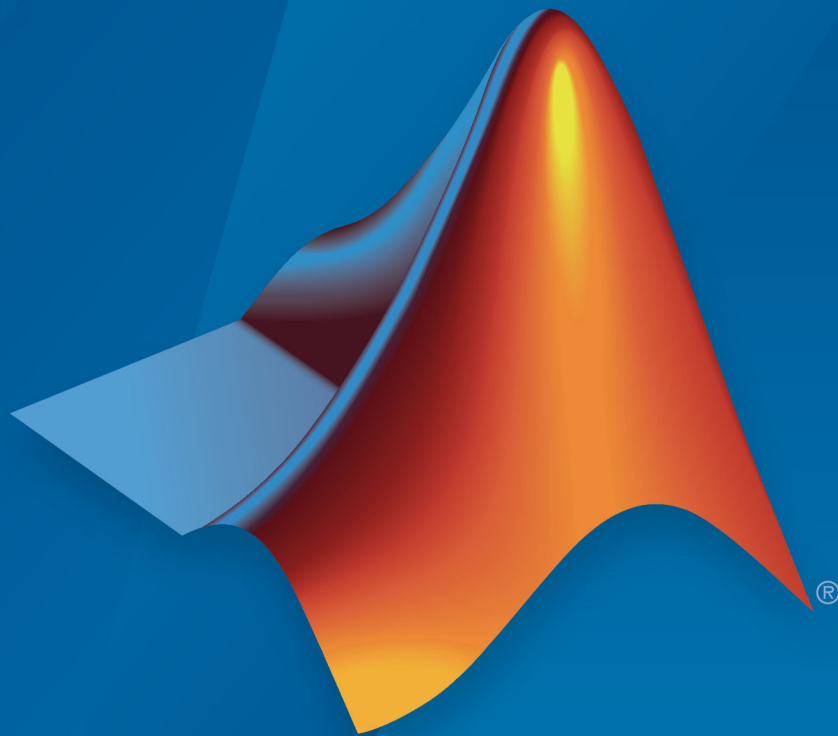


# Simulink® Coverage™

Reference



# MATLAB® & SIMULINK®

R2019a



# How to Contact MathWorks



Latest news: [www.mathworks.com](http://www.mathworks.com)  
Sales and services: [www.mathworks.com/sales\\_and\\_services](http://www.mathworks.com/sales_and_services)  
User community: [www.mathworks.com/matlabcentral](http://www.mathworks.com/matlabcentral)  
Technical support: [www.mathworks.com/support/contact\\_us](http://www.mathworks.com/support/contact_us)



Phone: 508-647-7000



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

## *Simulink® Coverage™ Reference*

© COPYRIGHT 2017–2019 by The MathWorks, Inc.

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

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

## **Trademarks**

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

## **Patents**

MathWorks products are protected by one or more U.S. patents. Please see [www.mathworks.com/patents](http://www.mathworks.com/patents) for more information.

## **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)

## Functions – Alphabetical List

1

## Simulink Coverage Settings

2

<b>Coverage Pane</b> .....	<b>2-2</b>
Coverage Pane Overview .....	2-3
Enable coverage analysis .....	2-3
Scope of coverage analysis .....	2-4
Select Models .....	2-5
Select Subsystem .....	2-5
Record coverage for MATLAB files .....	2-7
Record coverage for C/C++ S-functions .....	2-7
Structural coverage level .....	2-8
Lookup table .....	2-9
Signal range .....	2-10
Signal size .....	2-10
Objectives and constraints .....	2-11
Saturation on integer overflow .....	2-12
Relational boundary .....	2-12
Relational boundary coverage absolute tolerance .....	2-13
Relational boundary coverage relative tolerance .....	2-13
Restrict coverage recording interval .....	2-14
Coverage interval start time .....	2-15
Coverage interval stop time .....	2-15
Force block reduction off .....	2-16
Treat Simulink logic blocks as short-circuited .....	2-16
MCDC mode .....	2-17
Warn when unsupported blocks exist in model .....	2-18
Coverage filter filename .....	2-18
Coverage metric settings .....	2-19

Record coverage for this model .....	2-20
Record coverage for referenced models .....	2-21
Include top model .....	2-22
<b>Coverage Pane: Results .....</b>	<b>2-23</b>
Coverage Results Pane Overview .....	2-24
Show Results Explorer .....	2-24
Display coverage results using model coloring .....	2-25
Generate report automatically after analysis .....	2-26
Save last run in workspace variable .....	2-27
Last coverage run variable name .....	2-27
Increment variable name with each simulation .....	2-28
Autosave data file name .....	2-29
Output directory .....	2-29
Coverage report options .....	2-30
Additional data to include in coverage report .....	2-32
Update coverage results on pause .....	2-32
Save output data .....	2-33
Enable cumulative data collection .....	2-33
Include cumulative data in coverage report .....	2-34
Save cumulative coverage results in workspace variable .....	2-35
Cumulative coverage variable name .....	2-36

## Class Reference

# 3

# Functions — Alphabetical List

---

## 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.cvdatabgroup;
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
<code>BlockPath</code>	Full path to a model or block
<code>BlockHandle</code>	Handle to a model or block
<code>slobj</code>	Handle to a Simulink® API object



Object Specification	Description
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 and 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`]:

<code>total_complexity</code>	Cyclomatic complexity coverage for <code>object</code> and its descendants (if any)
<code>local_complexity</code>	Cyclomatic complexity coverage for <code>object</code>

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

## Examples

Open the `sldemo_fuelsys` model and create the test specification object `testObj`. Enable decision, condition, and MCDC coverage for `sldemo_fuelsys` and execute `testObj` using `cvsim`. Use `complexityinfo` to retrieve cyclomatic complexity results

for the Throttle subsystem. The Throttle subsystem itself does not record cyclomatic complexity coverage results, but the contents of the subsystem do record cyclomatic complexity coverage.

```
mdl = 'sldemo_fuelsys';
open_system(mdl);
testObj = cvtest(mdl)
testObj.settings.decision = 1;
testObj.settings.condition = 1;
testObj.settings.mcdc = 1;
data = cvsim(testObj);
blk_handle = get_param([mdl, ...
    '/Engine Gas Dynamics/Throttle & Manifold/Throttle'],...
    'Handle');
coverage = complexityinfo(data, blk_handle);
coverage
```

## Alternatives

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

- 1 Open the model.
- 2 In the Model Editor, select **Analysis > Coverage > Settings**.
- 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](#) | [getCoverageInfo](#) | [mcdcinfo](#) | [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:

<code>BlockPath</code>	Full path to a Simulink model or block
<code>BlockHandle</code>	Handle to a Simulink model or block
<code>sObj</code>	Handle to a Simulink API object
<code>sfID</code>	Stateflow ID
<code>sfObj</code>	Handle to a Stateflow API object
<code>{BlockPath, sfID}</code>	Cell array with the path to a Stateflow chart or atomic subchart and the ID of an object contained in that chart or subchart
<code>{BlockPath, sfObj}</code>	Cell array with the path to a Stateflow chart or atomic subchart and a Stateflow object API handle contained in that chart or subchart
<code>{BlockHandle, sfID}</code>	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
<code>{BlockPath, fName}</code>	Cell array with the path to an S-Function block and the name of a source file.
<code>{BlockHandle, fName}</code>	Cell array with an S-Function block handle and the name of a source file.
<code>{BlockPath, fName, funName}</code>	Cell array with the path to an S-Function block, the name of a source file, and a function name.
<code>{BlockHandle, fName, funName}</code>	Cell array with an S-Function block handle, the name of a source file and 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:

<code>covered_outcomes</code>	Number of condition outcomes satisfied for object
<code>total_outcomes</code>	Total number of condition outcomes for object

### **description**

A structure array with the following fields:

<code>condition</code>	A structure array containing condition info for individual condition outcomes
<code>isFiltered</code>	Whether the block is filtered
<code>filterRationale</code>	The filtering rationale
<code>justifiedCoverage</code>	The justified coverage conditions
<code>isJustified</code>	Whether the block is justified

## **Examples**

The following example opens the `slvndemo_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 = 'slvndemo_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 **Analysis > Coverage > Settings**.
- 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 On the **Coverage > Results** pane, specify the output you need.
- 6 Click **OK** to close the Configuration Parameters dialog box and save your changes.
- 7 Simulate the model and review the results.

## See Also

[complexityinfo](#) | [cvsim](#) | [decisioninfo](#) | [getCoverageInfo](#) | [mcdcinfo](#) | [overflowsaturationinfo](#) | [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 (MATLAB) 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 an instantiation of the `cv.cvdatagroup` class (`cvdg`) that contains the `cvdata` objects `cvdo1`, `cvdo2`, etc. A `cvdata` object contains results of the simulation runs.

### **Examples**

Create an instantiation of the `cv.cvdatagroup` class and add two `cvdata` objects to it:

```
a = cvdata;  
b = cvdata;  
cvdg = cv.cvdatagroup(a, b);
```

# 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.
'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 the Simulation Modes Associated with Three `cvdata` Sets

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

```
a = cvdata;  
b = cvdata;  
c = cvdata;  
cvdg = cv.cvdatabgroup;  
add(cvdg, a, b, c);  
model_simModes = allSimulationModes(cvdg)
```

# 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, cvdo2, ...)
cvhtml(file, cvdo1, cvdo2, ..., 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, cvdo2, ...)` 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, cvdo2, ..., 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`.

### Input Arguments

#### **cvdo** — Class instance

object

`cv.cvdatagroup` object.



**file – HTML file**

character vector or string

The HTML file in the MATLAB current folder where `cvhtml` stores the results. You can also specify the absolute path or relative path and the HTML file where `cvhtml` stores the results.

**options – Report options**

character vector or string

Specify the report options that you specify in `options`:

- 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 vector or string

Simulation mode associated with the models. 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

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);
```

## 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 **Analysis > Coverage > Settings**.
- 3 On the **Coverage** pane of the Configuration Parameters dialog box, select **Enable coverage analysis**.
- 4 On the **Coverage > Results** pane, select **Generate report automatically after analysis**.
- 5 Click **OK** to close the Configuration Parameters dialog box and save your changes.

- 6 Simulate the model and review the generated report.

## See Also

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

## Topics

“Create HTML Reports with cvhtml”

**Introduced before R2006a**

## cvload

Load coverage tests and stored results into memory

### Syntax

```
[tests, data] = cvload(filename)
[tests, data] = cvload(filename, restoretotal)
```

### Description

`[tests, data] = cvload(filename)` loads the tests and data stored in the text file `filename.cvt`. `tests` is a cell array of `cvtest` objects that are loaded. `data` is a cell array of `cvdata` objects that are loaded. `data` has the same size as `tests`, but if a particular test has no results, `data` can contain empty elements.

`[tests, data] = cvload(filename, restoretotal)` restores or clears the cumulative results from prior runs, depending on the value of `restoretotal`. If `restoretotal` is 1, `cvload` restores the cumulative results from prior runs. If `restoretotal` is unspecified or 0, `cvload` clears the model's cumulative results.

The following are special considerations for using the `cvload` command:

- If a model with the same name exists in the coverage database, the software loads only 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, the coverage tool resolves the links to the existing models.
- When you are loading several files that reference the same model, the software loads only the results that are consistent with the earlier files.

### Examples

Store coverage results in `cvtest` and `cvdata` objects:

```
[test_objects, data_objects] = cvload(test_results, 1);
```

## **See Also**

cvsave

## **Topics**

“Load Stored Coverage Test Results with cvload”

**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** — Class instance

object

`cv.cvdatagroup` object.

**simMode** — Simulation mode

character vector or string

Simulation modes associated with the models. 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

Open the `slvndemo_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 = 'slvndemo_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 **Analysis > Coverage > Settings**.
- 3 On the **Coverage** pane of the Configuration Parameters dialog box, select **Enable coverage analysis**.
- 4 On the **Coverage > Results** pane, select **Display coverage results using model coloring**.
- 5 Click **OK** to close the Configuration Parameters dialog box and save your changes.
- 6 Simulate the model 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 tests and results to file

### **Syntax**

