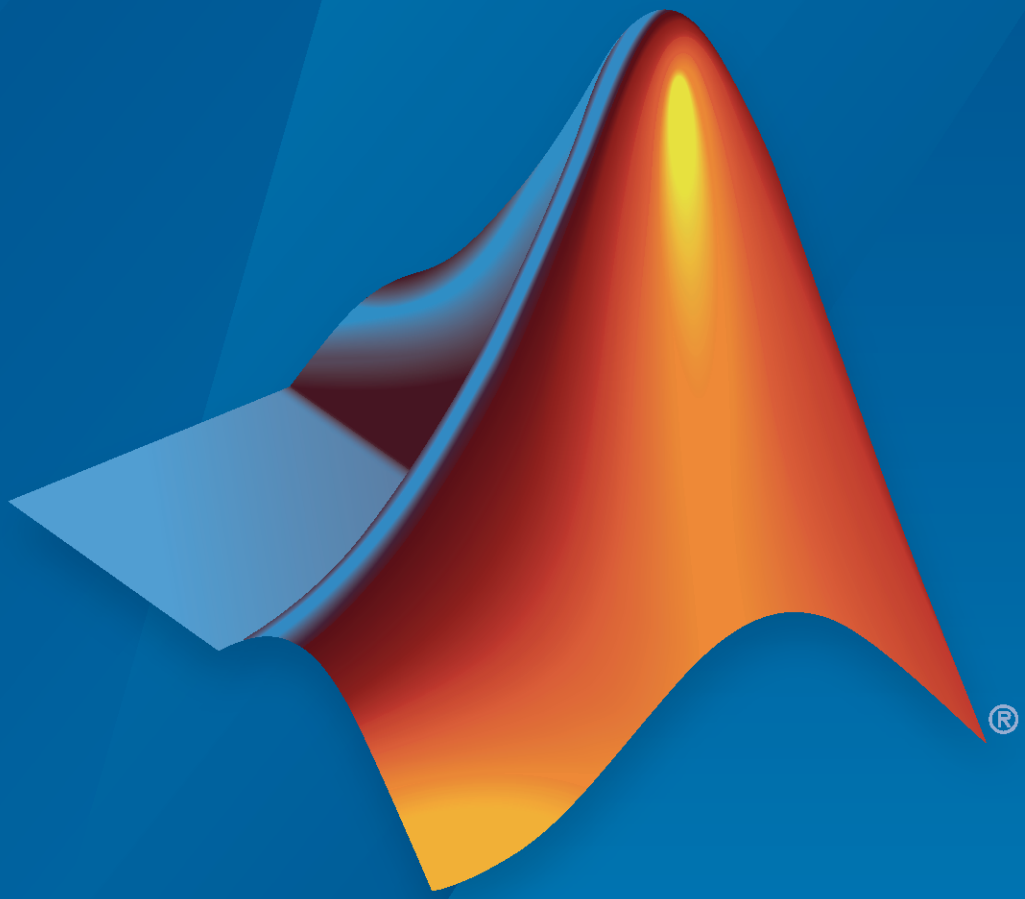
Simulink® Coverage™

Reference



MATLAB® & SIMULINK®

R2021a

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® Coverage™ Reference

Revision History

September 2017	Online only	New for Version 4.0 (Release 2017b)
March 2018	Online only	Revised for Version 4.1 (Release 2018a)
September 2018	Online only	Revised for Version 4.2 (Release 2018b)
March 2019	Online only	Revised for Version 4.3 (Release R2019a)
September 2019	Online only	Revised for Version 4.4 (Release R2019b)
March 2020	Online only	Revised for Version 5.0 (Release R2020a)
September 2020	Online only	Revised for Version 5.1 (Release R2020b)
March 2021	Online only	Revised for Version 5.2 (Release R2021a)

Contents

1 Functions

2 Simulink Coverage Settings

3 Classes

Functions

allNames

Class: `cv.cvdatagroup`
Package: `cv`

Get names of all models associated with `cvdata` objects in `cv.cvdatagroup`

Syntax

```
models = allNames(cvdg)
models = allNames(cvdg, simMode)
```

Description

Get names of all models associated with `cvdata` objects in `cv.cvdatagroup`.

`models = allNames(cvdg)` returns a cell array of character vectors or strings identifying all model names associated with the `cvdata` objects in `cvdg`, an instantiation of the `cv.cvdatagroup` class.

`models = allNames(cvdg, simMode)` returns a cell array of character vectors or strings identifying all model names having the simulation mode `simMode` associated with the `cvdata` objects in `cvdg`, an instantiation of the `cv.cvdatagroup` class.

Input Arguments

`cvdg` — Class instance

object

Instance of class `cv.cvdatagroup`.

`simMode` — Simulation mode

character vector or string

Simulation mode associated with the `cvdata` objects in `cvdg`. Valid values include the following:

Object Specification	Description
`'Normal'`	Model in Normal simulation mode.
`'SIL' (or 'PIL')`	Model in Software-in-the-Loop (SIL) or Processor-in-the-Loop (PIL) simulation mode.
`'ModelRefSIL' (or 'ModelRefPIL')`	Model reference in Software-in-the-Loop (SIL) or Processor-in-the-Loop (PIL) simulation mode.
`'ModelRefTopSIL' (or 'ModelRefTopPIL')`	Model reference in Software-in-the-Loop (SIL or Processor-in-the-Loop (PIL) simulation mode with code interface set to top model.

Output Arguments

`models` — Model names

cell array of character vectors or strings

All model names associated with the `cvdata` objects in `cvdg`.

Examples

Add three `cvdata` objects to `cvdg` and return a cell array of model names:

```
a = cvdata;
b = cvdata;
c = cvdata;
cvdg = cv.cvdatagroup;
add (cvdg, a, b, c);
model_names = allNames(cvdg);
model_names_sim_mode = allnames(cvdg, 'ModelRefSIL')
```

complexityinfo

Retrieve cyclomatic complexity coverage information from `cvdata` object

Syntax

```
complexity = complexityinfo(cvdo, object)
complexity = complexityinfo(cvdo, object, mode)
```

Description

`complexity = complexityinfo(cvdo, object)` returns complexity coverage results from the `cvdata` object `cvdo` for the model component `object`.

`complexity = complexityinfo(cvdo, object, mode)` returns complexity coverage results from the `cvdata` object `cvdo` for the model component `object` for the simulation mode `mode`.

Input Arguments

`cvdo`

`cvdata` object

`object`

The `object` argument specifies an object in the model or Stateflow® chart that received decision coverage. Valid values for `object` include the following:

Object Specification	Description
`BlockPath`	Full path to a model or block
`BlockHandle`	Handle to a model or block
`slObj`	Handle to a Simulink® API object
`sfID`	Stateflow ID
`sfObj`	Handle to a Stateflow API object from a singly instantiated Stateflow chart
`{BlockPath, sfID}`	Cell array with the path to a Stateflow chart or atomic subchart and the ID of an object contained in that chart or subchart
`{BlockPath, sfObj}`	Cell array with the path to a Stateflow chart or subchart and a Stateflow object API handle contained in that chart or subchart
`{BlockHandle, sfID}`	Cell array with a handle to a Stateflow chart or atomic subchart and the ID of an object contained in that chart or subchart

When specifying an S-function block, valid values for `object` include the following:

Object Specification	Description
`{BlockPath, fName}`	Cell array with the path to an S-Function block and the name of a source file.
`{BlockHandle, fName}`	Cell array with an S-Function block handle and the name of a source file.
`{BlockPath, fName, funName}`	Cell array with the path to an S-Function block, the name of a source file, and a function name.
`{BlockHandle, fName, funName}`	Cell array with an S-Function block handle, the name of a source file an a function name.

For coverage data collected during Software-in-the-Loop (SIL) mode or Processor-in-the-Loop (PIL) simulation mode, valid values for `object` include the following:

Object Specification	Description
`{fileName, funName}`	Cell array with the name of a source file and a function name.
`{Model, fileName}`	Cell array with a model name (or model handle) and the name of a source file.
`{Model, fileName, funName}`	Cell array with a model name (or model handle), the name of a source file, and a function name.

`mode`

The `mode` argument specifies the simulation mode for coverage. Valid values for `mode` include the following:

Object Specification	Description
`'Normal'`	Model in Normal simulation mode.
`'SIL' (or 'PIL')`	Model in Software-in-the-Loop (SIL) or Processor-in-the-Loop (PIL) simulation mode.
`'ModelRefSIL' (or 'ModelRefPIL')`	Model reference in Software-in-the-Loop (SIL) or Processor-in-the-Loop (PIL) simulation mode.
`'ModelRefTopSIL' (or 'ModelRefTopPIL')`	Model reference in Software-in-the-Loop (SIL or Processor-in-the-Loop (PIL) simulation mode with code interface set to top model.

Output Arguments

`complexity`

If `cvdo` does not contain cyclomatic complexity coverage results for `object`, `complexity` is empty.

If `cvdo` contains cyclomatic complexity coverage results for `object`, `complexity` is a two-element vector of the form `[total_complexity local_complexity]`:

`total_complexity`	Cyclomatic complexity coverage for `object` and its descendants (if any)

`local_complexity`	Cyclomatic complexity coverage for `object`

If `object` has variable-size signals, `complexity` also contains the variable complexity.

Examples

Retrieve Cyclomatic Complexity Data from Coverage Object

This example shows how to retrieve cyclomatic complexity information for the Gain subsystem of the `slvnvdemo_cv_small_controller` model.

Load the `slvnvdemo_cv_small_controller` model.

```
modelName = 'slvnvdemo_cv_small_controller';
load_system(modelName);
```

Create a test specification object and enable decision, condition, and MCDC coverage. Then, simulate the model using `cvsim`.

```
testObj = cvtest(modelName);
testObj.settings.decision = 1;
testObj.settings.condition = 1;
testObj.settings.mcdc = 1;
covData = cvsim(testObj);
```

Retrieve cyclomatic complexity information for the Gain subsystem.

```
gainPath = [modelName,'/Gain'];
gainComplexity = complexityinfo(covData,gainPath)
```

```
gainComplexity =

     1     0
```

The Gain subsystem itself does not record cyclomatic complexity, but the contents of the subsystem do. This can be seen in the results because the total complexity is 1, which includes the subsystem and all of its descendants. In contrast, the local complexity is 0, indicating that the one point of complexity comes from one of the descendants, in this case a Switch block.

```
switchPath = [modelName,'/Gain/Switch'];
switchComplexity = complexityinfo(covData,switchPath)
```

```
switchComplexity =

     1     1
```

Alternatives

Use the coverage settings to collect and display cyclomatic complexity coverage results in the coverage report:

1. Open the model.
2. In the Simulink Editor, select **Model Settings** on the **Modeling** tab.
3. On the **Coverage** pane of the Configuration Parameters dialog box, select **Enable coverage analysis**.
4. Under **Coverage metrics**, select **MCDC** as the structural coverage level.
5. Click **OK** to close the Configuration Parameters dialog box and save your changes.
6. Simulate the model and review the results in the HTML report.

See Also

`conditioninfo` | `cvsim` | `decisioninfo` | `executioninfo` | `getCoverageInfo` | `mcdcinfo` | `overflowsaturationinfo` | `relationalboundaryinfo` | `sigrangeinfo` | `sigsizeinfo` | `tableinfo`

Topics

“Cyclomatic Complexity”

Introduced in R2011a

conditioninfo

Retrieve condition coverage information from `cvdata` object

Syntax

```
coverage = conditioninfo(cvdo, object)
coverage = conditioninfo(cvdo, object, mode)
coverage = conditioninfo(cvdo, object, ignore_descendants)
[coverage, description] = conditioninfo(cvdo, object)
```

Description

`coverage = conditioninfo(cvdo, object)` returns condition coverage results from the `cvdata` object `cvdo` for the model component specified by `object`.

`coverage = conditioninfo(cvdo, object, mode)` returns condition coverage results from the `cvdata` object `cvdo` for the model component specified by `object` for the simulation mode `mode`.

`coverage = conditioninfo(cvdo, object, ignore_descendants)` returns condition coverage results for `object`, depending on the value of `ignore_descendants`.

`[coverage, description] = conditioninfo(cvdo, object)` returns condition coverage results and textual descriptions of each condition in `object`.

Input Arguments

`cvdo`

`cvdata` object

`object`

An object in the Simulink model or Stateflow diagram that receives decision coverage. Valid values for `object` are as follows:

`BlockPath`	Full path to a Simulink model or block
`BlockHandle`	Handle to a Simulink model or block
`slObj`	Handle to a Simulink API object
`sfID`	Stateflow ID
`sfObj`	Handle to a Stateflow API object
`{BlockPath, sfID}`	Cell array with the path to a Stateflow chart or atomic subchart and the ID of an object contained in that chart or subchart
`{BlockPath, sfObj}`	Cell array with the path to a Stateflow chart or atomic subchart and a Stateflow object API handle contained in that chart or subchart

`{BlockHandle, sfID}`	Cell array with a handle to a Stateflow chart or atomic subchart and the ID of an object contained in that chart or subchart

When specifying an S-function block, valid values for `object` include the following:

Object Specification	Description
`{BlockPath, fName}`	Cell array with the path to an S-Function block and the name of a source file.
`{BlockHandle, fName}`	Cell array with an S-Function block handle and the name of a source file.
`{BlockPath, fName, funName}`	Cell array with the path to an S-Function block, the name of a source file, and a function name.
`{BlockHandle, fName, funName}`	Cell array with an S-Function block handle, the name of a source file an a function name.

For coverage data collected during Software-in-the-Loop (SIL) mode or Processor-in-the-Loop (PIL) simulation mode, valid values for `object` include the following:

Object Specification	Description
`{fileName, funName}`	Cell array with the name of a source file and a function name.
`{Model, fileName}`	Cell array with a model name (or model handle) and the name of a source file.
`{Model, fileName, funName}`	Cell array with a model name (or model handle), the name of a source file, and a function name.

`mode`

The `mode` argument specifies the simulation mode for coverage. Valid values for `mode` include the following:

Object Specification	Description
`'Normal'`	Model in Normal simulation mode.
`'SIL' (or 'PIL')`	Model in Software-in-the-Loop (SIL) or Processor-in-the-Loop (PIL) simulation mode.
`'ModelRefSIL' (or 'ModelRefPIL')`	Model reference in Software-in-the-Loop (SIL) or Processor-in-the-Loop (PIL) simulation mode.
`'ModelRefTopSIL' (or 'ModelRefTopPIL')`	Model reference in Software-in-the-Loop (SIL or Processor-in-the-Loop (PIL) simulation mode with code interface set to top model.

`ignore_descendants`

Logical value that specifies whether to ignore the coverage of descendant objects
`1` to ignore coverage of descendant objects
`0` (default) to collect coverage of descendant objects

Output Arguments

coverage

The value of `coverage` is a two-element vector of form `[covered_outcomes total_outcomes]`. `coverage` is empty if `cvdo` does not contain condition coverage results for `object`. The two elements are:

`covered_outcomes`	Number of condition outcomes satisfied for `object`
`total_outcomes`	Total number of condition outcomes for `object`

description

A structure array with the following fields:

`text`	Description of the condition measured
`condition`	A structure array containing condition info for individual condition outcomes
`isFiltered`	Whether the block is filtered
`filterRationale`	The filtering rationale
`justifiedCoverage`	The justified coverage conditions
`isJustified`	Whether the block is justified

Examples

The following example opens the `slvnvdemo_cv_small_controller` example model, creates the test specification object `testObj`, enables condition coverage for `testObj`, and executes `testObj`. Then retrieve the condition coverage results for the Logic block (in the Gain subsystem) and determine its percentage of condition outcomes covered:

```
mdl = 'slvnvdemo_cv_small_controller';
open_system(mdl)
testObj = cvtest(mdl)
testObj.settings.condition = 1;
data = cvsim(testObj)
blk_handle = get_param([mdl, '/Gain/Logic'], 'Handle');
cov = conditioninfo(data, blk_handle)
percent_cov = 100 * cov(1) / cov(2)
```

Alternatives

Use the coverage settings to collect condition coverage for a model:

1. Open the model for which you want to collect condition coverage.
2. In the Model Editor, select **Model Settings** on the **Modeling** tab.
3. On the **Coverage** pane of the Configuration Parameters dialog box, select **Enable coverage analysis**.
4. Under **Coverage metrics**, select **Condition** as the structural coverage level.

5 Click **OK** to close the Configuration Parameters dialog box and save your changes.

6 Simulate the model by clicking the **Run** button and review the results.

See Also

`complexityinfo` | `cvsim` | `decisioninfo` | `executioninfo` | `getCoverageInfo` | `mcdcinfo` | `overflowsaturationinfo` | `relationalboundaryinfo` | `sigrangeinfo` | `sigsizeinfo` | `tableinfo`

Topics

“Condition Coverage (CC)”

Introduced in R2006b

cv.cvdatagroup class

Package: cv

Collection of cvdata objects

Description

Instances of this class contain a collection of cvdata objects. Each cvdata object contains coverage results for a particular model in the model hierarchy.

Construction

cv.cvdatagroup	Create collection of cvdata objects for model reference hierarchy

Methods

allNames	Get names of all models associated with cvdata objects in cv.cvdatagroup
allSimulationModes	Get names of all simulation modes associated with cvdata objects in cv.cvdatagroup
get	Get cvdata object
getAll	Get all cvdata objects

Properties

name	cv.cvdatagroup object name

Copy Semantics

Handle. To learn how this affects your use of the class, see Copying Objects in the MATLAB® Programming Fundamentals documentation.

cv.cvdatagroup

Class: `cv.cvdatagroup`
Package: `cv`

Create collection of `cvdata` objects for model reference hierarchy

Syntax

```
cvdg = cv.cvdatagroup(cvdo1, cvdo2,...)
```

Description

`cvdg = cv.cvdatagroup(cvdo1, cvdo2,...)` creates a `cv.cvdatagroup` object that contains the specified `cvdata` objects. A `cvdata` object contains coverage results of one or more simulations where coverage is enabled.

Examples

Create a cv.cvdatagroup Object

This example shows how to create a `cvdatagroup` object using two `cvdata` objects.

Record coverage for `slvnvdemo_cv_small_controller`.

```
model_1 = 'slvnvdemo_cv_small_controller';
load_system(model_1)
cvdo1 = cvsim(model_1);
```

Record coverage for `slvnvdemo_powerwindow_controller`.

```
model_2 = 'slvnvdemo_powerwindow_controller';
load_system(model_2)
cvdo2 = cvsim(model_2);
```

Add the two `cvdata` objects to a `cv.cvdatagroup` object.

```
cvdg = cv.cvdatagroup(cvdo1,cvdo2);
```

allSimulationModes

Class: `cv.cvdatagroup`
Package: `cv`

Get names of all simulation modes associated with `cvdata` objects in `cv.cvdatagroup`

Syntax

```
simModes= allSimulationModes(cvdg)
simModes= allSimulationModes(cvdg, modelName)
```

Description

Get names of all simulation modes associated with `cvdata` objects in `cv.cvdatagroup`.

`simModes= allSimulationModes(cvdg)` returns a cell array of character vectors or strings identifying all simulation modes associated with the `cvdata` objects in `cvdg`, an instantiation of the `cv.cvdatagroup` class.

`simModes= allSimulationModes(cvdg, modelName)` returns a cell array of character vectors or strings identifying all simulation modes associated with the model `modelName`in `cvdg`, an instantiation of the `cv.cvdatagroup` class.

Input Arguments

`cvdg` — Class instance

object

Instance of class `cv.cvdatagroup`.

`modelName` — Name of the model

character vector or string

Model with which simulation modes are associated.

Output Arguments

`simModes` — Simulation modes

cell array of character vectors or strings

All simulation modes associated with `cvdg`. Valid values include the following:

Object Specification	Description
`'Normal'`	Model in Normal simulation mode.
`'SIL' (or 'PIL')`	Model in Software-in-the-Loop (SIL) or Processor-in-the-Loop (PIL) simulation mode.

Object Specification	Description
`'ModelRefSIL' (or 'ModelRefPIL')`	Model reference in Software-in-the-Loop (SIL) or Processor-in-the-Loop (PIL) simulation mode.
`'ModelRefTopSIL' (or 'ModelRefTopPIL')`	Model reference in Software-in-the-Loop (SIL or Processor-in-the-Loop (PIL) simulation mode with code interface set to top model.

Examples

Get All Simulation Modes from cvdatagroup

This example shows how to query the simulation modes of the coverage results inside a `cvdatagroup` object.

Record coverage for slvnvdemo_cv_small_controller.

```
model_1 = 'slvnvdemo_cv_small_controller';
load_system(model_1)
cvdo1 = cvsim(model_1);
```

Record coverage for slvnvdemo_powerwindow_controller.

```
model_2 = 'slvnvdemo_powerwindow_controller';
load_system(model_2)
cvdo2 = cvsim(model_2);
```

Record coverage for slvnvdemo_counter.

```
model_3 = 'slvnvdemo_counter';
load_system(model_3)
cvdo3 = cvsim(model_3);
```

Add the three cvdata objects to a cv.cvdatagroup object.

```
cvdg = cv.cvdatagroup(cvdo1,cvdo2,cvdo3);
```

Get the simulation modes by using `allSimulationModes`.

```
simModes = allSimulationModes(cvdg)
```

```
simModes =

  1x1 cell array

    {'Normal'}
```

cvexit

Exit model coverage environment

Syntax

```
cvexit
```

Description

`cvexit` exits the model coverage environment. Issuing this command closes the Coverage Display window and removes coloring from a block diagram that displays its model coverage results.

Introduced in R2006b

cvhtml

Produce HTML report from model coverage objects

Syntax

```
cvhtml(file,cvdo)
cvhtml(file,cvdo1,...,cvdoN)
cvhtml(file,cvdo1,...,cvdoN,options)
cvhtml(file,cvdo,simMode)
```

Description

`cvhtml(file,cvdo)` creates an HTML report of the coverage results in the `cvdata` or `cv.cvdatagroup` object `cvdo` when you run model coverage in simulation. `cvhtml` saves the coverage results in `file`. The model must be open when you use `cvhtml` to generate its coverage report.

`cvhtml(file,cvdo1,...,cvdoN)` creates a combined report of several `cvdata` objects. The results from each object appear in a separate column of the HTML report. Each `cvdata` object must correspond to the same root model or subsystem. Otherwise, the function fails.

`cvhtml(file,cvdo1,...,cvdoN,options)` creates a combined report of several `cvdata` objects using the report options specified by `options`.

`cvhtml(file,cvdo,simMode)` creates an HTML report for the models having the simulation mode `simMode`.

Examples

Create a Coverage Report

Make sure you have write access to the default MATLAB folder. Create a cumulative coverage report for the `slvnvdemo_cv_small_controller` mode and save it as `ratelim_coverage.html`:

```
model = 'slvnvdemo_cv_small_controller';
open_system(model);
cvt = cvtest(model);
cvd = cvsim(cvt);
outfile = 'ratelim_coverage.html';
cvhtml(outfile, cvd);
```

Input Arguments

cvdo — Coverage data

`object`

Coverage data, specified as a `cvdata` object or `cv.cvdatagroup` object.

`file` — HTML file name
`character array | string array`

HTML file name, specified as a character or string array. You can specify the absolute path or relative path and the HTML file where `cvhtml` stores the report.

`options` — Report options
`character array | string array`

Reporting options, specified as a character or string array.

- To enable an option, set it to `1` (e.g., `'-hTR=1'`).
- To disable an option, set it to `0` (e.g., `'-bRG=0'`).
- To specify multiple report options, list individual options in a single `options` character vector or string separated by commas or spaces (e.g., `'-hTR=1 -bRG=0 -scm=0'`).

Option	Description	Default
`-sRT`	Show report	`on`
`-sVT`	Web view mode	`off`
`-aTS`	Include each test in the model summary	`on`
`-bRG`	Produce bar graphs in the model summary	`on`
`-bTC`	Use two color bar graphs (red, blue)	`on`
`-hTR`	Display hit/count ratio in the model summary	`off`
`-xEM`	Exclude execution metric details from report	`off`
`-nFC`	Exclude fully covered model objects from report	`off`
`-nFD`	Exclude fully covered model object details from report	`on`
`-scm`	Include cyclomatic complexity numbers in summary	`on`
`-bcm`	Include cyclomatic complexity numbers in block details	`on`
`-xEv`	Filter Stateflow events from report	`off`

`simMode` — Simulation mode
`character array | string array`

Simulation mode, specified as a character or string array. Valid values include the following:

Object Specification	Description
`'Normal'`	Model in Normal simulation mode.
`'SIL' (or 'PIL')`	Model in Software-in-the-Loop (SIL) or Processor-in-the-Loop (PIL) simulation mode.
`'ModelRefSIL' (or 'ModelRefPIL')`	Model reference in Software-in-the-Loop (SIL) or Processor-in-the-Loop (PIL) simulation mode.
`'ModelRefTopSIL' (or 'ModelRefTopPIL')`	Model reference in Software-in-the-Loop (SIL or Processor-in-the-Loop (PIL) simulation mode with code interface set to top model.

Alternatives

Use the coverage settings to create a model coverage report in an HTML file:

1. Open the model for which you want a model coverage report.
2. In the Simulink Editor, select **Model Settings** on the **Modeling** tab.
3. On the **Coverage** pane of the Configuration Parameters dialog box, select **Enable coverage analysis**.
4. Click **OK** to close the Configuration Parameters dialog box and save your changes.
5. Simulate the model by clicking the **Run** button and review the generated report.

See Also

`cv.cvdatagroup` | `cvmodelview` | `cvsim`

Topics

“Automating Model Coverage Tasks”

Introduced before R2006a

cvload

Load coverage tests and stored results into memory

Syntax

```
[covSettings,covData] = cvload(fileName)
[covSettings,covData] = cvload(fileName,restoreTotal)
```

Description

`[covSettings,covData] = cvload(fileName)` loads the tests and data stored in the specified file.

`[covSettings,covData] = cvload(fileName,restoreTotal)` restores or clears the cumulative results from prior runs depending on the value of `restoreTotal`.

Note When using the `cvload` command:

- If a model with the same name exists in the coverage database, `cvload` only loads the compatible results that reference the existing model to prevent duplication.
- If the Simulink models referenced from the file are open, but do not exist in the coverage database, `cvload` resolves the links to the existing models.
- When you are loading several files that reference the same model, `cvload` only loads the results that are consistent with the earlier files.
- Starting in R2020b, you can load coverage data created in R2017b or later. You can aggregate coverage data from two or more `cvdata` objects for the same model if the `dbVersion` properties match.

Examples

Load Coverage Data and Preserve Cumulative Data

Load the file `myCovData.cvt` while maintaining cumulative coverage results.

```
[covSettings,covData] = cvload('myCovData',1);
```

Input Arguments

`fileName` — Name of coverage data file

`character array | string array`

Name of coverage data file, specified as a character array or string array. `fileName` must be a coverage data file with the `.cvt` extension. You do not need to include the extension in `fileName`.

Example: `'myCoverageData'`

Data Types: `char` | `string`

`restoreTotal` — Cumulative data restoration setting
`0` (default) | `1`

Cumulative data restoration setting, specified as `1` or `0`. If `restoreTotal` is set to `1`, `cvload` restores the cumulative results from prior runs. If `restoreTotal` is set to `0` or unspecified, `cvload` clears the cumulative results.

Data Types: `double`

Output Arguments

`covSettings` — Coverage settings
`cell array`

Coverage settings, returned as a cell array of `cvtest` objects. The coverage settings are returned as `cvtest` objects even if you did not use `cvtest` and `cvsim` to collect the original data.

Data Types: `cell`

`covData` — Coverage data
`cell array`

Coverage data, returned as a cell array of `cvdata` objects. `covData` has the same size as `covSettings`, but if a settings entry has no results, `covData` can contain empty elements.

Alternatives

You can load existing coverage data in the Coverage Results window:

1. Open the model for which you want to load existing coverage data.
2. In the **Apps** tab, select **Coverage Analyzer**.
3. In the **Coverage** tab, select **Results Explorer**.
4. In the Coverage Results window, right click **Data Repository** and select **Load coverage data**.
5. Select the coverage data file that you want to load.

See Also

cvdata Properties | `cvsave` | `cvtest`

Topics
“Retrieve Coverage Details from Results”

Introduced before R2006a

cvmodelview

Display model coverage results with model coloring

Syntax

```
cvmodelview(cvdo)
cvmodelview(cvdo, simMode)
```

Description

`cvmodelview(cvdo)` displays coverage results from the `cvdata` object `cvdo` by coloring the objects in the model that have model coverage results.

`cvmodelview(cvdo, simMode)` displays coverage results from the `cvdata` object `cvdo` by coloring the objects in the model that have model coverage results for the specified simulation mode.

Input Arguments

`cvdo` — Coverage data

object

Coverage data, specified as a `cvdata` object or `cv.cvdatagroup` object.

`simMode` — Simulation mode

`character array` | `string array`

Simulation mode, specified as a character or string array. Valid values include the following:

Object Specification	Description
`'Normal'`	Model in Normal simulation mode.
`'SIL' (or 'PIL')`	Model in Software-in-the-Loop (SIL) or Processor-in-the-Loop (PIL) simulation mode.
`'ModelRefSIL' (or 'ModelRefPIL')`	Model reference in Software-in-the-Loop (SIL) or Processor-in-the-Loop (PIL) simulation mode.
`'ModelRefTopSIL' (or 'ModelRefTopPIL')`	Model reference in Software-in-the-Loop (SIL or Processor-in-the-Loop (PIL) simulation mode with code interface set to top model.

Examples

Open the `slvnvdemo_cv_small_controller` example model, create the test specification object `testObj`, and execute `testObj` to collect model coverage. Run `cvmodelview` to color the model objects for which you collect model coverage information:

```
mdl = 'slvnvdemo_cv_small_controller';
open_system(mdl)
testObj = cvtest(mdl)
```

```
data = cvsim(testObj)
cvmodelview(data)
```

Alternatives

Use the coverage settings to display model coverage results by coloring objects:

1. Open the model.
2. Select **Model Settings** on the **Modeling** tab.
3. On the **Coverage** pane of the Configuration Parameters dialog box, select **Enable coverage analysis**.
4. Click **OK** to close the Configuration Parameters dialog box and save your changes.
5. Simulate the model by clicking the **Run** button and review the results.

See Also

`cvhtml` | `cvsim`

Topics

“View Coverage Results in a Model”

Introduced in R2006b

cvresults

Returns active coverage data, clears and loads active coverage data from a file

Syntax

```
[CVDATA, CVCUMDATA] = cvresults(MODELNAME)
cvresults(MODELNAME, 'clear')
cvresults(MODELNAME, 'load', filename)
```

Description

`[CVDATA, CVCUMDATA] = cvresults(MODELNAME)` returns the active single-run coverage data `CVDATA` and cumulative coverage data `CVCUMDATA`.

`cvresults(MODELNAME, 'clear')` clears the active coverage data.

`cvresults(MODELNAME, 'load', filename)` loads the active coverage data from a `.cvt` file.

See Also

Introduced in R2016a

cvsave

Save coverage settings and results to file

Syntax

```
cvsave(fileName,model)
cvsave(fileName,covData)
cvsave(fileName,covSettings1,...,covSettingsN)
```

Description

`cvsave(fileName,model)` saves all the coverage settings and results related to `model` in the file `fileName`.

`cvsave(fileName,covData)` saves all the coverage settings and results contained in the `cvdata` object `covData`.

`cvsave(fileName,covSettings1,...,covSettingsN)` saves multiple `cvtest` objects and information about any referenced models.

Examples

Save Coverage Results

This example shows how to save coverage data to a file.

Start by loading the model into memory.

```
modelName = 'slvnvdemo_cv_small_controller';
load_system(modelName);
```

Simulate the model with the coverage settings that are saved with the model.

```
covData = cvsim(modelName);
```

Save a coverage data file called `coverage_data`, containing the coverage data in the `cvdata` object `covData`.

```
cvsave('coverage_data',covData);
```

Save Cumulative Coverage Data to a File

This example shows how to save more than one coverage data object to a single coverage data file.

