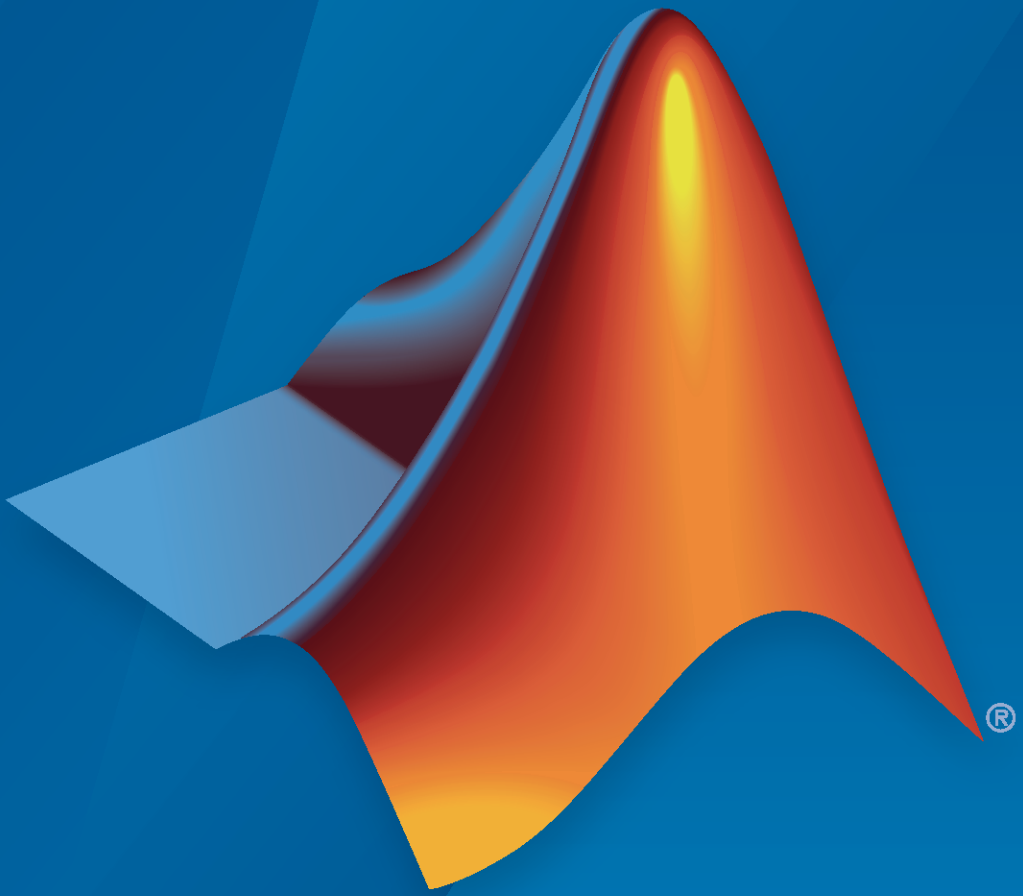
Simulink® Coverage™

Reference



MATLAB®&SIMULINK®

R2023a

MathWorks®

How to Contact MathWorks

Latest news:	`www.mathworks.com`
Sales and services:	`www.mathworks.com/sales_and_services`
User community:	`www.mathworks.com/matlabcentral`
Technical support:	`www.mathworks.com/support/contact_us`
Phone:	508-647-7000

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

Simulink® Coverage™ Reference

Revision History

September 2017	Online only	New for Version 4.0 (Release 2017b)
March 2018	Online only	Revised for Version 4.1 (Release 2018a)
September 2018	Online only	Revised for Version 4.2 (Release 2018b)
March 2019	Online only	Revised for Version 4.3 (Release R2019a)
September 2019	Online only	Revised for Version 4.4 (Release R2019b)
March 2020	Online only	Revised for Version 5.0 (Release R2020a)
September 2020	Online only	Revised for Version 5.1 (Release R2020b)
March 2021	Online only	Revised for Version 5.2 (Release R2021a)
September 2021	Online only	Revised for Version 5.3 (Release R2021b)
March 2022	Online only	Revised for Version 5.4 (Release R2022a)
September 2022	Online only	Revised for Version 5.5 (Release R2022b)
March 2023	Online only	Revised for Version 5.6 (Release R2023a)

Contents

1

Functions

allNames

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

Get names of `cvdata` objects in `cv.cvdatagroup` object

Syntax

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

Description

`names = allNames(cvdg)` returns the `cvdata` object names in the `cv.cvdatagroup` object `cvdg`.