```
cvsave(filename, model)
cvsave(filename, cvd)
cvsave(filename, cvto1, cvto2, ...)
cvsave(filename, cell_array{ :})
```

### **Description**

`cvsave(filename, model)` saves all the tests (`cvtest` objects) and results (`cvdata` objects) related to `model` in the text file `filename.cvt`. `model` is a handle to or name of a Simulink model.

`cvsave(filename, cvd)` saves all the results (`cvdata` objects) for the active model in the text file `filename.cvt`. `cvsave` also saves information about any referenced models.

`cvsave(filename, cvto1, cvto2, ...)` saves multiple `cvtest` objects in the text file `filename.cvt`. `cvsave` also saves information about any referenced models.

`cvsave(filename, cell_array{ :})` saves the test results stored in each element of `cell_array` to the file `filename.cvt`. Each element in `cell_array` contains test results for a `cvdata` object.

### **Input Arguments**

#### **filename**

Character vector or string containing the name of the file in which to save the data. `cvsave` appends the extension `.cvt` to the name of the file when saving it.

**model**

Handle to a Simulink model

**cvd**

cvdata object

**cvto**

cvtest object

**cell\_array**

Cell array of cvtest objects

## Examples

Save coverage results for the `slvndemo_cv_small_controller` model in `ratelim_testdata.cvt`:

```
model = 'slvndemo_cv_small_controller';  
open_system(model);  
cvt = cvtest(model);  
cvd = cvsim(cvt);  
cvsave('ratelim_testdata', model);
```

Save cumulative coverage results for the Adjustable Rate Limiter subsystem in the `slvndemo_ratelim_harness` model from two simulations:

```
% Open model and subsystem  
mdl = 'slvndemo_ratelim_harness';  
mdl_subsys = ...  
    'slvndemo_ratelim_harness/Adjustable Rate Limiter';  
open_system(mdl);  
open_system(mdl_subsys);  
  
% Create data files  
t_gain = (0:0.02:2.0)';  
u_gain = sin(2*pi*t_gain);  
t_pos = [0;2];  
u_pos = [1;1];
```

```
t_neg = [0;2];
u_neg = [-1;-1];
save('within_lim.mat','t_gain','u_gain','t_pos','u_pos', ...
     't_neg','u_neg');

t_gain = [0;2];
u_gain = [0;4];
t_pos = [0;1;1;2];
u_pos = [1;1;5;5]*0.02;
t_neg = [0;2];
u_neg = [0;0];
save('rising_gain.mat','t_gain','u_gain','t_pos','u_pos', ...
     't_neg','u_neg');

% Specify coverage options in cvtest object
testObj1 = cvtest mdl_subsys);
testObj1.label = 'Gain within slew limits';
testObj1.setupCmd = 'load(''within_lim.mat'');';
testObj1.settings.mcdc = 1;
testObj1.settings.condition = 1;
testObj1.settings.decision = 1;

testObj2 = cvtest mdl_subsys);
testObj2.label = ...
    'Rising gain that temporarily exceeds slew limit';
testObj2.setupCmd = 'load(''rising_gain.mat'');';
testObj2.settings.mcdc = 1;
testObj2.settings.condition = 1;
testObj2.settings.decision = 1;

% Simulate the model with both cvtest objects
[dataObj1,simOut1] = cvsim(testObj1);
[dataObj2,simOut2] = cvsim(testObj2,[0 2]);

cumulative = dataObj1+dataObj2;
cvsave('ratelim_testdata',cumulative);
```

As in the preceding example, save cumulative coverage results for the Adjustable Rate Limiter subsystem in the `slvnvdemo_ratelim_harness` model from two simulations. Save the results in a cell array and then save the data to a file:

```
% Open model and subsystem
mdl = 'slvnvdemo_ratelim_harness';
mdl_subsys = ...
```

```
    'slvndemo_ratelim_harness/Adjustable Rate Limiter';
open_system mdl;
open_system mdl_subsys);

% Create data files
t_gain = (0:0.02:2.0)';
u_gain = sin(2*pi*t_gain);
t_pos = [0;2];
u_pos = [1;1];
t_neg = [0;2];
u_neg = [-1;-1];
save('within_lim.mat','t_gain','u_gain','t_pos','u_pos', ...
     't_neg','u_neg');

t_gain = [0;2];
u_gain = [0;4];
t_pos = [0;1;1;2];
u_pos = [1;1;5;5]*0.02;
t_neg = [0;2];
u_neg = [0;0];
save('rising_gain.mat','t_gain','u_gain','t_pos','u_pos', ...
     't_neg','u_neg');

% Specify coverage options in cvtest object
testObj1 = cvtest mdl_subsys);
testObj1.label = 'Gain within slew limits';
testObj1.setupCmd = 'load(''within_lim.mat'')';
testObj1.settings.mcdc = 1;
testObj1.settings.condition = 1;
testObj1.settings.decision = 1;

testObj2 = cvtest mdl_subsys);
testObj2.label = ...
    'Rising gain that temporarily exceeds slew limit';
testObj2.setupCmd = 'load(''rising_gain.mat'')';
testObj2.settings.mcdc = 1;
testObj2.settings.condition = 1;
testObj2.settings.decision = 1;

% Simulate the model with both cvtest objects
[dataObj1,simOut1] = cvsim(testObj1);
[dataObj2,simOut2] = cvsim(testObj2,[0 2]);

% Save the results in the cell array
```

```
cov_results{1} = dataObj1;  
cov_results{2} = dataObj2;  
  
% Save the results to a file  
cvsave('ratelim_testdata', cov_results{ :});
```

## Alternatives

Use the coverage settings to save cumulative coverage results for a model:

- 1 Open the model for which you want to save cumulative coverage results.
- 2 In the Model Editor, select **Analysis > Coverage > Settings**.
- 3 On the **Coverage** pane of the Configuration Parameters dialog box, select **Enable coverage analysis**.
- 4 On the **Coverage > Results** pane, select **Save last run in workspace variable**.
- 5 Click **OK** to close the Configuration Parameters dialog box and save your changes.
- 6 Simulate the model and review the results.

## See Also

cvload

## Topics

“Save Test Runs to File with cvsave”

**Introduced before R2006a**

## cvsim

Simulate and return model coverage results for test objects

### Syntax

```
cvdo = cvsim(modelName)
cvdo = cvsim(cvto)
[cvdo,simOut] = cvsim(cvto,Name1,Value1,Name2,Value2,...)
[cvdo,simOut] = cvsim(cvto,ParameterStruct)
[cvdo1,cvdo2,...] = cvsim(cvto1,cvto2,...)
```

### Description

`cvdo = cvsim(modelName)` simulates the model and returns the coverage results for the model. `cvsim` saves the coverage results in the `cvdata` object, `cvdo`. However, when recording coverage for multiple models in a hierarchy, `cvsim` returns the coverage results in a `cv.cvdatalogroup` object.

`cvdo = cvsim(cvto)` simulates the model and returns the coverage results for the `cvtest` object, `cvto`. `cvsim` saves the coverage results in the `cvdata` object, `cvdo`. However, when recording coverage for multiple models in a hierarchy, `cvsim` returns the coverage results in a `cv.cvdatalogroup` object.

`[cvdo,simOut] = cvsim(cvto,Name1,Value1,Name2,Value2,...)` specifies the model parameters and simulates the model. `cvsim` returns the coverage results in the `cvdata` object, `cvdo`, and returns the simulation outputs in the `Simulink.SimulationOutput` object, `simOut`.

`[cvdo,simOut] = cvsim(cvto,ParameterStruct)` sets the model parameters specified in a structure `ParameterStruct`, simulates the model, returns the coverage results in `cvdo`, and returns the simulation outputs in `simOut`.

`[cvdo1,cvdo2,...] = cvsim(cvto1,cvto2,...)` simulates the model and returns the coverage results for the test objects, `cvto1`, `cvto2`, ... `cvdo1` contains the coverage results for `cvto1`, `cvdo2` contains the coverage results for `cvto2`, and so on.

---

**Note** Even if you have not enabled coverage recording for the model, you can execute the `cvsim` command to record coverage for your model.

---

## Input Arguments

### **modelName**

Name of model specified as a character vector or string

### **cvto**

`cvtest` object that specifies coverage options for the simulation

### **ParameterStruct**

Model parameters specified as a structure

## 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`.

### **ParameterName**

Name of the model parameter to be specified for simulation

### **ParameterValue**

Value of the model parameter

---

**Note** For a complete list of model parameters, see “Model Parameters” (Simulink).

---

## Output Arguments

### **cvdo**

`cvdata` object



## simOut

A `Simulink.SimulationOutput` object that contains the simulation outputs.

## Examples

Open the `sldemo_engine` example model, create the test object, set the model parameters, and simulate the model. `cvsim` returns the coverage data in `cvdo` and the simulation outputs in the `Simulink.SimulationOutput` object, `simOut`:

```
model = 'sldemo_engine';
open_system(model);
testObj = cvtest(model); % Get test data
testObj.settings.decision = 1;
paramStruct.AbsTol = '1e-5';
paramStruct.SaveState = 'on';
paramStruct.StateSaveName = 'xoutNew';
paramStruct.SaveOutput = 'on';
paramStruct.OutputSaveName = 'youtNew';
[cvdo,simOut] = cvsim(testObj,paramStruct); % Get coverage
cvhtml('CoverageReport.html', cvdo); % Create HTML Report
```

## See Also

`cv.cvdagroup` | `cvtest` | `sim`

**Introduced before R2006a**

## **cvtest**

Create model coverage test specification object

### **Syntax**

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

### **Description**

`cvto = cvtest(root)` creates a test specification object with the handle `cvto`. Simulate `cvto` with the `cvsim` command.

`cvto = cvtest(root, label)` creates a test object with the label `label`, which is used for reporting results.

`cvto = cvtest(root, label, setupcmd)` creates a test object with the setup command `setupcmd`.

### **Input Arguments**

#### **root**

Name or handle for a Simulink model or a subsystem. Only the specified model or subsystem and its descendants are subject to model coverage testing.

#### **label**

Label for test object

**setupcmd**

Setup command for creating test object. The setup command is executed in the base MATLAB workspace just prior to running the simulation. This command is useful for loading data prior to a test.

**Output Arguments****cvto**

A test specification object with the following structure.

Field	Description
<code>id</code>	Read-only internal ID
<code>modelcov</code>	Read-only internal ID
<code>rootPath</code>	Name of system or subsystem for analysis
<code>label</code>	String used when reporting results
<code>setupCmd</code>	Command executed in base workspace prior to simulation
<code>settings.condition</code>	Set to 1 for condition coverage.
<code>settings.decision</code>	Set to 1 for decision coverage.
<code>settings.designverifier</code>	Set to 1 for coverage for Simulink Design Verifier™ blocks.
<code>settings.mcdc</code>	Set to 1 for MCDC coverage.
<code>settings.overflowsaturation</code>	Set to 1 for saturate on integer overflow coverage.
<code>settings.relationalop</code>	Set to 1 for relational boundary coverage. Use <code>options.covBoundaryAbsTol</code> and <code>options.covBoundaryRelTol</code> for specifying tolerances for this coverage.  For more information, see “Relational Boundary Coverage”.
<code>settings.sigrange</code>	Set to 1 for signal range coverage.

Field	Description
<code>settings.sigsize</code>	Set to 1 for signal size coverage.
<code>settings.tableExec</code>	Set to 1 for lookup table coverage.
<code>modelRefSettings. enable</code>	<ul style="list-style-type: none"> <li>• 'off' — Disables coverage for all referenced models.</li> <li>• 'all' or on — Enables coverage for all referenced models.</li> <li>• 'filtered' — Enables coverage only for referenced models not listed in the <code>excludedModels</code> subfield.</li> </ul>
<code>modelRefSettings. excludeTopModel</code>	Set to 1 to exclude coverage for the top model
<code>modelRefSettings. excludedModels</code>	Character vector or string specifying a comma-separated list of referenced models for which coverage is disabled.
<code>emlSettings. enableExternal</code>	Set to 1 to enable coverage for external program files called by MATLAB functions in your model.
<code>sfcnSettings. enableSfcn</code>	Set to 1 to enable coverage for C/C++ S-Function blocks in your model.
<code>options. forceBlockReduction</code>	Set to 1 to override the Simulink <b>Block reduction</b> parameter if it is enabled.
<code>options. covBoundaryRelTol</code>	Set to the value of relative tolerance for relational boundary coverage.  For more information, see “Relational Boundary Coverage”.
<code>options. covBoundaryAbsTol</code>	Set to the value of absolute tolerance for relational boundary coverage.  For more information, see “Relational Boundary Coverage”.

Field	Description
options.mcdcmode	<ul style="list-style-type: none"> <li>'Masking' — Use the masking modified condition and decision coverage (MCDC) definition for recording MCDC coverage results.</li> <li>'UniqueCause' — Use the unique cause modified condition and decision coverage (MCDC) definition for recording MCDC coverage results</li> </ul> <p>For more information, see “Modified Condition and Decision Coverage (MCDC) Definitions in Simulink Coverage”.</p>
options.useTimeInterval	<p>Set to 1 to restrict model coverage recording only inside a specified simulation time interval.</p> <p>For more information see “Specify Coverage Options”.</p>
options.intervalStartTime	Value of the coverage recording interval start time.
options.intervalStopTime	Value of the coverage recording interval stop time.
filter.fileName	Character vector or string specifying name of coverage filter file, if you have excluded objects from coverage recording. See “Coverage Filter Rules and Files”.

## Examples

Create a cvtest object for the Adjustable Rate Limiter block in the slvndemo\_ratelim\_harness model. Simulate and get coverage data using cvsim.