Load the Model

Load the model into memory.

```
modelName = 'slvnvdemo_ratelim_harness';
load_system(modelName);
```

Set Model Parameters for Coverage

Create a `Simulink.SimulationInput` object to set coverage parameters.

```
covSet = Simulink.SimulationInput(modelName);
covSet = covSet.setModelParameter('CovEnable','on');
covSet = covSet.setModelParameter('CovMetricStructuralLevel','MCDC');
covSet = covSet.setModelParameter('CovScope','Subsystem');
covSet = covSet.setModelParameter('CovPath','/Adjustable Rate Limiter');
covSet = covSet.setModelParameter('CovSaveSingleToWorkspaceVar','on');
```

Simulate the Model to Collect Coverage Data

Load the data files and then simulate the model to collect two sets of coverage data.

```
load within_lim.mat
covSet = covSet.setModelParameter('CovSaveName','covData1');
simOut1 = sim(covSet);
```

Simulate the model a second time using the second data file.

```
load rising_gain.mat
covSet = covSet.setModelParameter('CovSaveName','covData2');
simOut2 = sim(covSet);
```

Save the Coverage Data to a File

Save the results in a cell array.

```
cov_results{1} = covData1;
cov_results{2} = covData2
```

```
cov_results =

  1x2 cell array

    {1x1 cvdata}    {1x1 cvdata}
```

Save the results to a file.

```
cvsave('ratelim_testdata',cov_results{:});
```

Input Arguments

`fileName` — Name of coverage data file

`character array` | `string array`

Name of coverage data file, specified as a character array or a string array. `cvsave` appends the extension `.cvt` to the name of the file when saving it.

Example: `'myCoverageDataFile'`

Data Types: `char` | `string`

`model` — Simulink model that has coverage data
`character array` | `string array`

Simulink model that has coverage data, specified as a character array or a string array. `model` can be the name of a model or a handle to a model.

Example: `'mySimulinkModel'`

Data Types: `char` | `string`

`covSettings` — Coverage settings
`cvtest` object | `cell array`

Coverage settings, specified as a `cvtest` object, or a cell array of `cvtest` objects.

Data Types: `cvtest` | `cell`

`covData` — Coverage data
`cvdata` object | `cell array`

Coverage data, specified as a `cvdata` object or a cell array of `cvdata` objects.

Data Types: `cvdata` | `cell`

Alternatives

You can save coverage results to a MATLAB workspace variable when you run your model in Simulink:

1. Open the model for which you want to save cumulative coverage results.
2. On the **Modeling tab**, select **Model Settings**.
3. In the left pane of the Configuration Parameters dialog box, select **Coverage**.
4. Select **Enable coverage analysis**.
5. In the **Results** section, select **Save last run in workspace variable**.
6. Click **OK** to close the Configuration Parameters dialog box and save your changes.
7. Simulate the model by clicking the **Run** button and review the results.

See Also

`cv.cvdatagroup` | `cvdata Properties` | `cvload` | `cvtest`

Introduced before R2006a

cvsim

Simulate and return model coverage results for test objects

Syntax

```
cvdo = cvsim(modelName)
cvdo = cvsim(testObj)
[cvdo,simOut] = cvsim(__,Name,Value)
[cvdo,simOut] = cvsim(testObj,paramStruct)
[cvdo1,...,cvdoN] = cvsim(testObj1,...,testObjN)
```

Description

`cvdo = cvsim(modelName)` simulates the model and returns the coverage results in the `cvdata` object, `cvdo`.

`cvdo = cvsim(testObj)` simulates the model and returns the coverage results for the `cvtest` object, `testObj`.

`[cvdo,simOut] = cvsim(__,Name,Value)` specifies the model parameters, simulates the model, and returns the coverage results in the `cvdata` object, `cvdo`, and the simulation outputs in the `Simulink.SimulationOutput` object, `simOut`.

Note `cvsim` ignores model parameters listed in the **Coverage** pane of the Configuration Parameters window. Create a `cvtest` object to set coverage options, or use `sim` instead.

`[cvdo,simOut] = cvsim(testObj,paramStruct)` sets the model parameters specified in a structure `paramStruct`.

`[cvdo1,...,cvdoN] = cvsim(testObj1,...,testObjN)` simulates the model for `N` test objects, and returns the coverage results for each object.

Note `cvsim` will record coverage even if coverage is not enabled in the model parameters.

Examples

Record Coverage Data Using `cvsim`

This example shows how to use `cvsim` with a `cvtest` object input and a structure of model parameters.

Load the `slvnvdemo_cv_small_controller` example model.

```
modelName = 'slvnvdemo_cv_small_controller';
load_system(modelName)
```

Create a `cvtest` object and turn on decision coverage.

```
testObj = cvtest(modelName);
testObj.settings.decision = 1;
```

Create a structure that defines the following model parameters:

- Set the absolute tolerance, `AbsTol`, to `1e-5`.
- Enable the simulation to save states to the workspace with `SaveState`.
- Declare the variable name to save the state information in with `SaveStateName`.
- Enable Simulink® to save simulation output data to the workspace with `SaveOutput`.
- Declare the variable name in which to store the simulation output data with `OutputSaveName`.

```
paramStruct.AbsTol =         '1e-5';
paramStruct.SaveState =      'on';
paramStruct.StateSaveName =  'xoutNew';
paramStruct.SaveOutput =     'on';
paramStruct.OutputSaveName = 'youtNew';
```

Simulate the model with `cvsim` and generate a coverage report with `cvhtml`.

```
[covData,simOut] = cvsim(testObj,paramStruct);
cvhtml('CoverageReport.html',covData,'-sRT=0');
```

Input Arguments

`modelName` — Name of Simulink model

character array | string array

Name of a Simulink Model, specified as a character array or string array. `cvsim` simulates the model with the current coverage settings.

Data Types: `char` | `string`

`testObj` — Coverage test settings

`cvtest` object

Coverage test settings, specified as a `cvtest` object. `cvsim` collects coverage using the settings specified in `testObj`.

Data Types: `cvtest`

`paramStruct` — Model parameters

structure

Model parameters, specified as a structure. You can specify model parameters as a structure and use the structure instead of name-value pair arguments to set multiple parameters.

`paramStruct` fields are the names of model parameters and the values are the corresponding parameter values.

Example: `paramStruct.AbsTol = '1e-5';`

Data Types: `struct`

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: `[cvdo,simOut] = cvsim(testObj,'AbsTol','1e-5');` specifies that the model is simulated using an absolute tolerance of `1e-5` with coverage settings specified in `testObj`.

`cvsim` supports all model parameters that are supported by `sim`, except for parameters in the **Coverage** pane of the Configuration Parameters window.

`ModelParameter` — Model parameter and value

character array | string array

Name of a model parameter, specified as a character array or string array. The value of the parameter is specified as the next argument.

Data Types: `char` | `string`

Output Arguments

`cvdo` — Coverage data object

`cvdata`

`cvdo`, returned as a `cvdata` object. When recording coverage for multiple models in a hierarchy, `cvdo` is a `cv.cvdatagroup` object instead. `cvdo` contains the coverage data from the simulated system.

See “Evaluate Coverage Results” for a list of `info` functions.

See `cvdata Properties` for the object structure.

`simOut` — Simulation data object

`Simulink.SimulationOutput`

`simOut`, returned as a `Simulink.SimulationOutput` object.

See Also

`cv.cvdatagroup` | `cvdata Properties` | `cvhtml` | `cvtest` | `sim`

Introduced before R2006a

cvtest

Create model coverage test specification object

Description

Use `cvtest` to create a test specification object that stores model coverage settings. Pass the `cvtest` object to the `cvsim` function to execute coverage analysis based on your settings.

Creation

Syntax

```
cvto = cvtest(root)
cvto = cvtest(root,label)
cvto = cvtest(root,label,setupCmd)
```

Description

`cvto = cvtest(root)` creates a `cvtest` object with default coverage settings. `root` can be the name of a model or the handle to a model. `root` can also be the name or handle to a subsystem within the model, in which case only the specified subsystem and its descendents are analyzed for coverage.

`cvto = cvtest(root,label)` creates a `cvtest` object with the designated label.

`cvto = cvtest(root,label,setupCmd)` creates a `cvtest` object with the setup command setupCmd. The setup command is executed in the base MATLAB workspace before running coverage analysis.

Input Arguments

`root` — Name or handle of model or path to subsystem

`character array | string array`

Model name or handle, or path to a subsystem, specified as a character array or string array.

Properties

`id` — Internal Model ID

`scalar`

This property is read-only.

Internal model ID, returned as a scalar.

`modelcov` — Internal Coverage Configuration ID

`scalar`

This property is read-only.

Internal coverage configuration ID, returned as a scalar.

`rootPath` — Name or handle of system to analyze
`character array` | `string array`

This property is read-only.

Name of the system you specified to analyze, returned as a character array or string array.

Data Types: `char` | `string`

`label` — Test label
`character array` | `string array`

Test label, specified as a character array or a string array. This label appears in the coverage report as the test name.

Data Types: `char` | `string`

`setupCmd` — Command executed in base MATLAB workspace before simulation
`character array` | `string array`

Command executed in base MATLAB workspace before simulation, specified as a character array or string array.

The setup command is executed before each simulation.

Data Types: `char` | `string`

`settings` — Coverage settings
`structure`

Types of coverage to collect, specified as a structure.

`settings` includes the following fields:

Property	Description	Values
`settings.decision`	Enable decision coverage data.	`1` (default) | `0`
`settings.condition`	Enable condition coverage data.	`1` | `0` (default)
`settings.mcdc`	Enable modified condition decision coverage (MCDC) data. If `settings.mcdc` is enabled, you can also choose which definition of MCDC to use with the `options.mcdcmode` property.	`1` | `0` (default)
`settings.designverifier`	Enable coverage data from Simulink Design Verifier™ blocks.	`1` | `0` (default)

Property	Description	Values
`settings.tableExec`	Enable coverage data for lookup tables.	`1` | `0` (default)
`settings.sigrange`	Enable signal range data.	`1` | `0` (default)
`settings.sigsize`	Enable signal size data.	`1` | `0` (default)
`settings.overflowsaturation`	Enable saturation on integer overflow coverage data.	`1` | `0` (default)
`settings.relationalop`	Enable relational boundary coverage data. Use `options.covBoundaryRelTol` and `options.covBoundaryAbsTol` to specify tolerances for this type of coverage. For more information, see "Relational Boundary Coverage"	`1` | `0` (default)

`options` — Advanced coverage options
`structure`

Advanced coverage options, specified as a structure.

`options` includes the following fields:

Property	Description	Values
`options.covBoundaryRelTol`	Relative tolerance for relational boundary coverage. For more information, see "Relational Boundary Coverage".	`0.01` (default) | `scalar`
`options.CovBoundaryAbsTol`	Absolute tolerance for relational boundary coverage. For more information, see "Relational Boundary Coverage".	`1e-5` (default) | `scalar`

Property	Description	Values
`options.useTimeInterval`	Whether to restrict model coverage recording to a specified simulation time interval. Use `options.intervalStartTime` and `options.intervalStopTime` to specify the time interval. For more information, see "Specify Coverage Options"	`1 | 0` (default)
`options.intervalStartTime`	When to start recording coverage. Specify this property if `options.useTimeInterval` is enabled.	`0` (default) `| scalar`
`options.intervalStopTime`	When to stop recording coverage. Specify this property if `options.useTimeInterval` is enabled.	`0` (default) `| scalar`

Property	Description	Values
`options.forceBlockReduction`	Whether to record coverage for blocks flagged with the **Block Reduction** parameter. • `1` (default) — Override the Simulink **Block Reduction** parameter if it is enabled. Coverage is recorded for every supported block in the model. The value of the configuration parameter **Block Reduction** is ignored. • `0` — Use the value for the configuration parameter **Block Reduction**. If **Block Reduction** is enabled, coverage is not recorded for blocks that are effectively removed from the model because of block reduction. For instance, coverage is not recorded for a block that is reduced by dead code elimination. For more information, see “Block reduction”	`1` (default) | `0`
`options.mcdcMode`	Which MCDC definition to apply to the model, specified as one of the following options: • `'masking'` — Use the `masking` definition of MCDC coverage. • `'unique cause'` — Use the `unique cause` definition of MCDC coverage. For more information, see “Modified Condition and Decision Coverage (MCDC) Definitions in Simulink Coverage”.	`'masking'` (default) | `'unique cause'`

`filter` — Coverage filter
`structure`

Coverage filter, specified as a structure.

`filter` has one field, `filter.fileName`. `filter.fileName` is the name of a coverage filter file to apply to coverage analysis, specified as a character array or string array.

For more information, see “Coverage Filter Rules and Files”

`modelRefSettings` — Model reference settings
`structure` (default)

Model reference settings, specified as a structure.

`modelRefSettings` includes the following fields:

Property	Description	Values
`modelRefSettings.enable`	Model reference coverage setting, specified as one of the following options: • `'off'` — Disable coverage for all referenced models. • `'all'` or `'on'` — Enable coverage for all supported referenced models. • `'filtered'` — Enable coverage for all supported referenced models except those listed in the `excludedModels` field.	`'off'` (default) | `'on'` | `'all'` | `'filtered'`
`modelRefSettings.excludeTopModel`	Whether to exclude the top model from coverage analysis, specified as a numeric or logical `1 (true)` or `0 (false)`.	`1` (default) | `0`
`modelRefSettings.excludedModels`	Referenced models to exclude from coverage analysis, specified as a single character or string array of comma-separated model names. To use this field, set `modelRefSettings.enable` to `'filtered'`.	`char` | `string`

`emlSettings` — Whether to collect coverage for external program files called by MATLAB functions
`structure`

Whether to collect coverage for external program files called by MATLAB functions in your model, specified as a structure.

`emlSettings` has one field, `emlSettings.enableExternal`. `emlSettings.enableExternal` is whether to collect external program files called by MATLAB functions, specified as a numeric or logical `1 (true)(default)` or `0 (false)`.

`sfcnSettings` — Whether to collect coverage for C/C++ S-Function blocks
`structure` (default)

Whether to collect coverage for C/C++ S-Function blocks in your model, specified as a structure.

`sfcnSettings` has one field, `sfcnSettings.enableSfcn`. `sfcnSettings.enableSfcn` is whether to collect coverage S-Function coverage, specified as a logical `1 (true)(default)` or `0 (false)`.

For more information, see S-Function.

Examples

Create `cvtest` Object

In this example, you create a `cvtest` object for the Adjustable Rate Limiter block in the `slvnvdemo_ratelim_harness` model. Simulate the model to get decision coverage and saturation on integer overflow coverage data.

Open the `slvnvdemo_ratelim_harness` model and define the test object using `cvtest`.

```
open_system('slvnvdemo_ratelim_harness');
testObj = cvtest(['slvnvdemo_ratelim_harness', ...
          '/Adjustable Rate Limiter']);
testObj.label = 'Gain within slew limits';
```

Add a setup command to `testObj`. The setup command is executed in the base MATLAB workspace before running the coverage analysis. In this case, the setup command loads data into the workspace that is required for the simulation.

```
testObj.setupCmd = ...
    'load slvnvdemo_ratelim_harness_data.mat';
```

To collect decision coverage and saturation on integer overflow coverage, enable the `decision` and `overflowsaturation` fields in the `settings` structure by setting the fields to `1`.

```
testObj.settings.decision = 1;
testObj.settings.overflowsaturation = 1;
```

Finally, simulate the model with the coverage analysis by providing the `cvtest` object to the `cvsim` function.

```
cvdo = cvsim(testObj);
```

See Also

`cv.cvdatagroup` | `cvsim`

Topics

“Automating Model Coverage Tasks”

Introduced before R2006a

decisioninfo

Retrieve decision coverage information from `cvdata` object

Syntax

```
coverage = decisioninfo(cvdo, object)
coverage = decisioninfo(cvdo, object, mode)
coverage = decisioninfo(cvdo, object, ignore_descendants)
[coverage, description] = decisioninfo(cvdo, object)
```

Description

`coverage = decisioninfo(cvdo, object)` returns decision coverage results from the `cvdata` object `cvdo` for the model component specified by `object`.

`coverage = decisioninfo(cvdo, object, mode)` returns decision coverage results from the `cvdata` object `cvdo` for the model component specified by `object` for the simulation mode `mode`.

`coverage = decisioninfo(cvdo, object, ignore_descendants)` returns decision coverage results for `object`, depending on the value of `ignore_descendants`.

`[coverage, description] = decisioninfo(cvdo, object)` returns decision coverage results and text descriptions of decision points associated with `object`.

Input Arguments

`cvdo`

`cvdata` object

`object`

The `object` argument specifies an object in the model or Stateflow chart that received decision coverage. Valid values for `object` include the following:

Object Specification	Description
`BlockPath`	Full path to a model or block
`BlockHandle`	Handle to a model or block
`slObj`	Handle to a Simulink API object
`sfID`	Stateflow ID
`sfObj`	Handle to a Stateflow API object from a singly instantiated Stateflow chart
`{BlockPath, sfID}`	Cell array with the path to a Stateflow chart or atomic subchart and the ID of an object contained in that chart or subchart

Object Specification	Description
`{BlockPath, sfObj}`	Cell array with the path to a Stateflow chart or subchart and a Stateflow object API handle contained in that chart or subchart
`{BlockHandle, sfID}`	Cell array with a handle to a Stateflow chart or atomic subchart and the ID of an object contained in that chart or subchart

When specifying an S-function block, valid values for `object` include the following:

Object Specification	Description
`{BlockPath, fName}`	Cell array with the path to an S-Function block and the name of a source file.
`{BlockHandle, fName}`	Cell array with an S-Function block handle and the name of a source file.
`{BlockPath, fName, funName}`	Cell array with the path to an S-Function block, the name of a source file, and a function name.
`{BlockHandle, fName, funName}`	Cell array with an S-Function block handle, the name of a source file an a function name.

For coverage data collected during Software-in-the-Loop (SIL) mode or Processor-in-the-Loop (PIL) simulation mode, valid values for `object` include the following:

Object Specification	Description
`{fileName, funName}`	Cell array with the name of a source file and a function name.
`{Model, fileName}`	Cell array with a model name (or model handle) and the name of a source file.
`{Model, fileName, funName}`	Cell array with a model name (or model handle), the name of a source file, and a function name.

`mode`

The `mode` argument specifies the simulation mode for coverage. Valid values for `mode` include the following:

Object Specification	Description
`'Normal'`	Model in Normal simulation mode.
`'SIL' (or 'PIL')`	Model in Software-in-the-Loop (SIL) or Processor-in-the-Loop (PIL) simulation mode.
`'ModelRefSIL' (or 'ModelRefPIL')`	Model reference in Software-in-the-Loop (SIL) or Processor-in-the-Loop (PIL) simulation mode.
`'ModelRefTopSIL' (or 'ModelRefTopPIL')`	Model reference in Software-in-the-Loop (SIL or Processor-in-the-Loop (PIL) simulation mode with code interface set to top model.

`ignore_descendants`

Specifies to ignore the coverage of descendant objects if `ignore_descendants` is set to `1`.

Output Arguments

coverage

The value of `coverage` is a two-element vector of the form `[covered_outcomes total_outcomes]`.`coverage` is empty if `cvdo` does not contain decision coverage results for `object`. The two elements are:

`covered_outcomes`	Number of decision outcomes satisfied for `object`
`total_outcomes`	Number of decision outcomes for `object`

description

`description` is a structure array containing the following fields:

`text`	Description of the decision measured
`decision`	Structure array describing individual decisions, including filtering information. `decision.outcome` is a structure array describing individual decision outcomes, including filtering information for outcomes
`isFiltered`	Whether the block is filtered
`filterRationale`	The filtering rationale
`justifiedCoverage`	The justified decision conditions
`isJustified`	Whether the block is justified

Examples

Open the `slvnvdemo_cv_small_controller` model and create the test specification object `testObj`. Enable decision coverage for `slvnvdemo_cv_small_controller` and execute `testObj` using `cvsim`. Use `decisioninfo` to retrieve the decision coverage results for the Saturation block and determine the percentage of decision outcomes covered:

```
mdl = 'slvnvdemo_cv_small_controller';
open_system(mdl)
testObj = cvtest(mdl)
testObj.settings.decision = 1;
data = cvsim(testObj)
blk_handle = get_param([mdl, '/Saturation'], 'Handle');
cov = decisioninfo(data, blk_handle)
percent_cov = 100 * cov(1) / cov(2)
```

Alternatives

Use the coverage settings to collect and display decision coverage results:

1. Open the model.
2. In the Model Editor, select **Model Settings** on the **Modeling** tab.
3. On the **Coverage** pane of the Configuration Parameters dialog box, select **Enable coverage analysis**.

4 Under **Coverage metrics**, select **Decision** as the structural coverage level.

5 Click **OK** to close the Configuration Parameters dialog box and save your changes.

6 Simulate the model by clicking the **Run** button and review the results.

See Also

`complexityinfo` | `conditioninfo` | `cvsim` | `executioninfo` | `getCoverageInfo` | `mcdcinfo` | `overflowsaturationinfo` | `relationalboundaryinfo` | `sigrangeinfo` | `sigsizeinfo` | `tableinfo`

Topics

“Decision Coverage (DC)”

Introduced in R2006b

executioninfo

Retrieve execution coverage information from `cvdata` object

Syntax

```
coverage = executioninfo(cvdo, object)
coverage = executioninfo(cvdo, object, mode)
coverage = executioninfo(cvdo, object, ignore_descendants)
[coverage, description] = executioninfo(cvdo, object)
```

Description

`coverage = executioninfo(cvdo, object)` returns execution coverage results from the `cvdata` object `cvdo` for the model component specified by `object`.

`coverage = executioninfo(cvdo, object, mode)` returns execution coverage results from the `cvdata` object `cvdo` for the model component specified by `object` for the simulation mode `mode`.

`coverage = executioninfo(cvdo, object, ignore_descendants)` returns execution coverage results for `object`, depending on the value of `ignore_descendants`.

`[coverage, description] = executioninfo(cvdo, object)` returns execution coverage results and text descriptions of execution points associated with `object`.

Input Arguments

`cvdo`

`cvdata` object

`object`

The `object` argument specifies an object in the model or Stateflow chart that received execution coverage. Valid values for `object` include the following:

Object Specification	Description
`BlockPath`	Full path to a model or block
`BlockHandle`	Handle to a model or block
`slObj`	Handle to a Simulink API object
`sfID`	Stateflow ID
`sfObj`	Handle to a Stateflow API object from a singly instantiated Stateflow chart
`{BlockPath, sfID}`	Cell array with the path to a Stateflow chart or atomic subchart and the ID of an object contained in that chart or subchart

Object Specification	Description
`{BlockPath, sfObj}`	Cell array with the path to a Stateflow chart or subchart and a Stateflow object API handle contained in that chart or subchart
`{BlockHandle, sfID}`	Cell array with a handle to a Stateflow chart or atomic subchart and the ID of an object contained in that chart or subchart

When specifying an S-function block, valid values for `object` include the following:

Object Specification	Description
`{BlockPath, fName}`	Cell array with the path to an S-Function block and the name of a source file.
`{BlockHandle, fName}`	Cell array with an S-Function block handle and the name of a source file.
`{BlockPath, fName, funName}`	Cell array with the path to an S-Function block, the name of a source file, and a function name.
`{BlockHandle, fName, funName}`	Cell array with an S-Function block handle, the name of a source file an a function name.

For coverage data collected during Software-in-the-Loop (SIL) mode or Processor-in-the-Loop (PIL) simulation mode, valid values for `object` include the following:

Object Specification	Description
`{fileName, funName}`	Cell array with the name of a source file and a function name.
`{Model, fileName}`	Cell array with a model name (or model handle) and the name of a source file.
`{Model, fileName, funName}`	Cell array with a model name (or model handle), the name of a source file, and a function name.

`mode`

The `mode` argument specifies the simulation mode for coverage. Valid values for `mode` include the following:

Object Specification	Description
`'Normal'`	Model in Normal simulation mode.
`'SIL' (or 'PIL')`	Model in Software-in-the-Loop (SIL) or Processor-in-the-Loop (PIL) simulation mode.
`'ModelRefSIL' (or 'ModelRefPIL')`	Model reference in Software-in-the-Loop (SIL) or Processor-in-the-Loop (PIL) simulation mode.
`'ModelRefTopSIL' (or 'ModelRefTopPIL')`	Model reference in Software-in-the-Loop (SIL or Processor-in-the-Loop (PIL) simulation mode with code interface set to top model.

`ignore_descendants`

Specifies to ignore the coverage of descendant objects if `ignore_descendants` is set to `1`.

Output Arguments

coverage

The value of `coverage` is a two-element vector of the form `[covered_outcomes total_outcomes]`.`coverage` is empty if `cvdo` does not contain execution coverage results for `object`. The two elements are:

`covered_outcomes`	Number of execution outcomes satisfied for `object`
`total_outcomes`	Number of execution outcomes for `object`

description

`description` is a structure array containing the following fields:

`decision.text`	Structure array describing block execution counts
`isFiltered`	Whether the block is filtered
`filterRationale`	The filtering rationale
`justifiedCoverage`	The justified decision conditions
`isJustified`	Whether the block is justified

Examples

Open the `slvnvdemo_cv_small_controller` model and create the test specification object `testObj`. Enable execution coverage for `slvnvdemo_cv_small_controller` and execute `testObj` using `cvsim`. Use `executioninfo` to retrieve the execution coverage results for the Saturation block and determine the percentage of execution outcomes covered:

```
mdl = 'slvnvdemo_cv_small_controller';
open_system(mdl)
testObj = cvtest(mdl)
data = cvsim(testObj)
blk_handle = get_param([mdl, '/Saturation'], 'Handle');
cov = executioninfo(data, blk_handle)
percent_cov = 100 * cov(1) / cov(2)
```

Alternatives

Use the coverage settings to collect and display execution coverage results:

1. Open the model.
2. In the Model Editor, select **Model Settings** on the **Modeling** tab.
3. On the **Coverage** pane of the Configuration Parameters dialog box, select **Enable coverage analysis**.
4. Under **Coverage metrics**, select **Block Execution** as the structural coverage level.
5. Click **OK** to close the Configuration Parameters dialog box and save your changes.
6. Simulate the model by clicking the **Run** button and review the results.

See Also

`complexityinfo` | `conditioninfo` | `cvsim` | `decisioninfo` | `getCoverageInfo` | `mcdcinfo` | `overflowsaturationinfo` | `relationalboundaryinfo` | `sigrangeinfo` | `sigsizeinfo` | `tableinfo`

Topics

“Execution Coverage (EC)”

Introduced in R2006b

get

Class: `cv.cvdatagroup`
Package: `cv`

Get `cvdata` object

Syntax

```
get(cvdg, model_name)
get(cvdg, model_name, simMode)
```

Description

Get `cvdata` object.

`get(cvdg, model_name)` returns the `cvdata` object in the `cv.cvdatagroup` object `cvdg` that corresponds to the model specified in `model_name`.

`get(cvdg, model_name, simMode)` returns the `cvdata` object in the `cv.cvdatagroup` object `cvdg` that corresponds to the model specified in `model_name` having the simulation mode `simMode`.

Input Arguments

`cvdg` — Class instance

object

Instance of class `cv.cvdatagroup`.

`model_name` — Name of the model

character vector or string

Model to which the `cvdata` object in the `cv.cvdatagroup` object `cvdg` corresponds.

`simMode` — Simulation mode

character vector or string

Simulation mode for the `cvdata` object in the `cv.cvdatagroup` object. Valid values include the following:

Object Specification	Description
`'Normal'`	Model in Normal simulation mode.
`'SIL' (or 'PIL')`	Model in Software-in-the-Loop (SIL) or Processor-in-the-Loop (PIL) simulation mode.
`'ModelRefSIL' (or 'ModelRefPIL')`	Model reference in Software-in-the-Loop (SIL) or Processor-in-the-Loop (PIL) simulation mode.
`'ModelRefTopSIL' (or 'ModelRefTopPIL')`	Model reference in Software-in-the-Loop (SIL or Processor-in-the-Loop (PIL) simulation mode with code interface set to top model.

Examples

Get a `cvdata` object from the specified Simulink model:

```
get(cvdg, 'slvnvdemo_cv_small_controller');
get(cvdg, 'slvnvdemo_cv_small_controller', 'ModelRefSIL');
```

getAll

Class: `cv.cvdatagroup`
Package: `cv`

Get all `cvdata` objects

Syntax

```
getAll(cvdg)
getAll(cvdg, simMode)
```

Description

Get all `cvdata` objects.

`getAll(cvdg)` returns all `cvdata` objects in the `cv.cvdatagroup` object `cvdg`.

`getAll(cvdg, simMode)` returns all `cvdata` objects in the `cv.cvdatagroup` object `cvdg` having the simulation mode `simMode`.

Input Arguments

`cvdg` — Class instance

object

Instance of class `cv.cvdatagroup`.

`simMode` — Simulation mode

character vector or string

Simulation mode associated with the `cvdata` objects in `cvdg`. Valid values include the following:

Object Specification	Description
`'Normal'`	Model in Normal simulation mode.
`'SIL' (or 'PIL')`	Model in Software-in-the-Loop (SIL) or Processor-in-the-Loop (PIL) simulation mode.
`'ModelRefSIL' (or 'ModelRefPIL')`	Model reference in Software-in-the-Loop (SIL) or Processor-in-the-Loop (PIL) simulation mode.
`'ModelRefTopSIL' (or 'ModelRefTopPIL')`	Model reference in Software-in-the-Loop (SIL or Processor-in-the-Loop (PIL) simulation mode with code interface set to top model.

Examples

Return all `cvdata` objects from the specified Simulink model:

```
getAll(cvdg, 'slvnvdemo_cv_small_controller');
getAll(cvdg, 'slvnvdemo_cv_small_controller', 'ModelRefSIL');
```

extract

Extract subsystem coverage data from system-level coverage data

Syntax

```
excvd = extract(cvdo,subsystem)
```

Description

`excvd = extract(cvdo,subsystem)` extracts coverage data for `subsystem` from system-level coverage data `cvdo`.

Examples

Create HTML Coverage Report for a Subsystem from Model Coverage Data

Record coverage data for the `sf_car` model.

```
model = 'sf_car'
open(model)
cvdo = cvsim(model);
```

Extract the coverage data for the `shift_logic` subsystem from the coverage data from the top-level model `sf_car`.

```
excvd = extract(cvdo,'sf_car/shift_logic');
```

Create a coverage report from the extracted coverage data.

```
cvhtml('tmp', excvd)
```

Input Arguments

`cvdo` — System-level coverage data object

object (default)

System-level coverage data object from a top-level model, specified as an object.

Data Types: `object`

`subsystem` — Subsystem in a Simulink model

character vector (default)

Full name or path of a subsystem in an open or loaded Simulink model, specified as a character vector.

Data Types: `character vector`

Output Arguments

excvd — Extracted coverage data object

object

Coverage data object for a subsystem extracted from a Simulink model, returned as an object.

See Also

`cv.cvdatagroup` | `cvhtml` | `cvsim`

Topics

“Automating Model Coverage Tasks”

Introduced in R2019b

getCoverageInfo

Retrieve coverage information for Simulink Design Verifier blocks from `cvdata` object

Syntax

```
[coverage, description] = getCoverageInfo(cvdo, object)
[coverage, description] = getCoverageInfo(cvdo, object, metric)
[coverage, description] = getCoverageInfo(cvdo, object, metric,
ignore_descendants)
```

Description

`[coverage, description] = getCoverageInfo(cvdo, object)` collects Simulink Design Verifier coverage for `object`, based on coverage results in `cvdo`. `object` is a handle to a block, subsystem, or Stateflow chart. `getCoverageData` returns coverage data only for Simulink Design Verifier library blocks in `object`'s hierarchy.

`[coverage, description] = getCoverageInfo(cvdo, object, metric)` returns coverage data for the block type specified in `metric`. If `object` does not match the block type, `getCoverageInfo` does not return data.

`[coverage, description] = getCoverageInfo(cvdo, object, metric, ignore_descendants)` returns coverage data about `object`, omitting coverage data for its descendant objects if `ignore_descendants` equals `1`.