`names = allNames(cvdg,simMode)` returns the `cvdata` object names that have the simulation mode `simMode`.

Input Arguments

cvdg — Coverage data group
`cv.cvdatagroup` object

Coverage data group, specified as a `cv.cvdatagroup` object.

Data Types: `cv.cvdatagroup`

`simMode` — Simulation mode
`"Normal"` | `"SIL"` | `"PIL"` | `"ModelRefSIL"` | `"ModelRefPIL"` | `"ModelRefTopSIL"` | `"ModelRefTopPIL"`

Simulation mode that corresponds to one or more `cvdata` objects in the `cv.cvdatagroup` object, specified as one of 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 the code interface set to the top model.

If the function does not find a match, it returns an empty array. If the function finds multiple matches, it returns a cell array.

Data Types: `char` | `string`

Output Arguments

names — Coverage data object names
cell array

Coverage data object names, returned as a cell array of character arrays. `names` contains the model, subsystem, or external file names associated with the `cvdata` objects contained in `cvdg`. The object names refer to the name of the top-level model object that you analyzed for coverage.

Examples

Get Coverage Data from cvdatagroup Objects

This example shows how to retrieve a specified coverage data object, all coverage data objects, or all coverage data object names from a `cv.cvdatagroup` object.

Load the example model.

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

Use the `Simulink.SimulationInput` class to set up the simulation scenario with coverage enabled.

```
simIn = Simulink.SimulationInput(modelName);
simIn = setModelParameter(simIn,'CovEnable','on');
simIn = setModelParameter(simIn,'CovMetricStructuralLevel','MCDC');
simIn = setModelParameter(simIn,'CovSaveSingleToWorkspaceVar','on');
simIn = setModelParameter(simIn,'CovSaveName','covData');
simIn = setModelParameter(simIn,'CovSaveOutputData','off');
```

Simulate the model by passing the `SimulationInput` object to the `sim` function, and then retrieve the coverage data from the `SimulationOutput` object.

```
simOut = sim(simIn);
covData = simOut.covData
```

```
covData = ... cv.cvdatagroup

  slcoverageExternalFile (simulation mode: Normal)
  slvnvdemo_eml_model_coverage_demo (simulation mode: Normal)
```

The example model contains a MATLAB® Function block that calls an external MATLAB function. Because Simulink Coverage returns the coverage data for the MATLAB function in a separate `cvdata` object, `sim` returns a `cv.cvdatagroup` object that contains the coverage results for both the model and the MATLAB function.

To determine the name associated with each `cvdata` object inside the `cv.cvdatagroup`, use the class method `allNames`.

```
covDataNames = allNames(covData)
covDataNames =

  2x1 cell array

    {'slcoverageExternalFile'           }
    {'slvnvdemo_eml_model_coverage_demo'}
```

The `cv.cvdatagroup` contains the coverage results for the model as well as the MATLAB function.

To retrieve the coverage data for the model, use the `get` class method.

```
modelCovData = get(covData,modelName)
```

```
modelCovData = ... cvdata
            version: (R2023a)
                 id: 1314
               type: TEST_DATA
               test: cvtest object
             rootID: 1316
           checksum: [1x1 struct]
          modelinfo: [1x1 struct]
          startTime: 03-Mar-2023 13:40:11
           stopTime: 03-Mar-2023 13:40:11
  intervalStartTime: 0
   intervalStopTime: 0
simulationStartTime: 0
 simulationStopTime: 1
             filter: 
            simMode: Normal
```

To retrieve coverage data for both the model and the external file, use the `getAll` class method.

```
covDataArray = getAll(covData)
```

```
covDataArray =

  2x1 cell array

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

Version History

Introduced in R2007b

complexityinfo

Retrieve cyclomatic complexity coverage information from `cvdata` object

Syntax

```
complexity = complexityinfo(cvdo,modelObject)
complexity = complexityinfo(cvdo,modelObject,simMode)
```

Description

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

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

Examples

Retrieve Cyclomatic Complexity Data from Coverage Object

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

Load the `slvnvdemo_cv_small_controller` model.

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

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

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

Retrieve cyclomatic complexity information for the Gain subsystem.

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

```
gainComplexity =

     1     0
```

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

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