```
open_system('slvndemo_ratelim_harness');
testObj = cvtest(['slvndemo_ratelim_harness', ...
    '/Adjustable Rate Limiter']);
testObj.label = 'Gain within slew limits';
testObj.setupCmd = ...
    'load(''slvndemo_ratelim_harness_data.mat'');';
testObj.settings.decision = 1;
testObj.settings.overflowsaturation = 1;
cvdo = cvsim(testObj);
```

## **See Also**

`cv.cvdatagroup` | `cvsim`

## **Topics**

“Create Tests with `cvtest`”

**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:

<b>Object Specification</b>	<b>Description</b>
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:

<b>Object Specification</b>	<b>Description</b>
{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 and 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:

<b>Object Specification</b>	<b>Description</b>
{fileName, funName}	Cell array with the name of a source file and a function name.



Object Specification	Description
{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:

<code>covered_outcomes</code>	Number of decision outcomes satisfied for object
-------------------------------	--



- 1 Open the model.
- 2 In the Model Editor, select **Analysis > Coverage > Settings**.
- 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 On the **Coverage > Results** pane, specify the output you need.
- 6 Click **OK** to close the Configuration Parameters dialog box and save your changes.
- 7 Simulate the model and review the results.

## See Also

[complexityinfo](#) | [conditioninfo](#) | [cvsim](#) | [getCoverageInfo](#) | [mcdcinfo](#) | [overflowsaturationinfo](#) | [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
<code>BlockPath</code>	Full path to a model or block
<code>BlockHandle</code>	Handle to a model or block
<code>sObj</code>	Handle to a Simulink API object
<code>sfID</code>	Stateflow ID
<code>sfObj</code>	Handle to a Stateflow API object from a singly instantiated Stateflow chart
<code>{BlockPath, sfID}</code>	Cell array with the path to a Stateflow chart or atomic subchart and the ID of an object contained in that chart or subchart
<code>{BlockPath, sfObj}</code>	Cell array with the path to a Stateflow chart or subchart and a Stateflow object API handle contained in that chart or subchart
<code>{BlockHandle, sfID}</code>	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
<code>{BlockPath, fName}</code>	Cell array with the path to an S-Function block and the name of a source file.
<code>{BlockHandle, fName}</code>	Cell array with an S-Function block handle and the name of a source file.
<code>{BlockPath, fName, funName}</code>	Cell array with the path to an S-Function block, the name of a source file, and a function name.
<code>{BlockHandle, fName, funName}</code>	Cell array with an S-Function block handle, the name of a source file and 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:

<b>Object Specification</b>	<b>Description</b>
{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:

<b>Object Specification</b>	<b>Description</b>
'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:

<code>covered_outcomes</code>	Number of execution outcomes satisfied for object
<code>total_outcomes</code>	Number of execution outcomes for object

### description

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

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

## Examples

Open the `slvndemo_cv_small_controller` model and create the test specification object `testObj`. Enable execution coverage for `slvndemo_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 = 'slvndemo_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 **Analysis > Coverage > Settings**.
- 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 On the **Coverage > Results** pane, specify the output you need.
- 6 Click **OK** to close the Configuration Parameters dialog box and save your changes.
- 7 Simulate the model and review the results.

## See Also

[complexityinfo](#) | [conditioninfo](#) | [cvsim](#) | [decisioninfo](#) | [getCoverageInfo](#) | [mcdcinfo](#) | [overflowsaturationinfo](#) | [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:

<b>Object Specification</b>	<b>Description</b>
'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, 'slvndemo_cv_small_controller');  
getAll(cvdg, 'slvndemo_cv_small_controller', 'ModelRefSIL');
```

## 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`.

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

<code>test</code>	Test Objective block
<code>proof</code>	Proof Objective block
<code>condition</code>	Test Condition block
<code>assumption</code>	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*].

<i>covered_outcomes</i>	Number of test objectives satisfied for <code>object</code>
<i>total_outcomes</i>	Total number of test objectives for <code>object</code>

`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 **Analysis > Coverage > Settings**.
- 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](#) | [mcdcinfo](#) | [overflowsaturationinfo](#) | [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
<code>BlockPath</code>	Full path to a model or block
<code>BlockHandle</code>	Handle to a model or block
<code>slObj</code>	Handle to a Simulink API object
<code>sfID</code>	Stateflow ID
<code>sfObj</code>	Handle to a Stateflow API object
<code>{BlockPath, sfID}</code>	Cell array with the path to a Stateflow chart or atomic subchart and the ID of an object contained in that chart or subchart
<code>{BlockPath, sfObj}</code>	Cell array with the path to a Stateflow chart or atomic subchart and a Stateflow object API handle contained in that chart or subchart
<code>{BlockHandle, sfID}</code>	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
<code>{BlockPath, fName}</code>	Cell array with the path to an S-Function block and the name of a source file.
<code>{BlockHandle, fName}</code>	Cell array with an S-Function block handle and the name of a source file.
<code>{BlockPath, fName, funName}</code>	Cell array with the path to an S-Function block, the name of a source file, and a function name.
<code>{BlockHandle, fName, funName}</code>	Cell array with an S-Function block handle, the name of a source file and 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:

<b>Object Specification</b>	<b>Description</b>
<code>{fileName, funName}</code>	Cell array with the name of a source file and a function name.
<code>{Model, fileName}</code>	Cell array with a model name (or model handle) and the name of a source file.
<code>{Model, fileName, funName}</code>	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:

<b>Object Specification</b>	<b>Description</b>
<code>'Normal'</code>	Model in Normal simulation mode.
<code>'SIL'</code> (or <code>'PIL'</code> )	Model in Software-in-the-Loop (SIL) or Processor-in-the-Loop (PIL) simulation mode.
<code>'ModelRefSIL'</code> (or <code>'ModelRefPIL'</code> )	Model reference in Software-in-the-Loop (SIL) or Processor-in-the-Loop (PIL) simulation mode.
<code>'ModelRefTopSIL'</code> (or <code>'ModelRefTopPIL'</code> )	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:

<code>covered_outcomes</code>	Number of condition/decision outcomes satisfied for <code>object</code>
-------------------------------	---

*total\_outcomes* Total number of condition/decision outcomes for object

### description

A structure array containing the following fields:

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

```
mdl = 'slvndemo_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 **Analysis > Coverage > Settings**.
- 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 On the **Coverage > Results** pane, specify the output you need.
- 6 Click **OK** to close the Configuration Parameters dialog box and save your changes.
- 7 Simulate the model and review the MCDC coverage results.

### See Also

[complexityinfo](#) | [conditioninfo](#) | [cvsim](#) | [decisioninfo](#) | [getCoverageInfo](#) | [overflowsaturationinfo](#) | [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(cvdata, object)
coverage = overflowsaturationinfo(cvdata, object,
ignore_descendants)
[coverage, description] = overflowsaturationinfo(cvdata, object)
```

## Description

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

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

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

## Examples

### Collect Saturation on Integer Overflow Coverage for MinMax Block

Collect saturation on integer overflow coverage information for a MinMax block in the example model `sldemo_fuelsys`.

Open the `sldemo_fuelsys` example model. Create a model coverage test specification object for the Mixing & Combustion subsystem of the Engine Gas Dynamics subsystem.

```
open_system('sldemo_fuelsys');
testObj = cvtest('sldemo_fuelsys/Engine Gas Dynamics/', ...
    'Mixing & Combustion');
```

In the model coverage test specification object, specify to collect saturation on overflow coverage.

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

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

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

Get the saturation on overflow coverage results for the MinMax block in the Mixing & Combustion subsystem. The coverage results are stored in a two-element vector of the form `[covered_outcomes total_outcomes]`.

```
blockHandle = get_param('sldemo_fuelsys/' ...
    'Engine Gas Dynamics/Mixing & Combustion/MinMax', 'Handle');
covResults = overflowsaturationinfo(dataObj, blockHandle)
```

```
covResults =
    1     2
```

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.

## Collect Saturation on Integer Overflow Coverage and Description for Example Model

Collect saturation on integer overflow coverage for the example model `slvndemo_saturation_on_overflow_coverage`. Review collected coverage results and description for Sum block in Controller subsystem.

Open the `slvndemo_saturation_on_overflow_coverage` example model.

```
open_system('slvndemo_saturation_on_overflow_coverage');
```



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

```
dataObj = cvsim('slvndemo_saturation_on_overflow_coverage');
```

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

```
[covResults, covDesc] = overflowsaturationinfo(dataObj, ...
        'slvndemo_saturation_on_overflow_coverage/Test Unit /' ...
        'Controller/Sum')
```

```
covResults =
```

```
    1    2
```

```
covDesc =
```

```
    isFiltered: 0
    decision: [1x1 struct]
```

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.

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

```
ans =
```

```
    executionCount: 3
                text: 'false'
```

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

```
ans =
```

```
    executionCount: 0
                text: 'true'
```

During simulation, integer overflow did not occur in the Sum block.