Input Arguments

`cvdo`

`cvdata` object

`object`

In the model or Stateflow chart, object that received Simulink Design Verifier coverage. The following are valid values for `object`.

`BlockPath`	Full path to a model or block
`BlockHandle`	Handle to a model or block
`slObj`	Handle to a Simulink API object
`sfID`	Stateflow ID from a singly instantiated Stateflow chart
`sfObj`	Handle to a Stateflow API object from a singly instantiated Stateflow chart
`{BlockPath, sfID}`	Cell array with the path to a Stateflow chart or atomic subchart and the ID of an object contained in that chart or subchart

`{BlockPath, sfObj}`	Cell array with the path to a Stateflow chart or atomic subchart and a Stateflow object API handle contained in that chart or subchart
`{BlockHandle, sfID}`	Cell array with a handle to a Stateflow chart or atomic subchart and the ID of an object contained in that chart or subchart

Default:

`metric`

`cvmetric.Sldv` enumeration object, or a cell array of enumeration objects, with values that correspond to Simulink Design Verifier library blocks. If you don't specify a metric, `getCoverageInfo` returns coverage information for all available metrics for the specified `object`.

`test`	Test Objective block
`proof`	Proof Objective block
`condition`	Test Condition block
`assumption`	Proof Assumption block

`ignore_descendants`

Boolean value that specifies to ignore the coverage of descendant objects if set to `1`.

Output Arguments

`coverage`

Two-element vector of the form [*`covered_outcomes total_outcomes`*].

`covered_outcomes`	Number of test objectives satisfied for `object`
`total_outcomes`	Total number of test objectives for `object`

`coverage` is empty if `cvdo` does not contain decision coverage results for `object`.

Note If `object` receives coverage for multiple metrics, then the output argument `coverage` is a cell array with each cell corresponding to the objective outcomes for a metric. Each cell contains a two-element vector of the form [*`covered_outcomes total_outcomes`*].

`description`

Structure array containing descriptions of each objective, and descriptions and execution counts for each outcome within `object`.

Note If `object` receives coverage for multiple metrics, then the output argument `description` is a cell array with each cell corresponding to the descriptions for a metric. Each cell contains a structure array containing descriptions of each objective, and descriptions and execution counts for each outcome within `object`.

Examples

Get coverage for all Proof Objective blocks in `Verification Subsystem1`

```
mdl = 'sldvdemo_powerwindow_vs';
open_system(mdl)
set_param(mdl, 'StopTime', '10')
testObj = cvtest(mdl);
testObj.settings.designverifier = 1;
data = cvsim(testObj);
verifSubsys = [mdl '/Verification Subsystem1'];
covProof = getCoverageInfo(data, verifSubsys, cvmetric.Sldv.proof)
```

`covProof` is a two-element vector of the form `[covered_outcomestotal_outcomes]` showing 1 covered outcome out of 1 total proof objective outcome.

Get coverage for a specific Test Objective block in `Verification Subsystem1`

```
mdl = 'sldvdemo_powerwindow_vs';
open_system(mdl)
set_param(mdl, 'StopTime', '10')
testObj = cvtest(mdl);
testObj.settings.designverifier = 1;
data = cvsim(testObj);
verifSubsys = [mdl '/Verification Subsystem1'];
testObjBlock = [verifSubsys '/Test Objective2'];
covTest = getCoverageInfo(data, testObjBlock)
```

`covTest` is a two-element vector of the form `[covered_outcomes total_outcomes]` showing 0 covered outcomes out of 1 total test objective outcome.

Get coverage data and descriptions for all available metrics recorded in `Verification Subsystem1`

```
mdl = 'sldvdemo_powerwindow_vs';
open_system(mdl)
set_param(mdl, 'StopTime', '10')
testObj = cvtest(mdl);
testObj.settings.designverifier = 1;
data = cvsim(testObj);
verifSubsys = [mdl '/Verification Subsystem1'];
[covAll, descrAll] = getCoverageInfo(data, verifSubsys, ...
{cvmetric.Sldv.proof, cvmetric.Sldv.test})
```

`covAll` is a cell array with cells corresponding to the objective outcomes for each metric. `descrAll` is a cell array with cells corresponding to descriptions of each metric.

```
covAll{1}
covAll{2}
```

`covAll{1}` is a two-element vector of the form [covered_outcomes total_outcomes] showing 1 covered outcomes out of 1 total proof objective outcomes. `covAll{2}` is a two-element vector of the form [covered_outcomes total_outcomes] showing 0 covered outcomes out of 1 total test objective outcomes.

```
descrAll{1}
descrAll{2}
```

`descrAll{1}` is a structure array containing descriptions of each proof objective, and descriptions and execution counts for each outcome. `descrAll{2}` is a structure array containing descriptions of each test objective, and descriptions and execution counts for each outcome.

Alternatives

Use the coverage settings to collect and display coverage results for Simulink Design Verifier library blocks:

1. Open the model.
2. In the Model Editor, select **Model Settings** on the **Modeling** tab.
3. On the **Coverage** pane of the Configuration Parameters dialog box, select **Enable coverage analysis**.
4. Under **Coverage metrics**, select **Objectives and constraints**.
5. Click **OK** to close the Configuration Parameters dialog box and save your changes.
6. Simulate the model and review the results.

See Also

`complexityinfo` | `conditioninfo` | `cvsim` | `decisioninfo` | `executioninfo` | `mcdcinfo` | `overflowsaturationinfo` | `relationalboundaryinfo` | `sigrangeinfo` | `sigsizeinfo` | `tableinfo`

Topics

“Simulink Design Verifier Coverage”

Introduced in R2009b

mcdcinfo

Retrieve modified condition/decision coverage information from `cvdata` object

Syntax

```
coverage = mcdcinfo(cvdo, object)
coverage = mcdcinfo(cvdo, object, mode)
coverage = mcdcinfo(cvdo, object, ignore_descendants)
[coverage, description] = mcdcinfo(cvdo, object)
```

Description

`coverage = mcdcinfo(cvdo, object)` returns modified condition/decision coverage (MCDC) results from the `cvdata` object `cvdo` for the model component specified by `object`.

`coverage = mcdcinfo(cvdo, object, mode)` returns modified condition/decision coverage (MCDC) results from the `cvdata` object `cvdo` for the model component specified by `object` for the simulation mode `mode`.

`coverage = mcdcinfo(cvdo, object, ignore_descendants)` returns MCDC results for `object`, depending on the value of `ignore_descendants`.

`[coverage, description] = mcdcinfo(cvdo, object)` returns MCDC results and text descriptions of each condition/decision in `object`.

Input Arguments

`cvdo`

`cvdata` object

`ignore_descendants`

Logical value specifying whether to ignore the coverage of descendant objects
`1` — Ignore coverage of descendant objects
`0` — Collect coverage for descendant objects

`object`

The `object` argument specifies an object in the Simulink model or Stateflow diagram that receives decision coverage. Valid values for `object` include the following:

Object Specification	Description
`BlockPath`	Full path to a model or block
`BlockHandle`	Handle to a model or block
`slObj`	Handle to a Simulink API object
`sfID`	Stateflow ID

Object Specification	Description
`sfObj`	Handle to a Stateflow API object
`{BlockPath, sfID}`	Cell array with the path to a Stateflow chart or atomic subchart and the ID of an object contained in that chart or subchart
`{BlockPath, sfObj}`	Cell array with the path to a Stateflow chart or atomic subchart and a Stateflow object API handle contained in that chart or subchart
`{BlockHandle, sfID}`	Cell array with a handle to a Stateflow chart or atomic subchart and the ID of an object contained in that chart or subchart

When specifying an S-function block, valid values for `object` include the following:

Object Specification	Description
`{BlockPath, fName}`	Cell array with the path to an S-Function block and the name of a source file.
`{BlockHandle, fName}`	Cell array with an S-Function block handle and the name of a source file.
`{BlockPath, fName, funName}`	Cell array with the path to an S-Function block, the name of a source file, and a function name.
`{BlockHandle, fName, funName}`	Cell array with an S-Function block handle, the name of a source file an a function name.

For coverage data collected during Software-in-the-Loop (SIL) mode or Processor-in-the-Loop (PIL) simulation mode, valid values for `object` include the following:

Object Specification	Description
`{fileName, funName}`	Cell array with the name of a source file and a function name.
`{Model, fileName}`	Cell array with a model name (or model handle) and the name of a source file.
`{Model, fileName, funName}`	Cell array with a model name (or model handle), the name of a source file, and a function name.

`mode`

The `mode` argument specifies the simulation mode for coverage. Valid values for `mode` include the following:

Object Specification	Description
`'Normal'`	Model in Normal simulation mode.
`'SIL' (or 'PIL')`	Model in Software-in-the-Loop (SIL) or Processor-in-the-Loop (PIL) simulation mode.
`'ModelRefSIL' (or 'ModelRefPIL')`	Model reference in Software-in-the-Loop (SIL) or Processor-in-the-Loop (PIL) simulation mode.

Object Specification	Description
`'ModelRefTopSIL' (or 'ModelRefTopPIL')`	Model reference in Software-in-the-Loop (SIL or Processor-in-the-Loop (PIL) simulation mode with code interface set to top model.

Output Arguments

`coverage`

Two-element vector of the form `[`*`covered_outcomes total_outcomes`*`]`. `coverage` is empty if `cvdo` does not contain modified condition/decision coverage results for `object`. The two elements are:

`covered_outcomes`	Number of condition/decision outcomes satisfied for `object`
`total_outcomes`	Total number of condition/decision outcomes for `object`

`description`

A structure array containing the following fields:

`text`	Description of the condition/decision measured
`condition`	A structure array containing condition/decision info for individual condition outcomes
`isFiltered`	Whether the block is filtered
`filterRationale`	The filtering rationale
`justifiedCoverage`	The justified coverage conditions
`isJustified`	Whether the block is justified

Examples

Collect MCDC coverage for the `slvnvdemo_cv_small_controller` model and determine the percentage of MCDC coverage collected for the Logic block in the Gain subsystem:

```
mdl = 'slvnvdemo_cv_small_controller';
open_system(mdl)
%Create test specification object
testObj = cvtest(mdl)
%Enable MCDC coverage
testObj.settings.mcdc = 1;
%Simulate model
data = cvsim(testObj)
%Retrieve MCDC results for Logic block
blk_handle = get_param([mdl, '/Gain/Logic'], 'Handle');
cov = mcdcinfo(data, blk_handle)
%Percentage of MCDC outcomes covered
percent_cov = 100 * cov(1) / cov(2)
```

Alternatives

Use the coverage settings to collect MCDC coverage for a model:

1. Open the model.
2. In the Model Editor, select **Model Settings** on the **Modeling** tab.
3. On the **Coverage** pane of the Configuration Parameters dialog box, select **Enable coverage analysis**.
4. Under **Coverage metrics**, select **MCDC** as the structural coverage level.
5. Click **OK** to close the Configuration Parameters dialog box and save your changes.
6. Simulate the model by clicking the **Run** button and review the results.

See Also

`complexityinfo` | `conditioninfo` | `cvsim` | `decisioninfo` | `executioninfo` | `getCoverageInfo` | `overflowsaturationinfo` | `relationalboundaryinfo` | `sigrangeinfo` | `sigsizeinfo` | `tableinfo`

Topics

“Modified Condition/Decision Coverage (MCDC)”
“MCDC Analysis”

Introduced in R2006b

overflowsaturationinfo

Retrieve saturation on integer overflow coverage from `cvdata` object

Syntax

```
coverage = overflowsaturationinfo(covdata, object)
coverage = overflowsaturationinfo(covdata, object, ignore_descendants)
[coverage, description] = overflowsaturationinfo(covdata, object)
```

Description

`coverage = overflowsaturationinfo(covdata, object)` returns saturation on integer overflow coverage results from the `cvdata` object `covdata` for the model object specified by `object` and its descendants.

`coverage = overflowsaturationinfo(covdata, object, ignore_descendants)` returns saturation on integer overflow coverage results from the `cvdata` object `covdata` for the model object specified by `object` and, depending on the value of `ignore_descendants`, descendant objects.

`[coverage, description] = overflowsaturationinfo(covdata, object)` returns saturation on integer overflow coverage results from the `cvdata` object `covdata` for the model object specified by `object`, and textual descriptions of each coverage outcome.

Examples

Extract Saturation on Integer Overflow Data

This example shows how to use `overflowsaturationinfo` to extract saturation on integer overflow data for a `MinMax` block from a `cvdata` object.

Generate coverage data

Open the model and set coverage settings.

```
load_system('slcoverage_fuelsys');
```

Create a `Simulink.SimulationInput` object to change configuration parameters without modifying the model.

```
covSet = Simulink.SimulationInput('slcoverage_fuelsys');
```

Turn on coverage analysis and configure Simulink® to save the coverage data in a separate `cvdata` object in the workspace.

```
covSet = covSet.setModelParameter('CovEnable','on');
covSet = covSet.setModelParameter('CovSaveSingleToWorkspaceVar','on');
covSet = covSet.setModelParameter('CovSaveName','covData');
covSet = covSet.setModelParameter('CovScope','EntireSystem');
```

Enable collection of saturation on integer overflow Coverage and simulate the model by calling `sim` with the `SimulationInput` object as the input.

```
covSet = covSet.setModelParameter('CovMetricSaturateOnIntegerOverflow','on');
simOut = sim(covSet);
```

Extract saturation on integer overflow results

Get the block handle to the MinMax block using `get_param` and then get the saturation on integer overflow results.

```
blockHandle = get_param(['slcoverage_fuelsys/',...
    'Engine Gas Dynamics/Mixing & Combustion/MinMax'],'Handle');
saturationResults = overflowsaturationinfo(covData,blockHandle)
percentSaturationCoverage = 100 * saturationResults(1)/saturationResults(2)
```

```
saturationResults =

     1     2


percentSaturationCoverage =

    50
```

One out of two saturation on integer overflow decision outcomes were satisfied for the `MinMax` block in the `Mixing & Combustion` subsystem, so it received 50% saturation on integer overflow coverage.

Determine Individual Integer Overflow Outcomes

This example shows how to use `overflowsaturationinfo` to determine whether or not integer overflow occurs for a block in a model.

Move the current MATLAB® directory to the location that contains the example files.

```
openExample('slcoverage/DetermineIndividualIntegerOverflowOutcomesExample');
```

Generate coverage data

Load the `slvnvdemo_saturation_on_overflow_coverage` example model.

```
load_system('slvnvdemo_saturation_on_overflow_coverage');
```

Set coverage setting using a `Simulink.SimulationInput` object. Turn coverage on and configure Simulink® to output a `cvdata` object into the workspace.

```
covSet = Simulink.SimulationInput('slvnvdemo_saturation_on_overflow_coverage');
covSet = covSet.setModelParameter('CovEnable','on');
covSet = covSet.setModelParameter('CovSaveSingleToWorkspaceVar','on');
covSet = covSet.setModelParameter('CovSaveName','covData');
covSet = covSet.setModelParameter('CovScope','EntireSystem');
```

Extract saturation on integer overflow results

Retrieve saturation on integer overflow coverage results and description for the `Sum` block in the `Controller` subsystem of the `Test Unit` subsystem.

```
covSet = covSet.setModelParameter('CovMetricSaturateOnIntegerOverflow','on');
simOut = sim(covSet);
[covResults, covDesc] = overflowsaturationinfo(covData, ...
        ['slvnvdemo_saturation_on_overflow_coverage/Test Unit /' ...
        'Controller/Sum'])
percentSaturation = 100 * covResults(1)/covResults(2)
```

```
covResults =

     1     2


covDesc =

  struct with fields:

           isFiltered: 0
          isJustified: 0
    justifiedCoverage: 0
      filterRationale: ''
             decision: [1x1 struct]


percentSaturation =

    50
```

One out of two saturation on integer overflow decision outcomes were satisfied for the Sum block, so it received 50% saturation on integer overflow coverage.

Review the number of times the `Sum` block evaluated to each saturation on integer overflow outcome during simulation.

```
outcome1 = covDesc.decision.outcome(1)
outcome2 = covDesc.decision.outcome(2)
```

```
outcome1 =

  struct with fields:

          execCount: 3
     executionCount: 3
               text: 'false'
         isFiltered: 0
        isJustified: 0
    filterRationale: ''
         executedIn: []


outcome2 =
```

```
  struct with fields:

          execCount: 0
     executionCount: 0
               text: 'true'
         isFiltered: 0
        isJustified: 0
    filterRationale: ''
         executedIn: []
```

During simulation, integer overflow did *not* occur in the `Sum` block because the `'true'` outcome has an execution count of `0`.

If integer overflow is not possible for a block in your model, consider clearing the **Saturate on integer overflow** block parameter to optimize efficiency of your generated code.

Input Arguments

`covdata` — Coverage results data

`cvdata` object

Coverage results data, specified as a `cvdata` object.

`object` — Model or model component

full path | handle

Model or model component, specified as a full path, handle, or array of paths or handles.

Object Specification	Description
`BlockPath`	Full path to a model or block
`BlockHandle`	Handle to a model or block
`slObj`	Handle to a Simulink API object
`sfID`	Stateflow ID
`sfObj`	Handle to a Stateflow API object
`{BlockPath, sfID}`	Cell array with the path to a Stateflow chart or atomic subchart and the ID of an object contained in that chart or subchart
`{BlockPath, sfObj}`	Cell array with the path to a Stateflow chart or atomic subchart and a Stateflow object API handle contained in that chart or subchart
`{BlockHandle, sfID}`	Cell array with a handle to a Stateflow chart or atomic subchart and the ID of an object contained in that chart or subchart

Example: `'slvnvdemo_saturation_on_overflow_coverage'`

Example: `get_param('slvnvdemo_cv_small_controller/Saturation', 'Handle')`

`ignore_descendants` — Preference to ignore coverage of descendant objects

`0` (default) | `1`

Preference to ignore coverage of descendant objects, specified as a logical value.
`1` — Ignore coverage of descendant objects
`0` — Collect coverage for descendant objects

Data Types: `logical`

Output Arguments

`coverage` — Saturation on overflow coverage results for `object`

numerical vector

Saturation on overflow coverage results, stored in a two-element vector of the form `[covered_outcomes total_outcomes]`. The two elements are:

`covered_outcomes`	Number of saturation on integer overflow outcomes satisfied for `object`
`total_outcomes`	Total number of saturation on integer overflow outcomes for `object`

Data Types: `double`

`description` — Textual description of coverage outcomes

structure array

Textual description of coverage outcomes for the model component specified by `object`, returned as a structure array. Depending on the types of model coverage collected, the structure array can have different fields. If only saturation on overflow coverage is collected, the structure array contains the following fields:

`isFiltered`	`0` if the model component specified by `object` is not excluded from coverage recording. `1` if the model component specified by `object` is excluded from coverage recording. For more information about excluding objects from coverage, see “Coverage Filtering”.
`decision.text`	`'Saturate on integer overflow'`
`decision.outcome`	Structure array containing two fields for each coverage outcome:

`executionCount`	Number of times saturation on integer overflow for `object` evaluated to the outcome described by `text`.
`text`	`'true'` or `'false'`

Saturation on integer overflow has two possible outcomes, `'true'` and `'false'`.

`decision.isFiltered`	`0` if the model component specified by `object` is not excluded from coverage recording. `1` if the model component specified by `object` is excluded from coverage recording. For more information about excluding objects from coverage, see “Coverage Filtering”.
`decision.filterRationale`	Rationale for filtering the model component specified by `object`, if `object` is excluded from coverage and a rationale is specified. For more information about excluding objects from coverage, see “Coverage Filtering”.

Data Types: `struct`

See Also

`complexityinfo` | `conditioninfo` | `cvsim` | `cvtest` | `decisioninfo` | `executioninfo` | `getCoverageInfo` | `mcdcinfo` | `relationalboundaryinfo` | `sigrangeinfo` | `sigsizeinfo` | `tableinfo`

Topics

“Command Line Verification Tutorial”
“Saturate on Integer Overflow Coverage”

Introduced in R2013a

relationalboundaryinfo

Retrieve relational boundary coverage from `cvdata` object

Syntax

```
coverage = relationalboundaryinfo(covdata, object)
coverage = relationalboundaryinfo(covdata, object,mode)
coverage = relationalboundaryinfo(covdata, object, ignore_descendants)
[coverage, description] = relationalboundaryinfo(covdata, object)
```

Description

`coverage = relationalboundaryinfo(covdata, object)` returns relational boundary coverage results from the `cvdata` object `covdata` for the model object specified by `object` and its descendants.

`coverage = relationalboundaryinfo(covdata, object,mode)` returns relational boundary coverage results from the `cvdata` object `covdata` for the model object specified by `object` and its descendants for the simulation mode `mode`.

`coverage = relationalboundaryinfo(covdata, object, ignore_descendants)` returns relational boundary coverage results from the `cvdata` object `covdata` for the model object specified by `object` and, depending on the value of `ignore_descendants`, descendant objects.

`[coverage, description] = relationalboundaryinfo(covdata, object)` returns relational boundary coverage results from the `cvdata` object `covdata` for the model object specified by `object`, and textual descriptions of each coverage outcome.

Examples

Collect Relational Boundary Coverage for Supported Block in Model

This example shows how to collect relational boundary coverage information for a Saturation block in a model. For more information on blocks supported for relational boundary coverage, see “Model Objects That Receive Coverage”.

Open the `slvnvdemo_cv_small_controller` model. Create a model coverage test specification object for the model.

```
open_system('slvnvdemo_cv_small_controller');
testObj = cvtest('slvnvdemo_cv_small_controller');
```

In the model coverage test specification object, activate relational boundary coverage.

```
testObj.settings.relationalop = 1;
```

Simulate the model and collect coverage results in a `cvdata` object.

```
dataObj = cvsim(testObj);
```

Obtain relational boundary coverage results for the Saturation block in `slvnvdemo_cv_small_controller`. The coverage results are stored in a two-element vector of the form `[covered_outcomes total_outcomes]`.

```
blockHandle = get_param('slvnvdemo_cv_small_controller/Saturation','Handle');;
[covResults, covDesc] = relationalboundaryinfo(dataObj, blockHandle)
```

```
covResults =

     2     4


covDesc =

    isFiltered: 0
      decision: [1x2 struct]
```

The field `decision` is a `1 X 2` structure. Each element of `decision` corresponds to a relational operation in the block. The Saturation block contains two comparisons. The first comparison is with a lower limit and the second with an upper limit. Therefore, `decision` is a 2-element structure.

View the first operation in the block that receives relational boundary coverage. For the Saturation block, the first relational operation is `input > lowerlimit`.

```
covDesc.decision(1)
```

```
ans =

            outcome: [1x2 struct]
               text: 'input - lowerlimit'
         isFiltered: 0
    filterRationale: ''
```

The `text` field shows the two operands. The `isFiltered` field is set to 1 if the block is filtered from relational boundary coverage. For more information, see “Coverage Filtering”.

View results for the first relational operation in the block.

```
for(i=1:2)
  covDesc.decision(1).outcome(i)
end
```

```
ans =

     isActive: 1
    execCount: 0
         text: '[-tol..0]'


ans =

     isActive: 1
    execCount: 0
         text: '(0..tol]'
```

View the second operation in the block that receives relational boundary coverage. For the Saturation block, the second relational operation is `input < upperlimit`.

```
covDesc.decision(2)
```

```
ans =

            outcome: [1x2 struct]
               text: 'input - upperlimit'
         isFiltered: 0
    filterRationale: ''
```

View results for the second relational operation in the block.

```
for(i=1:2)
  covDesc.decision(2).outcome(i)
end
```

```
ans =

     isActive: 1
    execCount: 1
         text: '[-tol..0)'


ans =

     isActive: 1
    execCount: 2
         text: '[0..tol]'
```

Input Arguments

`covdata` — Coverage results data

`cvdata` object

Coverage results data, specified as a `cvdata` object.

`object` — Model or model component

full path | handle

Model or model component, specified as a full path, handle, or array of paths or handles.

Object Specification	Description
`BlockPath`	Full path to a model or block
`BlockHandle`	Handle to a model or block
`slObj`	Handle to a Simulink API object
`sfID`	Stateflow ID
`sfObj`	Handle to a Stateflow API object
`{BlockPath, sfID}`	Cell array with the path to a Stateflow chart or atomic subchart and the ID of an object contained in that chart or subchart
`{BlockPath, sfObj}`	Cell array with the path to a Stateflow chart or atomic subchart and a Stateflow object API handle contained in that chart or subchart

Object Specification	Description
`{BlockHandle, sfID}`	Cell array with a handle to a Stateflow chart or atomic subchart and the ID of an object contained in that chart or subchart

When specifying an S-function block, valid values for `object` include the following:

Object Specification	Description
`{BlockPath, fName}`	Cell array with the path to an S-Function block and the name of a source file.
`{BlockHandle, fName}`	Cell array with an S-Function block handle and the name of a source file.
`{BlockPath, fName, funName}`	Cell array with the path to an S-Function block, the name of a source file, and a function name.
`{BlockHandle, fName, funName}`	Cell array with an S-Function block handle, the name of a source file an a function name.

For coverage data collected during Software-in-the-Loop (SIL) mode or Processor-in-the-Loop (PIL) simulation mode, valid values for `object` include the following:

Object Specification	Description
`{fileName, funName}`	Cell array with the name of a source file and a function name.
`{Model, fileName}`	Cell array with a model name (or model handle) and the name of a source file.
`{Model, fileName, funName}`	Cell array with a model name (or model handle), the name of a source file, and a function name.

Example: `get_param('slvnvdemo_cv_small_controller/Saturation', 'Handle')`

`mode` — The `mode` argument specifies the simulation mode for coverage
character vector or string

Valid values for `mode` include the following:

Object Specification	Description
`'Normal'`	Model in Normal simulation mode.
`'SIL' (or 'PIL')`	Model in Software-in-the-Loop (SIL) or Processor-in-the-Loop (PIL) simulation mode.
`'ModelRefSIL' (or 'ModelRefPIL')`	Model reference in Software-in-the-Loop (SIL) or Processor-in-the-Loop (PIL) simulation mode.
`'ModelRefTopSIL' (or 'ModelRefTopPIL')`	Model reference in Software-in-the-Loop (SIL or Processor-in-the-Loop (PIL) simulation mode with code interface set to top model.

`ignore_descendants` — Preference to ignore coverage of descendant objects
`0` (default) | `1`

Preference to ignore coverage of descendant objects, specified as a logical value.

`1` — Ignore coverage of descendant objects
`0` — Collect coverage for descendant objects

Data Types: `logical`

Output Arguments

`coverage` — Relational boundary coverage results for `object`

numerical vector

Relational boundary coverage results, stored in a two-element vector of the form `[covered_outcomes total_outcomes]`. The two elements are:

`covered_outcomes`	Number of relational boundary outcomes satisfied for `object`
`total_outcomes`	Total number of relational boundary outcomes for `object`

Data Types: `double`

`description` — Textual description of coverage outcomes

structure array

Textual description of coverage outcomes for the model component specified by `object`, returned as a structure array. Depending on the types of model coverage collected, the structure array can have different fields. If only relational boundary coverage is collected, the structure array contains the following fields:

`isFiltered`	`0` if the model component specified by `object` is not excluded from coverage recording. `1` if the model component specified by `object` is excluded from coverage recording. For more information about excluding objects from coverage, see “Coverage Filtering”.
`decision.text`	Character vector or string of the form: *op_1*`-`*op_2* • *op_1* is the left operand in the relational operation. • *op_2* is the right operand in the relational operation.

`decision.outcome`	Structure array containing two fields for each coverage outcome: `isActive` — Boolean variable. If this variable is `false`, it indicates that decisions were not evaluated during simulation due to variable signal size. `execCount` — Number of times *op_1*-*op_2* fell in the range described by `text` `text` — The range around the relational boundary considered for coverage. For more information, see "Relational Boundary".
`decision.isFiltered`	`0` if the model component specified by `object` is not excluded from coverage recording. `1` if the model component specified by `object` is excluded from coverage recording. For more information about excluding objects from coverage, see "Coverage Filtering".
`decision.filterRationale`	Rationale for filtering the model component specified by `object`, if `object` is excluded from coverage and a rationale is specified. For more information about excluding objects from coverage, see "Coverage Filtering".

Data Types: `struct`

See Also

`complexityinfo` | `conditioninfo` | `cvsim` | `cvtest` | `decisioninfo` | `executioninfo` | `getCoverageInfo` | `mcdcinfo` | `overflowsaturationinfo` | `relationalboundaryinfo` | `sigrangeinfo` | `sigsizeinfo` | `tableinfo`

Topics

"Command Line Verification Tutorial"
"Relational Boundary Coverage"

Introduced in R2014b

sigrangeinfo

Retrieve signal range coverage information from `cvdata` object

Syntax

```
[min, max] = sigrangeinfo(cvdo, object)
[min, max] = sigrangeinfo(cvdo, object, portID)
```

Description

`[min, max] = sigrangeinfo(cvdo, object)` returns the minimum and maximum signal values output by the model component `object` within the `cvdata` object `cvdo`.

`[min, max] = sigrangeinfo(cvdo, object, portID)` returns the minimum and maximum signal values associated with the output port `portID` of the Simulink block `object`.

Input Arguments

`cvdo`

`cvdata` object

`object`

An object in the model or Stateflow chart that receives signal range coverage. Valid values for `object` include the following:

Object Specification	Description
`BlockPath`	Full path to a model or block
`BlockHandle`	Handle to a model or block
`slObj`	Handle to a Simulink API object
`sfID`	Stateflow ID
`sfObj`	Handle to a Stateflow API object
`{BlockPath, sfID}`	Cell array with the path to a Stateflow chart or atomic subchart and the ID of an object contained in that chart or subchart
`{BlockPath, sfObj}`	Cell array with the path to a Stateflow chart or atomic subchart and a Stateflow object API handle contained in that chart or subchart
`{BlockHandle, sfID}`	Cell array with a handle to a Stateflow chart or atomic subchart and the ID of an object contained in that chart or subchart

`portID`

Output port of the block `object`

Output Arguments

`max`

Maximum signal value output by the model component `object` within the `cvdata` object, `cvdo`. If `object` outputs a vector, `min` and `max` are also vectors.

`min`

Minimum signal value output by the model component `object` within the `cvdata` object, `cvdo`. If `object` outputs a vector, `min` and `max` are also vectors.

Examples

Collect Signal Range Data for a Block

This example shows how to extract signal range info from a coverage data object.

Load the model and set up coverage options

Load the model into memory. This example uses a small controller model.

```
modelName = 'slvnvdemo_cv_small_controller';
load_system(modelName)
```

Declare coverage settings using a structure of parameter names and values. For a complete list of coverage parameters and their possible values, see “Coverage Settings” on page 2-2.

```
covOpts.CovEnable = 'on';
covOpts.CovSaveSingleToWorkspaceVar = 'on';
covOpts.CovSaveName = 'covData';
covOpts.CovMetricSignalRange = 'on';
```

Simulate the model using `sim` with the model name and the parameter structure as inputs.

```
simOut = sim(modelName,covOpts);
```

Get signal range data

Get the block handle of the Product block using `get_param`.

```
bHandle = get_param([modelName,'/Product'],'Handle');
```

Get the signal range data by calling `sigrangeinfo` with the `cvdata` object and the block handle as inputs.

```
[minVal, maxVal] = sigrangeinfo(covData,bHandle)
```

```
minVal =

     0


maxVal =
```

```
25.0000
```

Alternatives

Use the coverage settings to collect signal range coverage for a model:

1. Open the model for which you want to collect signal range coverage.
2. In the Model Editor, select **Model Settings** on the **Modeling** tab.
3. On the **Coverage** pane of the Configuration Parameters dialog box, select **Enable coverage analysis**.
4. Under **Coverage metrics**, select **Signal Range**.
5. Click **OK** to close the Configuration Parameters dialog box and save your changes.
6. Simulate the model by clicking the **Run** button and review the results.