```
switchComplexity =

     1     1
```

Input Arguments

cvdo — Coverage data
cvdata object

Coverage data, specified as a `cvdata` object.

Data Types: cvdata

modelObject — Model object
character array | string array | Simulink® handle | Stateflow® ID | cell array

Model object, specified as a character array, string array, Simulink handle, Stateflow ID, or cell array.

To specify a model object, such as a block or a Stateflow chart, use one of these formats:

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

To specify an S-Function block or its contents, use one of these formats:

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

Object Specification	Description
`{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

To specify a code coverage result, such as coverage data collected during software-in-the-loop (SIL) or processor-in-the-loop (PIL) analysis, use one of these formats:

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

Data Types: `char` | `string` | `cell` | `Stateflow.State` | `Stateflow.Transition`

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

Simulation mode during coverage analysis, specified as one of these options:

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 SIL or PIL simulation mode.
`"ModelRefTopSIL"` or `"ModelRefTopPIL"`	Model reference in SIL or PIL simulation mode with the code interface set to top model.

Data Types: `char` | `string`

Output Arguments

`complexity` — Cyclomatic complexity
`scalar`

Cyclomatic complexity, returned as a two-element array of the form `[total_complexity,local_complexity]` if `cvdo` contains cyclomatic complexity coverage results, or an empty array if it does not.

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

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

Data Types: `double`

Alternatives

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

1. Open the model.
2. In the Simulink Editor, select **Model Settings** on the **Modeling** tab.
3. On the **Coverage** pane of the Configuration Parameters dialog box, select **Enable coverage analysis**.
4. Under **Coverage metrics**, select **MCDC** as the structural coverage level.
5. Click **OK** to close the Configuration Parameters dialog box and save your changes.
6. Simulate the model.
7. In the docked **Coverage Details** pane, the coverage report shows the cyclomatic complexity for the model as well as each model object.

Version History

Introduced in R2011a

See Also

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

Topics

"Cyclomatic Complexity"

conditioninfo

Retrieve condition coverage information from `cvdata` object

Syntax

```
covInfo = conditioninfo(cvdo,modelObject)
covInfo = conditioninfo(cvdo,modelObject,simMode)
covInfo = conditioninfo(cvdo,modelObject,ignoreDescendants)
[covInfo,description] = conditioninfo(cvdo,modelObject)
```

Description

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

`covInfo = conditioninfo(cvdo,modelObject,simMode)` returns the condition coverage results from the `cvdata` object `cvdo` for the simulation mode `simMode`.

`covInfo = conditioninfo(cvdo,modelObject,ignoreDescendants)` returns the condition coverage results for `modelObject`, including or ignoring descendant objects based on the value of `ignoreDescendants`.

`[covInfo,description] = conditioninfo(cvdo,modelObject)` returns the condition coverage results and textual descriptions for each condition in `modelObject`.

Examples

View Condition Coverage Data

This example shows how to view condition coverage for a block in your model.

Load the model.

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

Configure the coverage settings for the model by using a `Simulink.SimulationInput` object.

```
simIn = Simulink.SimulationInput(modelName);
simIn = simIn.setModelParameter('CovEnable','on');
simIn = simIn.setModelParameter('CovMetricStructuralLevel','MCDC');
simIn = simIn.setModelParameter('CovSaveSingleToWorkspaceVar','on');
simIn = simIn.setModelParameter('CovSaveName','covData');
```

Simulate the model by passing `simIn` as the input to `sim`.

```
simOut = sim(simIn);
covData = simOut.covData;
```

View the condition coverage results for the Logic block in the Gain subsystem by calling `conditioninfo` with the block path.

```
condCov = conditioninfo(covData,[modelName,'/Gain/Logic'])
```

```
condCov =

     2     4
```

`conditioninfo` returns an array with two scalars. The first value is the number of satisfied condition outcomes, and the second value is the number of total condition outcomes. Use these to determine the percentage of satisfied condition outcomes for the Logic block.