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: 'slvndemo\_saturation\_on\_overflow\_coverage'

Example: `get_param('slvndemo_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:

<code>covered_outcomes</code>	Number of saturation on integer overflow outcomes satisfied for <code>object</code>
<code>total_outcomes</code>	Total number of saturation on integer overflow outcomes for <code>object</code>

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:

<code>isFiltered</code>	0 if the model component specified by <code>object</code> is not excluded from coverage recording. 1 if the model component specified by <code>object</code> 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` | `getCoverageInfo` | `mcdcinfo` | `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(cvdata, object)
coverage = relationalboundaryinfo(cvdata, object, mode)
coverage = relationalboundaryinfo(cvdata, object,
ignore_descendants)
[coverage, description] = relationalboundaryinfo(cvdata, object)
```

### Description

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

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

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

`[coverage, description] = relationalboundaryinfo(cvdata, object)` returns relational boundary coverage results from the `cvdata` object `cvdata` 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 `slvndemo_cv_small_controller` model. Create a model coverage test specification object for the model.

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

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

```
testObj.settings.relatiop = 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 `slvndemo_cv_small_controller`. The coverage results are stored in a two-element vector of the form `[covered_outcomes total_outcomes]`.

```
blockHandle = get_param('slvndemo_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

*covdata* object

Coverage results data, specified as a *covdata* 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

Object Specification	Description
{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 and 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:						
	<table> <tr> <td data-bbox="798 348 926 371">isActive</td> <td data-bbox="1069 348 1325 569">Boolean variable. If this variable is <code>false</code>, it indicates that decisions were not evaluated during simulation due to variable signal size.</td> </tr> <tr> <td data-bbox="798 586 940 609">execCount</td> <td data-bbox="1069 586 1314 708">Number of times <code>op_1-op_2</code> fell in the range described by <code>text</code></td> </tr> <tr> <td data-bbox="798 725 866 748">text</td> <td data-bbox="1069 725 1325 944">The range around the relational boundary considered for coverage. For more information, see “Relational Boundary”.</td> </tr> </table>	isActive	Boolean variable. If this variable is <code>false</code> , it indicates that decisions were not evaluated during simulation due to variable signal size.	execCount	Number of times <code>op_1-op_2</code> fell in the range described by <code>text</code>	text	The range around the relational boundary considered for coverage. For more information, see “Relational Boundary”.
isActive	Boolean variable. If this variable is <code>false</code> , it indicates that decisions were not evaluated during simulation due to variable signal size.						
execCount	Number of times <code>op_1-op_2</code> fell in the range described by <code>text</code>						
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 <code>object</code> is not excluded from coverage recording. 1 if the model component specified by <code>object</code> 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 <code>object</code> , if <code>object</code> 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` |  
`getCoverageInfo` | `mcdcinfo` | `overflowsaturationinfo` | `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
<code>BlockPath</code>	Full path to a model or block
<code>BlockHandle</code>	Handle to a model or block
<code>slObj</code>	Handle to a Simulink API object

<b>Object Specification</b>	<b>Description</b>
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 the Product block in the slvndemo\_cv\_small\_controller model:

```
mdl = 'slvndemo_cv_small_controller';  
open_system(mdl)  
%Create test spec object
```



```
testObj = cvtest mdl)
%Enable signal range coverage
testObj.settings.sigrange = 1;
%Simulate the model
data = cvsim(testObj)
blk_handle = get_param([mdl, '/Product'], 'Handle');
%Get signal range data
[minVal, maxVal] = sigrangeinfo(data, blk_handle)
```

## 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 **Analysis > Coverage > Settings**.
- 3 On the **Coverage** pane of the Configuration Parameters dialog box, select **Enable coverage analysis**.
- 4 Under **Coverage metrics**, select **Signal Range**.
- 5 On the **Coverage > Results** pane, specify the output you need.
- 6 Click **OK** to close the Configuration Parameters dialog box and save your changes.
- 7 Simulate the model and review the results.

## See Also

[complexityinfo](#) | [conditioninfo](#) | [cvsim](#) | [decisioninfo](#) | [getCoverageInfo](#) | [mcdinfo](#) | [overflowsaturationinfo](#) | [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:

<b>Object Specification</b>	<b>Description</b>
BlockPath	Full path to a Simulink model or block
BlockHandle	Handle to a Simulink model or block
slobj	Handle to a Simulink API object

Object Specification	Description
<code>sfID</code>	Stateflow ID
<code>sfObj</code>	Handle to a Stateflow API object
<code>{BlockPath, sfID}</code>	Cell array with the path to a Stateflow chart or atomic subchart and the ID of an object contained in that chart or subchart
<code>{BlockPath, sfObj}</code>	Cell array with the path to a Stateflow chart or atomic subchart and a Stateflow object API handle contained in that chart or subchart
<code>{BlockHandle, sfID}</code>	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 **Analysis > Coverage > Settings**.
- 3 On the **Coverage** pane of the Configuration Parameters dialog box, select **Enable coverage analysis**.
- 4 Under **Coverage metrics**, select **Signal Size**.
- 5 On the **Coverage > Results** pane, specify the output you need.
- 6 Click **OK** to close the Configuration Parameters dialog box and save your changes.
- 7 Simulate the model and review the results.

## See Also

`complexityinfo` | `conditioninfo` | `cvsim` | `decisioninfo` | `mcdcinfo` | `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_mdhref_conversion` model and copy it into a new model:

```
open_system('sldemo_mdhref_conversion');  
newmodel = slvnvextract('sldemo_mdhref_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**

# slvnharnessopts

Generate default options for `slvnvmakeharness`

## Syntax

`harnessopts = slvnharnessopts`

## Description

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

## Output Arguments

**harnessopts** — Default configuration for `slvnvmakeharness` structure

The `harnessopts` structure can have the following fields. If you do not specify any values, default values are used.

Field	Description
<code>harnessFilePath</code>	Specifies the file path for creating the harness model. If an invalid path is specified, <code>slvnvmakeharness</code> does not save the harness model, but it creates and opens the harness model. If you do not specify this option, <code>slvnvmakeharness</code> generates a new harness model and saves it in the MATLAB current folder.  Default: ''

Field	Description
<code>modelRefHarness</code>	Generates the test harness model that includes <code>model</code> in a Model block. When <code>false</code> , the test harness model includes a copy of <code>model</code> .  Default: <code>true</code>
<code>usedSignalsOnly</code>	When <code>true</code> , 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 <code>model</code> must be compatible with that software to detect the input signals.  Default: <code>false</code>

## Examples

### Create a Test Harness with Default Options

```
% Create a test harness for the sldemo_mdhref_house model
% using the default options:
open_system('sldemo_mdhref_house');
harnessOpts = slvnharnessopts;
[harnessfile] = slvnvmakeharness('sldemo_mdhref_house',...
    '', harnessOpts);
```

## See Also

`slvnvmakeharness`

**Introduced in R2010b**



# slvnlvlogssignals

Log test data for component or model during simulation

## Syntax

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

## Description

`data = slvnlvlogssignals(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 = slvnlvlogssignals(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 `slvnlvmakeharness`.

`data = slvnlvlogssignals(harness_model, test_case_index)` simulates every test case in the Signal Builder block of the `harness_model` specified by `test_case_index`. `slvnlvlogssignals` logs the input signals to the Test Unit block in the harness model. If you omit `test_case_index`, `slvnlvlogssignals` 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 `slvnmakeharness` 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_mdhref_counter model, in the context of the
% sldemo_mdhref_basic model and log the data:
open_system('sldemo_mdhref_basic');
data = slvnlvsignals('sldemo_mdhref_basic/CounterB');

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

```
slvnmakeharness('slldemo_mdref_counter', data, ...  
harness0pts);
```

## See Also

[sldvmakeharness](#) | [slvnmakeharness](#) | [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 slvnvlogs signals 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_md1ref_house` model using the default options:

```
open_system('sldemo_md1ref_house');  
harness0pts = slvnharnessopts;  
[harnessfile] = slvnvmakeharness('sldemo_md1ref_house', '', harness0pts);
```

## See Also

`slvnharnessopts` | `slvnvmmergeharness`

**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_mdhref_basic model, which contains three Model blocks
% that reference the sldemo_mdhref_counter model:
sldemo_mdhref_basic;

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

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

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

### See Also

[sldvrun](#) | [slvnlvlogs](#) | [slvnmmakeharness](#) | [slvnruncgvtest](#) | [slvnruntest](#)

**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 `slvnmergeharness` for `name` maintain the structure and initialization from the earlier run. If `name` matches an existing Simulink model, `slvnmergeharness` 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 `slvnmergeharness`.

### **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

`slvnmergeharness` 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_mdref_basic example model
% that each reference the same model:
open_system('sldemo_mdref_basic');
data1 = slvnvlogsignals('sldemo_mdref_basic/CounterA');
data2 = slvnvlogsignals('sldemo_mdref_basic/CounterB');
data3 = slvnvlogsignals('sldemo_mdref_basic/CounterC');
open_system('sldemo_mdref_counter');

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

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

## See Also

slvnlvlogsignals | slvnmmakeharness

**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 `slvnvlogssignals` function.

### runOpts — Specify the configuration of slvnvruncgvttest

structure

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

Field Name	Description
<code>testIdx</code>	<p>Test case index array to simulate from <code>dataFile</code>.</p> <p>If <code>testIdx = []</code> (the default), <code>slvnvruncgvttest</code> simulates all test cases.</p>
<code>allowCopyModel</code>	<p>If you have not configured your model to execute test cases with the CGV API, this field specifies creating and configuring the model.</p> <p>If <code>true</code> and you have not configured your model to execute test cases with the CGV API, <code>slvnvruncgvttest</code> copies the model, fixes the configuration, and executes the test cases on the copied model.</p> <p>If <code>false</code> (the default), an error occurs if the tests cannot execute with the CGV API.</p> <hr/> <p><b>Note</b> If you have not configured the top-level model or any referenced models to execute test cases, <code>slvnvruncgvttest</code> does not copy the model, even if <code>allowCopyModel</code> is <code>true</code>. An error occurs.</p>
<code>cgvCompType</code>	<p>Defines the software-in-the-loop (SIL) or processor-in-the-loop (PIL) approach for CGV:</p> <ul style="list-style-type: none"> <li>• <code>'topmodel'</code> (default)</li> <li>• <code>'modelblock'</code></li> </ul>
<code>cgvConn</code>	<p>Specifies mode of execution for CGV:</p> <ul style="list-style-type: none"> <li>• <code>'sim'</code> (default)</li> <li>• <code>'sil'</code></li> <li>• <code>'pil'</code></li> </ul>

**Note** `runOpts = slvnvrntestopts('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
<code>tout_slvnvruncgvtest</code>	Simulation time
<code>xout_slvnvruncgvtest</code>	State data
<code>yout_slvnvruncgvtest</code>	Output signal data
<code>logout_slvnvruncgvtest</code>	Signal logging data for: <ul style="list-style-type: none"> <li>• Signals connected to outports</li> <li>• Signals that are configured for logging data on the model</li> </ul>