See Also

`complexityinfo` | `conditioninfo` | `cvsim` | `decisioninfo` | `executioninfo` | `getCoverageInfo` | `mcdcinfo` | `overflowsaturationinfo` | `relationalboundaryinfo` | `sigsizeinfo` | `tableinfo`

Introduced in R2006b

sigsizeinfo

Retrieve signal size coverage information from `cvdata` object

Syntax

```
[min, max, allocated] = sigsizeinfo(data, object)
[min, max, allocated] = sigsizeinfo(data, object, portID)
```

Description

`[min, max, allocated] = sigsizeinfo(data, object)` returns the minimum, maximum, and allocated signal sizes for the outputs of model component `object` within the coverage data object `data`, if `object` supports variable size signals.

`[min, max, allocated] = sigsizeinfo(data, object, portID)` returns the minimum and maximum signal sizes associated with the output port `portID` of the model component `object`.

Input Arguments

`data`

`cvdata` object

`object`

An object in the model or Stateflow chart that receives signal size coverage. Valid values for `object` include the following:

Object Specification	Description
`BlockPath`	Full path to a Simulink model or block
`BlockHandle`	Handle to a Simulink model or block
`slObj`	Handle to a Simulink API object
`sfID`	Stateflow ID
`sfObj`	Handle to a Stateflow API object
`{BlockPath, sfID}`	Cell array with the path to a Stateflow chart or atomic subchart and the ID of an object contained in that chart or subchart
`{BlockPath, sfObj}`	Cell array with the path to a Stateflow chart or atomic subchart and a Stateflow object API handle contained in that chart or subchart
`{BlockHandle, sfID}`	Cell array with a handle to a Stateflow chart or atomic subchart and the ID of an object contained in that chart or subchart

`portID`

Output port number of the model component `object`

Output Arguments

`max`

Maximum signal size output by the model component `object` within the `cvdata` object `data`. If `object` has multiple outputs, `max` is a vector.

`min`

Minimum signal size output by the model component `object` within the `cvdata` object `data`. If `object` has multiple outputs, `min` is a vector.

`allocated`

Allocated signal size output by the model component `object` within the `cvdata` object `data`. If `object` has multiple outputs, `allocated` is a vector.

Examples

Collect signal size coverage data for the Switch block in the `sldemo_varsize_basic` model:

```
mdl = 'sldemo_varsize_basic';
open_system(mdl);
%Create test spec object
testObj = cvtest(mdl);
%Enable signal size coverage
testObj.settings.sigsize=1;
%Simulate the model
data = cvsim(testObj);
%Set the block handle
blk_handle = get_param([mdl, '/Switch'], 'Handle');
%Get signal size data
[minVal, maxVal, allocVal] = sigsizeinfo(data, blk_handle);
```

Alternatives

Use the coverage settings to collect signal size coverage for a model:

1. Open the model for which you want to collect signal size coverage.
2. In the Simulink Editor, select **Model Settings** on the **Modeling** tab.
3. On the **Coverage** pane of the Configuration Parameters dialog box, select **Enable coverage analysis**.
4. Under **Coverage metrics**, select **Signal Size**.
5. Click **OK** to close the Configuration Parameters dialog box and save your changes.
6. Simulate the model by clicking the **Run** button and review the results.

See Also

`complexityinfo` | `conditioninfo` | `cvsim` | `decisioninfo` | `executioninfo` | `getCoverageInfo` | `mcdcinfo` | `overflowsaturationinfo` | `relationalboundaryinfo` | `sigrangeinfo` | `tableinfo`

Introduced in R2010b

slvnvextract

Extract subsystem or subchart contents into new model

Syntax

```
newModel = slvnvextract(subcomponent)
newModel = slvnvextract(subcomponent,showModel)
```

Description

`newModel = slvnvextract(subcomponent)` extracts the contents of the Atomic Subsystem block or atomic subchart `subcomponent` and creates a model. `slvnvextract` returns the name of the new model in `newModel`. If the model name already exists,`slvnvextract` uses the subsystem or subchart name for the model name, appending a numeral to the model name.

Note If an atomic subchart calls an exported graphical function that is outside the subchart, `slvnvextract` creates the model, but the new model does not compile.

`newModel = slvnvextract(subcomponent,showModel)` opens the extracted model if you set `showModel` to `true`. The extracted model is loaded only if you set `showModel` to `false`.

Input Arguments

`subcomponent` — Subsystem or subchart whose contents are extracted

character vector or string

The full path to the atomic subsystem or atomic subchart whose contents are extracted.

`showModel` — Display extracted model

`true` (default) | `false`

Specify if you want the extracted model to be displayed.

Output Arguments

`newModel` — The name of the new extracted model

character vector or string

Reports the name of the new extracted model created by `slvnvextract`.

Examples

Extract Subsystem and Copy to a New Model

Extract the Atomic Subsystem block, Bus Counter, from the `sldemo_mdlref_conversion` model and copy it into a new model:

```
open_system('sldemo_mdlref_conversion');
newmodel = slvnvextract('sldemo_mdlref_conversion/Bus Counter', true);
```

Extract Subchart and Copy to a New Model

Extract the Atomic Subchart block, Sensor1, from the `sf_atomic_sensor_pair` model and copy it into a new model:

```
open_system('sf_atomic_sensor_pair');
newmodel = slvnvextract('sf_atomic_sensor_pair/RedundantSensors/Sensor1', true);
```

Introduced in R2010b

slvnvharnessopts

Generate default options for `slvnvmakeharness`

Syntax

```
harnessopts = slvnvharnessopts
```

Description

`harnessopts = slvnvharnessopts` generates the default configuration for running `slvnvmakeharness`.

Output Arguments

`harnessopts` — Default harness options

structure

Default harness options, returned as a structure. The `harnessopts` structure has the following fields. If you do not specify any values, default values are used.

Field	Description
`harnessFilePath`	Specifies the file path for creating the harness model. If an invalid path is specified, `slvnvmakeharness` does not save the harness model, but it creates and opens the harness model. If you do not specify this option, `slvnvmakeharness` generates a new harness model and saves it in the MATLAB current folder. Default: `''`
`modelRefHarness`	Generates the test harness model that includes `model` in a Model block. When `false`, the test harness model includes a copy of `model`. Default: `true`
`usedSignalsOnly`	When `true`, the Signal Builder block in the harness model has signals only for input signals in the model. You must have the Simulink Design Verifier software and `model` must be compatible with that software to detect the input signals. Default: `false`

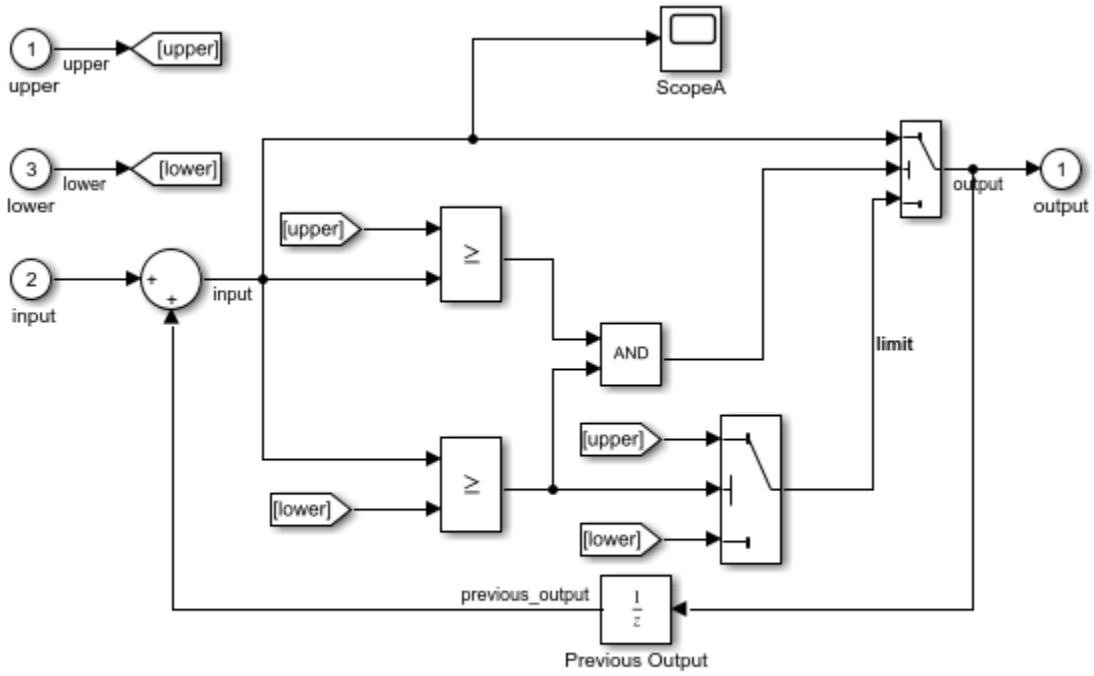
Examples

Create a Coverage Harness with Default Options

This example shows how to set harness options for a coverage harness.

Open the model using `open_system`.

```
open_system('slvnvdemo_counter')
```



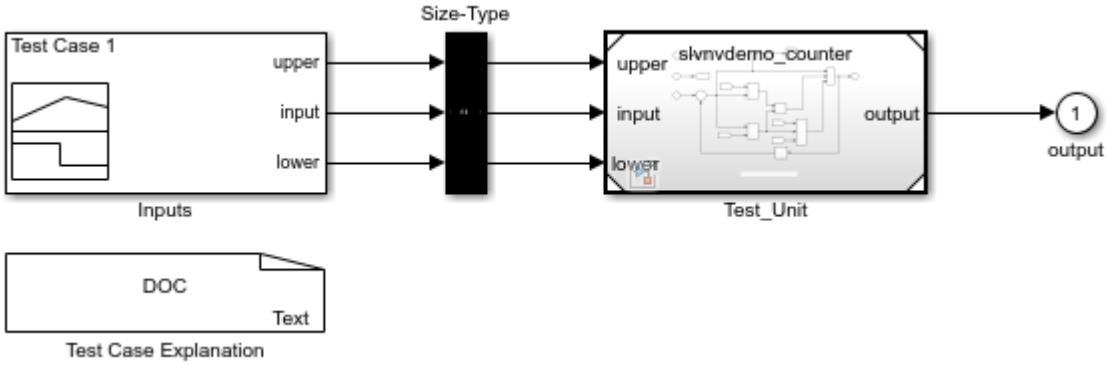
Create the harness options object using `slvnvharnessopts`.

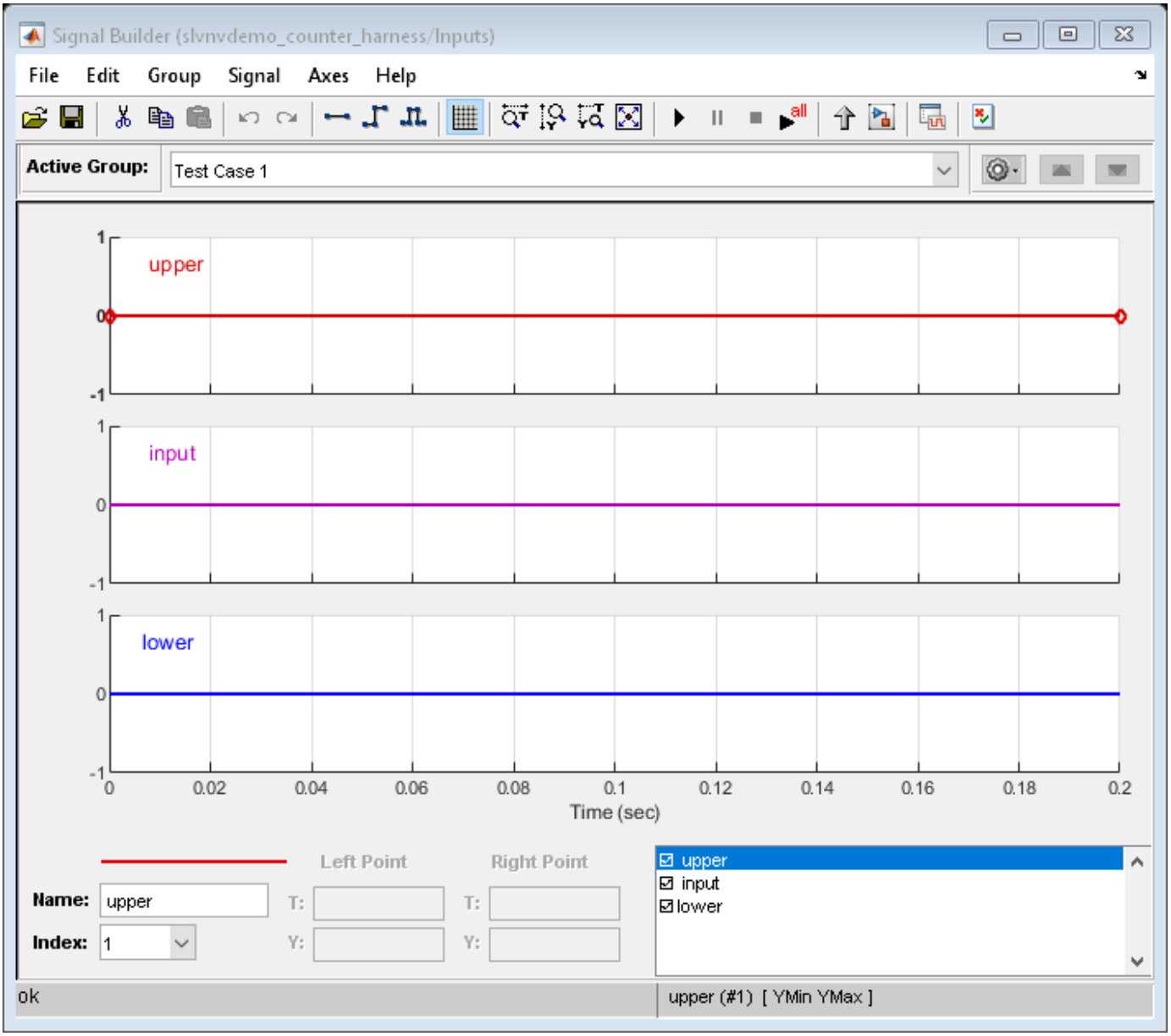
```
harnessOpts = slvnvharnessopts
```

```
harnessOpts =

  struct with fields:

    harnessFilePath: ''
    modelRefHarness: 1
    usedSignalsOnly: 0
      harnessSource: 'Signal Builder'
```

Use the `harnessOpts` to create the harness file using `slvnvmakeharness`.

```
harnessFile = slvnvmakeharness('slvnvdemo_counter', ...
              '', harnessOpts);
```


Size-Type
Test Case 1
upper
input
lower
Inputs
upper
input
lower
slvnvdemo_counter
output
Test_Unit
1
output
DOC
Text
Test Case Explanation



See Also

`slvnvmakeharness`

Introduced in R2010b

slvnvlogsignals

Log test data for component or model during simulation

Syntax

```
data = slvnvlogsignals(model_block)
data = slvnvlogsignals(harness_model)
data = slvnvlogsignals(harness_model, test_case_index)
```

Description

`data = slvnvlogsignals(model_block)` simulates the model that contains `model_block` and logs the input signals to the `model_block` block. `model_block` must be a Simulink Model block.

`data = slvnvlogsignals(harness_model)` simulates every test case in `harness_model` and logs the input signals to the Test Unit block in the harness model. Generate `harness_model` by using the Simulink Design Verifier analysis, `sldvmakeharness`, or `slvnvmakeharness`.

`data = slvnvlogsignals(harness_model, test_case_index)` simulates every test case in the Signal Builder block of the `harness_model` specified by `test_case_index`. `slvnvlogsignals` logs the input signals to the Test Unit block in the harness model. If you omit `test_case_index`, `slvnvlogsignals` simulates every test case in the Signal Builder.

Input Arguments

`model_block` — Component or model

character vector or string | handle

The full block path name or handle to a Simulink Model block, specified as a character vector or string.

`harness_model` — Harness name

character vector or string | handle

Name or handle to a harness model that the Simulink Design Verifier software, `sldvmakeharness`, or `slvnvmakeharness` creates, specified as a character vector or string.

`test_case_index` — Indices of test cases to be simulated

Integer array

Array of integers that specifies which test cases in the Signal Builder block of the harness model to simulate.

Output Arguments

`data` — Output data

structure

Structure that contains the logged data.

Examples

Log and Visualize Simulation Data

Log simulation data for a Model block. Use the logged data to create a harness model and visualize the data in the referenced model.

```
% Simulate the CounterB Model block, which references the
% sldemo_mdlref_counter model, in the context of the
% sldemo_mdlref_basic model and log the data:
open_system('sldemo_mdlref_basic');
data = slvnvlogsignals('sldemo_mdlref_basic/CounterB');

% Create a harness model for sldemo_mdlref_counter using the
% logged data and the default harness options:
load_system('sldemo_mdlref_counter');
harnessOpts = slvnvharnessopts
[harnessFilePath]  = ...
    slvnvmakeharness('sldemo_mdlref_counter', data, ...
    harnessOpts);
```

See Also

`sldvmakeharness` | `slvnvmakeharness` | `slvnvruncgvtest` | `slvnvruntest`

Introduced in R2010b

slvnvmakeharness

Generate Simulink Coverage harness model

Syntax

```
harnessFilePath = slvnvmakeharness(model)
harnessFilePath = slvnvmakeharness(model, dataFile)
harnessFilePath = slvnvmakeharness(model, dataFile, harnessOpts)
```

Description

`harnessFilePath = slvnvmakeharness(model)` generates a test harness from `model`, which is a handle to a Simulink model or a character vector or string with the model name. `slvnvmakeharness` returns the path and file name of the generated harness model in `harnessFilePath`. `slvnvmakeharness` creates a harness model containing the Model block, a Signal Builder block, and a size-type conversion block, by default. The test harness includes one default test case that specifies the default values for all input signals.

`harnessFilePath = slvnvmakeharness(model, dataFile)` generates a test harness from the data file `dataFile`.

`harnessFilePath = slvnvmakeharness(model, dataFile, harnessOpts)` generates a test harness from `model` by using the `dataFile` and `harnessOpts`, which specifies the harness creation options. Requires `''` for `dataFile` if `dataFile` is not available. The default `dataFile` argument creates a test harness with a single test case with default values for the inputs.

Input Arguments

`model` — Simulink model

character vector or string | handle

Simulink model or the model name.

`dataFile` — Structure created by slvnvlogsignals or slvnvmergedata

`''` (default) | structure

Contains information about the model, its input and output ports, and any preexisting test signals. This argument can be either the structure itself or the name of the `.mat` file containing this structure. Use this parameter when you have previously logged test data that you want to import into a new test harness.

`harnessOpts` — Configuration for slvnvmakeharness

structure

A structure whose fields specify the configuration for `slvnvmakeharness`.

Field	Description
`harnessFilePath`	Specifies the file path for creating the harness model. If an invalid path is specified, `slvnvmakeharness` does not save the harness model, but it creates and opens the harness model. If you do not specify this option, the `slvnvoptions` object is used. Also, `slvnvmakeharness` generates a new harness model and saves it in the MATLAB current folder. Default: `''`
`modelRefHarness`	Generates the test harness model that includes `model` in a Model block. When `false`, the test harness model includes a copy of `model`. Default: `true`
`usedSignalsOnly`	When `true`, the Signal Builder block in the harness model has signals for input signals in the model. You must have the Simulink Design Verifier software and `model` must be compatible with that software to detect the input signals. Default: `false`

Note To create a default `harnessOpts` object, at the MATLAB command prompt, type:

```
slvnvharnessopts
```

Output Arguments

`harnessFilePath` — Generated harness model

Character vector or string

The path and file name of the generated harness model.

Examples

Create a Test Harness Using the Default Options

Create a test harness for the `sldemo_mdlref_house` model using the default options:

```
open_system('sldemo_mdlref_house');
harnessOpts = slvnvharnessopts;
[harnessfile] = slvnvmakeharness('sldemo_mdlref_house', '', harnessOpts);
```

See Also

`slvnvharnessopts` | `slvnvmergeharness`

Introduced in R2010b

slvnvmergedata

Combine test data from data files

Syntax

```
merged_data = slvnvmergedata(data1,data2,...)
```

Description

`merged_data = slvnvmergedata(data1,data2,...)` combines two or more test cases and counterexamples `data` into a single test case data structure `merged_data`.

Input Arguments

`data` — Structure that contains test case or counterexample data

structure

Generated by running `slvnvlogsignals` or by running a Simulink Design Verifier analysis.

Output Arguments

`merged_data` — The merged test cases or counterexamples

structure

Structure that contains the merged test cases or counterexamples.

Examples

Log Signals and Merge Logged Data

```
% Open the sldemo_mdlref_basic model, which contains three Model blocks
% that reference the sldemo_mdlref_counter model:
sldemo_mdlref_basic;

% Log the input signals to the three Model blocks:
data1 = slvnvlogsignals('sldemo_mdlref_basic/CounterA');
data2 = slvnvlogsignals('sldemo_mdlref_basic/CounterB');
data3 = slvnvlogsignals('sldemo_mdlref_basic/CounterC');

% Merge the logged data:
merged_data = slvnvmergedata(data1, data2, data3);

% Simulate the referenced model, sldemo_mdlref_counter,for coverage with
% the merged data and display the coverage results in an HTML file.
open_system('sldemo_mdlref_counter');
runOpts = slvnvruntestopts;
runOpts.coverageEnabled = true;
[ outData, initialCov ] = slvnvruntest('sldemo_mdlref_counter', ...
     merged_data, runOpts);
cvhtml('Initial coverage', initialCov);
```

See Also

`sldvrun` | `slvnvlogsignals` | `slvnvmakeharness` | `slvnvruncgvtest` | `slvnvruntest`

Introduced in R2011a

slvnvmergeharness

Combine test data from harness models

Syntax

```
status = slvnvmergeharness(name, models, initialization_commands)
initialization_commands
slvnvmergeharness
```

Description

`status = slvnvmergeharness(name, models, initialization_commands)` collects the test data and initialization commands from each test harness model and saves them in a handle to the new model.

`initialization_commands` is a cell array of character vectors or strings that are the same length as `models`. It defines parameter settings for the test cases of each test harness model.

`slvnvmergeharness` assumes that `name` and the rest of the models in `models` have only one Signal Builder block on the top level. If a model in `models` does not meet this restriction or its top-level Signal Builder block does not have the same number of signals as the top-level Signal Builder block in `name`, `slvnvmergeharness` does not merge that model's test data into `name`.

Input Arguments

`name` — Name of the new harness model, to be stored in the default MATLAB folder
character vector or string

If `name` does not exist, `slvnvmergeharness` creates it as a copy of the first model in `models`. `slvnvmergeharness` then merges data from other models listed in `models` into this model. If you create `name` from a previous `slvnvmergeharness` run, subsequent runs of `slvnvmergeharness` for `name` maintain the structure and initialization from the earlier run. If `name` matches an existing Simulink model, `slvnvmergeharness` merges the test data from `models` into `name`.

`models` — Harness model names
cell array of character vectors or strings

Names of harness models that are inputs to `slvnvmergeharness`.

`initialization_commands` — Parameter settings for the test cases of each test harness model
cell array of character vectors or strings

Cell array of character vectors or strings that is the same length as `models`.

Output Arguments

`status` — Status of data and initialization commands getting saved
1 | 0

`slvnvmergeharness` returns a `status` of `1` if the data and initialization commands are saved in `name`. Otherwise, it returns `0`.

Examples

Log Signals and Merge Test Harnesses

```
% Log the input signals to the three Model blocks in the sldemo_mdlref_basic example model
% that each reference the same model:
open_system('sldemo_mdlref_basic');
data1 = slvnvlogsignals('sldemo_mdlref_basic/CounterA');
data2 = slvnvlogsignals('sldemo_mdlref_basic/CounterB');
data3 = slvnvlogsignals('sldemo_mdlref_basic/CounterC');
open_system('sldemo_mdlref_counter');

% Make three test harnesses using the logged signals:
harness1FilePath = slvnvmakeharness('sldemo_mdlref_counter', data1);
harness2FilePath = slvnvmakeharness('sldemo_mdlref_counter', data2);
harness3FilePath = slvnvmakeharness('sldemo_mdlref_counter', data3)
[~, harness1] = fileparts(harness1FilePath);
[~, harness2] = fileparts(harness2FilePath);
[~, harness3] = fileparts(harness3FilePath);

% Merge the three test harnesses:
slvnvmergeharness('new_harness_model',{harness1, harness2, harness3});
```

See Also

`slvnvlogsignals` | `slvnvmakeharness`

Introduced in R2010b

slvnvruncgvtest

Invoke Code Generation Verification (CGV) API and execute model

Syntax

```
cgvObject = slvnvruncgvtest(model, dataFile)
cgvObject = slvnvruncgvtest(model, dataFile, runOpts)
```

Description

`cgvObject = slvnvruncgvtest(model, dataFile)` invokes the Code Generation Verification (CGV) API methods and executes the `model` by using all test cases in `dataFile`. `cgvObject` is a `cgv.CGV` object that `slvnvruncgvtest` creates during the execution of the `model`. `slvnvruncgvtest` sets the execution mode for `cgvObject` to `'sim'` by default.

`cgvObject = slvnvruncgvtest(model, dataFile, runOpts)` invokes CGV API methods and executes the `model` by using test cases in `dataFile`. `runOpts` defines the options for executing the test cases. The settings in `runOpts` determine the configuration of `cgvObject`.

Input Arguments

`model` — Model to execute

character vector or string

Name of the Simulink model that you execute.

`dataFile` — Input data

structure | character vector or string

Name of the data file or a structure that contains the input data. Generate data by either:

- Using the Simulink Design Verifier software to analyze the model.
- Using the `slvnvlogsignals` function.

`runOpts` — Specify the configuration of slvnvruncgvtest

structure

The fields of `runOpts` specify the configuration of `slvnvruncgvtest` .

Field Name	Description
`testIdx`	Test case index array to simulate from `dataFile`. If `testIdx = []` (the default), `slvnvruncgvtest` simulates all test cases.

Field Name	Description
`allowCopyModel`	If you have not configured your model to execute test cases with the CGV API, this field specifies creating and configuring the model. If `true` and you have not configured your `model` to execute test cases with the CGV API, `slvnvruncgvtest` copies the model, fixes the configuration, and executes the test cases on the copied model. If `false` (the default), an error occurs if the tests cannot execute with the CGV API. **Note** If you have not configured the top-level model or any referenced models to execute test cases, `slvnvruncgvtest` does not copy the model, even if `allowCopyModel` is `true`. An error occurs.
`cgvCompType`	Defines the software-in-the-loop (SIL) or processor-in-the-loop (PIL) approach for CGV: • `'topmodel'` (default) • `'modelblock'`
`cgvConn`	Specifies mode of execution for CGV: • `'sim'` (default) • `'sil'` • `'pil'`

Note `runOpts = slvnvruntestopts('cgv')` returns a `runOpts` structure with the default values for each field.

Output Arguments

`cgvObject` — Object created by `slvnvruncgv` test during the execution of `model`

`cgv.CGV` object

`cgv.CGV` object that `slvnvruncgvtest` creates during the execution of `model`.

`slvnvruncgvtest` saves the following data for each test case executed in an array of `Simulink.SimulationOutput` objects inside `cgvObject`.

Field	Description
`tout_slvnvruncgvtest`	Simulation time
`xout_slvnvruncgvtest`	State data
`yout_slvnvruncgvtest`	Output signal data

Field	Description
`logsout_slvnvruncgvtest`	Signal logging data for: • Signals connected to outports • Signals that are configured for logging data on the model

Examples

Log Signals, Run Tests, and Compare Results by Using the CGV API

```
% Open the sldemo_mdlref_basic example model and log the input signals to the CounterA Model block:
open_system('sldemo_mdlref_basic');
load_system('sldemo_mdlref_counter');
loggedData = slvnvlogsignals('sldemo_mdlref_basic/CounterA');

% Create the default configuration object for slvnvruncgvtest,and allow the model to be configured to
% execute test cases with the CGV API:
runOpts = slvnvruntestopts('cgv');
runOpts.allowCopyModel = true;

% Using the logged signals, execute slvnvruncgvtest — first in simulation mode, and then in
% Software-in-the-Loop (SIL) mode — to invoke the CGV API and execute the specified test
% cases on the generated code for the model:
cgvObjectSim = slvnvruncgvtest('sldemo_mdlref_counter', loggedData, runOpts);
runOpts.cgvConn = 'sil';
cgvObjectSil = slvnvruncgvtest('sldemo_mdlref_counter', loggedData, runOpts);

% Use the CGV API to compare the results of the first test case:
simout = cgvObjectSim.getOutputData(1);
silout = cgvObjectSil.getOutputData(1);
[matchNames, ~, mismatchNames, ~ ] = cgv.CGV.compare(simout, silout);
fprintf('\nTest Case:  %d Signals match, %d Signals mismatch', ...
        length(matchNames), length(mismatchNames));
```

Tips

To run `slvnvruncgvtest`, you must have the Embedded Coder® software.

If your model has parameters that are not configured for executing test cases with the CGV API, `slvnvruncgvtest` reports warnings about the invalid parameters. If you see these warnings, do one of the following:

- Modify the invalid parameters and rerun `slvnvruncgvtest`.
- Set `allowCopyModel` in `runOpts` to be `true` and rerun `slvnvruncgvtest`. `slvnvruncgvtest` makes a copy of your model configured for executing test cases, and invokes the CGV API.

See Also

`cgv.CGV` | `slvnvlogsignals` | `slvnvruntest` | `slvnvruntestopts`

Introduced in R2010b

slvnvruntest

Simulate model by using input data

Syntax

```
outData = slvnvruntest(model, dataFile)
outData = slvnvruntest(model, dataFile, runOpts)
[outData, covData] = slvnvruntest(model, dataFile, runOpts)
```

Description

`outData = slvnvruntest(model, dataFile)` simulates `model` by using all the test cases in `dataFile`. `outData` is an array of `Simulink.SimulationOutput` objects. Each array element contains the simulation output data of the corresponding test case.

`outData = slvnvruntest(model, dataFile, runOpts)` simulates `model` by using all the test cases in `dataFile`. `runOpts` defines the options for simulating the test cases.

`[outData, covData] = slvnvruntest(model, dataFile, runOpts)` simulates `model` by using the test cases in `dataFile`. When the `runOpts` field `coverageEnabled` is `true`, the Simulink Coverage™ software collects model coverage information during the simulation. `slvnvruntest` returns the coverage data in the `cvdata` object `covData`.

Input Arguments

`model` — Simulink model that you simulate

character vector or string | handle

The Simulink model to simulate.

`dataFile` — Input data

character vector or string | structure

Name of the data file or structure that contains the input data. You can generate `dataFile` with Simulink Design Verifier software, or by running the `slvnvlogsignals` function.

`runOpts` — Configuration specification

structure

A structure whose fields specify the configuration of `slvnvruntest`.

Field	Description
`testIdx`	Test case index array to simulate from `dataFile`. If `testIdx` is `[]`, `slvnvruntest` simulates all test cases. **Default:** `[]`

Field	Description
`coverageEnabled`	If `true`, specifies that the Simulink Coverage software collects model coverage data during simulation. **Default:** `false`
`coverageSetting`	`cvtest` object for collecting model coverage. If `[]`, `slvnvruntest` uses the existing coverage settings for `model`. **Default:** `[]`
`fastRestart`	If `true`, Simulink Coverage uses fast restart mode for model simulation. **Default:** `true`
`useParallel`	If `true`, Simulink Coverage simulates test cases with parallel computing. This option requires a Parallel Computing Toolbox™ license. **Default:** `false`

Output Arguments

`outData` — Output objects obtained after simulating the test cases

array of `Simulink.SimulationOutput` objects

Each `Simulink.SimulationOutput` object has the following fields.

Field Name	Description
`tout_slvnvruntest`	Simulation time
`xout_slvnvruntest`	State data
`yout_slvnvruntest`	Output signal data
`logsout_slvnvruntest`	Signal logging data for: • Signals connected to outports • Signals that are configured for logging on the model

`covData` — Object that contains model coverage data

`cvdata` object

`cvdata` object that contains the model coverage data collected during simulation.

Note `covdata` references a file containing the coverage results. The coverage data from the referenced file is automatically loaded into memory when `covdata` is used by a coverage function. This file gets stored in the `sldv_covoutput` folder inside the current directory.