```
percentCondCov = 100 * condCov(1) / condCov(2)
```

```
percentCondCov =

    50
```

Input Arguments

cvdo — Coverage data

`cvdata` object

Coverage data, specified as a `cvdata` object.

Data Types: `cvdata`

modelObject — Model object

`character array` | `string array` | `Simulink handle` | `Stateflow ID` | `cell array`

Model object, specified as a character array, string array, Simulink handle, Stateflow ID, or cell array.

To specify a model object, such as a block or a Stateflow chart, use one of these formats:

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

To specify an S-Function block or its contents, use one of these formats:

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

To specify a code coverage result, such as coverage data collected during software-in-the-loop (SIL) or processor-in-the-loop (PIL) analysis, use one of these formats:

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

Data Types: `char` | `string` | `cell` | `Stateflow.State` | `Stateflow.Transition`

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

Simulation mode during coverage analysis, specified as one of these options:

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 SIL or PIL simulation mode.
`"ModelRefTopSIL"` or `"ModelRefTopPIL"`	Model reference in SIL or PIL simulation mode with the code interface set to top model.

Data Types: `char` | `string`

`ignoreDescendants` — Whether to ignore descendants in coverage results
`false` or `0` (default) | `true` or `1`

Whether to ignore descendants in coverage results, specified as a numeric or logical `1 (true)` or `0 (false)`, where:

- `0 (false)` includes coverage results of descendant objects.
- `1 (true)` ignores coverage results of descendant objects.

Data Types: `single` | `double` | `logical`

Output Arguments

`covInfo` — Coverage information
`scalar`

Coverage information, returned as a two-element array of the form `[covered_outcomes,total_outcomes]` if `cvdo` contains condition coverage data, or an empty array if it does not.

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

Data Types: `double`

`description` — Condition coverage description
`structure`

Condition coverage description, returned as a structure with the following fields:

`isFiltered` — Block exclusion flag
`0 | 1`

Block exclusion flag, returned as `1` if the block is excluded and `0` if it is not.

Data Types:

`filterRationale` — Block coverage filter rationale
`character array`

Block coverage filter rationale, returned as a character array.

Data Types: `char`

`justifiedCoverage` — Number of justified coverage objective outcomes
`scalar`

Number of justified coverage objective outcomes, returned as a scalar double.

Data Types: `double`

`isJustified` — Block justification flag
`0 | 1`

Block justification flag, returned as `1` if the block is justified or `0` if it is not.

Data Types: `double`

`condition` — Information for individual condition outcomes
`structure`

Information for individual condition outcomes, returned as a structure with the following fields:

`isFiltered` — Condition exclusion flag
`0 | 1`

Condition exclusion flag, returned as `1` if the condition is excluded or `0` if it is not.

Data Types: `double`

`isJustified` — Condition justification flag
`0 | 1`

Condition justification flag, returned as `1` if the condition is justified or `0` if it is not.

Data Types: `double`

`filterRationale` — Coverage filter rationale
`character array`

Coverage filter rationale, returned as a character array.

Data Types: `char`

`text` — Description of condition
`character array`

Description of the condition, returned as a character array.

Data Types: `char`

`trueCnts` — Number of time steps condition is `true`
`scalar`

Number of time steps the condition is `true`, returned as a scalar.

Data Types: `double`

`falseCnts` — Number of time steps condition is `false`
`scalar`

Number of time steps the condition is `false`, returned as a scalar.

Data Types: `double`

`trueOutcomeFilter` — Coverage filter information for `true` condition outcome
`structure array`

Coverage filter information for the `true` condition outcome, returned as a structure array with the following fields:

`isFiltered`	Filter flag for the condition outcome, returned as `1` if the outcome is filtered or `0` if it is not.
`isJustified`	Justification flag for the condition outcome, returned as `1` if the outcome is justified or `0` if it is not.
`filterRationale`	The filter rationale, returned as a character array.

Data Types: `struct`

`falseOutcomeFilter` — Coverage filter information for `false` condition outcome
`structure array`

Coverage filter information for the `false` condition outcome, returned as a structure array with the following fields:

`isFiltered`	Filter flag for the condition outcome, returned as `1` if the outcome is filtered or `0` if it is not.
`isJustified`	Justification flag for the condition outcome, returned as `1` if the outcome is justified or `0` if it is not.
`filterRationale`	The filter rationale, returned as a character array.

Data Types: `struct`

`trueExecutedIn` — Aggregated coverage test case traceability information
`structure array` | `[]`

Aggregated coverage test case traceability information, returned as a structure array. If your coverage data does not contain aggregated results from multiple simulations, `trueExecutedIn` is an empty array. If your coverage data contains aggregated results, `trueExecutedIn` indicates which test runs executed the `true` condition outcome and has the following fields:

`uniqueId`	Unique identifier for the `cvdata` object created by the test case that executed this condition outcome, returned as a character array.
`analyzedModel`	The name of the model analyzed, returned as a character array.
`description`	The test case description, returned as a character array.
`date`	The date and time of the simulation that created the `cvdata` object, returned as a character array.
`traceLabel`	The short name of the test case, returned as a character array.
`testRunInfo`	Test details, returned as a structure array.
`testRunInfo.runid`	The identifier of the run that generated the coverage result, returned as a scalar double.
`testRunInfo.runName`	The name of the test case that generated the coverage result, returned as a character array.
`testRunInfo.testId`	Identifying data for the test case that generated the coverage result, returned as a structure array.
`testRunInfo.testId.uuid`	The unique identifier for the test case that generated the coverage result, returned as a character array.
`testRunInfo.testId.contextType`	The test context in which the coverage result was obtained, returned as one of these values: `'RE'` for the Coverage Results Explorer `'ST'` for the Test Manager in Simulink Test™

Data Types: `struct`

`falseExecutedIn` — Aggregated coverage test case traceability information
`structure array | []`

Aggregated coverage test case traceability information, returned as a structure array. If your coverage data does not contain aggregated results from multiple simulations, `falseExecutedIn` is an empty array. If your coverage data contains aggregated results, `falseExecutedIn` indicates which test runs executed the `false` condition outcome and has the following fields:

`uniqueId`	Unique identifier for the `cvdata` object created by the test case that executed this condition outcome, returned as a character array.
`analyzedModel`	The name of the model analyzed, returned as a character array.
`description`	The test case description, returned as a character array.
`date`	The date and time of the simulation that created the `cvdata` object, returned as a character array.
`traceLabel`	The short name of the test case, returned as a character array.
`testRunInfo`	Test details, returned as a structure array.
`testRunInfo.runid`	The identifier of the run that generated the coverage result, returned as a scalar double.
`testRunInfo.runName`	The name of the test case that generated the coverage result, returned as a character array.
`testRunInfo.testId`	Identifying data for the test case that generated the coverage result, returned as a structure array.
`testRunInfo.testId.uuid`	The unique identifier for the test case that generated the coverage result, returned as a character array.
`testRunInfo.testId.contextType`	The test context in which the coverage result was obtained, returned as one of these values: `'RE'` for the Coverage Results Explorer `'ST'` for the Test Manager in Simulink Test

Data Types: `struct`

Data Types: `struct`

Data Types: `struct`

Alternatives

You can also collect condition coverage for a model in the Simulink Editor. Set the model configuration parameters and run the model:

1. Open the model for which you want to collect condition coverage.
2. In the Simulink Editor, in the **Modeling** tab, select **Model Settings**.
3. On the **Coverage** pane of the Configuration Parameters dialog box, select **Enable coverage analysis**.
4. Under **Coverage metrics**, select `Condition Decision` as the structural coverage level.
5. Click **OK** to close the Configuration Parameters dialog box and save your changes.
6. Simulate the model by clicking **Run**. Review the results.

Version History

Introduced in R2006b

See Also

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

Topics

"Condition Coverage (CC)"

cv.cvdatagroup class

Package: `cv`

Collection of `cvdata` objects

Description

A `cv.cvdatagroup` object contains one or more `cvdata` objects. Each `cvdata` object contains coverage results for a model in the model hierarchy.

Simulink Coverage™ creates a `cvdatagroup` object when you simulate a model containing model references when coverage is enabled, or if your model uses external code files and you selected external files for coverage analysis. Use the `get` and `getAll` class methods to retrieve coverage data for specific models in your model reference hierarchy. Use the `allNames` method to retrieve the `cvdata` object names to use with the `get` class method.

The `cv.cvdatagroup` class is a `handle` class.

Creation

Description

`cvdg = cv.cvdatagroup(cvdo1,cvdo2,...)` creates a `cvdatagroup` object that contains the specified `cvdata` objects.

Input Arguments

cvdo — Coverage data
`cvdata` object

Coverage data, specified as one or more `cvdata` objects.

Data Types: `cvdata`

Properties

name — Object name
`''` (default) | character array

Object name, specified as a character array.

Example: `cvdg.name = 'myDataGroup';`

Attributes:

`GetAccess`	`public`
`SetAccess`	`public`

Data Types: `char`

Methods

Public Methods

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

Examples

Get Coverage Data from `cvdatagroup` Objects

This example shows how to retrieve a specified coverage data object, all coverage data objects, or all coverage data object names from a `cv.cvdatagroup` object.

Load the example model.

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

Use the `Simulink.SimulationInput` class to set up the simulation scenario with coverage enabled.