## Examples

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

```
% Open the sldemo_mdref_basic example model and log the input signals to the CounterA Model block:
open_system('sldemo_mdref_basic');
load_system('sldemo_mdref_counter');
loggedData = slvnvlogsignals('sldemo_mdref_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 = slvnvrntestopts('cgv');
runOpts.allowCopyModel = true;
```

```
% Using the logged signals, execute slvnvruncgvttest – 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 = slvnvruncgvttest('sldemo_mdhref_counter', loggedData, runOpts);
runOpts.cgvConn = 'sil';
cgvObjectSil = slvnvruncgvttest('sldemo_mdhref_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 `slvnvruncgvttest`, you must have the Embedded Coder® software.

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

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

## See Also

`cgv.CGV` | `slvnvlogssignals` | `slvnvruntest` | `slvnvruntestopts`

**Introduced in R2010b**

## slvnvrntest

Simulate model by using input data

### Syntax

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

### Description

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

`[outData, covData] = slvnvrntest(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. `slvnvrntest` 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 `slvnvlogssignals` function.

### **runOpts — Configuration specification**

structure

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

Field	Description
<code>testIdx</code>	Test case index array to simulate from <code>dataFile</code> . If <code>testIdx</code> is <code>[]</code> , <code>slvnvrntest</code> simulates all test cases.  <b>Default:</b> <code>[]</code>
<code>coverageEnabled</code>	If <code>true</code> , specifies that the Simulink Coverage software collects model coverage data during simulation.  <b>Default:</b> <code>false</code>
<code>coverageSetting</code>	<code>cvtest</code> object for collecting model coverage. If <code>[]</code> , <code>slvnvrntest</code> uses the existing coverage settings for <code>model</code> .  <b>Default:</b> <code>[]</code>

## 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
<code>tout_slvnvrntest</code>	Simulation time
<code>xout_slvnvrntest</code>	State data
<code>yout_slvnvrntest</code>	Output signal data

Field Name	Description
logout_slvnvrntest	Signal logging data for: <ul style="list-style-type: none"> <li>• Signals connected to outputs</li> <li>• Signals that are configured for logging on the model</li> </ul>

**covData — Object that contains model coverage data**

cvdata object

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

## Examples

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

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

% Using the logged signals, simulate the model referenced in the Counter block (sldemo_mdref_counter):
runOpts = slvnvrntestopts;
runOpts.coverageEnabled = true;
open_system('sldemo_mdref_counter');
[ outData ] = slvnvrntest('sldemo_mdref_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('logout_slvnvrntest')});
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.

## **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`.

`runOpts = slvnvruntestopts('cgv')` generates 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
<code>testIdx</code>	Test case index array to simulate or execute from data file. If <code>testIdx = []</code> , all test cases are simulated or executed. <b>Default:</b> <code>[]</code>

Field Name	Description
signalLoggingSaveFormat	<p>Available only for slvnvruntest.</p> <p>Specifies the format of signal logging data for signals that connects to the output of the model and for intermediate signals that are configured for logging.</p> <p>If you specify <code>Dataset</code>, data is stored in the <code>Simulink.SimulationData.Dataset</code> objects.</p> <p>If you specify <code>ModelDataLogs</code>, data is stored in <code>Simulink.ModelDataLogs</code> objects.</p> <p><b>Default:</b> 'Dataset'</p>
coverageEnabled	<p>Available only for slvnvruntest.</p> <p>If <code>true</code>, <code>slvnvruntest</code> collects model coverage data during simulation.</p> <p><b>Default:</b> <code>false</code></p>
coverageSetting	<p>Available only for slvnvruntest.</p> <p><code>cvtest</code> object for collecting model coverage.</p> <p>If <code>coverageSetting</code> is <code>[]</code>, <code>slvnvruntest</code> uses the coverage settings for the model specified in the call to <code>slvnvruntest</code>.</p> <p><b>Default:</b> <code>[]</code></p>

Field Name	Description
allowCopyModel	<p>Available only for <code>slvnvruncgvtest</code>.</p> <p>If you have not configured your model to execute test cases with the CGV API, this field specifies creating and configuring the model.</p> <p>If <code>true</code> and you have not configured the model to execute test cases with the CGV API, <code>slvnvruncgvtest</code> copies the model, fixes the configuration, and executes the test cases on the copied model.</p> <p>If <code>false</code>, an error occurs if the tests cannot execute with the CGV API.</p> <hr/> <p><b>Note</b> If you have not configured the top-level model or any referenced models to execute test cases, <code>slvnvruncgvtest</code> does not copy the model, even if <code>allowCopyModel</code> is <code>true</code>. An error occurs.</p> <hr/> <p><b>Default:</b> <code>false</code></p>
cgvCompType	<p>Available only for <code>slvnvruncgvtest</code>.</p> <p>Defines the software-in-the-loop (SIL) or processor-in-the-loop (PIL) approach for CGV:</p> <ul style="list-style-type: none"> <li>• <code>'topmodel'</code></li> <li>• <code>'modelblock'</code></li> </ul> <p><b>Default:</b> <code>'topmodel'</code></p>

Field Name	Description
cgvConn	<p>Available only for <code>slvnvruncgvtest</code>.</p> <p>Specifies mode of execution for CGV:</p> <ul style="list-style-type: none"> <li>• 'sim'</li> <li>• 'sil'</li> <li>• 'pil'</li> </ul> <p><b>Default:</b> 'sim'</p>

## 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', cvdata

## **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:

<code>covered_intervals</code>	Number of interpolation/extrapolation intervals satisfied for <code>object</code>
<code>total_intervals</code>	Total number of interpolation/extrapolation intervals for <code>object</code>

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

```
mdl = 'slvndemo_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 MDCD 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 **Analysis > Coverage > Settings**.
- 3 On the **Coverage** pane of the Configuration Parameters dialog box, select **Enable coverage analysis**.
- 4 Under **Coverage metrics**, select **Lookup Table**.
- 5 On the **Coverage > Results** pane, specify the output you need.
- 6 Click **OK** to close the Configuration Parameters dialog box and save your changes.
- 7 Simulate the model and review the results.

## See Also

[complexityinfo](#) | [conditioninfo](#) | [cvsim](#) | [decisioninfo](#) | [getCoverageInfo](#) | [mcdcinfo](#) | [overflowsaturationinfo](#) | [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” (Simulink)

“Templates for C S-Functions” (Simulink)

“Coverage for Custom C/C++ Code in Simulink Models”

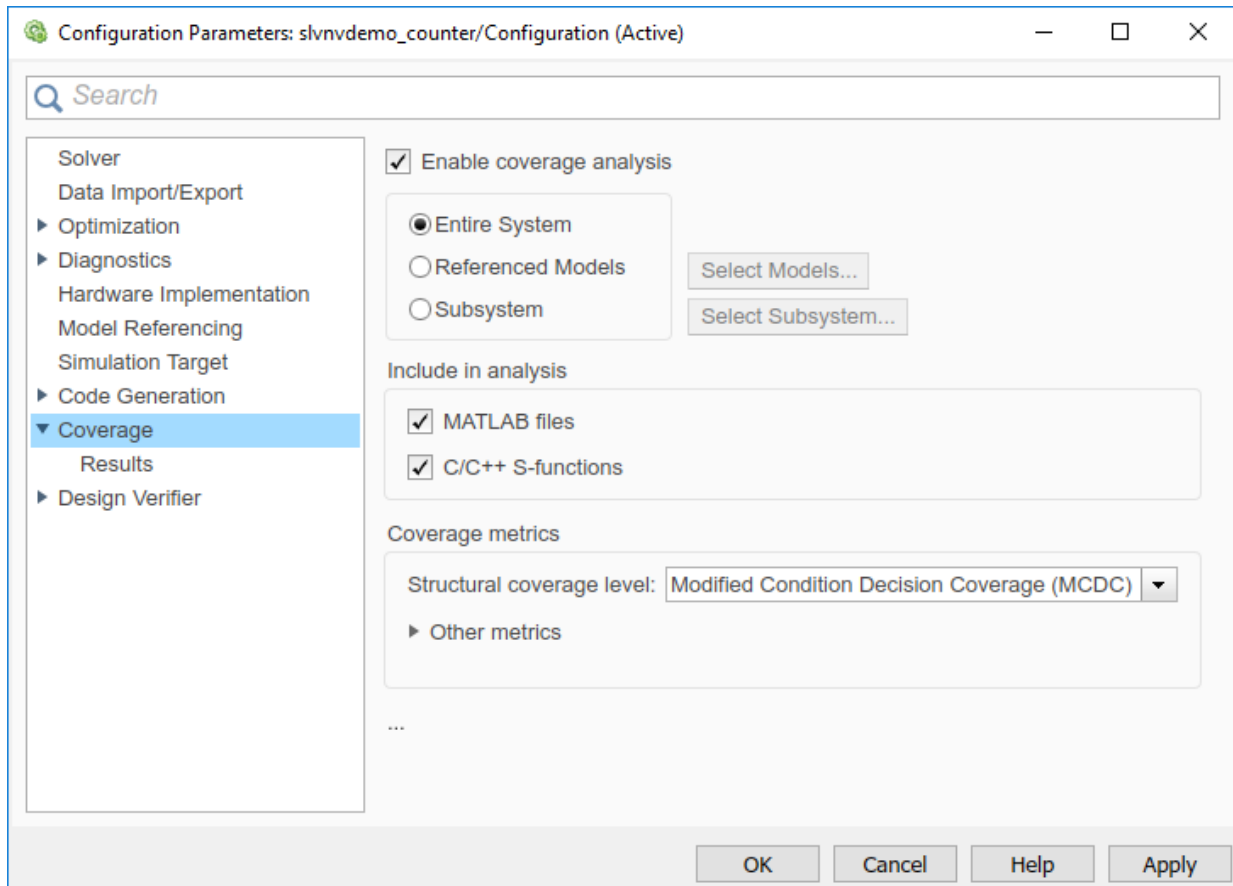
“View Coverage Results for Custom C/C++ Code in S-Function Blocks”

**Introduced in R2015a**

# Simulink Coverage Settings

---

# Coverage Pane



### In this section...

- “Coverage Pane Overview” on page 2-3
- “Enable coverage analysis” on page 2-3
- “Scope of coverage analysis” on page 2-4
- “Select Models” on page 2-5
- “Select Subsystem” on page 2-5

**In this section...**

“Record coverage for MATLAB files” on page 2-7

“Record coverage for C/C++ S-functions” on page 2-7

“Structural coverage level” on page 2-8

“Lookup table” on page 2-9

“Signal range” on page 2-10

“Signal size” on page 2-10

“Objectives and constraints” on page 2-11

“Saturation on integer overflow” on page 2-12

“Relational boundary” on page 2-12

“Relational boundary coverage absolute tolerance” on page 2-13

“Relational boundary coverage relative tolerance” on page 2-13

“Restrict coverage recording interval” on page 2-14

“Coverage interval start time” on page 2-15

“Coverage interval stop time” on page 2-15

“Force block reduction off” on page 2-16

“Treat Simulink logic blocks as short-circuited” on page 2-16

“MCDC mode” on page 2-17

“Warn when unsupported blocks exist in model” on page 2-18

“Coverage filter filename” on page 2-18

“Coverage metric settings” on page 2-19

“Record coverage for this model” on page 2-20

“Record coverage for referenced models” on page 2-21

“Include top model” on page 2-22

**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 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 2-3.

### 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 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.

### Dependency

To enable this parameter, select:

- “Enable coverage analysis” on page 2-3
- 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'.

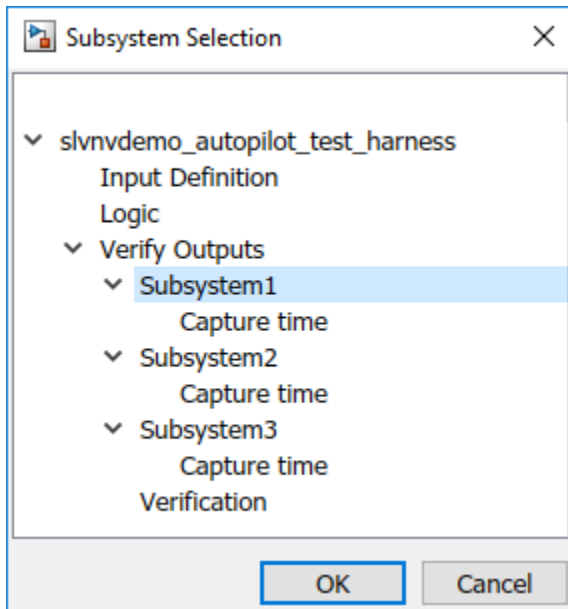
**Default:** ''

## 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 2-3
- “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 2-3.

### 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 2-3
- “Record coverage for this model” on page 2-20 or “Record coverage for referenced models” on page 2-21 (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 2-3.

### 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 2-3.

### 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 2-3.

#### 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” (Simulink).

Off (default)

Signal size information is not recorded.

## Dependency

To enable this parameter, select “Enable coverage analysis” on page 2-3.

### 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 2-3.

### 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 2-3.

#### 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 2-3.

**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” (MATLAB).

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 2-3
- “Relational boundary” on page 2-12

**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  $lhs \leq rhs$  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 2-3
- “Relational boundary” on page 2-12

### Command-Line Information

**Parameter:** CovBoundaryRelTol

**Type:** Floating-point number

**Value:** Relative tolerance value such as 0.001

**Default:** 0.01

## 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 2-3.

### 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 2-3
- “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 2-3
- “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” (Simulink).

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 2-3.

### 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 2-3.

## 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.

### UniqueCause

Use unique-cause MCDC analysis.

### Dependency

To enable this parameter, select:

- “Enable coverage analysis” on page 2-3
- Specify Modified Condition/Decision Coverage (MCDC) for “Structural coverage level” on page 2-8.