Examples

Analyze the Model and Examine the Output Data with the Simulation Data Inspector

```
% Analyze the sldemo_mdlref_basic model and log the input signals to the CounterA Model block:
open_system('sldemo_mdlref_basic');
loggedData = slvnvlogsignals('sldemo_mdlref_basic/CounterA');

% Using the logged signals, simulate the model referenced in the Counter block (sldemo_mdlref_counter):
runOpts = slvnvruntestopts;
runOpts.coverageEnabled = true;
open_system('sldemo_mdlref_counter');
[ outData ] = slvnvruntest('sldemo_mdlref_counter',...
    loggedData, runOpts);

% Examine the output data from the first test case using the Simulation Data Inspector:
Simulink.sdi.createRun('Test Case 1 Output', 'namevalue',...
    {'output'}, {outData(1).find('logsout_slvnvruntest')});
Simulink.sdi.view;
```

Tips

The `dataFile` that you create with a Simulink Design Verifier analysis or by running `slvnvlogsignals` contains time values and data values. When you simulate a model by using these test cases, you might see missing coverage. This issue occurs when the time values in the `dataFile` are not aligned with the current simulation time step due to numeric calculation differences. You see this issue more frequently with multirate models—models that have multiple sample times.

Tips

- For `useParallel`, the following points must be considered when simulating test cases using parallel computing:
  - Starting a parallel pool can take time, which impacts the overall analysis time. To reduce the analysis time:
    - Make sure that the parallel pool is already running before you run a test generation analysis. By default, the parallel pool shuts down after being idle for a specified number of minutes. To change the setting, see “Specify Your Parallel Preferences” (Parallel Computing Toolbox).
    - Load Simulink on all the parallel pool workers.
  - The simulation occurs sequentially when:
    - The cluster is not `local`. Configure parallel preferences to use the `local` cluster only. See “Specify Your Parallel Preferences” (Parallel Computing Toolbox).
    - The model is in `dirty state` prior to launching the SLDV analysis.
    - The model has `ToFile` blocks.
    - The model in Software-in-the-loop (SIL) simulation mode.
    - The model is an internal harness.

See Also

`cvsim` | `cvtest` | `sim` | `slvnvruntestopts`

Introduced in R2010b

slvnvruntestopts

Generate simulation or execution options for `slvnvruntest` or `slvnvruncgvtest`

Syntax

```
runOpts = slvnvruntestopts
runOpts = slvnvruntestopts('cgv')
```

Description

`runOpts = slvnvruntestopts` generates a `runOpts` structure for `slvnvruntest`.

generat`runOpts = slvnvruntestopts('cgv')`es a `runOpts` structure for `slvnvruncgvtest`.

Output Arguments

`runOpts` — Configuration specification of `slvnvruntest` or `slvnvruncgvtest`
structure

`runOpts` can have the following fields. If you do not specify a field, `slvnvruncgvtest` or `slvnvruntest` uses the default value.

Field Name	Description
`testIdx`	Test case index array to simulate or execute from data file. If `testIdx = []`, all test cases are simulated or executed. **Default:** `[]`
`signalLoggingSaveFormat`	Available only for `slvnvruntest`. Specifies the format of signal logging data for signals that connect to the outport of the model and for intermediate signals that are configured for logging. If you specify `Dataset`, data is stored in the `Simulink.SimulationData.Dataset` objects. **Default:** `'Dataset'`
`coverageEnabled`	Available only for `slvnvruntest`. If `true`, `slvnvruntest` collects model coverage data during simulation. **Default:** `false`

Field Name	Description
`coverageSetting`	Available only for `slvnvruntest`. `cvtest` object for collecting model coverage. If `coverageSetting` is `[]`, `slvnvruntest` uses the coverage settings for the model specified in the call to `slvnvruntest`. **Default:** `[]`
`allowCopyModel`	Available only for `slvnvruncgvtest`. If you have not configured your model to execute test cases with the CGV API, this field specifies creating and configuring the model. If `true` and you have not configured the model to execute test cases with the CGV API, `slvnvruncgvtest` copies the model, fixes the configuration, and executes the test cases on the copied model. If `false`, an error occurs if the tests cannot execute with the CGV API. **Note** If you have not configured the top-level model or any referenced models to execute test cases, `slvnvruncgvtest` does not copy the model, even if `allowCopyModel` is `true`. An error occurs. **Default:**`false`
`cgvCompType`	Available only for `slvnvruncgvtest`. Defines the software-in-the-loop (SIL) or processor-in-the-loop (PIL) approach for CGV: • `'topmodel'` • `'modelblock'` **Default:**`'topmodel'`
`cgvConn`	Available only for `slvnvruncgvtest`. Specifies mode of execution for CGV: • `'sim'` • `'sil'` • `'pil'` **Default:**`'sim'`

Field Name	Description
`fastRestart`	Available only for `slvnvruntest`. If `true`, Simulink Coverage uses fast restart mode for model simulation. **Default:** `true`
`useParallel`	Available only for `slvnvruntest`. If `true`, Simulink Coverage simulates test cases with parallel computing. This option requires a Parallel Computing Toolbox license. **Default:** `false`

Examples

Create `runOpts` Objects for `slvnvruntest` and `slvnvruncgvtest`

```
% Create runOpts objects for slvnvruntest
runtest_opts = slvnvruntestopts;

% Create runOpts objects for slvnvruncgvtest
runcgvtest_opts = slvnvruntestopts('cgv')
```

Alternatives

Create a `runOpts` object at the MATLAB command line.

See Also

`slvnvruncgvtest` | `slvnvruntest`

Introduced in R2010b

slwebview_cov

Export Simulink models to Web views with coverage

Syntax

```
filename = slwebview_cov(sysname)
filename = slwebview_cov(sysname,Name,Value)
```

Description

`filename = slwebview_cov(sysname)` exports the system `sysname` and its children to a web page `filename` with contextual coverage information for the system displayed on a separate panel of the layered model structure Web view.

`filename = slwebview_cov(sysname,Name,Value)` uses additional options specified by one or more `Name,Value` pair arguments.

Note You can use `slwebview_cov` only if you have also installed Simulink Report Generator™.

Examples

Export All Layers

Export all the layers (including libraries and masks) from the system `gcs` to the file `filename`

```
filename = slwebview_cov(gcs, 'LookUnderMasks', 'all', 'FollowLinks', 'on')
```

Input Arguments

`sysname` — The system to export to a Web view file

character vector or string containing the path to the system | handle to a subsystem or block diagram | handle to a chart or subchart

Exports the specified system or subsystem and its child systems to a Web view file, with contextual coverage information for the system displayed on a separate panel of the layered model structure Web view. By default, child systems of the `sysname` system are also exported. Use the `SearchScope` name-value pair to export other systems, in relation to `sysname`.

Example: 'sysname'

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:
`slwebview_cov(gcs,'SearchScope','CurrentAndBelow','LookUnderMasks','all','FollowLinks','on')`

`SearchScope` — Systems to export, relative to the `sysname` system
`'CurrentAndBelow'` (default) | `'Current'` | `'CurrentAndAbove'` | `'All'`

`'CurrentAndBelow'` exports the Simulink system or the Stateflow chart specified by `sysname` and all systems or charts that it contains.

`'Current'` exports only the Simulink system or the Stateflow chart specified by `sysname`.

`'CurrentAndAbove'` exports the Simulink system or the Stateflow chart specified by the `sysname` and all systems or charts that contain it.

`'All'` exports all Simulink systems or Stateflow charts in the model that contains the system or chart specified by `sysname`.

Data Types: `char`

`LookUnderMasks` — Specifies whether to export the ability to interact with masked blocks
`'none'` (default) | `'all'`

`'none'` does not export masked blocks in the Web view. Masked blocks are included in the exported systems, but you cannot access the contents of the masked blocks.

`'all'` exports all masked blocks.

Data Types: `char`

`FollowLinks` — Specifies whether to follow links into library blocks
`'off'` (default) | `'on'`

`'off'` does not allow you to follow links into library blocks in a Web view.

`'on'` allows you to follow links into library blocks in a Web view.

Data Types: `char`

`FollowModelReference` — Specifies whether to access referenced models in a Web view
`'off'` (default) | `'on'`

`'off'` does not allow you to access referenced models in a Web view.

`'on'` allows you to access referenced models in a Web view.

Data Types: `char`

`ViewFile` — Specifies whether to display the Web view in a Web browser when you export the Web view
`'on'` (default) | `'off'`

`'on'` displays the Web view in a Web browser when you export the Web view.

`'off'` does not display the Web view in a Web browser when you export the Web view.

Data Types: `char`

`ShowProgressBar` — Specifies whether to display the status bar when you export a Web view
'on' (default) | 'off'

`'on'` displays the status bar when you export a Web view.

`'off'` does not display the status bar when you export a Web view.

Data Types: `char`

`CovData` — `cvdata` objects to use
cvdata

The coverage data to use, specified as the comma-separated pair consisting of `'CovData'` and the `cvdata` objects to use.

Example: `'CovData', covdata`

Output Arguments

`filename` — The name of the HTML file for displaying the Web view
character vector or string

Reports the name of the HTML file for displaying the Web view. Exporting a Web view creates the supporting files, in a folder.

Tips

A Web view is an interactive rendition of a model that you can view in a Web browser. You can navigate a Web view hierarchically to examine specific subsystems and to see properties of blocks and signals.

You can use Web views to share models with people who do not have Simulink installed.

Web views require a Web browser that supports Scalable Vector Graphics (SVG).

See Also
`slwebview_req`

Introduced in R2015a

tableinfo

Retrieve lookup table coverage information from `cvdata` object

Syntax

```
coverage = tableinfo(cvdo, object)
coverage = tableinfo(cvdo, object, ignore_descendants)
[coverage, exeCounts] = tableinfo(cvdo, object)
[coverage, exeCounts, brkEquality] = tableinfo(cvdo, object)
```

Description

`coverage = tableinfo(cvdo, object)` returns lookup table coverage results from the `cvdata` object `cvdo` for the model component `object`.

`coverage = tableinfo(cvdo, object, ignore_descendants)` returns lookup table coverage results for `object`, depending on the value of `ignore_descendants`.

`[coverage, exeCounts] = tableinfo(cvdo, object)` returns lookup table coverage results and the execution count for each interpolation/extrapolation interval in the lookup table block `object`.

`[coverage, exeCounts, brkEquality] = tableinfo(cvdo, object)` returns lookup table coverage results, the execution count for each interpolation/extrapolation interval, and the execution counts for breakpoint equality.

Input Arguments

`cvdo`

`cvdata` object

`ignore_descendants`

Logical value specifying whether to ignore the coverage of descendant objects
`1` — Ignore coverage of descendant objects
`0` — Collect coverage for descendant objects

`object`

Full path or handle to a lookup table block or a model containing a lookup table block.

Output Arguments

`brkEquality`

A cell array containing vectors that identify the number of times during simulation that the lookup table block input was equivalent to a breakpoint value. Each vector represents the breakpoints along a different lookup table dimension.

coverage

The value of `coverage` is a two-element vector of form `[covered_intervals total_intervals]`, the elements of which are:

`covered_intervals`	Number of interpolation/extrapolation intervals satisfied for `object`
`total_intervals`	Total number of interpolation/extrapolation intervals for `object`

`coverage` is empty if `cvdo` does not contain lookup table coverage results for `object`.

exeCounts

An array having the same dimensionality as the lookup table block; its size has been extended to allow for the lookup table extrapolation intervals.

Examples

Collect lookup table coverage for the `slvnvdemo_cv_small_controller` model and determine the percentage of interpolation/extrapolation intervals coverage collected for the Gain Table block in the Gain subsystem:

```
mdl = 'slvnvdemo_cv_small_controller';
open_system(mdl)
%Create test spec object
testObj = cvtest(mdl)
%Enable lookup table coverage
testObj.settings.tableExec = 1;
%Simulate the model
data = cvsim(testObj)
blk_handle = get_param([mdl, '/Gain/Gain Table'], 'Handle');
%Retrieve l/u table coverage
cov = tableinfo(data, blk_handle)
%Percent MCDC outcomes covered
percent_cov = 100 * cov(1) / cov(2)
```

Alternatives

Use the coverage settings to collect lookup table coverage for a model:

1. Open the model.
2. In the Model Editor, select **Model Settings** on the **Modeling** tab.
3. On the **Coverage** pane of the Configuration Parameters dialog box, select **Enable coverage analysis**.
4. Under **Coverage metrics**, select **Lookup Table**.
5. Click **OK** to close the Configuration Parameters dialog box and save your changes.
6. Simulate the model by clicking the **Run** button and review the results.

See Also

`complexityinfo` | `conditioninfo` | `cvsim` | `decisioninfo` | `executioninfo` | `getCoverageInfo` | `mcdcinfo` | `overflowsaturationinfo` | `relationalboundaryinfo` | `sigrangeinfo` | `sigsizeinfo`

Topics

“Lookup Table Coverage”

Introduced in R2006b

name property

Class: cv.cvdatagroup
Package: cv

cv.cvdatagroup object name

Values

name

Description

The name property specifies the name of the cv.cvdatagroup object.

Examples

```
cvdg = cvsim(topModelName);
cvdg.name = 'My_Data_Group';
```

slcovmex

Build coverage-compatible MEX-function from C/C++ code

Syntax

```
slcovmex(sourceFile1,...,sourceFileN)
slcovmex(sourceFile1,...,sourceFileN,-sldv)
slcovmex(sourceFile1,...,sourceFileN,Name,Value)
slcovmex(argumentSet1,...,argumentSetN)
```

Description

`slcovmex(sourceFile1,...,sourceFileN)` compiles level 2 C/C++ MEX S-Function to work with coverage.

`slcovmex(sourceFile1,...,sourceFileN,-sldv)` compiles level 2 C/C++ MEX S-Function to work with coverage, and with support enabled for Simulink Design Verifier.

`slcovmex(sourceFile1,...,sourceFileN,Name,Value)` uses additional options specified by one or more `Name,Value` pair arguments.

`slcovmex(argumentSet1,...,argumentSetN)` combines several `mex` function calls, each with one set of arguments.

Input Arguments

`sourceFile1,...,sourceFileN` — One or more file names

character vectors or strings

Comma-separated source file names with each name specified as a character vector or string.

If the files are not in the current folder, the file names must include the full path or relative path. Use `pwd` to find the current folder and `cd` to change the current folder.

Example: `'file1.c'`, `'file1.c','file2.c'`

`argumentSet1,...,argumentSetN` — One or more sets of `mex` arguments

Cell arrays of character vectors or strings

Comma-separated `mex` argument sets, with each set specified as a cell array.

If you invoke `mex` multiple times, you can invoke `slcovmex` once and pass the arguments for each `mex` invocation as a cell array of character vectors.

For example, if you use the following sequence of `mex` commands:

```
mex -c file1.c
mex -c file2.c
mex file1.o file2.o -output sfcnOutput
```

You can replace the sequence with one `slcovmex` invocation:

```
slcovmex({'-c','file1.c'},{'-c','file2.c'},{'file1.o','file2.o',
'-output','sfcnOutput'})
```

Example: `{'-c','file1.c'},{'-c','file2.c'},{'file1.o','file2.o','-output','sfcnOutput'}`

`-sldv` — Option to enable support for Simulink Design Verifier

character vector or string

Option to enable support for your compiled MEX-function in Simulink Design Verifier.

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: You can use all the name-value pair arguments that are allowed for the `mex` function. In addition, you can use the following options that are specific to model coverage.

`-ifile` — File ignored for coverage

character vector or string

File name, specified as a character vector or string.

Example: `'myFile.c'`

`-ifcn` — Function ignored for coverage

character vector or string

Function name, specified as a character vector or string.

Example: `'myFunc'`

`-idir` — Folder ignored for coverage

character vector or string

Folder name, specified as a character vector or string.

All files in the folder are ignored for coverage.

Example: `'C:\Libraries\'`

See Also

Topics

“Create a Basic C MEX S-Function”
“Templates for C S-Functions”
“Coverage for Custom C/C++ Code in Simulink Models”
“View Coverage Results for Custom C/C++ Code in S-Function Blocks”

Introduced in R2015a

cvdata Properties

Store Simulink Coverage data for use in other coverage functions

Description

`cvdata` objects store model coverage data. A `cvdata` object is generated automatically when you simulate a model that has coverage enabled. The coverage data can be accessed by other coverage functions. The `cvdata` object becomes invalid if its parent model is closed or modified.

Properties

Properties

`dbVersion` — Coverage data origin release

`character array`

This property is read-only.

Coverage data origin release, returned as a character array.

Data Types: `char`

`id` — Internal coverage data ID

`scalar`

This property is read-only.

Internal coverage data ID, returned as a scalar.

Data Types: `double`

`type` — Internal coverage data type

`TEST_DATA` | `DERIVED_DATA`

This property is read-only.

Internal coverage type, returned as either `TEST_DATA` for a single coverage simulation, or `DERIVED_DATA` for aggregated or cumulative coverage data.

Data Types: `char`

`test` — Test data

`cvtest` object

This property is read-only.

Test data, returned as a `cvtest` object. This property describes the coverage configuration.

Data Types: `cvtest`

`rootID` — Internal root ID

`scalar`

This property is read-only.

Internal root ID, returned as a scalar.

Data Types: `double`

`checksum` — Coverage data checksum

`struct`

This property is read-only.

Coverage data checksum, returned as a struct. The checksum is based on the structure of the model being analyzed. It can be used to determine if two `cvdata` objects would be compatible for data aggregation.

Data Types: `struct`

`modelinfo` — Model information

`struct`

This property is read-only.

Model information, returned as a struct. This property contains metadata about the model analyzed for coverage.

`cvdata.modelinfo` has the following fields:

Field	Description	Values
`modelVersion`	Version of the model analyzed for coverage.	`character array` containing version number
`creator`	Original creator of the model.	System name or organization name
`lastModifiedDate`	Date and time the model was last modified.	`character array` containing date and time
`defaultParameterBehavior`	Indicates the default parameter behavior setting.	`'Tunable' | 'Inlined'`
`blockReductionStatus`	Indicates whether Block Reduction is enabled. See “Block Reduction” for more information.	`'off' | 'on'`
`conditionallyExecuteInputs`	Conditional input execution switch. A value of `1` is `'on'`, and a value of `0` is `'off'`.	`1 | 0`

Field	Description	Values
`mdcdMode`	Definition used for modified condition decision coverage (MCDC) analysis. A value of `1` indicates the model used the masking definition of MCDC and a value of `0` indicates the model used the unique-cause definition of MCDC. For more information, see "Modified Condition and Decision Coverage (MCDC) Definitions in Simulink Coverage".	`1 | 0`
`analyzedModel`	Name of the analyzed model or model object. If analysis is scoped to a subsystem, Stateflow Chart, or other model object, this is the path to that model object.	`character array`
`reducedBlocks`	List of blocks reduced by the block reduction parameter, if it is enabled and any blocks are reduced.	`character array`
`ownerModel`	Model that is or contains the component under test. If you have a block diagram harness, this is the model that the harness tests. If you have a subsystem harness, this is the model that contains that subsystem.	`character array`
`ownerBlock`	If the model includes a subsystem harness, this is the subsystem that the harness is testing.	`character array`
`harnessModel`	Harness model name. If you have data aggregated from multiple test runs, where each run used a different harness with the same `ownerModel`, this field shows `Not Unique`.	`character array`
`logicBlkShortcircuit`	Indicates whether the short-circuiting option is enabled.	`0 | 1`

Data Types: `struct`

`startTime` — System time at simulation start
`character array`

System time at simulation start, returned as a character array.

Data Types: `char`

stopTime — System time at simulation stop
scalar

System time at simulation stop, returned as a character array.

Data Types: char

intervalStartTime — Coverage interval start time
scalar

Coverage interval start time, returned as a scalar. This value comes from the CovStartTime parameter. For more information, see “Coverage interval start time” on page 2-15.

Data Types: double

intervalStopTime — Coverage interval start time
scalar

Coverage interval stop time, returned as a scalar. This value comes from the CovStopTime parameter. For more information, see “Coverage interval stop time” on page 2-15.

Data Types: double

filter — Coverage filter file name
character array | cell array

Coverage filter file name, returned as a character array or a cell array of character arrays. This property contains the coverage filter file name. If there is no coverage filter, this field is empty. You can apply a coverage filter after simulation by assigning the name of a valid filter file to this property.

Data Types: char | cell

simMode — Simulation mode
character array

Simulation mode, returned as a character array. For more information, see “Simulation mode”.

Data Types: char

See Also

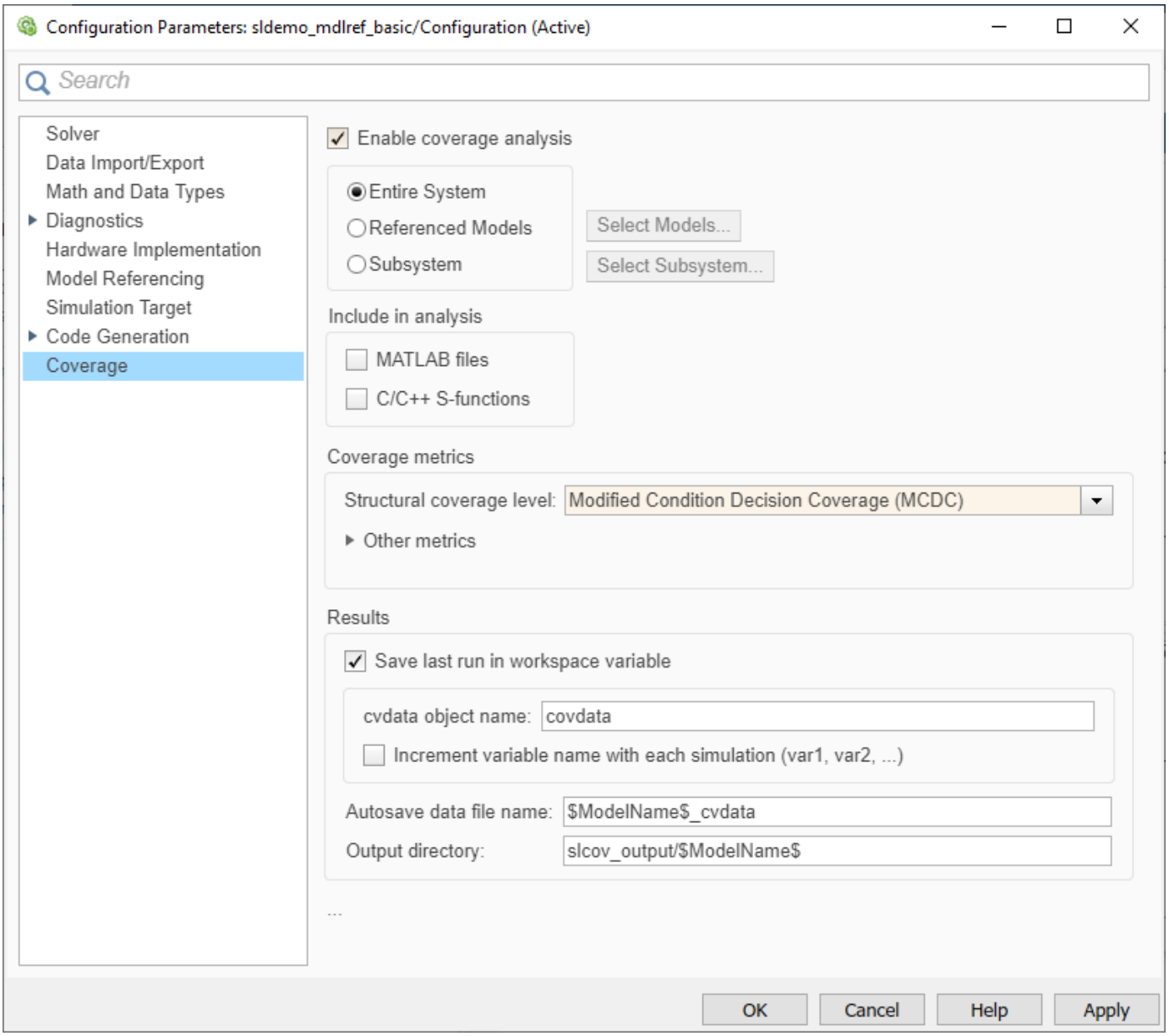
cv.cvdatagroup | cvhtml | cvsim

Topics
“Collect Coverage Data”
“Evaluate Coverage Results”

Introduced before R2006a

Simulink Coverage Settings

Coverage Settings



In this section...

In this section...

Coverage Pane Overview

Specify the Simulink Coverage analysis options.

Enable coverage analysis

Enable coverage analysis. See “Specify Coverage Options”.

Settings

☑ On

Coverage data is collected during simulation.

☐ Off (default)

Coverage data is not collected during simulation.

Command-Line Information

Parameter: `CovEnable`
Type: Character vector or string
Value: `'on'|'off'`
Default: `'off'`

Scope of coverage analysis

Specify whether the analysis must collect coverage data for the entire system, or a specific referenced model, Observer model or subsystem.

Settings

Entire System (default)

Coverage data is collected for the top-level model, as well as all supported subsystems and model references.

Referenced Models

Coverage data is collected for one or more referenced models. To specify the referenced models, use the parameter “Select Models” on page 2-5. You can also specify the top-level model itself.

Subsystem

Coverage data is collected for a specific subsystem. To specify a subsystem, use the parameter “Select Subsystem” on page 2-5.

Dependency

To enable this parameter, select “Enable coverage analysis” on page 2-4.

Command-Line Information

Parameter: `CovScope`
Type: Character vector or string
Value: `'EntireSystem'` | `'ReferencedModels'` | `'Subsystem'`
Default: `'EntireSystem'`

Select Models

Specify the referenced models for which you want coverage.

Settings

In the **Select Models for Coverage Analysis** dialog box, select the referenced models and Observer models for which you want coverage. You can also select the top-level model. The icon next to the model name indicates the simulation mode: `Normal`, `SIL`, or `PIL`. Only Observer models in `Normal` mode are analyzed for coverage.

Dependency

To enable this parameter, select:

- “Enable coverage analysis” on page 2-4
- Specify referenced models for “Scope of coverage analysis” on page 2-4.

Command-Line Information

Note Unlike in the user interface, on the command line, you *exclude* models from coverage instead of including them.

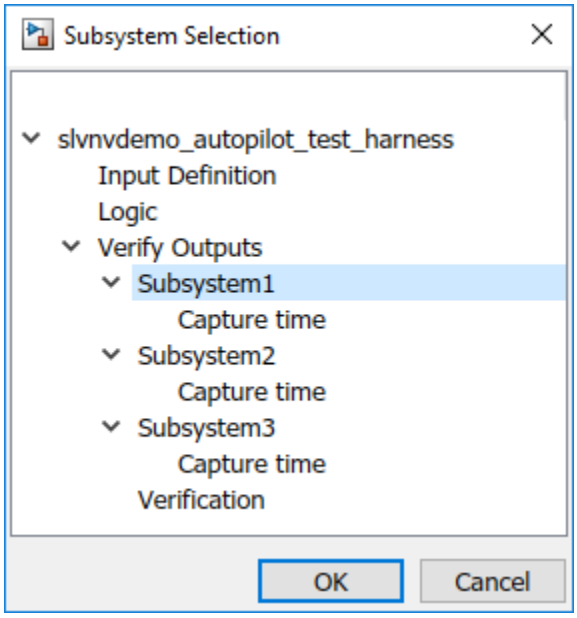
Parameter: `CovModelRefExcluded`
Type: Character vector or string
Value: Comma-separated list of model names, for instance, `'mRefA, mRefB, mRefC'`. If the same model is referenced in two simulation modes, you can distinguish between them using `:`, for instance, `'mRefA:normal, mRefA:sil'`.
Default: `''`

Select Subsystem

Specify the path to the subsystem for which Simulink Coverage collects coverage data. Specify the path relative to the top model.

Settings

Select the subsystem for which you want coverage.



Dependency

To enable this parameter, select:

- “Enable coverage analysis” on page 2-4
- “Scope of coverage analysis” on page 2-4

Command-Line Information
Parameter: `CovPath`
Type: Character vector or string
Value: Path to subsystem relative to (and excluding) the top-level Simulink system, for instance, `'Subsys1/subsys2'`
Default: `'/'`. Coverage data is reported for the entire system.

Record coverage for MATLAB files

Enable coverage for MATLAB functions in external MATLAB files. The functions can be invoked from MATLAB Function blocks or Stateflow charts in your model. See “Model Coverage for MATLAB Functions”.

Settings

☑ On (default)

Coverage data is collected for MATLAB functions in external MATLAB files. The functions can be called from MATLAB Function blocks or Stateflow charts in the model.

☐ Off

Coverage data is not collected for external MATLAB files.

Dependency

To enable this parameter, select “Enable coverage analysis” on page 2-4.

Command-Line Information
Parameter: `CovExternalEMLEnable`
Type: Character vector or string
Value: `'on'|'off'`
Default: `'on'`

Record coverage for C/C++ S-functions

Enable coverage for C/C++ code in S-Function blocks in your model. See also “Coverage for Custom C/C++ Code in Simulink Models”.

Settings

☑ On (default)

Coverage data is collected for C/C++ code in S-Function blocks in the model.

☐ Off

Coverage data is not collected for C/C++ code used in the model.

Dependency

To enable this parameter, select:

- “Enable coverage analysis” on page 2-4
- “Record coverage for this model” on page 2-19 or “Record coverage for referenced models” on page 2-20 (enter `on`)

Command-Line Information
Parameter: `CovSFcnEnable`
Type: Character vector or string
Value: `'on'|'off'`
Default: `'on'`

Structural coverage level

Select the type of coverage data collected. See also “Types of Model Coverage”.

Settings

Decision (default)

The analysis computes decision coverage during simulation

Decision coverage analysis checks blocks that perform an action based on whether an operation evaluates to true or false. For instance, the Abs block first evaluates if the input is less than zero and acts accordingly. For each operation that can evaluate to true or false, the analysis reports what fraction of the outcomes was true during simulation and what fraction was false.

See “Decision Coverage (DC)”.

Condition/Decision

The analysis computes condition and decision coverage during simulation.

Condition coverage analysis checks blocks that output a logical combination of their inputs (such as Logical Operator blocks). For each block, the analysis records what fraction of the inputs was true during simulation and what fraction was false.

See “Condition Coverage (CC)”.

Modified Condition/Decision Coverage (MCDC)

The analysis computes Modified Condition/Decision Coverage (MCDC) during simulation.

See “Modified Condition/Decision Coverage (MCDC)”.

Block Execution

The analysis checks if each block executes at least once during simulation.

See “Execution Coverage (EC)”.

Dependency

To enable this parameter, select “Enable coverage analysis” on page 2-4.

Command-Line Information

Parameter: `CovMetricStructuralLevel`
Type: Character vector or string
Value: `'BlockExecution'` | `'ConditionDecision'` | `'Decision'` | `'MCDC'`
Default: `'Decision'`

Lookup table

Enable lookup table coverage. See “Types of Model Coverage”.

Settings

☑ On

Blocks with lookup tables are checked for coverage. A test case achieves full coverage of a lookup table if it executes each interval of the table at least once.

☐ Off (default)

Lookup table coverage is not recorded.

Dependency

To enable this parameter, select “Enable coverage analysis” on page 2-4.

Command-Line Information

Parameter: `CovMetricLookupTable`
Type: Character vector or string
Value: `'on'|'off'`
Default: `'off'`

Signal range

Enable signal range coverage. See “Types of Model Coverage”.

Settings

☑ On

Maximum and minimum signal values are recorded for each block that has an output signal.

☐ Off (default)

Signal range information is not recorded.

Dependency

To enable this parameter, select “Enable coverage analysis” on page 2-4.

Command-Line Information

Parameter: `CovMetricSignalRange`
Type: Character vector or string
Value: `'on'|'off'`
Default: `'off'`

Signal size

Enable signal size coverage. See “Types of Model Coverage”.

Settings

☑ On

Maximum, minimum and allocated signal size are recorded for each block that has a variable-size output signal. See “Variable-Size Signal Basics”.

☐ Off (default)

Signal size information is not recorded.

