanager.TestCase` object.

Output Arguments

`resultObj` — Results set object

object

Test results, returned as a results set object, `sltest.testmanager.ResultSet`.

Examples

Run a Test Case

```
% 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');

% Run the test case and return an object with results data
ro = run(tc);
```

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)
```

Description

`resultObj = run(tf)` runs the enabled test cases in the test file and returns a results set object.

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.

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

```
% Create the test file, test suite, and test case structure
tf = sltest.testmanager.TestFile('My Test File');
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');

% Run the test file and return an object with results data
ro = run(tf);
```

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)
```

Description

`resultObj = run(ts)` runs the enabled test cases in the test suite and returns a results set object.

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.

Output Arguments

`resultObj` — Result set

`sltest.testmanager.ResultSet` object

Test results, returned as a `sltest.testmanager.ResultSet` results set object.

Examples

Run a Test Suite

```
% Create the test file, test suite, and test case structure
tf = sltest.testmanager.TestFile('My Test File');
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');

% Run the test suite and return an object with results data
ro = run(ts);
```

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

```
% 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

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 or handle of a model or block

character vector | handle

Handle or name of a model or block, specified as a numeric handle or a character vector. A numeric handle must be a scalar. You can also set parameters of lines and ports, but you must use numeric handles to specify them.

Example: `'vdp/Fcn'`

`paramName` — Model or block parameter name
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
`set_param` | `sltest.testmanager.TestIteration`

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 comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'StopTime',100`

`Model` — System Under Test model name

empty 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` (default) | `true`

Override SIL/PIL simulation mode of model blocks to Normal simulation mode, specified as a logical. If this property is `true`, the associated check box in the Simulation Settings Overrides section of the Test Manager is selected.

`OverrideStartTime` — Override model start time
`false` (default) | `true`

Indicate if the test case overrides the model start time, specified as `true` or `false`.

`StartTime` — Model start time
`0` (default) | scalar

Model start time, specified as a scalar value.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`OverrideStopTime` — Override model stop time
`false` (default) | `true`

Indicate if the test case overrides the model stop time, specified as `true` or `false`.

`StopTime` — Model stop time
`10` (default) | scalar

Model stop time, specified as a scalar value.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`OverrideInitialState` — Override model initial state
`false` (default) | `true`

Indicate if the test case overrides the model initial state, specified as `true` or `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.

`UseSignalBuilderGroup` — Override Signal Builder group
`false` (default) | `true`

Indicate if the test case overrides the model and uses a different Signal Builder group in the Inputs section, specified as `true` or `false`.

`SignalBuilderGroup` — Signal Builder group name
empty character vector (default)

Signal Builder group name, specified as a character vector. To return to the default model settings, specify an empty character vector, `''`.

`OverrideModelOutputSettings` — Override model output settings
`false` (default) | `true`

Indicate if the test case overrides the model settings under the Outputs section, specified as `true` or `false`.

`SaveOutput` — Override saving output
`false` (default) | `true`

Indicate if the test case overrides saving model output, specified as `true` or `false`.

`SaveState` — Save output state values
`false` (default) | `true`

Indicate if the test case is set to save output state values, specified as `true` or `false`.

`SignalLogging` — Log signals
`true` (default) | `false`

Indicate if the test case is set to log signals marked for logging in the model, specified as `true` or `false`.

`DSMLogging` — Log Data Store variables
`true` (default) | `false`

Indicate if the test case is set to log Data Store variables, specified as `true` or `false`.

`SaveFinalState` — Save final state
`false` (default) | `true`

Indicate if the test case is set to store final state values, specified as `true` or `false`.

`SimulationIndex` — Equivalence test case simulation
1 (default) | 2

Simulation number that the property applies to, specified as an integer, 1 or 2. This setting applies to the simulation test case.

`ConfigSetOverrideSetting` — Configuration setting override
1 (default) | 2 | 3

Override the configuration settings, specified as an integer.

- 1 — No override
- 2 — Use a named configuration set in the model
- 3 — Use a configuration set specified in a file

`ConfigSetName` — Configuration set name
empty 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` (default) | `true`

Indicate if the test iterations run using fast restart mode, specified as `true` or `false`.

`SaveBaselineRunInTestResult` — Save baseline in test result
`false` (default) | `true`

Indicate if the test case saves the baseline used in the test result after test execution, specified as `true` or `false`.

`SaveInputRunInTestResult` — Save input in test result
`false` (default) | `true`

Enable saving external input run used in test result, specified as `true` or `false`.

`StopSimAtLastTimePoint` — Stop simulation at last input time
`false` (default) | `true`

Enable stopping the simulation at the final time value of the input, specified as `true` or `false`.

`LoadAppFrom` — Application location
1 (default) | 2 | 3

Location from which to load the application, specified as an integer. This property is available only in real-time test cases.

- 1 — Model
- 2 — Target application
- 3 — Target computer

For more information on real-time testing, see “Test Models in Real Time”.

`TargetComputer` — Target computer name
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”

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 comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'SetupCallback','a = 300; % set nominal value'`

`SetupCallback` — Setup callback script
empty (default) | 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 comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'SetupCallback','a = 300; % set nominal value'`

`SetupCallback` — Setup callback script

empty (default) | 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 parameter

Syntax