### 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 2-3.

### 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 2-3.

### 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 2-18
- e: Same result as disabling “Display coverage results using model coloring” on page 2-25

### Dependency

To enable this parameter, select:

- “Enable coverage analysis” on page 2-3
- One of these: “Record coverage for this model” on page 2-20, “Record coverage for referenced models” on page 2-21 (enter on) or “Record coverage for MATLAB files” on page 2-7

### 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 2-3.
  - 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 2-3.
  - 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.
- **off:** Coverage data is not collected for referenced models.
- **filtered:** Coverage data is collected for all referenced 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 2-3.
  - 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 2-3
- Specify referenced model for “Scope of coverage analysis” on page 2-4.

### Command-Line Information

**Parameter:** CovIncludeTopModel

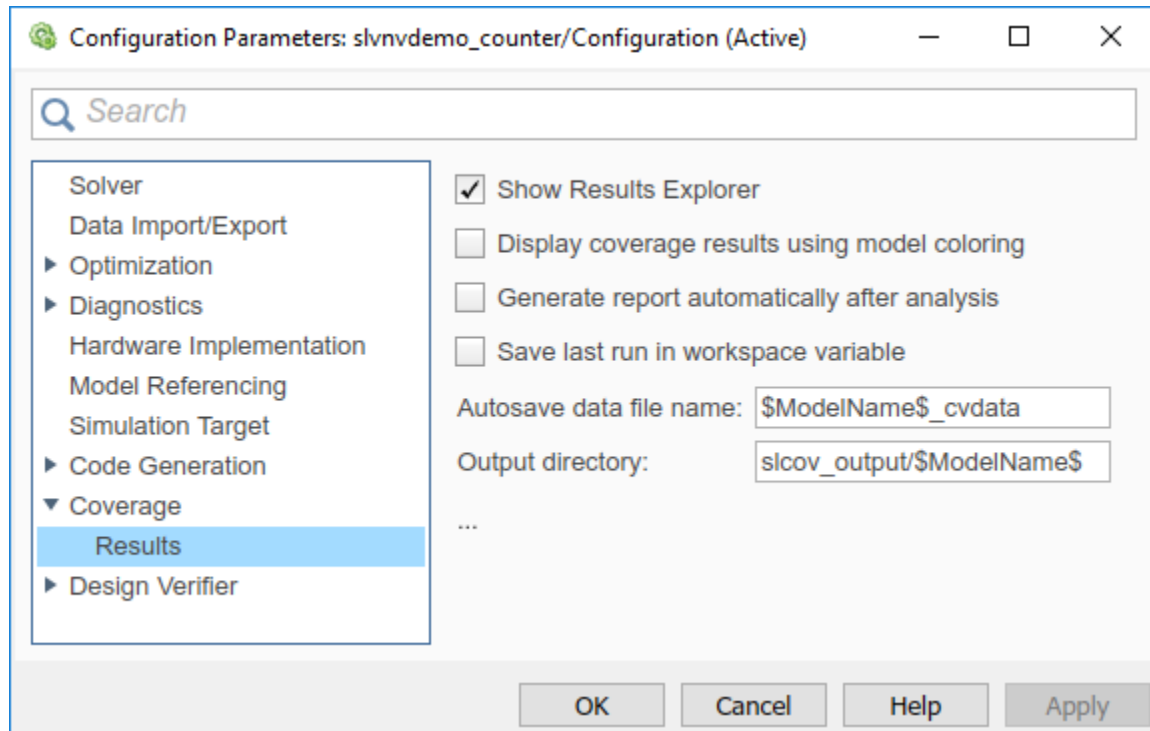
**Type:** Character vector or string

**Value:** 'on'|'off'

**Default:** 'on'



## Coverage Pane: Results



### In this section...

- “Coverage Results Pane Overview” on page 2-24
- “Show Results Explorer” on page 2-24
- “Display coverage results using model coloring” on page 2-25
- “Generate report automatically after analysis” on page 2-26
- “Save last run in workspace variable” on page 2-27
- “Last coverage run variable name” on page 2-27
- “Increment variable name with each simulation” on page 2-28
- “Autosave data file name” on page 2-29
- “Output directory” on page 2-29

### In this section...

“Coverage report options” on page 2-30

“Additional data to include in coverage report” on page 2-32

“Update coverage results on pause” on page 2-32

“Save output data” on page 2-33

“Enable cumulative data collection” on page 2-33

“Include cumulative data in coverage report” on page 2-34

“Save cumulative coverage results in workspace variable” on page 2-35

“Cumulative coverage variable name” on page 2-36

## Coverage Results Pane Overview

Specify Simulink Coverage coverage results options.

## Show Results Explorer

Show Coverage Results Explorer after simulation. When you run a simulation, the Coverage Results Explorer opens to show the most recent coverage run.

### Settings

On (default)

When you run a simulation, the Coverage Results Explorer opens to show the most recent coverage run. See “Access, Manage, and Accumulate Coverage Results”

Off

The Coverage Results Explorer does not open after simulation.

You can open it later. Select **Analysis > Coverage > Open Results Explorer**.

---

**Note** If you enable the Simulink Toolstrip tech preview and you use the toolbar buttons to simulate a model with coverage enabled, the Results Explorer does not appear after a simulation. You can access the Results Explorer from the Simulink Coverage contextual tabs, which appear when you open the **Coverage Analyzer** app, under **Verification, Validation, and Test**.

---

## Dependency

To enable this parameter, select “Enable coverage analysis” on page 2-3.

## Command-Line Information

**Parameter:** CovShowResultsExplorer

**Type:** Character vector or string

**Value:** 'on'|'off'

**Default:** 'on'

## Display coverage results using model coloring

Color blocks in the model based on coverage received during simulation.

### Settings

On

Coverage results are displayed on the model. If a model receives incomplete coverage during simulation, it is colored light red. If it receives complete coverage, it is green. See also “View Coverage Results in a Model”.

Off (default)

Coverage results are not displayed on the model.

You can enable coloring based on coverage later. Select **Analysis > Coverage > Open Results Explorer**. In the Coverage Results Explorer, select a coverage result from the data repository. Select **Highlight model with coverage results** below the coverage results summary.

---

**Note** If you use the toolbar buttons to simulate a model with coverage enabled, this setting is not honored and the model coloring for coverage results always appears after each simulation. You can click **Highlight model with coverage results** in the Results Explorer to enable or disable model coverage highlighting. You access the Results Explorer by selecting **Analysis > Coverage > Open Results Explorer**. For more information, see “Accessing Coverage Data from the Results Explorer”.

If you enabled the Simulink toolstrip tech preview, you enable an disable model highlighting from the Simulink Coverage contextual tabs, which appear when you open the **Coverage Analyzer** app, under **Verification, Validation, and Test**.

---

### Dependency

To enable this parameter, select “Enable coverage analysis” on page 2-3.

### Command-Line Information

**Parameter:** CovHighlightResults

**Type:** Character vector or string

**Value:** 'on'|'off'

**Default:** 'off'

## Generate report automatically after analysis

Create an HTML report containing coverage results after simulation.

### Settings

On

An HTML report containing coverage results opens after simulation. Specify the report location using the parameter “Output directory”.

Off (default)

The HTML report is not generated after simulation.

You can generate the report later. Select **Analysis > Coverage > Open Results Explorer**. In the Coverage Results Explorer, select a coverage result from the data repository. Select **Generate report** below the coverage results summary.

---

**Note** If you enable the Simulink Toolstrip tech preview and you use the toolbar buttons to simulate a model with coverage enabled, the HTML report does not appear after a simulation. You access the HTML report from the Simulink Coverage contextual tabs, which appear when you open the **Coverage Analyzer** app, under **Verification, Validation, and Test**.

---

### Dependency

To enable this parameter, select “Enable coverage analysis” on page 2-3.

**Command-Line Information****Parameter:** CovHtmlReporting**Type:** Character vector or string**Value:** 'on'|'off'**Default:** 'off'**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 2-27. Choose to create a new object for each simulation using the parameter “Increment variable name with each simulation” on page 2-28.

 Off (default)

Coverage data is not stored in a MATLAB variable.

**Dependency**

To enable this parameter, select “Enable coverage analysis” on page 2-3.

**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 2-28. 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 2-3
- “Save last run in workspace variable” on page 2-27

### 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 2-27.

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 2-3
- “Save last run in workspace variable” on page 2-27

## 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 2-3
- “Save output data” on page 2-33

## 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 2-3.

### Command-Line Information

**Parameter:** `CovOutputDir`

**Type:** Character vector or string

**Value:** Path to folder

**Default:** `'slcov_output/$modelName$'`

### 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.



## Summary





### Model Hierarchy/Complexity Test 1

	Decision	Condition	MCDC	Execution
1. <a href="#">slvnydemo_counter</a>	3 25% 	50% 	0% 	86% 

- ' -bTC=1 ' — Use two color bar graphs (red, blue).
- ' -hTR=1 ' — Display hit/count ratio in the model summary.

## Summary

### Model Hierarchy/Complexity Test 1

	Decision	Condition	MCDC	Execution
1. <a href="#">slvnydemo_counter</a>	3 25% (1/4) 	50% (4/8) 	0% (0/2) 	86% (6/7) 

- ' -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.

### Dependency

To enable this parameter, select:

- “Enable coverage analysis” on page 2-3
- “Generate report automatically after analysis” on page 2-26

### 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 2-27
- “Cumulative coverage variable name” on page 2-36

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 2-3
- “Generate report automatically after analysis” on page 2-26

#### 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 2-3.

### 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 2-29.

Off

Coverage data results are not saved to a file.

### Dependency

To enable this parameter, select “Enable coverage analysis” on page 2-3.

### 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 2-34. To save the data in one workspace variable, use the parameters “Save cumulative coverage results in workspace variable” on page 2-35 and “Cumulative coverage variable name” on page 2-36.

Off

Model coverage data is retained for the most recent simulation only.

### Dependency

To enable this parameter, select “Enable coverage analysis” on page 2-3.

### 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 2-3
- “Generate report automatically after analysis” on page 2-26
- “Enable cumulative data collection” on page 2-33

### 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 2-36.

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 2-3
- “Enable cumulative data collection” on page 2-33

### 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 2-3
- “Save cumulative coverage results in workspace variable” on page 2-35
- “Enable cumulative data collection” on page 2-33

### Command-Line Information

**Parameter:** `CovCumulativeVarName`

**Type:** Character vector or string  
**Value:** Name to be given to cvdata object  
**Default:** 'covCumulativeData'





# Class Reference

---

## 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'`

## 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 vector

This property is read-only.

Code used to create this selector object, returned as a character vector.

### **Description** — Description of the selector

character vector

This property is read-only.

Description of the selector, returned as a character vector. Simulink Coverage creates the description based on the selector.

**Id — Element identifier**

Simulink ID (default) | property | handle

This property is read-only.

Identifier of the model element, returned as character vector of the Simulink ID, model element property, or handle. This property is empty for the `slcoverage.CodeSelector` class.

**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 (MATLAB).

## 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, 'CovMetricSettings', 'dcme', '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”](#)

[“Get a Simulink Identifier” \(Simulink\)](#)

[“Create, Edit, and View Coverage Filter Rules”](#)

### Introduced in R2017b

# slcoverage.CodeSelector class

**Package:** slcoverage

Select custom C/C++ code for coverage filter

## Description

Specify custom C/C++ code selection criteria for a filter rule.

## Construction

`sel = slcoverage.CodeSelector(type, file)` creates the selector based on the C or C++ file.

`sel = slcoverage.CodeSelector(type, file, function)` creates the selector based on the C or C++ function in the file.

`sel = slcoverage.CodeSelector(type, file, function, expression, index)` creates the selector based on the index of the decision or condition expression.

## Input Arguments

**type — Code selector type**

`slcoverage.CodeSelectorType` value

Type of custom C/C++ code to select, as one of these values:

- `slcoverage.CodeSelectorType.File` — Custom C/C++ code file name.
- `slcoverage.CodeSelectorType.Function` — Custom C/C++ code function name.
- `slcoverage.CodeSelectorType.Decision` — A custom C/C++ code decision.
- `slcoverage.CodeSelectorType.Condition` — A custom C/C++ code condition.