```
simIn = Simulink.SimulationInput(modelName);
simIn = setModelParameter(simIn,'CovEnable','on');
simIn = setModelParameter(simIn,'CovMetricStructuralLevel','MCDC');
simIn = setModelParameter(simIn,'CovSaveSingleToWorkspaceVar','on');
simIn = setModelParameter(simIn,'CovSaveName','covData');
simIn = setModelParameter(simIn,'CovSaveOutputData','off');
```

Simulate the model by passing the `SimulationInput` object to the `sim` function, and then retrieve the coverage data from the `SimulationOutput` object.

```
simOut = sim(simIn);
covData = simOut.covData
```

```
covData = ... cv.cvdatagroup

  slcoverageExternalFile (simulation mode: Normal)
  slvnvdemo_eml_model_coverage_demo (simulation mode: Normal)
```

The example model contains a MATLAB® Function block that calls an external MATLAB function. Because Simulink Coverage returns the coverage data for the MATLAB function in a separate `cvdata` object, `sim` returns a `cv.cvdatagroup` object that contains the coverage results for both the model and the MATLAB function.

To determine the name associated with each `cvdata` object inside the `cv.cvdatagroup`, use the class method `allNames`.

```
covDataNames = allNames(covData)
covDataNames = 

  2x1 cell array

    {'slcoverageExternalFile'           }
    {'slvnvdemo_eml_model_coverage_demo'}
```

The `cv.cvdatagroup` contains the coverage results for the model as well as the MATLAB function.

To retrieve the coverage data for the model, use the `get` class method.

```
modelCovData = get(covData,modelName)
```

```
modelCovData = ... cvdata
            version: (R2023a)
                 id: 1314
               type: TEST_DATA
               test: cvtest object
             rootID: 1316
           checksum: [1x1 struct]
          modelinfo: [1x1 struct]
          startTime: 03-Mar-2023 13:40:11
           stopTime: 03-Mar-2023 13:40:11
  intervalStartTime: 0
   intervalStopTime: 0
simulationStartTime: 0
 simulationStopTime: 1
             filter: 
            simMode: Normal
```

To retrieve coverage data for both the model and the external file, use the `getAll` class method.

```
covDataArray = getAll(covData)
```

```
covDataArray = 

  2x1 cell array

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

Create a cv.cvdatagroup Object

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

Record coverage for `slvnvdemo_cv_small_controller`.

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

Record coverage for `slvnvdemo_powerwindow_controller`.

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

Create a `cv.cvdatagroup` object that contains two `cvdata` objects.

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

Version History

Introduced in R2007b

See Also

`cvsim` | `cvdata` | `sim`

Topics

“Types of Coverage Data”

allSimulationModes

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

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

Syntax

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

Description

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

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

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

Input Arguments

cvdg — Class instance
object

Instance of class `cv.cvdatagroup`.

`modelName` — Name of the model
character vector or string

Model with which simulation modes are associated.

Output Arguments

`simModes` — Simulation modes
cell array of character vectors or strings

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

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

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

Examples

Get All Simulation Modes from cvdatagroup

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

Record coverage for slvnvdemo_cv_small_controller.

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

Record coverage for slvnvdemo_powerwindow_controller.

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

Record coverage for slvnvdemo_counter.

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

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

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

Get the simulation modes by using `allSimulationModes`.

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

```
simModes =

  1x1 cell array

    {'Normal'}
```

cvexit

Exit coverage environment

Syntax

```
cvexit
```

Description

`cvexit` exits the model coverage environment. Using `cvexit` closes the Coverage Details pane and removes coloring from open models.

Examples

Exit the Coverage Environment

- Call `cvexit` to close the coverage environment

  ```
  cvexit;
  ```

Version History

Introduced in R2006b

cvhtml

Create HTML coverage report from model coverage objects

Syntax

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

Description

`cvhtml(file,cvdo)` creates an HTML report of the coverage results in the `cvdata` or `cv.cvdatagroup` object `cvdo` with the file name `file`. The model must be open when you call `cvhtml`.

`cvhtml(file,cvdo1,...,cvdoN)` creates a combined report from 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.

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

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

Examples

Create a Coverage Report

Set your current folder to a writeable directory.