Dependency

To enable this parameter, select “Enable coverage analysis” on page 2-4.

Command-Line Information

Parameter: `CovMetricSignalSize`
Type: Character vector or string
Value: `'on'|'off'`
Default: `'off'`

Objectives and constraints

Enable coverage of objectives and constraints specified in Simulink Design Verifier blocks. See “Types of Model Coverage”.

Settings

☑ On

Through Simulink Design Verifier blocks, you can specify objectives and constraints in your model. To check if these objectives are satisfied, you first generate test cases using these blocks.

You can execute these test cases on the original model and record whether the specified objective was satisfied at least once. To record this coverage, enable this parameter.

For an example, see “Simulink Design Verifier Coverage”.

☐ Off (default)

Coverage information is not recorded for Simulink Design Verifier blocks.

Dependency

To enable this parameter, select “Enable coverage analysis” on page 2-4.

Command-Line Information
Parameter: `CovMetricObjectiveConstraint`
Type: Character vector or string
Value: `'on'|'off'`
Default: `'off'`

Saturation on integer overflow

Enable saturation on integer overflow coverage. See “Types of Model Coverage”.

Settings

☑ On

For certain blocks, such as the Abs block, you can specify that they must saturate on integer overflow. If you enable this parameter, the number of times these blocks saturate during simulation is recorded.

☐ Off (default)

Saturation on integer overflow information is not recorded.

Dependency

To enable this parameter, select “Enable coverage analysis” on page 2-4.

Command-Line Information
Parameter: `CovMetricSaturateOnIntegerOverflow`
Type: Character vector or string
Value: `'on'|'off'`
Default: `'off'`

Relational boundary

Enable relational boundary coverage. See “Types of Model Coverage”.

Settings

☑ On

Certain blocks such as the Relational Operator or If block use a relational operation. If you enable this parameter, the coverage analysis checks if these operations are executed with equal (integer) or almost equal (floating-point) values.

Off (default)

Relational boundary coverage information is not recorded.

Dependency

To enable this parameter, select “Enable coverage analysis” on page 2-4.

Command-Line Information
Parameter: `CovMetricRelationalBoundary`
Type: Character vector or string
Value: `'on'|'off'`
Default: `'off'`

Relational boundary coverage absolute tolerance

Specify the value of absolute tolerance for relational boundary coverage. See “Relational Boundary Coverage”.

Settings

Enter a floating-point value. See “Floating-Point Numbers”.

Relational boundary coverage checks blocks with relational operations (such as the Relational Operator block). The analysis checks if the operations are executed with floating-point operands that differ by at most this value.

Dependency

To enable this parameter, select:

- “Enable coverage analysis” on page 2-4
- “Relational boundary” on page 2-10

Command-Line Information
Parameter: `CovBoundaryAbsTol`
Type: Floating-point number
Value: Absolute tolerance value such as 1e-06
Default: 1e-05

Relational boundary coverage relative tolerance

Specify the value of relative tolerance for relational boundary coverage. See “Relational Boundary Coverage”.

Settings

Enter a number less than 1.

Relational boundary coverage checks blocks with relational operations (such as the Relational Operator block). The analysis checks if the operations are executed with floating-point operands that differ by at most this fraction of the operands.

For instance, if you enter 0.01, the analysis checks if an operation `lhs < = rhs` in your model is executed with operands that differ by at most:

```
0.01 * max(|lhs|,|rhs|)
```

Dependency

To enable this parameter, select:

- "Enable coverage analysis" on page 2-4
- "Relational boundary" on page 2-10

Command-Line Information

Parameter: `CovBoundaryRelTol`
Type: Floating-point number
Value: Relative tolerance value such as 0.001
Default: 0.01

Save last run in workspace variable

Save the coverage data from simulation in a MATLAB variable.

You can retrieve coverage information from this variable later. For instance, to retrieve decision coverage information, use the `decisioninfo` function. For the full list of functions, see "Automate Coverage Workflows".

Settings

☑ On

Coverage data is stored in a `cvdata` object in the MATLAB workspace. Specify the object name using the parameter "Last coverage run variable name" on page 2-12. Choose to create a new object for each simulation using the parameter "Increment variable name with each simulation" on page 2-13.

☐ Off (default)

Coverage data is not stored in a MATLAB variable.

Dependency

To enable this parameter, select "Enable coverage analysis" on page 2-4.

Command-Line Information

Parameter: `CovSaveSingleToWorkspaceVar`
Type: Character vector or string
Value: `'on'|'off'`
Default: `'off'`

Last coverage run variable name

Specify a name for the `cvdata` object that contains coverage results from the last simulation.

Settings

Enter a name, for instance, `coverageData`.

If you want a new variable to store coverage results for each simulation, use the parameter "Increment variable name with each simulation" on page 2-13. The new variable name is created by appending a counter value to the original name, for instance, `coverageData1`, `coverageData2`, and so on.

The default variable name is `covdata`.

Dependency

To enable this parameter, select:

- "Enable coverage analysis" on page 2-4
- "Save last run in workspace variable" on page 2-12

Command-Line Information
Parameter: `CovSaveName`
Type: Character vector or string
Value: Name to be given to `cvdata` object
Default: `'covdata'`

Increment variable name with each simulation

Create a new variable to store coverage results for each new simulation.

Settings

☑ On

A new `cvdata` object stores coverage results for each simulation.

The new variable name is created by appending a counter value to the original variable name from the first simulation. Specify the original variable name using the parameter "Last coverage run variable name" on page 2-12.

☐ Off (default)

Each new simulation overwrites the coverage results from the previous simulation. A single `cvdata` object stores the coverage results from the most recent simulation.

Dependency

To enable this parameter, select:

- "Enable coverage analysis" on page 2-4
- "Save last run in workspace variable" on page 2-12

Command-Line Information
Parameter: `CovNameIncrementing`
Type: Character vector or string
Value: `'on'|'off'`
Default: `'off'`

Autosave data file name

Specify name of `.cvt` file to which coverage data is automatically saved.

Settings

Enter file name. The default name is `$ModelName$_cvdata`, where `$ModelName$` is the model name.

Dependency

To enable this parameter, select:

- "Enable coverage analysis" on page 2-4
- "Save output data" on page 2-23

Command-Line Information
Parameter: `CovDataFileName`
Type: Character vector or string
Value: Name to be given to `.cvt` file
Default: `'$ModelName$_cvdata'`

Output directory

Specify a folder in which coverage output files are saved.

Settings

Enter path to folder. You can enter the absolute path or path relative to the current working folder.

By default, the files are saved in a subfolder `slcov_output/$ModelName$` relative to the current working folder. Here `$ModelName$` is the model name.

Dependency

To enable this parameter, select "Enable coverage analysis" on page 2-4.

Command-Line Information
Parameter: `CovOutputDir`
Type: Character vector or string
Value: Path to folder
Default: `'slcov_output/$ModelName$'`

Restrict coverage recording interval

Record coverage only for a specified time interval.

For instance, you might want to restrict model coverage recording if your model has transient effects early in simulation, or if you need model coverage reported only for a particular model operation.

Settings

☑ On

Coverage is recorded only for the time interval that you specify. To specify a time interval, use these parameters:

- “Coverage interval start time” on page 2-15
- “Coverage interval stop time” on page 2-15

☐ Off (default)

Coverage is recorded for the entire duration of simulation.

Dependency

To enable this parameter, select “Enable coverage analysis” on page 2-4.

Command-Line Information
Parameter: `CovUseTimeInterval`
Type: Character vector or string
Value: `'on'|'off'`
Default: `'off'`

Coverage interval start time

Specify when coverage data collection must begin.

Settings

Enter a time value (in seconds).

Dependency

To enable this parameter, select:

- “Enable coverage analysis” on page 2-4
- “Restrict coverage recording interval” on page 2-14

Command-Line Information
Parameter: `CovStartTime`
Type: Floating-point number
Value: Time in seconds, for instance, 2
Default: 0

Coverage interval stop time

Specify when coverage data collection must end.

Settings

Enter a time value (in seconds).

Dependency

To enable this parameter, select:

- “Enable coverage analysis” on page 2-4
- “Restrict coverage recording interval” on page 2-14

Command-Line Information

Parameter: `CovStopTime`
Type: Floating-point number
Value: Time in seconds, for instance, 4
Default: 0

Force block reduction off

Report coverage for every block in the model that is supported for coverage.

Settings

☑ On (default)

Coverage is recorded for every supported block in the model. The value of the configuration parameter **Block reduction** is ignored. See “Block reduction”.

☐ Off

Coverage is not recorded for blocks that are effectively removed from the model because of block reduction. For instance, coverage is not recorded for a block that is reduced by dead code elimination.

Dependency

To enable this parameter, select “Enable coverage analysis” on page 2-4.

Command-Line Information

Parameter: `CovForceBlockReductionOff`
Type: Character vector or string
Value: `'on'|'off'`
Default: `'on'`

Treat Simulink logic blocks as short-circuited

Specify that coverage must take into account the order of operands in blocks that perform a logical operation, for instance, Logical operator blocks.

For instance, if the order of the two inputs to a Logical AND block is taken into account, the second input is redundant when the first input is false. Therefore, for cases where the first input is false, the paths that lead to the second input are not considered for coverage.

Settings

☑ On

Coverage analysis does not consider the input to a logical operation that is rendered redundant by another input.

☐ Off (default)

Coverage analysis considers all inputs to a logical operation.

Dependency

To enable this parameter, select “Enable coverage analysis” on page 2-4.

Command-Line Information
Parameter: `CovLogicBlockShortCircuit`
Type: Character vector or string
Value: `'on'|'off'`
Default: `'off'`

MCDC mode

Specify the definition of Modified Condition/Decision Coverage (MCDC) to use during coverage analysis. See “Modified Condition and Decision Coverage (MCDC) Definitions in Simulink Coverage”.

Settings

`Masking`

Use masking MCDC analysis. To establish the independence of inputs, masking MCDC analysis does not require that all other inputs be strictly held constant while one input is varied. Therefore, masking MCDC analysis allows you to satisfy greater number of objectives in a given simulation.

`Unique-Cause`

Use unique-cause MCDC analysis.

Dependency

To enable this parameter, select:

- “Enable coverage analysis” on page 2-4
- Specify Modified Condition/Decision Coverage (MCDC) for “Structural coverage level” on page 2-7.

Command-Line Information
Parameter: `CovMcdcMode`
Type: Character vector or string
Value: `'Masking'|'UniqueCause'`
Default: `'Masking'`

Warn when unsupported blocks exist in model

Warn when unsupported blocks exist in model.

Settings

☑ On (default)

Provide a warning when blocks in the model are not supported for coverage analysis.

Off

Do not provide a warning for unsupported blocks.

Dependency

To enable this parameter, select “Enable coverage analysis” on page 2-4.

Command-Line Information
Parameter: `CovUnsupportedBlockWarning`
Type: Character vector or string
Value: `'on'|'off'`
Default: `'on'`

Coverage filter filename

Specify a filter file to exclude certain model objects from coverage analysis during simulation.

You can use a command-line API to create filtering rules for blocks. Selection criteria for filtering includes filtering by individual block ID, filtering for all blocks of the same type, filtering certain decisions, conditions, and outcomes of a block, and more. You can also filter S-Function C++ code by code coverage outcome.

For an example of filtering, see:

- User interface: “Create, Edit, and View Coverage Filter Rules”.
- Command line: R2017b release notes for Simulink Coverage.

Settings

Enter full path to `.cvf` file with filter rules.

Dependency

To enable this parameter, select “Enable coverage analysis” on page 2-4.

Command-Line Information
Parameter: `CovFilter`
Type: Character vector or string
Value: Full path to `.cvf` file
Default:

Coverage metric settings

Specify the type of coverage metric to be recorded. See also “Types of Model Coverage”.

Settings

Enter a sequence of letters that describe the coverage metric types. For instance, the sequence `dc` indicates that the decision and condition coverage must be recorded.

The coverage metric types are:

- `d`: Decision coverage

- `c`: Condition coverage
- `m`: MCDC coverage
- `t`: Lookup table coverage
- `r`: Signal range coverage
- `o`: Coverage for Simulink Design Verifier blocks
- `b`: Relational boundary coverage
- `r`: Signal range coverage

Additionally, you can use these letters. The letters correspond to other parameters.

- `s`: “Treat Simulink logic blocks as short-circuited” on page 2-16
- `w`: “Warn when unsupported blocks exist in model” on page 2-17
- `e`: Disables display coverage results using model coloring

Note `CovHighlightResults` has been removed. Include `e` in `CovMetricSettings` to avoid a warning message.

Dependency

To enable this parameter, select:

- “Enable coverage analysis” on page 2-4
- One of these: “Record coverage for this model” on page 2-19, “Record coverage for referenced models” on page 2-20 (enter `on`) or “Record coverage for MATLAB files” on page 2-6

Command-Line Information

Parameter: `CovMetricSettings`
Value: Character vector or string where each character signifies a coverage metric. For instance, `'dc'` specifies decision and condition coverage.
Default: `'dwe'`

Record coverage for this model

Record model coverage data during simulation.

Note This parameter represents a deprecated workflow. Instead use these parameters:

- To enable coverage, use “Enable coverage analysis” on page 2-4.
- To perform coverage analysis for the entire model, use “Scope of coverage analysis” on page 2-4.

Settings

☑ On (default)

Simulink collects model coverage data during simulation.

☐ Off

Model coverage data is not collected or reported.

Command-Line Information
Parameter: `RecordCoverage`
Type: Character vector or string
Value: `'on'|'off'`
Default: `'on'`

Record coverage for referenced models

Record coverage data for referenced models during simulation.

Note This parameter represents a deprecated workflow. Instead use these parameters:

- To enable coverage, use "Enable coverage analysis" on page 2-4.
- To perform coverage analysis for referenced models, use "Scope of coverage analysis" on page 2-4.
- To specify the referenced models, use "Select Models" on page 2-5.

Settings

Enter one of these:

- `on`: Coverage data is collected for all referenced models and Observer models.
- `off`: Coverage data is not collected for referenced models.
- `filtered`: Coverage data is collected for all referenced models and Observer models except those excluded using the parameter "Select Models" on page 2-5.

Command-Line Information
Parameter: `CovModelRefEnable`
Type: Character vector or string
Value: `'on'|'off'|'filtered'`
Default: `'off'`

Include top model

Record coverage for the top-level model in addition to referenced models.

Note This parameter represents a deprecated workflow. Instead use these parameters:

- To enable coverage, use "Enable coverage analysis" on page 2-4.
- To perform coverage analysis for referenced models, use "Scope of coverage analysis" on page 2-4.
- To include or exclude the top-level model, use "Select Models" on page 2-5.

Settings

☑ On (default)

Coverage data is collected for the top-level model.

☐ Off

Coverage data is not collected for the top-level model.

Dependency

To enable this parameter, select:

- “Enable coverage analysis” on page 2-4
- Specify referenced model for “Scope of coverage analysis” on page 2-4.

Command-Line Information
Parameter: `CovIncludeTopModel`
Type: Character vector or string
Value: `'on'|'off'`
Default: `'on'`

Coverage report options

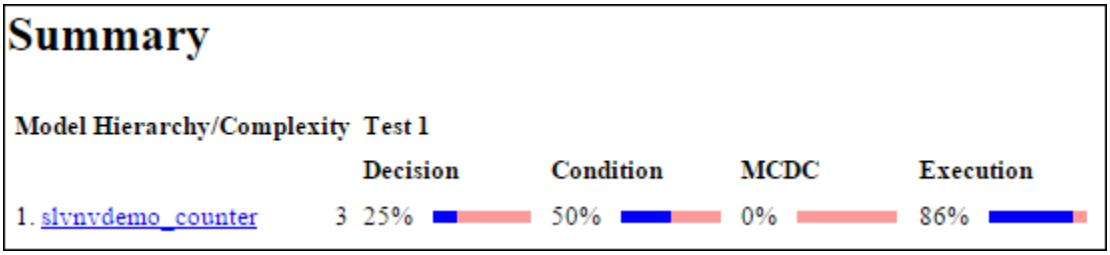
Specify the formatting of certain aspects of the coverage report (HTML).

Note For an easier way to specify report formatting, see Report from Results Explorer.

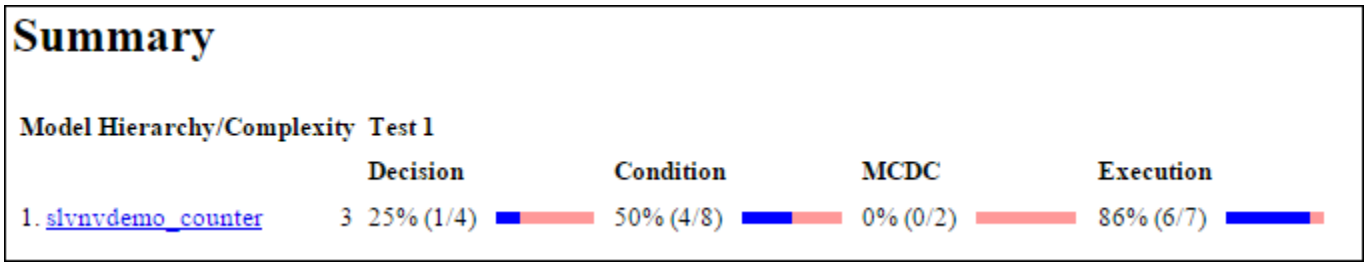
Settings

Enter a space-separated list of flags. The available flags are:

- `'-sRT=0'` — Do not show report
- `'-sVT=1'` — Open a web view of the report in a browser. See also “Export Model Coverage Web View”.
- `'-aTS=1'` — Show each test in the model summary.
- `'-bRG=1'` — Show bar graphs in the model summary.



- `'-bTC=1'` — Use two color bar graphs (red, blue).
- `'-hTR=1'` — Display hit/count ratio in the model summary.



- `'-nFC=0'` — Do not report fully covered model objects
- `'-scm=1'` — Include cyclomatic complexity numbers in summary. See also “Cyclomatic Complexity”.
- `'-bcm=1'` — Include cyclomatic complexity numbers in block details.
- `'-xEv=0'` — Filter Stateflow events from report.
- `'-agT=1'` — Show aggregated tests information in coverage report.
- `'-req=1'` — Include linked requirements in aggregated coverage report.

Dependency

To enable this parameter, select:

- “Enable coverage analysis” on page 2-4

Command-Line Information
Parameter: `CovHTMLOptions`
Type: Character vector or string
Value:
Default:

Additional data to include in coverage report

Include additional model coverage data from `cvdata` objects in the model coverage report.

Settings

Enter the name of a `cvdata` object associated with a simulation.

You get a `cvdata` object when you record coverage and save coverage data in a workspace variable. See:

- “Last coverage run variable name” on page 2-12
- “Cumulative coverage variable name” on page 2-25

You also get a `cvdata` object if you run simulation using the `cvsim` function. See `cvsim`.

Dependency

To enable this parameter, select:

- “Enable coverage analysis” on page 2-4

Command-Line Information
Parameter: `CovCompData`
Type: Character vector or string
Value: Name of `cvdata` object.
Default: No default

Update coverage results on pause

Update coverage report when you pause during simulation. The report is updated with coverage results up to the current pause or stop time.

Settings

☑ On (default)

Coverage report is updated when you pause simulation.

☐ Off

Coverage report is not updated when you pause simulation.

Dependency

To enable this parameter, select "Enable coverage analysis" on page 2-4.

Command-Line Information
Parameter: `CovReportOnPause`
Type: Character vector or string
Value: `'on'|'off'`
Default: `'on'`

Save output data

Save coverage data results to a file.

Settings

☑ On (default)

Coverage data results are saved to a file. Specify the file name using the parameter "Autosave data file name" on page 2-14.

☐ Off

Coverage data results are not saved to a file.

Dependency

To enable this parameter, select "Enable coverage analysis" on page 2-4.

Command-Line Information
Parameter: `CovSaveOutputData`
Type: Character vector or string
Value: `'on'|'off'`
Default: `'on'`

Enable cumulative data collection

Collect model coverage results from successive simulations. See also “Cumulative Coverage Data”.

Note For an easier way to accumulate coverage data from multiple simulations, see “Accumulating Coverage Data from the Results Explorer”.

Settings

☑ On (default)

Model coverage data from successive simulations are collected together.

To show the cumulative data in one report, use the parameter “Include cumulative data in coverage report” on page 2-24. To save the data in one workspace variable, use the parameters “Save cumulative coverage results in workspace variable” on page 2-25 and “Cumulative coverage variable name” on page 2-25.

☐ Off

Model coverage data is retained for the most recent simulation only.

Dependency

To enable this parameter, select “Enable coverage analysis” on page 2-4.

Command-Line Information

Parameter: `CovEnableCumulative`
Type: Character vector or string
Value: `'on'|'off'`
Default: `'on'`

Include cumulative data in coverage report

Show model coverage results from successive simulations in a single HTML report.

Note For an easier way to accumulate coverage data from multiple simulations, see “Accumulating Coverage Data from the Results Explorer”.

Settings

☑ On

The HTML report shows model coverage data from successive simulations.

☐ Off (default)

The HTML report shows model coverage data from the most recent simulation.

Dependency

To enable this parameter, select:

- “Enable coverage analysis” on page 2-4
- “Enable cumulative data collection” on page 2-24

Command-Line Information
Parameter: `CovCumulativeReport`
Type: Character vector or string
Value: `'on'|'off'`
Default: `'off'`

Save cumulative coverage results in workspace variable

Save model coverage data from successive simulations in a single `cvdata` object in the MATLAB workspace.

You can retrieve coverage information from this variable later. For instance, to retrieve decision coverage information, use the `decisioninfo` function. For the full list of functions, see “Automate Coverage Workflows”.

Note For an easier way to accumulate coverage data from multiple simulations, see “Accumulating Coverage Data from the Results Explorer”.

Settings

☑ On

A single `cvdata` object stores model coverage data from successive simulations. See “Cumulative Coverage Data”.

Specify the variable name using the parameter “Cumulative coverage variable name” on page 2-25.

☐ Off (default)

The `cvdata` object stores model coverage data from the most recent simulation.

Dependency

To enable this parameter, select:

- “Enable coverage analysis” on page 2-4
- “Enable cumulative data collection” on page 2-24

Command-Line Information
Parameter: `CovSaveCumulativeToWorkspaceVar`
Type: Character vector or string
Value: `'on'|'off'`
Default: `'off'`

Cumulative coverage variable name

Specify the name of the `cvdata` object that saves coverage data from successive simulations.

Note For an easier way to accumulate coverage data from multiple simulations, see "Accumulating Coverage Data from the Results Explorer".

Settings

Enter variable name, for instance, `cumulativeCoverageData`.

Dependency

To enable this parameter, select:

- "Enable coverage analysis" on page 2-4
- "Save cumulative coverage results in workspace variable" on page 2-25
- "Enable cumulative data collection" on page 2-24

Command-Line Information
Parameter: `CovCumulativeVarName`
Type: Character vector or string
Value: Name to be given to `cvdata` object
Default: `'covCumulativeData'`

Enable limited coverage analysis of accelerated models

Collect limited coverage for accelerated models. You can collect the following coverage metrics in accelerator mode:

- Stateflow elements — all coverage metrics that are supported for normal mode
- MATLAB function blocks — all coverage metrics that are supported for normal mode
- Other Simulink blocks — only execution coverage

Settings

☑ On (default)

Model coverage will be collected for models in accelerator mode.

Note Only execution coverage can be collected on Simulink blocks when the model simulation mode is set to `accelerator`.

☐ Off

Model coverage will not be collected for models in accelerator mode.

Dependency

To enable this parameter, select "Enable coverage analysis" on page 2-4.

Command-Line Information
Parameter: `CovAccelSimSupport`
Type: Character vector or string
Value: `'on'|'off'`

Default: `'on'`

Classes

slcoverage.BlockSelector class

Package: `slcoverage`

Select blocks for coverage filter

Description

Specify block selection criteria for a filter rule.

Construction

`sel = slcoverage.BlockSelector(type,element)` specifies the type of model elements to create the filter rule for and returns an `slcoverage.BlockSelector` object.

Input Arguments

`type` — Block selector type
`slcoverage.BlockSelectorType` value

Type of model element to select, specified as one of these values:

- `slcoverage.BlockSelectorType.BlockInstance` — An instance of a block.
- `slcoverage.BlockSelectorType.BlockType` — All blocks of the specified block type.
- `slcoverage.BlockSelectorType.Chart` — A Stateflow chart.
- `slcoverage.BlockSelectorType.MaskType` — Blocks that use the specified mask type.
- `slcoverage.BlockSelectorType.State` — A Stateflow state.
- `slcoverage.BlockSelectorType.StateAllContent` — Stateflow state and its contents.
- `slcoverage.BlockSelectorType.StateflowFunction` — A Stateflow function.
- `slcoverage.BlockSelectorType.Subsystem` — A subsystem block.
- `slcoverage.BlockSelectorType.SubsystemAllContent` — A subsystem and its contents.
- `slcoverage.BlockSelectorType.TemporalEvent` — A Stateflow temporal event.
- `slcoverage.BlockSelectorType.Transition` — A Stateflow transition.

Example: `slcoverage.BlockSelectorType.Transition`

`element` — Model element to select
property name | handle | Simulink ID

Model element to select, specified as a property name of the element, its handle, or its Simulink identifier. Use a handle or ID for selector types that select an instance. Use a property name, such as the value of a block's `'BlockType'` property, to select multiple model elements.

Example: `'sldemo_lct_bus:18'`, `'RelationalOperator'`

Attributes:

SetAccess
protected

Data Types: `char` | `string` | `handle` | `integer`

Outputs

`sel` — Selector object

`slcoverage.BlockSelector` object | array of `slcoverage.BlockSelector` objects

Selector object, returned as an `slcoverage.BlockSelector` object or array of `slcoverage.BlockSelector` objects.

Properties

`ConstructorCode` — Code used to create this selector object

character array

Code used to create this selector object, returned as a character vector.

Attributes

SetAccess

protected

`Description` — Description of the selector

character vector

Description of the selector, returned as a character vector. Simulink Coverage creates the description based on the selector.

Attributes

SetAccess

protected

`Id` — Model element identifier

Simulink ID (default) | property | handle

Model element identifier, specified as the property name of the element, the handle to an element, or the Simulink identifier of the element. Use a handle or ID for selector types that select an instance. Use a property name, such as the value of the `'BlockType'` property of a block, to select multiple model elements.

Attributes

SetAccess

protected

Data Types: `char` | `string` | `handle` | `integer`

`Type` — Block selector type

`slcoverage.BlockSelectorType` value

This property is read-only.

Selector type, returned as one of these `slcoverage.BlockSelectorType` values:

- `BlockInstance`

- `BlockType`
- `Chart`
- `MaskType`
- `State`
- `StateAllContent`
- `StateflowFunction`
- `Subsystem`
- `SubsystemAllContent`
- `TemporalEvent`
- `Transition`

Methods

allSelectors	Selectors for model or code element

Copy Semantics

Handle. To learn how handle classes affect copy operations, see Copying Objects.

Examples

Add Block Selector Rules to a Filter

Select multiple blocks to add a rule for and an instance of a block to add a rule for. The resulting filter has two rules. You can simulate your model for code coverage using the filter to see the effect.

Open the model. Specify coverage settings and turn on coverage recording.

```
modelName = 'sldemo_lct_bus';
open_system(modelName);
set_param(modelName,'CovMetricStructuralLevel','MCDC','RecordCoverage','on');
```

Select blocks that have the same block type as the `upper GE input` block to add a filter rule for.

```
type = get_param('sldemo_lct_bus/slCounter/upper GE input','BlockType');
bl = slcoverage.BlockSelector(slcoverage.BlockSelectorType.BlockType,type);
```

Create a filter object, create a rule based on the selector, and add the rule to the filter.

```
filt = slcoverage.Filter;
rule = slcoverage.FilterRule(bl,'Tested elsewhere',slcoverage.FilterMode.Exclude);
filt.addRule(rule);
```

Select a block instance and add a rule for the block instance to the filter. This rule uses the default filter mode of `Justify`.

```
id = Simulink.ID.getSID('sldemo_lct_bus/slCounter/And');
bl = slcoverage.BlockSelector(slcoverage.BlockSelectorType.BlockInstance,id);
rule = slcoverage.FilterRule(bl,'Edge case');
filt.addRule(rule);
```

Save the filter as `blfilter`. Simulate the model for code coverage. Add the filter file as the value to the `filter` property of the resulting `cvdata` object. Then generate the coverage report.

```
filt.save('blfilter');
csim = cvsim(modelName);
csim.filter = 'blfilter';
cvhtml('cov',csim);
```

Examine the HTML report to see information about the blocks that you added rules for.

See Also

`cv.cvdatagroup` | `getSimulinkBlockHandle` | `slcoverage.Filter` | `slcoverage.FilterRule` | `slcoverage.MetricSelector` | `slcoverage.SFcnSelector`

Topics

“Top-Level Model Coverage Report”
“Simulink Identifiers”
“Create, Edit, and View Coverage Filter Rules”

Introduced in R2017b

slcoverage.CodeSelector class

Package: `slcoverage`

Select custom C or C++ code for coverage filter

Description

Use objects of the `slcoverage.CodeSelector` class to specify custom C or C++ code selection criteria for a filter rule.

The `slcoverage.CodeSelector` class is a `handle` class.

Creation

`sel = slcoverage.CodeSelector(type,fileName)` creates `CodeSelector` object of the specified `type` based on the specified `fileName` and sets the `Type` and `FileName` properties.

`sel = slcoverage.CodeSelector(type,fileName,functionName)` creates a `CodeSelector` object based on the specified C or C++ `functionName` in the file and sets the `FunctionName` property.

`sel = slcoverage.CodeSelector(type,fileName,functionName,expr,exprIndex)` creates a `CodeSelector` object for the specified expression and expression index and sets the `Expr` and `ExprIndex` properties.

`sel = slcoverage.CodeSelector(type,fileName,functionName,expr,exprIndex,outcomeIndex)` creates a `CodeSelector` object based on the specified coverage outcome and sets the `OutcomeIndex` property.

`sel = slcoverage.CodeSelector(type,fileName,functionName,expr,exprIndex,outcomeIndex,parentExprIndex)` creates a `CodeSelector` object based on the specified coverage outcome that belongs to an expression owned by `parentExprIndex` and sets the `DecOrCondIndex` property to `parentExprIndex`.

Properties

Type — Type of custom C or C++ code to select

`slcoverage.CodeSelectorType` value

Type of custom C or C++ code to select, specified as an enumeration of the `slcoverage.CodeSelectorType` class:

Example: `slcoverage.CodeSelectorType.Function`

Attributes

SetAccess	protected

Data Types: `slcoverage.CodeSelectorType`

`FileName` — C or C++ file to select

character array | string array

C or C++ file to select, specified as a character array or string array.

Example: `'myfile.c'`

Attributes

SetAccess	protected

Data Types: `char` | `string`

`FunctionName` — C or C++ function to select

character array | string array

C or C++ function to select, specified as a character array or string array.

Example: `'counterbusFcn'`

Attributes

SetAccess	protected

Data Types: `char` | `string`

`Expr` — Decision or condition expression to select

character array | string array

Decision or condition expression to select, specified as a character array or string array.

Example: `'x | y'`

Attributes

SetAccess	protected

Data Types: `char` | `string`

`ExprIndex` — Expression index

integer

Expression index, specified as an integer. If you are filtering an outcome, this property is the index of the expression that owns that outcome. If you are filtering an expression, this property is the index of that expression inside the body of the function.

Example: 2

Attributes

SetAccess	protected

Data Types: `single` | `double` | `int`

`OutcomeIndex` — Index of outcome to select
integer

Index of outcome to select, specified as an integer:

Example: 2

Attributes

`SetAccess` `protected`

Data Types: `single` | `double` | `int`

`DecOrCondIndex` — Parent expression index
integer

Parent expression index, specified as an integer. Use this input when you are filtering an expression owned by a parent decision or condition. This property is the index of the parent decision or condition relative to the function.

Example: 2

Attributes

`SetAccess` `protected`

Data Types: `single` | `double` | `int`

`ConstructorCode` — Code used to create this selector object
character array

Code used to create this selector object, returned as a character vector.