Example: `slcoverage.CodeSelectorType.Function`

### **file — C or C++ file to select**

character vector or string

C or C++ file to select, specified as a character vector or string.

Example: 'myfile.c'

### **function — C or C++ function to select**

character vector or string

C or C++ function to select, specified as a character vector or string.

Example: 'counterbusFcn'

### **expression — Decision expression to select**

character vector or string

Decision or condition expression to select, specified as a character vector or string.

Example: 'inputGElower'

### **index — Matrix position of expression to select**

integer

Matrix position of expression to select, specified as an integer.

Example: 2

## Outputs

### **sel — Selector object**

slcoverage.CodeSelector object | array of slcoverage.CodeSelector objects

Selector object, returned as an slcoverage.CodeSelector object or array of slcoverage.CodeSelector objects.

## Properties

### **ConstructorCode — Code used to create this selector object**

character vector

This property is read-only.



Code used to create this selector object, returned as a character vector.

### **Description — Description of the selector**

character vector

This property is read-only.

Description of the selector, returned as a character vector. Simulink Coverage creates the description based on the selector.

### **Id — Element identifier**

Simulink ID (default) | property | handle

This property is read-only.

Identifier of the model element, returned as character vector of the Simulink ID, model element property, or handle. This property is empty for the `slcoverage.CodeSelector` class.

### **Type — Code selector type**

`slcoverage.CodeSelectorType` value

This property is read-only.

Selector type, returned as one of these `slcoverage.CodeSelectorType` values:

- File
- Function
- Decision
- Condition

## **Methods**

`allSelectors`                      Selectors for model or code element

## **Copy Semantics**

Handle. To learn how handle classes affect copy operations, see [Copying Objects \(MATLAB\)](#).

## Examples

### Add Code Selector Rules to a Filter

Select custom C/C++ code to add a rule for and select a code construct to add a rule for. The resulting filter has one rule. You can simulate your model for coverage by using the filter to see the effect.

Open the model. Specify coverage settings and turn on coverage recording.

```
modelName = 'sllexCCallerExample';  
open_system(modelName);  
set_param(modelName, 'SimAnalyzeCustomCode', 'on', 'CovMetricSettings', 'dcme', 'Record');
```

Add a filter rule for the custom C function `times2`.

```
sel = slcoverage.CodeSelector(slcoverage.CodeSelectorType.Function, 'my_func.c', 'times2');
```

Create a filter object, create a rule based on the selector, and add the rule to the filter.

```
filt = slcoverage.Filter;  
rule = slcoverage.FilterRule(sel, 'Tested elsewhere', slcoverage.FilterMode.Exclude);  
filt.addRule(rule);
```

Save the filter as `codefilter`. Simulate the model for code coverage. Add the filter file to the filter property of the resulting `cvdata` object.

```
filt.save('codefilter');  
csim = cvsim(modelName);  
csim.get('my_func.c').filter = 'codefilter';
```

Generate a coverage report.

```
cvhtml('cov', csim);
```

Review the report. Click the `my_func.c` Custom Code File(s) link and find the filter rule that you added under **Objects Filtered from Coverage Analysis**.

## Analysis Information

### Objects Filtered from Coverage Analysis

Code	Rationale
Function <code>times2</code> (line <a href="#">3</a> )	Tested elsewhere

## See Also

`cv.cvdatagroup` | `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

Create a coverage filter object to add filter rules to.

### Construction

`filt = slcoverage.Filter()` creates an `slcoverage.Filter` object.

`filt = slcoverage.Filter(filterFile)` adds the filter rules in `filterFile` to the filter.

### Input Arguments

**filterFile** — Filter file

path name

Filter file (.cvf file), specified as a character vector of the path name to the file. You do not need to include the extension.

Example: 'myfilt', 'filters/myfilt'

### Methods

<code>addRule</code>	Add coverage filtering rule to filter
<code>removeRule</code>	Remove rule from filter rule set
<code>rules</code>	Rules for filter

## Copy Semantics

Handle. To learn how handle classes affect copy operations, see Copying Objects (MATLAB).

## Examples

### Create and Use a Filter Object

Create a filter object and add a rule to it. In this example, you add a rule to exclude some blocks from coverage testing.

Open the model. Specify coverage settings and turn on coverage recording.

```
modelName = 'sldemo_lct_bus';
open_system(modelName);
set_param(modelName, 'CovMetricSettings', 'dcme', '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.

```
filt = slcoverage.Filter;
rule = slcoverage.FilterRule(bl, 'Tested elsewhere', slcoverage.FilterMode.Exclude);
filt.addRule(rule);
```

After you create a filter and add one or more rules to it, save the filter to a file. Simulate the model for code coverage. Add the filter file as the value to the `filter` property of the resulting `cvdata` object.

```
filt.save('blfilter');
csim = cvsim(modelName);
csim.filter = 'blfilter';
cvhtml('cov', csim);
```

Examine the HTML report and notice the rules that were added for the blocks. The coverage report shows the excluded blocks and the rationale.

### Add Rule to a Filter File

This example assumes that you have an existing filter file `myfilt.cvf` that you want to add a rule to. Create a filter object that uses that file. Add a rule to the filter object and then save the file again.

```
filt = slcoverage.Filter('myfilt');  
bl = slcoverage.BlockSelector(slcoverage.BlockSelectorType.BlockInstance, 'sldemo_lct_bus:23');  
  
rule = slcoverage.FilterRule(bl, 'Edge case');  
filt.addRule(rule);  
filt.save('myfilt')
```

### 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 \(MATLAB\)](#).

## 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, 'CovMetricSettings', 'dcme', '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

Specify metric selection criteria for a filter rule.

### Construction

`sel = slcoverage.MetricSelector(type, element, objIndex, outIndex)` specifies the model element and the block metrics to create the rule for. Specify whether the type of selector is a condition or decision outcome. Then select the object and outcome index combination to specify the metric that you want to write the filter rule for.

You can create only a justify rule for a metric selector. You cannot create an exclude rule.

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.RelationalBoundaryOutcome` |  
`slcoverage.MetricSelectorType.SaturationOverflowOutcome`

Metric selector type, specified as:

- `slcoverage.MetricSelectorType.ConditionOutcome` selects outcome metrics related to block inputs.
- `slcoverage.MetricSelectorType.DecisionOutcome` selects outcome metrics related to block outputs.

- `slcoverage.MetricSelectorType.RelationalBoundaryOutcome` selects outcome metrics related to relational boundary outcomes.
- `slcoverage.MetricSelectorType.SaturationOverflowOutcome` selects 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 model element Simulink identifier.

Example: `'sldemo_lct_bus:18'`

**objIndex — Matrix position of objective**

integer

Matrix position of objective to select, specified as an integer that corresponds to the row of the coverage table.

Example: 1

**outIndex — Matrix position of outcome**

integer

Matrix position of the outcome to select, specified as an integer that corresponds to the column of the coverage table.

Example: 2

## Properties

**ConstructorCode — Code used to create this selector object**

character vector

This property is read-only.

Code used to create this selector object, returned as a character vector.

**Description — Description of the selector**

character vector

This property is read-only.

Description of the selector, returned as a character vector. Simulink Coverage creates the description based on the selector.

**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 handle.

**ObjectiveIndex — Matrix position of objective**

integer

This property is read-only.

Matrix position of objective for this selector, returned as an integer.

**OutcomeIndex — Matrix position of outcome**

integer

This property is read-only.

Matrix position of outcome for this selector, returned as an integer.

**Type — Metric selector type**

ConditionOutcome | DecisionOutcome | RelationalBoundaryOutcome | SaturationOverflowOutcome

This property is read-only.

Selector type, returned as ConditionOutcome, DecisionOutcome, 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

`allSelectors`                      Selectors for model or code element

## Copy Semantics

Handle. To learn how handle classes affect copy operations, see Copying Objects (MATLAB).

## Examples

### **Add Matrix Selector Rule to Filter**


Select a metric, create a rule for it, and add the rule to a filter. Then you can simulate the model for code coverage using the filter to see the effect.

In this example, you create a rule to justify the untested condition for the And block.

Logic block "[And](#)"


[Justify or Exclude](#)

Parent: [sldemo\\_lct\\_bus/slCounter](#)

Uncovered Links: 

Metric	Coverage
Cyclomatic Complexity	0
Condition	75% (3/4) condition outcomes
MCDC	50% (1/2) conditions reversed the outcome
Execution	100% (1/1) objective outcomes

Conditions analyzed

Description	True	False
input port 1	190	111
input port 2	301	0 

Open the model. Specify coverage settings and turn on coverage recording.

```
modelName = 'sldemo_lct_bus';
open_system(modelName);
set_param(modelName, 'CovMetricSettings', 'dcme', 'RecordCoverage', 'on');
```

Get the identifier for the And block. Create the metric selector for the block. For this example, filter the second condition (input port 2) False outcome, whose index, then, is 2,2. Use the ConditionOutcome selector type and the index 2,2.

```
id = Simulink.ID.getSID('sldemo_lct_bus/slCounter/And');
metr = slcoverage.MetricSelector(slcoverage.MetricSelectorType.ConditionOutcome,id,2,2);
```

Create a filter object and create a rule using the default filter mode of justify. Add the rule to the filter.

```
filt = slcoverage.Filter;
rule = slcoverage.FilterRule(metr, 'Expected result');
filt.addRule(rule);
```

Save the filter as metrfilter. 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('metrfilter');
csim = cvsim(modelName);
csim.filter = 'metrfilter';
cvhtml('cov',csim);


```

Examine the HTML report and view the condition table for the And block. The report now shows 100% coverage for the condition and that the untested condition was justified.

### Logic block "[And](#)"

#### [Justify or Exclude](#)

Parent: [sldemo\\_1ct\\_bus/slCounter](#)

Uncovered Links: 

Metric	Coverage
Cyclomatic Complexity	0
Condition	100% ((3+1)/4) condition outcomes
MCDC	50% (1/2) conditions reversed the outcome
Execution	100% (1/1) objective outcomes

#### Conditions analyzed

Description	True	False
input port 1	190	111
input port 2	301	<a href="#">J1</a>

## See Also

[cv.cvdatagroup](#) | [getSimulinkBlockHandle](#) | [slcoverage.BlockSelector](#) | [slcoverage.Filter](#) | [slcoverage.FilterRule](#) | [slcoverage.SFcnSelector](#)

## Topics

"Top-Level Model Coverage Report"

"Get a Simulink Identifier" (Simulink)

"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 vector

This property is read-only.

Code used to create this selector object, returned as a character vector.

### **Description** — Description of the selector

character vector

This property is read-only.

Description of the selector, returned as a character vector. Simulink Coverage creates the description based on the selector.

### **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 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 (MATLAB).

## Examples

### Get All Selectors

Get all the selectors for the block. Then you can add a rule to exclude or justify a selector. (You can only justify metric selectors.)

Open the model and turn on coverage recording. Get all the selectors for the And block.

```
modelName = 'sldemo_lct_bus';  
open_system(modelName);  
set_param(modelName, 'CovMetricSettings', 'dcme', 'RecordCoverage', 'on');  
id = Simulink.ID.getSID('sldemo_lct_bus/slCounter/And');  
sel = slcoverage.Selector.allSelectors(id)
```

```
s =
```

```
1x6 heterogeneous Selector (BlockSelector, MetricSelector) array with properties:
```

```
  Description  
  Type  
  Id  
  ConstructorCode
```

The block has six selectors. You can index into each one to see the content. The sixth selector is the metric selector that you want to justify.

```
sel(6)
```

```

ans =

MetricSelector with properties:

    ObjectiveIndex: 2
    OutcomeIndex: 2
    Description: 'F outcome of input port 2 in Logic block "And"'
    Type: ConditionOutcome
    Id: 'sldemo_lct_bus:23'
    ConstructorCode: 'slcoverage.MetricSelector(slcoverage.MetricSelectorType.ConditionOutcome,'sldemo_lct_bus:23',2,2)'

```

Create a justify rule for the selector. Create a filter object and add the rule to it.

```

rule = slcoverage.FilterRule(sel(6), 'Expected result');
filt = slcoverage.Filter;
filt.addRule(rule);

```

Save and run the filter.

```

filt.save('metrfilter');
csim = cvsim(modelName);
csim.filter = 'metrfilter';
cvhtml('cov',csim);

```

## 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

Specify S-function selection criteria for a filter rule.