Open your model and collect coverage by using `cvsim`.

```
modelName = 'slvnvdemo_cv_small_controller';
open_system(modelName);
cvd = cvsim(modelName);
```

Create a coverage report for the model and save the report as `ratelim_coverage.html`.

```
outfile = 'ratelim_coverage.html';
cvhtml(outfile,cvd);
```

Input Arguments

cvdo — Coverage data
`cvdata` object | `cv.cvdatagroup` object

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

Data Types: `cvdata` | `cv.cvdatagroup`

`file` — File name of the generated report
`character array` | `string array`

File name of the generated report, specified as a character array or string array. Specify the absolute path or relative path and the file name.

`options` — Report options
`character vector` | `string`

Report options, specified as a character vector or string that consists of an option from the table, an equal sign, and `0` or `1`.

- To enable an option, use `1`.

Example: `"-hTR=1"`

- To disable an option, use `0`.

Example: `"-sRT=0"`

- To specify multiple report options, list the options in a single character vector or string separated by commas or spaces.

Example: `"-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`
`-agT`	Show aggregated tests information in coverage report	`on`
`-bRG`	Produce bar graphs in the model summary	`on`
`-bTC`	Use two-color bar graphs	`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	`off`
`-scm`	Include cyclomatic complexity numbers in summary	`on`
`-bcm`	Include cyclomatic complexity numbers in block details	`on`
`-xEv`	Filter Stateflow events from report	`off`
`-req`	Include linked requirements in coverage report	`on`

Data Types: `char` | `string`

`simMode` — Simulation mode
`"Normal"` | `"SIL"` | `"PIL"` | `"ModelRefSIL"` | `"ModelRefPIL"` | `"ModelRefTopSIL"` | `"ModelRefTopPIL"`

Simulation mode, specified as one of these values:

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 SIL or PIL simulation mode
`"ModelRefTopSIL" (or "ModelRefTopPIL")`	Model reference in SIL or PIL simulation mode with code interface set to top model

Alternatives

Use the Simulink Editor to generate a coverage report when you simulate the model:

1. Open the model for which you want a model coverage report.
2. In the Simulink Editor, in the **Modeling** tab, select **Model Settings**.
3. In the **Coverage** pane of the Configuration Parameters dialog box, select **Enable coverage analysis**.
4. Click **OK**.
5. Click **Run (Coverage)** to simulate the model. When the simulation completes, the coverage report opens in the **Coverage Details** pane.

Version History

Introduced before R2006a

R2023a: Changed default value of Exclude fully covered model object details from report to `off`

Behavior changed in R2023a

Changed the default value of the **Exclude fully covered model object details from report** setting to `off`. Previously, if you wanted to see the coverage tables in the report for fully covered model objects, you included `'-nFD=0'` in the `options` argument when you called `cvhtml`. Now you can omit this setting from `options`.

See Also

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

Topics

"Automating Model Coverage Tasks"

cvload

Load coverage tests and stored results into memory

Syntax

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

Description

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

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

Note When using the `cvload` command:

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

Examples

Load Coverage Data and Preserve Cumulative Data

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

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

Input Arguments

`fileName` — Name of coverage data file
`character array | string array`

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

Example: `'myCoverageData'`

Data Types: `char` | `string`

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

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

Data Types: `double`

Output Arguments

`covSettings` — Coverage settings
`cell array`

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

Data Types: `cell`

`covData` — Coverage data
`cell array`

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

Alternatives

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

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

Version History

Introduced before R2006a

R2020b: Load and view coverage data from previous releases

You can load and view coverage data from previous releases as far back as R2017b.

After you import coverage data from a previous release, you can extract information using the API. You can aggregate coverage data that have matching release versions.

See Also

`cvdata` | `cvsave` | `cvtest`

Topics
"Analyze Coverage Data Using A Script"

cvmodelview

Display model coverage results with model highlighting

Syntax

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

Description

`cvmodelview(cvdo)` highlights the model with coverage results from the `cvdata` object `cvdo`.

`cvmodelview(cvdo,simMode)` highlights the model with coverage results for the specified simulation mode.

Examples

Highlight Coverage Results in a Model

Use `cvsim` to simulate a model and return a `cvdata` object.