Attributes

`SetAccess` `protected`

`Description` — Description of the selector
character vector

Description of the selector, returned as a character vector. Simulink Coverage creates the description based on the selector.

Attributes

`SetAccess` `protected`

`Id` — Model element identifier
Simulink ID (default) | property | handle

This property is empty for the `slcoverage.CodeSelector` class.

Attributes

`SetAccess` `protected`

Data Types: `char` | `string` | `handle` | `integer`

Methods

Public Methods

allSelectors Selectors for model or code element

Examples

Add Code Selector Rules to a Filter

This example shows how to select custom C or C++ code for which you want to add a filter rule.

Load the model.

```
modelName = 'slcovCCallerExample';
Simulink.importExternalCTypes('my_func.h','EnumClass','dynamic');
load_system(modelName)
```

Configure coverage settings using a `Simulink.SimulationInput` object.

```
covSet = Simulink.SimulationInput(modelName);
covSet = covSet.setModelParameter('CovEnable','on');
covSet = covSet.setModelParameter('CovMetricStructuralLevel','MCDC');
covSet = covSet.setModelParameter('CovSFcnEnable','on');
covSet = covSet.setModelParameter('CovSaveSingleToWorkspaceVar','on');
covSet = covSet.setModelParameter('CovSaveName','covData');
covSet = covSet.setModelParameter('SimAnalyzeCustomCode','on');
```

Simulate the model using `covSet` object as the input.

```
simOut = sim(covSet);
covData = simOut.covData;
```

Create a selector object to filter the custom C function `timesK`.

```
sel = slcoverage.CodeSelector(slcoverage.CodeSelectorType.Function, 'my_func.c', 'timesK');
```

Create a filter object and create a rule based on the selector, then add the rule to the filter.

```
filt = slcoverage.Filter;
rule = slcoverage.FilterRule(sel,'Tested elsewhere',...
           slcoverage.FilterMode.Exclude);
addRule(filt,rule);
setFilterName(filt,'Code Filter')
```

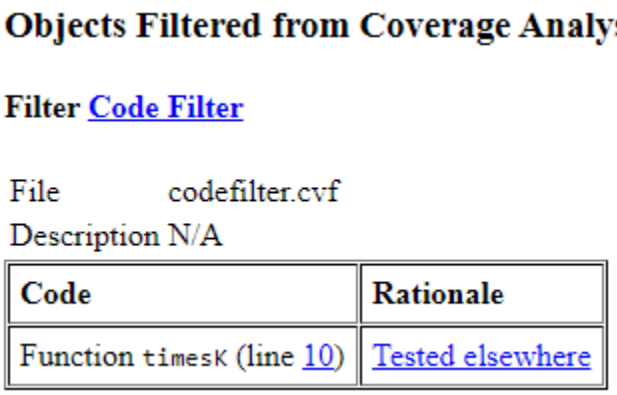
Save the filter as `codefilter` and add it to the `cvdata` object for `my_func.c`. Because the coverage data is stored in a `cv.cvdatagroup` object, use the `get` method to set the property.

```
save(filt,'codefilter');
covData.get('my_func.c').filter = 'codefilter';
```

Generate a coverage report.

```
cvhtml('codeCovReport',covData)
```

Review the report. Under **Custom Code File(s)**, click `my_func.c` and find the filter rule that you added under Objects Filtered from Coverage Analysis.



Objects Filtered from Coverage Analysis

Filter Code Filter

File codefilter.cvf
Description N/A

Code	Rationale
Function timesK (line 10)	Tested elsewhere

Create a C Code Outcome Selector

This example shows how to use an `slcoverage.CodeSelector` object to filter a code outcome in a custom C or C++ program called by a C Caller block.

Open the Model and Enable Coverage Analysis

Open the model.

```
modelName = 'slcovCCallerExample';
Simulink.importExternalCTypes('my_func.h','EnumClass','dynamic');
load_system(modelName)
```

Configure coverage settings using a `Simulink.SimulationInput` object.

```
covSet = Simulink.SimulationInput(modelName);
covSet = covSet.setModelParameter('CovEnable','on');
covSet = covSet.setModelParameter('CovMetricStructuralLevel','ConditionDecision');
covSet = covSet.setModelParameter('CovSFcnEnable','on');
covSet = covSet.setModelParameter('CovSaveSingleToWorkspaceVar','on');
covSet = covSet.setModelParameter('CovSaveName','covData');
```

Simulate the model using `covSet` as the input.

```
simOut = sim(covSet);
covData = simOut.covData;
```

The simulation returns the coverage data as a `cv.cvdatagroup` object when both the model and custom code are analyzed for coverage. To extract the code coverage data, use the `get` method of the `cvdatagroup` class.

```
codeCovData = get(covData,'my_func.c');
```

Justify the Missing Outcome

In this example, you justify the `F` outcome of the `inputGElower` condition in the `(u1->limits.upper_saturation_limit >= limit) && inputGElower` decision, which is located inside the `counterbusFcn` function in the `my_func.c` source file.

```
35
36  void counterbusFcn(const COUNTERBUS *u1, int32_T u2, COUNTERBUS *y1, int32_T *y2)
37  {
38      int32_T limit;
39      boolean_T inputGElower;
40      limit = u1->inputsignal.input + u2;
41      inputGElower = (limit >= u1->limits.lower_saturation_limit);
42      if((u1->limits.upper_saturation_limit >= limit) && inputGElower) {
43          *y2 = limit;
44      } else {
45          if(inputGElower) {
46              limit = u1->limits.upper_saturation_limit;
47          } else {
48            limit = u1->limits.lower_saturation_limit;
49          }
50          *y2 = limit;
51      }
52      y1->inputsignal.input = *y2;
53      y1->limits = u1->limits;
54  }
```

Create a selector object using `slcoverage.CodeSelector`. The first input is a `CodeSelectorType` enumeration. To justify a condition outcome, use a `ConditionOutcome` enumeration. The second input is the code source file, `my_func.c`. The third input is the name of the function that contains the outcome, `counterbusFcn`. The fourth input is the expression which contains the outcome, `(u1->limits.upper_saturation_limit >= limit) && inputGElower`. The fifth input is the index of the expression that owns the outcome. In this case, `inputGElower` is the second condition within its parent condition, so this input is 2. The sixth input is the condition outcome index, which is 1 for the F outcome of a Boolean expression. The seventh input is the index of the parent decision or condition, which is 1 for `(u1->limits.upper_saturation_limit >= limit) && inputGElower` because it is the first decision in the function.

```
enum = slcoverage.CodeSelectorType.ConditionOutcome;
sel = slcoverage.CodeSelector(enum,'my_func.c','counterbusFcn','(u1->limits.upper_saturation_lim
```

Create a `Filter` object and a `FilterRule` object and apply the rule to the filter.

```
filt = slcoverage.Filter;
rule = slcoverage.FilterRule(sel,'condition does not apply');
addRule(filt,rule);
```

Save the filter to a filter file and then apply the filter to the `cvdata` object.

```
save(filt,'codeOutcomeFilter');
codeCovData.filter = 'codeOutcomeFilter';
```

Review the Coverage Report

Verify the outcome is filtered by generating a coverage report using `cvhtml`.

```
cvhtml('filteredCodeCovReport',codeCovData)
```

6. Function counterbusFcn (line 36)

Justify or Exclude
File: my_func.c (code)
Uncovered Links:
Covered expressions: (u1->limits.upper_saturation_limit >= limit) && inputGElower (line 42)
inputGElower (line 45)
limit >= u1->limits.lower_saturation_limit (line 41)

Metric	Coverage
Cyclomatic Complexity	3
Decision	75% (3/4) decision outcomes
Condition	83% ((4+1)/6) condition outcomes
Statement	92% (12/13) covered statements (1 Function entry + 12 executable statements)

6.1. Decision/Condition (u1->limits.upper_saturation_limit >= limit) && inputGElower (line 42)

Justify or Exclude
Function: counterbusFcn
Uncovered Links:

Metric	Coverage
Decision	100% (2/2) decision outcomes
Condition	100% ((3+1)/4) condition outcomes

Conditions analyzed

Description	True	False
u1->limits.upper_saturation_limit >= limit	90	11
inputGElower	90	J1

See Also

`cv.cvdatagroup` | `cvdata` | `slcoverage.Filter` | `slcoverage.FilterRule` | `slcoverage.MetricSelector` | `slcoverage.SFcnSelector`

Topics

“Top-Level Model Coverage Report”
“Create, Edit, and View Coverage Filter Rules”

Introduced in R2018b

slcoverage.Filter class

Package: `slcoverage`

Coverage filter set

Description

Use an object of the `slcoverage.Filter` class to filter out unsatisfied coverage objectives by creating rules using the `slcoverage.FilterRule` class.

The `slcoverage.Filter` class is a `handle` class.

Creation

Description

`filt = slcoverage.Filter()` creates an `slcoverage.Filter` object.

`filt = slcoverage.Filter(filterFile)` creates an `slcoverage.Filter` object that contains the filter rules saved in `filterFile`.

Input Arguments

`filterFile` — Filter file
path name

Filter file in CVF format, specified as a character array of the path name to the file, or a cell array of character arrays. You do not need to include the file extension.

Example: `'myfilt'`, `'filters\myfilt'`, `{'myfilt1', 'myfilt2'}`

Methods

Public Methods

addRule	Add coverage filtering rule to filter
removeRule	Remove rule from filter rule set
rules	Rules for filter
setFilterName	Set name of coverage filter object
filterName	Get name of coverage filter object
setFilterDescription	Set description of coverage filter object
filterDescription	Get description of coverage filter object
save	Save coverage filter object to coverage filter file

Examples

Add Rule to a Filter File

This example shows how to add a rule to a coverage filter file.

Create a filter object and use the `BlockSelector` class to create a `BlockSelector` object for the Saturation block in the `slvnvdemo_covfilt` model.

```
filt = slcoverage.Filter;
blockSel = slcoverage.BlockSelector(slcoverage.BlockSelectorType.BlockInstance,...
           'slvnvdemo_covfilt/Saturation');
```

Create a rule to filter the Saturation block using `slcoverage.FilterRule` with the selector as the first input and the rationale as the second input.

```
rule = slcoverage.FilterRule(blockSel,'Edge case');
```

Use the `addRule` method of the `slcoverage.Filter` class to add the rule to the filter object.

```
addRule(filt,rule);
```

Save the filter with the new rule to a filter file using the `save` method of the `slcoverage.Filter` class.

```
save(filt,'myFilterFile')
```

Create and Use a Coverage Filter Object

This example shows how to create a filter object and add a rule to exclude a subsystem from coverage analysis.

Open the `slvnvdemo_covfilt` model. Use a `SimulationInput` object to enable coverage recording and specify coverage settings.

```
modelName = 'slvnvdemo_covfilt';
load_system(modelName)
simIn = Simulink.SimulationInput(modelName);
simIn = setModelParameter(simIn,'CovEnable','on');
simIn = setModelParameter(simIn,'CovMetricStructuralLevel','MCDC');
simIn = setModelParameter(simIn,'CovSaveSingleToWorkspaceVar','on');
simIn = setModelParameter(simIn,'CovSaveName','covData');
```

Simulate the model. The coverage data is stored in the value supplied for the `CovSaveName` property.

```
simOut = sim(simIn);
covData = simOut.covData;
```

You can filter a block by using the `slcoverage.BlockSelector` class. To exclude the Switchable config subsystem, use `slcoverage.BlockSelectorType.SubsystemAllContent` as the first input.

```
subsysSel = slcoverage.BlockSelector(...
                      slcoverage.BlockSelectorType.SubsystemAllContent,...
                      'slvnvdemo_covfilt/Switchable config');
```

Create the filter rule by passing the selector, rationale, and the exclude filter mode as inputs.

```
rule = slcoverage.FilterRule(subsysSel,...
                    'Unused configuration',...
                    slcoverage.FilterMode.Exclude);
```

Create an `slcoverage.Filter` object and then add the rule to it.

```
filt = slcoverage.Filter;
addRule(filt,rule);
```

Save the filter to a file called `blockFilter.cvf`. To create a report that uses this coverage filter, add the filter file as the value to the `filter` property of `covData`, and create a report called `coverageData.html` using `covData`.

```
save(filt,'blockFilter')
covData.filter = 'blockFilter';
cvhtml('coverageData',covData)
```

Use Multiple Coverage Filter Files for a Simulation

If you have multiple filter files that each contain their own set of rules, you can apply them to a coverage result set by creating a cell array of the filter file names or path names. In this example, you apply two filter files to a single `cvdata` object and then view the report to see that the filters are applied.

Load the `slvnvdemo_covfilt` model into memory.

```
modelName = 'slvnvdemo_covfilt';
load_system(modelName)
```

Set the coverage settings by using a `Simulink.SimulationInput` object and simulate the model using `sim`.

```
simIn = Simulink.SimulationInput(modelName);
simIn = setModelParameter(simIn,'CovEnable','on');
simIn = setModelParameter(simIn,'CovMetricStructuralLevel','MCDC');
simIn = setModelParameter(simIn,'CovSaveSingleToWorkspaceVar','on');
simIn = setModelParameter(simIn,'CovSaveName','covData');
simOut = sim(simIn);
```

Apply the two filters to the `cvdata` object by assigning them to the filter property as a cell array.

```
covData.filter = {'filter_1','filter_2'};
cvhtml('twoFiltersCovData',covData);
```

You can see the applied coverage filters in the **Objects Filtered from Coverage Analysis** section of the coverage report.

See Also

`slcoverage.BlockSelector` | `slcoverage.FilterRule` | `slcoverage.MetricSelector` | `slcoverage.SFcnSelector`

Topics

“Create, Edit, and View Coverage Filter Rules”

Introduced in R2017b

slcoverage.FilterRule class

Package: `slcoverage`

Create coverage filtering rule

Description

Create a coverage filtering rule that includes the selector and the rationale for filtering.

Construction

`rule = slcoverage.FilterRule(selector,rationale)` creates the filter rule object `rule` using the specified selector and rationale text.

`rule = slcoverage.FilterRule(selector,rationale,mode)` specifies whether the filter mode for this rule is justify or exclude. You can use only justify (the default) with metric selectors.

Input Arguments

`selector` — Selector for this rule
`slcoverage.BlockSelector` object | `slcoverage.MetricSelector` object | `slcoverage.SFcnSelector` object

Selector that determines the objects that this rule applies to, specified as an `slcoverage.BlockSelector` object, `slcoverage.MetricSelector` object, or `slcoverage.SFcnSelector` object.

`rationale` — Reason for adding the rule
character vector or string

Reason for adding the rule, specified as a character vector or string.

Example: `'value is never less than 0'`

`mode` — Filter mode
`slcoverage.FilterMode.Justify` (default) | `slcoverage.FilterMode.Exclude`

Filter mode for this rule, specified as `slcoverage.FilterMode.Justify` or `slcoverage.FilterMode.Exclude`.

Properties

`Mode` — Filter mode
`Justify` (default) | `Exclude`

This property is read-only.

Filter mode that was specified for this rule, returned as `Justify` or `Exclude`.

`Rationale` — Rationale text specified for this rule
character vector or string

This property is read-only.

Rationale text specified for this rule, returned as a character vector.

`Selector` — Selector object for this rule
`slcoverage.BlockSelector` object | `slcoverage.MetricSelector` object | `slcoverage.SFcnSelector` object

This property is read-only.

Selector object for this rule, returned as a `slcoverage.BlockSelector` object, `slcoverage.SFcnSelector` object, or `slcoverage.SFcnSelector` object.

Copy Semantics

Handle. To learn how handle classes affect copy operations, see Copying Objects.

Examples

Create Rule That Uses a Block Selector

Create a block selector object and a rule for it. Then add the rule to a filter.

Open the model. Specify coverage settings and turn on coverage recording.

```
modelName = 'sldemo_lct_bus';
open_system(modelName);
set_param(modelName,'CovMetricStructuralLevel','MCDC','RecordCoverage','on');
```

Select blocks with block type `'RelationalOperator'` to add a filter rule for.

```
bl = slcoverage.BlockSelector(slcoverage.BlockSelectorType.BlockType,'RelationalOperator');
```

Create a filter object, create a rule, and add the rule to the filter. This rule excludes the selection from coverage analysis.

```
filt = slcoverage.Filter;
rule = slcoverage.FilterRule(bl,'Tested elsewhere',slcoverage.FilterMode.Exclude);
filt.addRule(rule);
```

See Also

`slcoverage.BlockSelector` | `slcoverage.Filter` | `slcoverage.MetricSelector` | `slcoverage.SFcnSelector`

Topics
“Coverage Filter Rules and Files”
“Create, Edit, and View Coverage Filter Rules”

Introduced in R2017b

slcoverage.MetricSelector class

Package: `slcoverage`

Select metric criterion for coverage filter

Description

Use an object of the `slcoverage.MetricSelector` class to specify metric selection criteria for a coverage filter rule.

The `slcoverage.MetricSelector` class is a `handle` class.

Construction

`sel = slcoverage.MetricSelector(type,element,objIndex,outIndex)` creates a metric selector object of type `type` for the specified model element `element` at the objective index `objIndex` and outcome index `outIndex`.

You can only create a justify rule for a metric selector. For more information about the difference between justification and exclusion, see “Coverage Filtering”.

For more information on the condition and decision coverage tables produced in the report, see “Top-Level Model Coverage Report”.

Input Arguments

`type` — Metric selector type

`slcoverage.MetricSelectorType.ConditionOutcome` | `slcoverage.MetricSelectorType.DecisionOutcome` | `slcoverage.MetricSelectorType.MCDCOutcome` | `slcoverage.MetricSelectorType.RelationalBoundaryOutcome` | `slcoverage.MetricSelectorType.SaturationOverflowOutcome`

Metric selector type, specified as:

- `slcoverage.MetricSelectorType.ConditionOutcome` objects select condition metric objective outcomes.
- `slcoverage.MetricSelectorType.DecisionOutcome` objects select decision metric objective outcomes.
- `slcoverage.MetricSelectorType.MCDCOutcome` objects select MCDC metric objective outcomes.
- `slcoverage.MetricSelectorType.RelationalBoundaryOutcome` objects select outcome metrics related to relational boundary outcomes.
- `slcoverage.MetricSelectorType.SaturationOverflowOutcome` objects select outcome metrics related to saturation on integer overflow outcomes.

`element` — Model element to select

handle | Simulink ID

Model element to select, specified as a handle or the Simulink identifier of the model element.

Example: `'sldemo_lct_bus:18'`

`objIndex` — Index of objective
integer

Index of the objective that you want to filter, specified as an integer.

Example: 1

`outIndex` — Index of outcome
integer

Index of the outcome that you want to filter, specified as an integer.

Example: 2

Properties

`ConstructorCode` — Code used to create this selector object
character array

Code used to create this selector object, returned as a character vector.

Attributes

SetAccess
: protected

`Description` — Description of the selector
character vector

Description of the selector, returned as a character vector. Simulink Coverage creates the description based on the selector.

Attributes

SetAccess
: protected

`Id` — Element identifier
Simulink ID (default) | handle

This property is read-only.

Identifier of the model element, returned as character vector of the Simulink ID or a handle.

`ObjectiveIndex` — Index of objective
integer

This property is read-only.

Index of the objective for this selector, returned as an integer.

`OutcomeIndex` — Index of outcome
integer

This property is read-only.

Index of the outcome for this selector, returned as an integer.

Type — Metric selector type
`ConditionOutcome` | `DecisionOutcome` | `MCDCOutcome` | `RelationalBoundaryOutcome` | `SaturationOverflowOutcome`

This property is read-only.

Selector type, returned as `ConditionOutcome`, `DecisionOutcome`, `MCDCOutcome`, `RelationalBoundaryOutcome`, or `SaturationOverflowOutcome`.

Outputs

`sel` — Selector object
`slcoverage.MetricSelector` object | array of `slcoverage.MetricSelector` objects

Selector object, returned as an `slcoverage.MetricSelector` object or array of `slcoverage.MetricSelector` objects.

Methods

Public Methods

allSelectors	Selectors for model or code element

Examples

Add Metric Selector Rule to a Filter

This example shows how to select a metric and add a rule that uses that metric. In this example, you create a rule to justify an unsatisfied decision for a Saturation block.

Open the Model and Enable Coverage Analysis

Load the model into memory.

```
modelName = 'slvnvdemo_covfilt';
load_system(modelName);
```

Use a `Simulink.SimulationInput` object to configure coverage for the model.

```
covSet = Simulink.SimulationInput(modelName);
covSet = covSet.setModelParameter('CovEnable','on');
covSet = covSet.setModelParameter('CovMetricStructuralLevel','MCDC');
covSet = covSet.setModelParameter('CovSFcnEnable','on');
covSet = covSet.setModelParameter('StopTime','20');
covSet = covSet.setModelParameter('CovSaveSingleToWorkspaceVar','on');
covSet = covSet.setModelParameter('CovSaveName','covData');
```

Simulate the model using the `SimulationInput` object as the input.

```
simOut = sim(covSet);
```

View the coverage results before applying a filter. You can access the coverage using `decisioninfo`, or you can view the HTML report using `cvhtml`.

```
covInitial = decisioninfo(covData,[modelName,'/Saturation']);
percentInitial = 100 * covInitial(1)/covInitial(2)
```

```
percentInitial =

    50
```

```
cvhtml('covReportInitial',covData)
```

Saturate block "Saturation"

Justify or Exclude

Parent: /slvnvdemo_covfilt

Uncovered Links:

Metric	Coverage
Cyclomatic Complexity	2
Decision	50% (2/4) decision outcomes
Execution	100% (1/1) objective outcomes

Decisions analyzed

input > lower limit	50%
false	0/201
true	201/201
input >= upper limit	50%
false	201/201
true	0/201

Both `decisioninfo` and `cvhtml` show the same result of 50% decision coverage. If you don't intend your current tests to exercise this outcome, you can justify the outcome so it is no longer reported as missing coverage.

In this example, we justify the `false` decision outcome of the `input > lower limit` decision objective in the Saturation block.

Justify the Missing Condition Objective

`MetricSelector` objects accept the block path or the block handle as the second input. Get the block handle of the Saturation block by using `getSimulinkBlockHandle`.

```
id = getSimulinkBlockHandle([modelName,'/Saturation']);
```

Because the objective being justified is a decision outcome, the first input to the metric selector constructor is `slcoverage.MetricSelectorType.DecisionOutcome`. The second input is the block handle. The last two are the index of the objective to justify and the index of the outcome of that objective, respectively.

Because the `input > lower limit` decision objective is the first objective for the Saturation block, its objective index is `1`. Because the `false` outcome of this objective is the first outcome, its outcome index is also `1`. Therefore, the last two inputs are `1,1`.

```
metr = slcoverage.MetricSelector(slcoverage.MetricSelectorType.DecisionOutcome,id,1,1);
```

Create a filter and rule. In this case, we use the default filter mode of justify. Then add the rule to the filter using the `addRule` method.

```
filt = slcoverage.Filter;
rule = slcoverage.FilterRule(metr,'Expected result');
filt.addRule(rule);
```

Save the filter to a filter file using the `save` method. Then apply the filter file to the `cvdata` object by assigning the `filter` property to the new filter file.

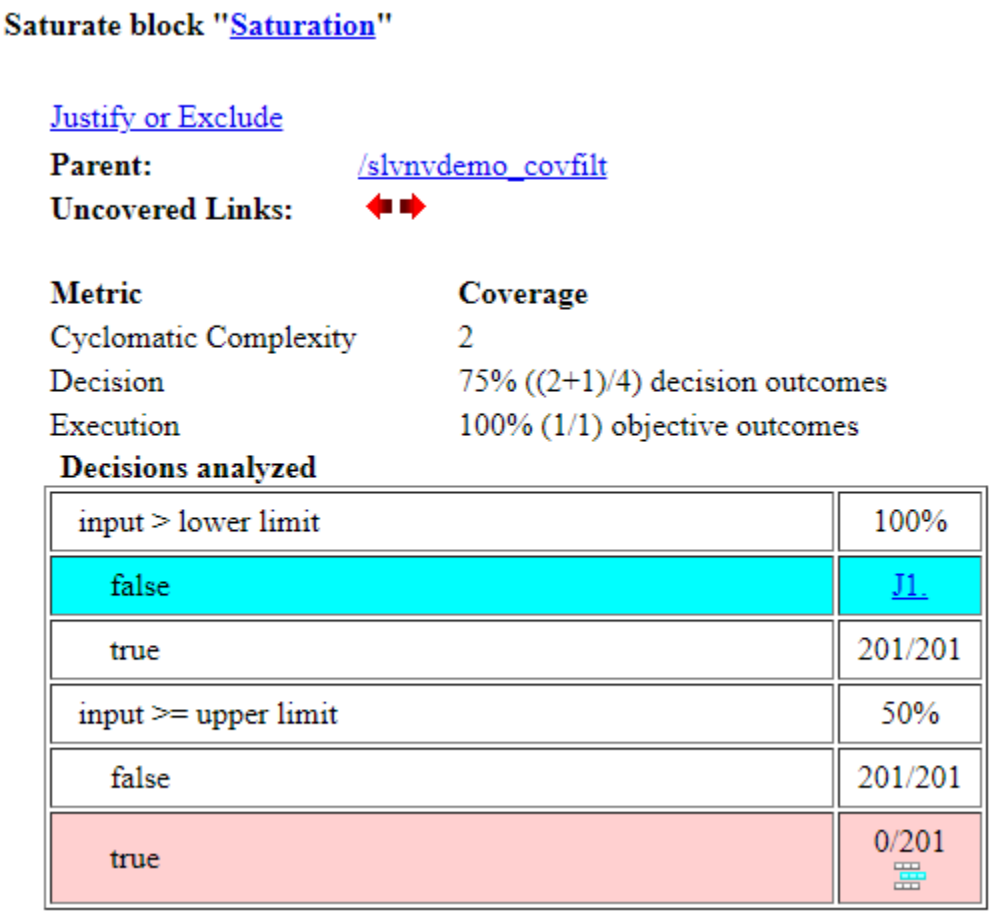
```
filt.save('metrfilter');
covData.filter = 'metrfilter';
```

Re-generate the coverage results for the Saturation block using the new filtered `cvdata` object.

```
covFiltered = decisioninfo(covData,[modelName,'/Saturation']);
percentInitial = 100 * covFiltered(1)/covFiltered(2)
```

```
percentInitial =

    75
```

```
cvhtml('covReportFiltered',covData)
```



Saturate block "Saturation"

Justify or Exclude

Parent: /slvnvdemo_covfilt

Uncovered Links:

Metric	Coverage
Cyclomatic Complexity	2
Decision	75% ((2+1)/4) decision outcomes
Execution	100% (1/1) objective outcomes

Decisions analyzed

input > lower limit	100%
false	J1.
true	201/201
input >= upper limit	50%
false	201/201
true	0/201

In the HTML report, the missing decision outcome is highlighted to indicate that it is justified. Decision coverage for the Saturation block is now 75%.

See Also

`getSimulinkBlockHandle` | `slcoverage.BlockSelector` | `slcoverage.Filter` | `slcoverage.FilterRule` | `slcoverage.SFcnSelector`

Topics

“Top-Level Model Coverage Report”
“Simulink Identifiers”
“Create, Edit, and View Coverage Filter Rules”

Introduced in R2017b

slcoverage.Selector class

Package: `slcoverage`

Get selectors of all types

Description

Use the `slcoverage.Selector` class with the `allSelectors` method to return all types of the selectors for a block.

Properties

`ConstructorCode` — Code used to create this selector object

character array

Code used to create this selector object, returned as a character vector.

Attributes

SetAccess

protected

`Description` — Description of the selector

character vector

Description of the selector, returned as a character vector. Simulink Coverage creates the description based on the selector.

Attributes

SetAccess

protected

`Id` — Element identifier

Simulink ID (default) | handle

This property is read-only.

Identifier of the model element, returned as character vector of the Simulink ID or a handle.

Type — Selector type

selector type value

This property is read-only.

Selector type, returned as a selector type of the corresponding selector.

Methods

allSelectors	Selectors for model or code element

Copy Semantics

Handle. To learn how handle classes affect copy operations, see Copying Objects.

Examples

Get All Selectors

This example shows how to get all the selectors for an And block and then add a rule to justify or exclude a selector. Metric selectors can only be justified.

Load the model and set coverage settings

```
modelName = 'slvnvdemo_covfilt';
load_system(modelName);
set_param(modelName,'CovEnable','on','CovMetricStructuralLevel','MCDC');
```

First, get the block handle for the And block.

```
id = getSimulinkBlockHandle([modelName,'/Saturation']);
```

Get the selectors using the block handle.

```
sel = slcoverage.Selector.allSelectors(id)
```

```
sel =

  1x10 heterogeneous Selector (BlockSelector, MetricSelector) array with properties:

    Description
    Type
    Id
    ConstructorCode
```

The block has ten selectors. You can index into each one to see their contents. In this example, you want to justify the sixth selector.

```
sel(6)
```

```
ans =

  MetricSelector with properties:

     ObjectiveIndex: 2
       OutcomeIndex: 2
        Description: 'T outcome of input >= upper limit in Saturate block "Saturation"'
               Type: DecisionOutcome
                 Id: 'slvnvdemo_covfilt:5'
    ConstructorCode: 'slcoverage.MetricSelector(slcoverage.MetricSelectorType.DecisionOutcome, '
```

Create a justify rule, then create a filter object and add the rule to it.

```
rule = slcoverage.FilterRule(sel(6),'Expected result');
filt = slcoverage.Filter;
filt.addRule(rule);
```

Save the filter and generate a coverage report.

```
filt.save('metrfilter');
csim = cvsim(modelName);
csim.filter = 'metrfilter';
cvhtml('cov',csim,'-sRT=0');
```

See Also

`slcoverage.BlockSelector` | `slcoverage.CodeSelector` | `slcoverage.MetricSelector` | `slcoverage.SFcnSelector`

Topics

“Create, Edit, and View Coverage Filter Rules”

Introduced in R2017b

slcoverage.SFcnSelector class

Package: `slcoverage`

Select S-function criterion for filtering rule

Description

Use objects of the `slcoverage.SFcnSelector` class to specify the S-function selection criteria for a filter rule.

The `slcoverage.SFcnSelector` class is a `handle` class.

Creation

`sel = slcoverage.SFcnSelector(type,id)` creates an `SFcnSelector` object of the specified `type` based on S-Function block `id` and sets the `Type` and `Id` properties.

`sel = slcoverage.SFcnSelector(type,id,fileName)` creates an `SFcnSelector` object based on the specified C or C++ `fileName`, and sets the `FileName` property.

`sel = slcoverage.SFcnSelector(type,id,fileName,functionName)` creates an `SFcnSelector` object based on the specified C or C++ `functionName` in the specified file and sets the `FunctionName` property.

`sel = slcoverage.SFcnSelector(type,id,fileName,functionName,expr,exprIndex)` creates an `SFcnSelector` object based on the specified expression and `exprIndex` and sets the `Expr` and `ExprIndex` properties.

`sel = slcoverage.SFcnSelector(type,id,fileName,functionName,expr,exprIndex,outcomeIndex)` creates an `SFcnSelector` object based on the specified coverage outcome and sets the `OutcomeIndex` property.

`sel = slcoverage.SFcnSelector(type,id,fileName,functionName,expr,exprIndex,outcomeIndex,parentExprIndex)` creates an `SFcnSelector` object based on the specified coverage outcome whose parent is another expression specified by `parentExprIndex` and sets the `DecOrCondIndex` property.

Properties

Type — Type of S-function

`slcoverage.SFcnSelectorType` enumeration

Type of S-function to select, specified as an enumeration of the `slcoverage.SFcnSelectorTjype` class:

Data Types: `slcoverage.SFcnSelectorType`

`Id` — Model element identifier
Simulink ID (default) | property | handle

Model element identifier, specified as the property name of the element, the handle to an element, or the Simulink identifier of the element. Use a handle or ID for selector types that select an instance. Use a property name, such as the value of the `'BlockType'` property of a block, to select multiple model elements.