```
setTestParam(obj,Name,Value)
```

Description

`setTestParam(obj,Name,Value)` sets the test iteration parameter with additional options specified by one or more `Name,Value` pair arguments.

Input Arguments

`obj` — Test iteration object
object

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

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

`ParameterSet` — Parameter set
Current test case setting (default) | character vector

Parameter set name, specified as the comma-separated pair consisting of `'ParameterSet'` and a character vector. Parameter set overrides are setup in the **Parameter Overrides** section of the test case.

Example: `'ParameterSet','Param Set 1'`

`Baseline` — Baseline criteria dataset
Current test case setting (default) | character vector

Baseline criteria dataset name, specified as the comma-separated pair consisting of `'Baseline'` and a character vector. Baseline criteria datasets are setup in the **Baseline Criteria** section of the test case. It is available only for baseline test cases.

Example: `'Baseline','BaselineSet_High'`

`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 setup in the **Inputs** section of the test case.

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 setup in the **Configuration Settings Overrides** section of the test case.

Example: `'ConfigSet','Solver 3'`

`SimulationIndex` — Equivalence test simulation index
Number of the simulation in an equivalence test case (default) | integer

Simulation index of an equivalence test case, specified as an integer.

Example: `'SimulationIndex',2`

`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: `'SignalBuilderGroup','Acceleration'`

`SignalBuilderGroup` — Signal Builder group
Current test case setting (default) | character vector

Signal Builder group input name, specified as the comma-separated pair consisting of `'SignalBuilderGroup'` and a character vector. Signal Builder group overrides are set up in the test case **Inputs** section.

Example: `'SignalBuilderGroup','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 setup 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 setup 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 setup 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 setup 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 setup in the **Description** section of the test case.

Example: `'Description','Test the autopilot controller for wind gusts'`

Examples

Set Test Parameter

```
setTestParam(obj,'Description','Test the autopilot controller for wind gusts');
```

See Also

`sltest.testmanager.TestIteration`

Topics

“Test Iterations”
“Create and Run Test Cases with Scripts”

Introduced in R2016a

setVariable

Class: `sltest.testmanager.TestIteration`
Package: `sltest.testmanager`

Set model variable override

Syntax

```
setVariable(obj,'Name',varName,'Source',srcName,'Value',value)
```

Description

`setVariable(obj,'Name',varName,'Source',srcName,'Value',value)` sets a model variable override for the test iteration. Specify the `sltest.testmanager.TestIteration` object, and then specify the variable name, source, and override value. The method overrides the variable in the test iteration and does not permanently change the model variable.

Input Arguments

`obj` — Test iteration object
object

The test iteration you want to apply the override to, specified as a `sltest.testmanager.TestIteration` object.

`varName` — Variable name
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

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.sldd'`.

For real-time test cases, the possible sources are model element paths, which might be an empty character vector.

`value` — Override value
numeric | logical | enum | struct

Value of the variable to override.

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

Blocks — Alphabetical List

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`

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

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

The Observer Reference block references a separate model called the Observer model, which houses your verification subsystem. Connect the Observer Reference block to Simulink models without signal lines to wirelessly debug and test the design of the system.

Ports

The Observer Reference block does not have inports or outports. Signals are mapped to the Observer Model through the Observer Port block.

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.

To view the model that you specified, click **Open**.

Programmatic Use
Parameter: `ObserverModelName`
Type: character vector
Value: `''` | `'<observer model name>'`
Default: `''`

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

See Also

Topics

“Access Model Data Wirelessly by Using Observers”
Observer Port

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, Atomic Subsystem, CodeReuse 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”
“Actions and Transitions”
“Signal Generation Functions”

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, Atomic Subsystem, CodeReuse 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”
“Actions and Transitions”
“Signal Generation Functions”

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.

Parameters

`Time Precision for Variable Step` — Adjust time increment precision
3 (default)

When using a variable step solver, change this parameter to adjust the time precision for the sequence viewer.

Programmatic Use
Block Parameter: `VariableStepTimePrecision`
Type: character vector
Values: `'3'` | scalar
Default: `'3'`

`History` — Maximum number of events to keep in viewer
5000 (default)

Programmatic Use
Block Parameter: `History`
Type: character vector
Values: `'1000'` | scalar
Default: `'1000'`

Block Characteristics

Data Types	`Boolean` | `bus` | `double` | `enumerated` | `fixed point` | `integer` | `single`
Direct Feedthrough	`no`
Multidimensional Signals	`yes`
Variable-Size Signals	`no`
Zero-Crossing Detection	`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 Block to Visualize Messages, Events, and Entities” (SimEvents)

Introduced in R2015b