### Construction

`sel = slcoverage.SFcnSelector(type,element)` specifies the type of selector to use for the selected model element.

`sel = slcoverage.SFcnSelector(type,element,file)` creates the selector based on the specified C or C++ file.

`sel = slcoverage.SFcnSelector(type,element,file,function)` creates the selector based on the specified C or C++ function in the specified file.

`sel = slcoverage.SFcnSelector(type,element,file,function,expression,index)` creates the selector based on the specified index of the specified decision expression.

### Input Arguments

#### **type — Type of S-function**

`slcoverage.SFcnSelectorType` value

Type of S-function to select, specified as one of these values:

- `slcoverage.SFcnSelectorType.SFcnName` selects the specified S-function.
- `slcoverage.SFcnSelectorType.SFcnInstanceCppFileName` selects the coverage data in the generated code file for this block. Use with the `file` argument.

- `slcoverage.SFcnSelectorType.SFcnInstanceCppFunction` selects an instance of a C or C++ function. Use with the `file` and `function` arguments.
- `slcoverage.SFcnSelectorType.SFcnInstanceCppCondition` selects a condition outcome of the specified code. Use with `file`, `function`, `expression`, and `index` arguments.
- `slcoverage.SFcnSelectorType.SFcnInstanceCppDecision` selects a decision outcome of the specified. Use with `file`, `function`, `expression`, and `index` arguments.

#### **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'

#### **file — C or C++ file to select**

character vector or string

C or C++ file to select, specified as a character vector or string.

Example: 'myfile.c'

#### **function — C or C++ function to select**

character vector or string

C or C++ function to select, specified as a character vector or string.

Example: 'counterbusFcn'

#### **expression — Decision expression to select**

character vector or string

Decision expression to select, specified as a character vector or string.

Example: 'inputGElower'

#### **index — Matrix position of expression to select**

integer

Matrix position of expression to select, specified as an integer.

Example: 2

## Outputs

### **sel — Selector object**

`slcoverage.SFcnSelector` object | array of `slcoverage.SFcnSelector` objects

Selector object, returned as an `slcoverage.SFcnSelector` object or array of `slcoverage.SFcnSelector` objects.

## Properties

### **ConstructorCode — Code used to create this selector object**

character vector

This property is read-only.

Code used to create this selector object, returned as a character vector.

### **Description — Description of the selector**

character vector

This property is read-only.

Description of the selector, returned as a character vector. Simulink Coverage creates the description based on the selector.

### **Expr — Decision expression for this selector**

character vector or string

This property is read-only.

Decision expression for this selector, returned as a character vector.

### **FileName — C or C++ file for selector**

character vector or string

This property is read-only.

C or C++ file for selector, returned as a character vector or string.

**FunctionName — C or C++ function name for selector**

character vector or string

This property is read-only.

C or C++ function name for this selector, returned as a character vector.

**Id — Element identifier**

Simulink ID (default) | property | handle

This property is read-only.

Identifier of the model element, returned as character vector of the Simulink ID, model element property, or handle. This property is empty for the `slcoverage.CodeSelector` class.

**Type — Selector type**`slcoverage.SFcnSelectorType` value

Selector type, returned as one of these `slcoverage.SFcnSelectorType` values:

- `SFcnName`
- `SFcnInstanceCppFileName`
- `SFcnInstanceCppFunction`
- `SFcnInstanceCppCondition`
- `SFcnInstanceCppDecision`

## Methods

`allSelectors`                      Selectors for model or code element

## Copy Semantics

Handle. To learn how handle classes affect copy operations, see [Copying Objects \(MATLAB\)](#).

## Examples

### Create and Run Filter for S-Function

Open and set up the model for S-function code coverage.

```
modelName = 'sldemo_lct_bus';
open_system(modelName);
set_param(modelName, 'CovMetricSettings', 'dcme', 'RecordCoverage', 'on', 'CovSFcnEnable', 'on');

def = legacy_code('initialize');
def.SFunctionName = 'sldemo_sfun_counterbus';
def.OutputFcnSpec = 'void counterbusFcn(COUNTERBUS u1[1],int32 u2,COUNTERBUS y1[1],int32 y2[1])';
def.HeaderFiles = {'counterbus.h'};
def.SourceFiles = {'counterbus.c'};
def.IncPaths = {'sldemo_lct_src'};
def.SrcPaths = {'sldemo_lct_src'};
def.Options.supportCoverageAndDesignVerifier = true;

legacy_code('generate_for_sim', def);
```

Get the S-function selectors from the sldemo\_sfun\_counterbus block. Examine the constructor code property value for all the selectors.

```
sel = slcoverage.SFcnSelector.allSelectors('sldemo_lct_bus/TestCounter/sldemo_sfun_counterbus');
sel.ConstructorCode

ans =

    'slcoverage.SFcnSelector(slcoverage.SFcnSelectorType.SFcnName, 'sldemo_sfun_counterbus')'

ans =

    'slcoverage.SFcnSelector(slcoverage.SFcnSelectorType.SFcnInstanceCppFileName, 'sldemo_lct_bus:15',
    'counterbus.c')'

ans =

    'slcoverage.SFcnSelector(slcoverage.SFcnSelectorType.SFcnInstanceCppFunction, 'sldemo_lct_bus:15',
    'counterbus.c', 'counterbusFcn')'

ans =

    'slcoverage.SFcnSelector(slcoverage.SFcnSelectorType.SFcnInstanceCppDecision, 'sldemo_lct_bus:15',
    'counterbus.c', 'counterbusFcn', '(u1->limits.upper_saturation_limit >= limit) && inputGELower', 1)'

ans =

    'slcoverage.SFcnSelector(slcoverage.SFcnSelectorType.SFcnInstanceCppDecision, 'sldemo_lct_bus:15',
    'counterbus.c', 'counterbusFcn', 'inputGELower', 2)'

ans =
```



```
'slcoverage.SFcnSelector(slcoverage.SFcnSelectorType.SFcnInstanceCppCondition,'sldemo_lct_bus:15',  
'counterbus.c','counterbusFcn','limit >= u1->limits.lower_saturation_limit',1)'
```

Create a filter object. Create a rule based on one of selectors by indexing into the selector object.

```
filt = slcoverage.Filter;  
rule = slcoverage.FilterRule(sel(6), 'OK to exclude', slcoverage.FilterMode.Exclude);  
filt.addRule(rule);
```

Save the filter as `sffilter`. 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('sffilter');  
csim = cvsim(modelName);  
csim.filter = 'sffilter';  
cvhtml('cov', csim);
```

Review the report. Click the `sldemo_sfun_counterbus` link and find the filter rule that you added under **Objects Filtered from Coverage Analysis**.

## Analysis Information

### Model Information

Model version	1.211
Author	The MathWorks, Inc.
Last saved	Fri Jun 30 07:52:25

[Justify or Exclude](#)

### Objects Filtered from Coverage Analysis

Code	Rationale
<code>Condition limit &gt;= u1-&gt;limits.lower_saturation_limit (line 13)</code>	OK to exclude

## See Also

`cv.cvdatagroup` | `getSimulinkBlockHandle` | `slcoverage.BlockSelector` | `slcoverage.Filter` | `slcoverage.FilterRule` | `slcoverage.MetricSelector`

## Topics

“Top-Level Model Coverage Report”  
“Get a Simulink Identifier” (Simulink)  
“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, 'CovMetricSettings', 'dcme', '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, 'CovMetricSettings', 'dcme', '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 =
```

```
1x2 FilterRule array with properties:
```

```
Selector
Mode
Rationale
```

```
ans =
```

```
FilterRule with properties:
```

```
Selector: [1x1 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: [1x1 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, 'CovMetricSettings', 'dcme', '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', 'F  
outcome'
```

### **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.BlockSelectorType`, `slcoverage.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` — Custom C/C++ code file name.
  - `slcoverage.CodeSelectorType.Function` — Custom C/C++ code function name.
  - `slcoverage.CodeSelectorType.Decision` — A custom C/C++ code decision.
  - `slcoverage.CodeSelectorType.Condition` — A custom C/C++ code condition.
- Metric selector types:
  - `slcoverage.MetricSelectorType.ConditionOutcome` selects outcome metrics related to block inputs.
  - `slcoverage.MetricSelectorType.DecisionOutcome` selects outcome metrics related to block outputs.
  - `slcoverage.MetricSelectorType.RelationalBoundaryOutcome` selects outcome metrics related to relational boundary outcomes.
  - `slcoverage.MetricSelectorType.SaturationOverflowOutcome` selects 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:

<b>Object Specification</b>	<b>Description</b>
'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

Get all the selectors for the block. Then you can add a rule to exclude or justify a selector. (You can only justify metric selectors.)

Open the model and turn on coverage recording. Get all the selectors for the And block.

```
modelName = 'sldemo_lct_bus';
open_system(modelName);
set_param(modelName, 'CovMetricSettings', 'dcme', 'RecordCoverage', 'on');
```

```
id = Simulink.ID.getSID('sldemo_lct_bus/slCounter/And');
sel = slcoverage.Selector.allSelectors(id)

s =
    1x6 heterogeneous Selector (BlockSelector, MetricSelector) array with properties:
        Description
        Type
        Id
        ConstructorCode
```

The block has six selectors. You can index into each one to see the content. The sixth selector is the metric selector that you want to justify.

```
sel(6)
```

```
ans =
    MetricSelector with properties:
        ObjectiveIndex: 2
        OutcomeIndex: 2
        Description: 'F outcome of input port 2 in Logic block "And"'
        Type: ConditionOutcome
        Id: 'sldemo_lct_bus:23'
        ConstructorCode: 'slcoverage.MetricSelector(slcoverage.MetricSelectorType.ConditionOutcome,'sldemo_lct_bus:23',2,2)'
```

Create a justify rule for the selector. Create a filter object and add the rule to it.

```
rule = slcoverage.FilterRule(sel(6), 'Expected result');
filt = slcoverage.Filter;
filt.addRule(rule);
```

Save and run the filter.

```
filt.save('metrfilter');
csim = cvsim(modelName);
csim.filter = 'metrfilter';
cvhtml('cov', csim);
```

### Get Selector by Type and Description

Get a particular metric selector.

Open the model and turn on code coverage.



```
modelName = 'sldemo_lct_bus';
open_system(modelName);
set_param(modelName, 'CovMetricSettings', 'dcme', 'RecordCoverage', 'on');
```

Get the condition selectors for the And block whose description includes the text F outcome.

```
id = Simulink.ID.getSID('sldemo_lct_bus/slCounter/And');
s = slcoverage.Selector.allSelectors(id,...
    'Type',slcoverage.MetricSelectorType.ConditionOutcome,'Description','F outcome')
s =
    1x2 MetricSelector array with properties:
        ObjectiveIndex
        OutcomeIndex
        Description
        Type
        Id
        ConstructorCode
```

Look at the constructor code for the two selectors that were returned.

**s.ConstructorCode**

```
ans =
    'slcoverage.MetricSelector(slcoverage.MetricSelectorType.ConditionOutcome,'sldemo_lct_bus:23',1,2)'
```

```
ans =
    'slcoverage.MetricSelector(slcoverage.MetricSelectorType.ConditionOutcome,'sldemo_lct_bus:23',2,2)'
```

## See Also

[slcoverage.BlockSelector](#) | [slcoverage.CodeSelector](#) | [slcoverage.MetricSelector](#) | [slcoverage.SFcnSelector](#) | [slcoverage.Selector](#)

**Introduced in R2017b**