Attributes

SetAccess	protected

Data Types: `char` | `string` | `handle` | `integer`

`FileName` — C or C++ file to select
`character array` | `string array`

C or C++ file to select, specified as a character array or string array.

Example: `'myfile.c'`

Attributes:

SetAccess	protected

Data Types: `char` | `string`

`FunctionName` — C or C++ function to select
`character array` | `string array`

C or C++ function to select, specified as a character array or string array.

Example: `'counterbusFcn'`

Attributes:

SetAccess	protected

Data Types: `char` | `string`

`Expr` — Expression to select
`character array` | `string array`

Expression to select, specified as a character array or string array.

Example: `'inputGElower'`

Attributes:

SetAccess	protected

Data Types: `char` | `string`

`ExprIndex` — Expression index
scalar

Expression index, specified as an integer. If you are filtering an outcome, this property is the index of the expression that owns that outcome. If you are filtering an expression, this property is the index of that expression inside the body of the function.

Example: 2

Attributes:

SetAccess	protected

Data Types: `single` | `double` | `int`

`OutcomeIndex` — Index of outcome to select

integer

Index of outcome to select, specified as an integer:

Example: 2

Attributes

SetAccess	protected

Data Types: `single` | `double` | `int`

`DecOrCondIndex` — Parent expression index

integer

Parent expression index, specified as an integer. Use this input to filter an expression that is owned by a parent decision or condition; this is the index of the parent decision or condition relative to the function.

Example: 2

Attributes

SetAccess	protected

Data Types: `single` | `double` | `int`

`Description` — Description of the selector

character vector

Description of the selector, returned as a character vector. Simulink Coverage creates the description based on the selector.

Attributes

SetAccess	protected

`ConstructorCode` — Code used to create this selector object

character array

Code used to create this selector object, returned as a character vector.

Attributes

SetAccess	protected

Methods

Public Methods

allSelectors Selectors for model or code element

Examples

Create an S-Function Selector

This example shows how to create an S-Function selector.

Load the model by using `load_system`.

```
load_system('slvnvdemo_covfilt');
```

Create an S-Function selector by using `slcoverage.SFcnSelector`. To select the S-Function based on its name, enter `slcoverage.SFcnSelectorType.SFcnName` as the first input. The second input is the path to the S-Function.

```
sel = slcoverage.SFcnSelector(slcoverage.SFcnSelectorType.SFcnName,...
                          'slvnvdemo_covfilt/Mode Logic/ SFunction ')
```

```
sel =

  SFcnSelector with properties:

           FileName: ''
       FunctionName: ''
               Expr: ''
       OutcomeIndex: []
    DecOrCondIndex: []
        Description: 'N/A'
               Type: SFcnName
                 Id: 'slvnvdemo_covfilt:6::46'
   ConstructorCode: 'slcoverage.SFcnSelector(slcoverage.SFcnSelectorType.SFcnName, 'slvnvdemo_co
```

Create an S-Function Outcome Selector

This example shows how to create a selector for an S-Function using the `slcoverage.SFcnSelector` class.

In this example, your model has an S-Function named `RejectDoublePress_sfun`. Inside the S-Function, suppose that the condition `rtb_AccelResSwOUT` is never false, and you want to justify this false outcome in the coverage report.

```
void RejectDoublePress_sfun_Outputs_wrapper(const boolean_T *CoastSetSwIn,
                                            const boolean_T *AccelResSwIn,
                                            boolean_T *CoastSetSwOut,
                                            boolean_T *AccelResSwOut)
{
/* Output_BEGIN */
int rtb_AccelResSwOUT;

  rtb_AccelResSwOUT = !(CoastSetSwIn[0] && AccelResSwIn[0]);

  CoastSetSwOut[0] = (rtb_AccelResSwOUT && CoastSetSwIn[0]);

  AccelResSwOut[0] = (rtb_AccelResSwOUT && AccelResSwIn[0]);
/* Output_END */
}

```

Load the model.

```
modelName = 'ex_cc_cruise_control_doublepress_sfun';
load_system(modelName)
```

Configure coverage settings using a `Simulink.SimulationInput` object.

```
covSet = Simulink.SimulationInput(modelName);
covSet = covSet.setModelParameter('CovEnable','on');
covSet = covSet.setModelParameter('CovMetricStructuralLevel','ConditionDecision');
covSet = covSet.setModelParameter('CovSFcnEnable','on');
covSet = covSet.setModelParameter('CovSaveSingleToWorkspaceVar','on');
covSet = covSet.setModelParameter('CovSaveName','covData');
covSet = covSet.setModelParameter('SimAnalyzeCustomCode','on');
```

Simulate the model using `covSet` as the input.

```
simOut = sim(covSet);
covData = simOut.covData;
```

```
Generating 'RejectDoublePress_sfun.c' ....Please wait

### Building S-function 'RejectDoublePress_sfun.c' for ex_cc_cruise_control_doublepress_sfun/Reje
mex -IC:\TEMP\Bdoc21a_1606923_5032\ib8F3FCD\13\tpa9d68794\ex97019589 -I\ -I\ C:\TEMP\Bdoc21a_1606
Building with 'Microsoft Visual C++ 2019 (C)'.
MEX completed successfully.
mex -IC:\TEMP\Bdoc21a_1606923_5032\ib8F3FCD\13\tpa9d68794\ex97019589 -I\ -I\ C:\TEMP\Bdoc21a_1606
Building with 'Microsoft Visual C++ 2019 (C)'.
MEX completed successfully.
```

Create a selector object for the outcome missing coverage by using the `slcoverage.SFcnSelector` class. To select a condition outcome inside an S-Function, use the `SFcnInstanceCppConditionOutcome` enumeration as the first input. The second input is the block ID or block path to the S-Function. The third input is the file name of the source file that contains the expression, `RejectDoublePress_sfun_wrapper.c`. The fourth input is the name of the function that contains the condition, `RejectDoublePress_sfun_Outputs_wrapper`. The fifth input is the

expression which owns the outcome, which is `rtb_AccelResSwOUT && CoastSetSwIn[0]`, because this parent decision owns the condition `rtb_AccelResSwOUT`.

The sixth input is the index of the expression that owns the outcome relative to its parent, and because `rtb_AccelResSwOUT` is the first condition in its parent decision, this input is `1`. The seventh input is the outcome index, which is `1` because this is the `F` case of a Boolean expression. The final input is the index of the parent expression relative to the function, and because `rtb_AccelResSwOUT && CoastSetSwIn[0]` is the second decision in the function, this input is `2`.

```
enum = slcoverage.SFcnSelectorType.SFcnInstanceCppConditionOutcome;
SFunID = Simulink.ID.getSID([modelName, '/RejectDoublePress']);
sel = slcoverage.SFcnSelector(enum,SFunID,'RejectDoublePress_sfun_wrapper.c',...
   'RejectDoublePress_sfun_Outputs_wrapper','rtb_AccelResSwOUT && CoastSetSwIn[0]',1,1,2);
```

Create a `Filter` object and a rule based on the selector, then add the rule to the filter.

```
filt = slcoverage.Filter;
rule = slcoverage.FilterRule(sel,'Tested elsewhere',...
           slcoverage.FilterMode.Exclude);
addRule(filt,rule);
setFilterName(filt,'S-Function Filter')
```

Save the filter as `sfunfilter` and add it to the cvdata object for `my_func.c` by setting the `filter` property to the filter file name.

```
save(filt,'sfunfilter');
covData.filter = 'sfunfilter';
```

Generate a coverage report.

```
cvhtml('codeCovReport',covData)
```

Review the report. Click the `RejectDoublePress_sfun` link under S-Function Code Coverage Results to see the filtered outcome under Objects Filtered from Coverage Analysis.

See Also

`cv.cvdatagroup` | `getSimulinkBlockHandle` | `slcoverage.BlockSelector` | `slcoverage.Filter` | `slcoverage.FilterRule` | `slcoverage.MetricSelector`

Topics

“Top-Level Model Coverage Report”
“Simulink Identifiers”
“Create, Edit, and View Coverage Filter Rules”

Introduced in R2017b

addRule

Class: `slcoverage.Filter`
Package: `slcoverage`

Add coverage filtering rule to filter

Syntax

```
result = addRule(filter,rule)
```

Description

`result = addRule(filter,rule)` adds the filter rule to the specified filter.

Input Arguments

`filter` — Filter object to add the rule to

`slcoverage.Filter` object

Filter object to add the rule to, specified as an `slcoverage.Filter` object.

`rule` — Rule to add to the filter

`slcoverage.FilterRule` object

Rule to add to the filter, specified as an `slcoverage.FilterRule` object.

Output Arguments

`result` — Rule addition result

logical

Rule addition result, returned as 0 or 1.

Examples

Add Rule to Filter Object

Create a block selector, a filter, and a rule for the selector. Then add the rule to the filter.

Open the model. Specify coverage settings and turn on coverage recording.

```
modelName = 'sldemo_lct_bus';
open_system(modelName);
set_param(modelName,'CovMetricStructuralLevel','MCDC','RecordCoverage','on');
```

Create a `BlockSelector` object, `bl`. This block selector selects all blocks in the model with the property `'RelationalOperator'`.

```
bl = slcoverage.BlockSelector(slcoverage.BlockSelectorType.BlockType,'RelationalOperator');
```

Create a filter object, create a rule object, and add the rule to the filter object.

```
filt = slcoverage.Filter;
rule = slcoverage.FilterRule(bl,'Tested elsewhere',slcoverage.FilterMode.Exclude);
filt.addRule(rule);
```

See Also

`removeRule` | `slcoverage.BlockSelector` | `slcoverage.Filter` | `slcoverage.FilterRule` | `slcoverage.MetricSelector` | `slcoverage.SFcnSelector`

Introduced in R2017b

removeRule

Class: `slcoverage.Filter`
Package: `slcoverage`

Remove rule from filter rule set

Syntax

```
result = removeRule(filter,rule)
```

Description

`result = removeRule(filter,rule)` removes the filter rule from the specified filter.

Input Arguments

`filter` — Filter object to remove rule from

`slcoverage.Filter` object

Filter object to remove the rule from, specified as an `slcoverage.Filter` object.

`rule` — Rule to remove from the filter

`slcoverage.FilterRule` object

Rule to remove from the filter, specified as an `slcoverage.FilterRule` object.

Output Arguments

`result` — Rule removal result

logical

Rule removal result, returned as 0 or 1.

Examples

Remove Rules from Filter Objects

Create a block selector, a filter, and a rule for the selector. Add rules to the filter. Then, remove a rule from a filter.

Open the model. Specify coverage settings and turn on coverage recording.

```
modelName = 'sldemo_lct_bus';
open_system(modelName);
set_param(modelName,'CovMetricStructuralLevel','MCDC','RecordCoverage','on');
```

Create two `BlockSelector` objects, `bl` and `bl1`.

```
bl = slcoverage.BlockSelector(slcoverage.BlockSelectorType.BlockType,'RelationalOperator');
id = Simulink.ID.getSID('sldemo_lct_bus/slCounter/And');
bl1 = slcoverage.BlockSelector(slcoverage.BlockSelectorType.BlockInstance,id);
```

Create a filter object, create two rule objects, and add each rule to the filter object.

```
filt = slcoverage.Filter;
rule = slcoverage.FilterRule(bl,'Tested elsewhere',slcoverage.FilterMode.Exclude);
rule1 = slcoverage.FilterRule(bl1,'Value is never greater than 0');
filt.addRule(rule);
filt.addRule(rule1);
```

Review the rules. Look the first rule in the array.

```
fi = filt.rules
fi(1)

fi =

  1×2 FilterRule array with properties:

    Selector
    Mode
    Rationale

ans =

  FilterRule with properties:

     Selector: [1×1 slcoverage.BlockSelector]
         Mode: Exclude
    Rationale: 'Tested elsewhere'
```

Remove the first rule that you added. Then review the rules to see that the first rule that you added is removed.

```
filt.removeRule(rule);
fi = filt.rules

fi =

  FilterRule with properties:

     Selector: [1×1 slcoverage.BlockSelector]
         Mode: Justify
    Rationale: 'Value is never greater than 0'
```

See Also

`addRule` | `slcoverage.Filter` | `slcoverage.FilterRule` | `slcoverage.filter.rules`

Introduced in R2017b

rules

Class: `slcoverage.Filter`
Package: `slcoverage`

Rules for filter

Syntax

```
fr = rules(filter)
fr = rules(filter,element)
```

Description

`fr = rules(filter)` returns all the rules assigned to the filter.

`fr = rules(filter,element)` returns only the rules for the specified model element.

Input Arguments

`filter` — Filter object whose rules to return

`slcoverage.Filter` object

Filter object whose rules to return, specified as an `slcoverage.Filter` object.

`element` — Element identifier

Simulink ID | property | handle

This property is read-only.

Identifier of the model element whose rules to return, specified as a character vector or string of the Simulink ID, model element property, or handle.

Output Arguments

`fr` — Filter rules

`slcoverage.FilterRule` object | array of `slcoverage.FilterRule` objects

Filter rules, returned as an `slcoverage.FilterRule` object or an array of `slcoverage.FilterRule` objects.

Examples

Get All Rules for Filter Object

Open a model. Specify coverage settings and turn on coverage recording.

```
modelName = 'sldemo_lct_bus';
open_system(modelName);
set_param(modelName,'CovMetricStructuralLevel','MCDC','RecordCoverage','on');
```

Create a `BlockSelector` object, `bl`. Create a filter object, create a rule, and add the rule to the filter.

```
bl = slcoverage.BlockSelector(slcoverage.BlockSelectorType.BlockType,'RelationalOperator');
filt = slcoverage.Filter;
rule = slcoverage.FilterRule(bl,'Tested elsewhere',slcoverage.FilterMode.Exclude);
filt.addRule(rule);
```

Create another rule and add it to the filter object.

```
id = Simulink.ID.getSID('sldemo_lct_bus/slCounter/And');
bl = slcoverage.BlockSelector(slcoverage.BlockSelectorType.BlockInstance,id);
rule = slcoverage.FilterRule(bl,'Value is never greater than 0');
filt.addRule(rule);
```

Use `rules` to return the filter rules. View first rule in the array.

```
fi = filt.rules
fi(1)
```

```
fi =

  1×2 FilterRule array with properties:

    Selector
    Mode
    Rationale

ans =

  FilterRule with properties:

     Selector: [1×1 slcoverage.BlockSelector]
         Mode: Exclude
    Rationale: 'Tested elsewhere'
```

Use `rules` to return the rule only for the And block.

```
filt.rules(id)
```

```
ans =

  FilterRule with properties:

     Selector: [1×1 slcoverage.BlockSelector]
         Mode: Justify
    Rationale: 'Value is never greater than 0'
```

See Also

`addRule` | `removeRule` | `slcoverage.Filter` | `slcoverage.FilterRule`

Introduced in R2017b

allSelectors

Class: `slcoverage.BlockSelector`, `slcoverage.CodeSelector`, `slcoverage.Selector`, `slcoverage.MetricSelector`, `slcoverage.SFcnSelector`
Package: `slcoverage`

Selectors for model or code element

Syntax

```
sel = slcoverage.Selector.allSelectors(element)
sel = slcoverage.BlockSelector.allSelectors(element)
sel = slcoverage.CodeSelector.allSelectors(element)
sel = slcoverage.CodeSelector.allSelectors(element,Name,Value)
sel = slcoverage.MetricSelector.allSelectors(element)
sel = slcoverage.SFcnSelector.allSelectors(element)
sel = slcoverage.Selector.allSelectors(element,Name,Value)
```

Description

`sel = slcoverage.Selector.allSelectors(element)` returns all the selectors for the model element.

`sel = slcoverage.BlockSelector.allSelectors(element)` returns all the block selectors for `element`.

`sel = slcoverage.CodeSelector.allSelectors(element)` returns all the custom C/C++ code selectors for `element`.

`sel = slcoverage.CodeSelector.allSelectors(element,Name,Value)` , where `element` is a model and `Name,Value` specifies the simulation mode, returns all the custom C/C++ code selectors for the model in the specified simulation mode.

`sel = slcoverage.MetricSelector.allSelectors(element)` returns all the metric selectors for `element`.

`sel = slcoverage.SFcnSelector.allSelectors(element)` returns all the S-function selectors for `element`.

`sel = slcoverage.Selector.allSelectors(element,Name,Value)` returns selectors for `element`, with additional options specified by one or more `Name,Value` pair arguments.

Input Arguments

`element` — Model element to select

handle | Simulink ID

Model element to select, specified as a handle or the model element Simulink identifier.

Example: `'sldemo_lct_bus:18'`

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: `'Type',slcoverage.BlockSelectorType.BlockInstance,'Description','Foutcome'`

Type — Selector type refinement

`slcoverage.BlockSelectorType` value | `slcoverage.CodeSelectorType` value | `slcoverage.MetricSelectorType` value | `slcoverage.SFcnSelectorType` value

Selector type refinement specified as one of the `slcoverage.BlockSelectorTypeslcoverage.CodeSelectorType`, `slcoverage.MetricSelectorType`, or `slcoverage.SFcnSelectorType` values. Possible values:

- Block selector types:
  - `slcoverage.BlockSelectorType.BlockInstance` — An instance of a block.
  - `slcoverage.BlockSelectorType.BlockType` — All blocks of the specified block type.
  - `slcoverage.BlockSelectorType.Chart` — A Stateflow chart.
  - `slcoverage.BlockSelectorType.MaskType` — Blocks that use the specified mask type.
  - `slcoverage.BlockSelectorType.State` — A Stateflow state.
  - `slcoverage.BlockSelectorType.StateAllContent` — Stateflow state and its contents.
  - `slcoverage.BlockSelectorType.StateflowFunction` — A Stateflow function.
  - `slcoverage.BlockSelectorType.Subsystem` — A subsystem block.
  - `slcoverage.BlockSelectorType.SubsystemAllContent` — A subsystem and its contents.
  - `slcoverage.BlockSelectorType.TemporalEvent` — A Stateflow temporal event.
  - `slcoverage.BlockSelectorType.Transition` — A Stateflow transition.
- Code selector types:
  - `slcoverage.CodeSelectorType.File` — A custom C or C++ code file name.
  - `slcoverage.CodeSelectorType.Function` — A custom C or C++ code function name.
  - `slcoverage.CodeSelectorType.Decision` — A custom C or C++ code decision.
  - `slcoverage.CodeSelectorType.Condition` — A custom C or C++ code condition.
  - `slcoverage.CodeSelectorType.DecisionOutcome` — A custom C or C++ code decision outcome.
  - `slcoverage.CodeSelectorType.ConditionOutcome` — A custom C or C++ code condition outcome.
  - `slcoverage.CodeSelectorType.MCDCOutcome` — A custom C or C++ code MCDC outcome.
  - `slcoverage.CodeSelectorType.RelationalBoundaryOutcome` — A custom C or C++ code relational boundary outcome.
- Metric selector types:

  - `slcoverage.MetricSelectorType.ConditionOutcome` objects select condition metric objective outcomes.
  - `slcoverage.MetricSelectorType.DecisionOutcome` objects select decision metric objective outcomes.
  - `slcoverage.MetricSelectorType.MCDCOutcome` objects select MCDC metric objective outcomes.
  - `slcoverage.MetricSelectorType.RelationalBoundaryOutcome` objects select outcome metrics related to relational boundary outcomes.
  - `slcoverage.MetricSelectorType.SaturationOverflowOutcome` objects select outcome metrics related to saturation on integer overflow outcomes.
- S-function selector types:
  - `slcoverage.SFcnSelectorType.SFcnName` selects the specified S-function.
  - `slcoverage.SFcnSelectorType.SFcnInstanceCppFileName` selects the coverage data in the generated code file for this block.
  - `slcoverage.SFcnSelectorType.SFcnInstanceCppFunction` selects a function.
  - `slcoverage.SFcnSelectorType.SFcnInstanceCppCondition` selects a condition outcome of the S-function block.
  - `slcoverage.SFcnSelectorType.SFcnInstanceCppDecision` selects a decision outcome of the S-function block.

`Description` — Description text to match
character vector or string

Description text to match for the selector that you want to return, specified as a character vector or string. For example, if you want to return only the selectors that include the text `F outcome` in the description, use this syntax:

```
s = slcoverage.Selector.allSelectors(id,'Description','F outcome')
```

`SimulationMode` — Simulation mode
character vector or string

Simulation mode to run when selecting code filters, entered as one of the following:

Object Specification	Description
`'normal'` (default)	Extract code selectors for custom code in normal simulation, such as custom code called from a C Caller block or a Stateflow chart.
`'sil'`	Extract code selectors for code generated in Simulation-in-the-Loop (SIL) mode and code selectors for the top model code interface
`'pil'`	Extract code selectors for code generated in Processor-in-the-Loop (PIL) mode and code selectors for the top model code interface

Object Specification	Description
`'xil'`	If SIL-mode code exists, extract code selectors for code generated in SIL mode and extract code selectors for the top model code interface; otherwise, extract code selectors for code generated in PIL mode and extract code selectors for the top model code interface
`'modelrefsil'`	Extract code selectors for the model reference code interface in SIL mode
`'modelrefpil'`	Extract code selectors for the model reference code interface in PIL mode
`'modelrefxil'`	If SIL-mode code exists, extract code selectors for the model reference code interface in SIL mode, if the model is in SIL mode; otherwise,extract code selectors for the model reference code interface in PIL mode

Output Arguments

`sel` — Selectors for the model or code element

array of `Selector` objects

Selectors for the model or code element, returned as an array of `Selector` objects.

Examples

Get All Selectors

This example shows how to get all the selectors for an And block and then add a rule to justify or exclude a selector. Metric selectors can only be justified.

Load the model and set coverage settings

```
modelName = 'slvnvdemo_covfilt';
load_system(modelName);
set_param(modelName,'CovEnable','on','CovMetricStructuralLevel','MCDC');
```

First, get the block handle for the And block.

```
id = getSimulinkBlockHandle([modelName,'/Saturation']);
```

Get the selectors using the block handle.

```
sel = slcoverage.Selector.allSelectors(id)
```

```
sel =

  1x10 heterogeneous Selector (BlockSelector, MetricSelector) array with properties:

    Description
    Type
    Id
```

```
    ConstructorCode
```

The block has ten selectors. You can index into each one to see their contents. In this example, you want to justify the sixth selector.

```
sel(6)
```

```
ans =

  MetricSelector with properties:

     ObjectiveIndex: 2
       OutcomeIndex: 2
        Description: 'T outcome of input >= upper limit in Saturate block "Saturation"'
               Type: DecisionOutcome
                 Id: 'slvnvdemo_covfilt:5'
    ConstructorCode: 'slcoverage.MetricSelector(slcoverage.MetricSelectorType.DecisionOutcome, '
```

Create a justify rule, then create a filter object and add the rule to it.

```
rule = slcoverage.FilterRule(sel(6),'Expected result');
filt = slcoverage.Filter;
filt.addRule(rule);
```

Save the filter and generate a coverage report.

```
filt.save('metrfilter');
csim = cvsim(modelName);
csim.filter = 'metrfilter';
cvhtml('cov',csim,'-sRT=0');
```

Get Selector by Type and Description

This example shows how to get a selector by type and description. In this example, you get all selectors for the False outcomes of the And block whose type is `ConditionOutcome`.

Load the model into memory.

```
modelName = 'sldemo_lct_bus';
load_system(modelName);
```

Get the false outcome condition selectors for the And block by searching for descriptions that include `F`.

```
id = getSimulinkBlockHandle([modelName,'/slCounter/And']);
sel = slcoverage.Selector.allSelectors(id, ...
         'Type',slcoverage.MetricSelectorType.ConditionOutcome, ...
         'Description','F')
```

```
sel =

  1x2 MetricSelector array with properties:
```

```
    ObjectiveIndex
    OutcomeIndex
    Description
    Type
    Id
    ConstructorCode
```

Look at the constructor code for the two selectors that were returned.

```
sel.ConstructorCode
```

```
ans =

    'slcoverage.MetricSelector(slcoverage.MetricSelectorType.ConditionOutcome, 'sldemo_lct_bus:23

ans =

    'slcoverage.MetricSelector(slcoverage.MetricSelectorType.ConditionOutcome, 'sldemo_lct_bus:23
```

See Also

`slcoverage.BlockSelector` | `slcoverage.CodeSelector` | `slcoverage.MetricSelector` | `slcoverage.SFcnSelector` | `slcoverage.Selector`

Introduced in R2017b

setFilterName

Class: `slcoverage.Filter`
Package: `slcoverage`

Set name of coverage filter object

Syntax

```
setFilterName(filterObj,filterName)
```

Description

`setFilterName(filterObj,filterName)` sets the name of the filter to the specified value.

Input Arguments

`filterObj` — Coverage filter

`slcoverage.Filter` object

Coverage filter, specified as an `slcoverage.Filter` object.

Data Types: `slcoverage.Filter`

`filterName` — Coverage filter name

`character array` | `string array`

Coverage filter name, specified as a character array or string array.

Data Types: `char` | `string`

Examples

Edit and View Coverage Filter Details

This example shows how to use the `slcoverage.Filter` methods to set and get filter names and descriptions.

Create a new filter object by using the `slcoverage.Filter` class.

```
filt = slcoverage.Filter;
```

Set the filter name and description by using `setFilterName` and `setFilterDescription`, respectively.

```
setFilterName(filt,'myCovFilter');
setFilterDescription(filt,'Justify missing coverage for unreachable outcomes');
```

Get the filter name and description by using `filterName` and `filterDescription` with the filter object as the input.

```
filtName = filterName(filt)
filtDescr = filterDescription(filt)
```

```
filtName =

    'myCovFilter'


filtDescr =

    'Justify missing coverage for unreachable outcomes'
```

Alternatives

You can also create, edit, and view filters in Simulink. See “Create, Edit, and View Coverage Filter Rules” for more information.

See Also

`addRule` | `filterDescription` | `filterName` | `removeRule` | `rules` | `save` | `setFilterDescription` | `slcoverage.Filter`

Topics

“Filter Coverage Results Using a Script”

Introduced in R2020a

filterName

Class: `slcoverage.Filter`
Package: `slcoverage`

Get name of coverage filter object

Syntax

```
filtName = filterName(filterObj)
```

Description

`filtName = filterName(filterObj)` returns the name of the specified coverage filter.

Input Arguments

`filterObj` — Coverage filter

`slcoverage.Filter` object

Coverage filter, specified as an `slcoverage.Filter` object.

Data Types: `slcoverage.Filter`

Output Arguments

`filtName` — Filter name

`character array` | `string array`

Filter name, returned as a character array or string array.

Data Types: `char` | `string`

Examples

Edit and View Coverage Filter Details

This example shows how to use the `slcoverage.Filter` methods to set and get filter names and descriptions.

Create a new filter object by using the `slcoverage.Filter` class.

```
filt = slcoverage.Filter;
```

Set the filter name and description by using `setFilterName` and `setFilterDescription`, respectively.

```
setFilterName(filt,'myCovFilter');
setFilterDescription(filt,'Justify missing coverage for unreachable outcomes');
```

Get the filter name and description by using `filterName` and `filterDescription` with the filter object as the input.

```
filtName = filterName(filt)
filtDescr = filterDescription(filt)
```

```
filtName =

    'myCovFilter'


filtDescr =

    'Justify missing coverage for unreachable outcomes'
```

Alternatives

You can also create, edit, and view filters in Simulink. See “Create, Edit, and View Coverage Filter Rules” for more information.

See Also

`addRule` | `filterDescription` | `removeRule` | `rules` | `save` | `setFilterDescription` | `setFilterName` | `slcoverage.Filter`

Topics

“Filter Coverage Results Using a Script”

Introduced in R2020a

setFilterDescription

Class: `slcoverage.Filter`
Package: `slcoverage`

Set description of coverage filter object

Syntax

```
setFilterDescription(filterObj,descr)
```

Description

`setFilterDescription(filterObj,descr)` sets the description of the filter to the specified value.

Input Arguments

`filterObj` — Coverage filter

`slcoverage.Filter` object

Coverage filter, specified as an `slcoverage.Filter` object.

Data Types: `slcoverage.Filter`

`descr` — Coverage filter description

`character array` | `string array`

Coverage filter description, specified as a character array or string array.

Data Types: `char` | `string`

Examples

Edit and View Coverage Filter Details

This example shows how to use the `slcoverage.Filter` methods to set and get filter names and descriptions.

Create a new filter object by using the `slcoverage.Filter` class.

```
filt = slcoverage.Filter;
```

Set the filter name and description by using `setFilterName` and `setFilterDescription`, respectively.

```
setFilterName(filt,'myCovFilter');
setFilterDescription(filt,'Justify missing coverage for unreachable outcomes');
```

Get the filter name and description by using `filterName` and `filterDescription` with the filter object as the input.

```
filtName = filterName(filt)
filtDescr = filterDescription(filt)
```

```
filtName =

    'myCovFilter'


filtDescr =

    'Justify missing coverage for unreachable outcomes'
```

Alternatives

You can also create, edit, and view filters in Simulink. See “Create, Edit, and View Coverage Filter Rules” for more information.

See Also

`addRule` | `filterDescription` | `filterName` | `removeRule` | `rules` | `save` | `setFilterName` | `slcoverage.Filter`

Topics

“Filter Coverage Results Using a Script”

Introduced in R2020a

filterDescription

Class: `slcoverage.Filter`
Package: `slcoverage`

Get description of coverage filter object

Syntax

```
descr = filterDescription(filterObj)
```

Description

`descr = filterDescription(filterObj)` returns the description of the specified coverage filter.

Input Arguments

`filterObj` — Coverage filter

`slcoverage.Filter`

Coverage filter, specified as an `slcoverage.Filter` object.

Data Types: `slcoverage.Filter`

Output Arguments

`descr` — Filter description

`character array` | `string array`

Filter description, returned as a character array or string array.

Data Types: `char` | `string`

Examples

Edit and View Coverage Filter Details

This example shows how to use the `slcoverage.Filter` methods to set and get filter names and descriptions.

Create a new filter object by using the `slcoverage.Filter` class.

```
filt = slcoverage.Filter;
```

Set the filter name and description by using `setFilterName` and `setFilterDescription`, respectively.

```
setFilterName(filt,'myCovFilter');
setFilterDescription(filt,'Justify missing coverage for unreachable outcomes');
```

Get the filter name and description by using `filterName` and `filterDescription` with the filter object as the input.

```
filtName = filterName(filt)
filtDescr = filterDescription(filt)
```

```
filtName =

    'myCovFilter'


filtDescr =

    'Justify missing coverage for unreachable outcomes'
```

Alternatives

You can also create, edit, and view filters in Simulink. See “Create, Edit, and View Coverage Filter Rules” for more information.

See Also

`addRule` | `filterName` | `removeRule` | `rules` | `save` | `setFilterDescription` | `setFilterName` | `slcoverage.Filter`

Topics

“Filter Coverage Results Using a Script”

Introduced in R2020a

save

Class: `slcoverage.Filter`
Package: `slcoverage`

Save coverage filter object to coverage filter file

Syntax

```
save(filterObj,fileName)
```

Description

`save(filterObj,fileName)` saves the specified filter object with the specified file name. The generated file will have the `.cvf` extension.

Input Arguments

`filterObj` — Coverage filter

`slcoverage.Filter` object

Coverage filter, specified as an `slcoverage.Filter` object.

Data Types: `slcoverage.Filter`

`fileName` — File name

`character array` | `string array`

File name, specified as a character array or string array.

Data Types: `char` | `string`

Examples

Create and Save a Coverage Filter

Create a filter object by using the `slcoverage.Filter` class, then set the filter name to `myCovFilter` using `setFilterName`.

```
filt = slcoverage.Filter;
setFilterName(filt,'myCovFilter')
```

Save the filter as `myCovFilter.cvf` by using `save`.

```
save(filt,'myCovFilter')
```

Alternatives

You can also create and save filters in Simulink. See “Create, Edit, and View Coverage Filter Rules” for more information.

See Also

`addRule` | `filterDescription` | `filterName` | `removeRule` | `rules` | `setFilterDescription` | `setFilterName` | `slcoverage.Filter`

Topics

“Filter Coverage Results Using a Script”

Introduced in R2020a