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

Use `cvmodelview` with the `cvdata` object to display the coverage results visually by highlighting the model.

```
cvmodelview(covData)
```

Input Arguments

cvdo — Coverage data
`cvdata` object | `cv.cvdatagroup` object

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

`simMode` — Simulation mode
`"Normal"` | `"SIL"` | `"PIL"` | `"ModelRefSIL"` | `"ModelRefPIL"` | `"ModelRefTopSIL"` | `"ModelRefTopPIL"`

Simulation mode, specified as one of these values:

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 SIL or PIL simulation mode

Object Specification	Description
`"ModelRefTopSIL"` (or `"ModelRefTopPIL"`)	Model reference in SIL or PIL simulation mode with code interface set to top model

Alternatives

Use the coverage settings to enable coverage highlighting:

1. Open the model.
2. On the **Modeling** tab, select **Model Settings**.
3. In the **Coverage** pane of the Configuration Parameters dialog box, select **Enable coverage analysis**.
4. Click **OK**.
5. Click **Run (Coverage)** to simulate the model. When the simulation completes, the model displays with coverage highlighting enabled.

Version History

Introduced in R2006b

See Also

`cvhtml` | `cvsim`

Topics

"View Coverage Results in Simulink Canvas"

cvresults

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

Syntax

```
[covData,cumulativeCovData] = cvresults(modelName)
cvresults(modelName,'clear')
cvresults(modelName,'load',fileName)
```

Description

`[covData,cumulativeCovData] = cvresults(modelName)` returns the active single-run coverage data `covData` and cumulative coverage data `cumulativeData`.

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

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

Examples

Retrieve Current Run Coverage Data

1. Simulate your model and collect coverage.

   ```
   cvsim(modelName);
   ```
2. Get the current coverage data.

   ```
   [covData,cumulativeCovData] = cvresults(modelName);
   ```

Input Arguments

`modelName` — Model name
character array | string array

Model name, specified as a character array or string array.

Data Types: `char` | `string`

`fileName` — Coverage data file name
character array | string array

Coverage data file name, specified as a character array or string array.

Data Types: `char` | `string`

Output Arguments

`covData` — Coverage data
`cvdata` object

Coverage data, returned as a `cvdata` object.

Data Types: `cvdata`

`cumulativeCovData` — Cumulative coverage data
`cvdata` object

Cumulative coverage data, returned as a `cvdata` object.

Data Types: `cvdata`

Version History

Introduced in R2016a

cvsave

Save coverage settings and results to file

Syntax

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

Description

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

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

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

Examples

Save Coverage Results

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

Start by loading the model into memory.

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

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

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

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

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

Save Multiple Coverage Data Objects to a File

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

Load the Model

Load the model into memory.

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

Set Model Parameters for Coverage

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

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

Simulate the Model to Collect Coverage Data

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

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

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

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

Save the Coverage Data to a File

Save the results in a cell array.

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

```
cov_results =

  1x2 cell array

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

Save the results to a file.

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

Input Arguments

`fileName` — Name of coverage data file
character array | string array

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

Example: `'myCoverageDataFile'`

Data Types: `char` | `string`

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

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

Example: `'mySimulinkModel'`

Data Types: `char` | `string`

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

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

Data Types: `cvtest` | `cell`

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

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

Data Types: `cvdata` | `cell`

Alternatives

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

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

Version History

Introduced before R2006a

See Also

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

cvsim

Simulate and return model coverage results for test objects

Syntax

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

Description

`cvdo = cvsim(modelName)` simulates the model and returns the coverage results in the `cvdata` object, `cvdo`. When called using `modelName`, `cvsim` uses the coverage metric settings specified in the model configuration parameters.

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

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

Note The coverage metric settings specified in `testObj` override coverage metric settings specified in the model configuration parameters.

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

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

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

Examples

Record Coverage Data Using `cvsim`

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

Load the `slvnvdemo_cv_small_controller` example model.

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

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

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

Create a structure that defines the following model parameters:

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

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

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

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

Input Arguments

`modelName` — Name of Simulink model

character array | string array

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

Data Types: `char` | `string`

`testObj` — Coverage test settings

`cvtest object`

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

The coverage metric settings specified in `testObj` override the coverage metric settings specified in the model configuration parameters. For example, if the **Structural coverage level** parameter in your model is set to `Modified Condition Decision Coverage (MCDC)`, but your `testObj.settings.mcdc` property is `0`, then `covData = cvsim(testObj)` will *not* record MCDC coverage.

Data Types: `cvtest`

`paramStruct` — Model parameters

structure

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

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

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

Data Types: `struct`

Name-Value Pair Arguments

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

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

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

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

`ModelParameter` — Model parameter and value
character array | string array

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

Data Types: `char` | `string`

Output Arguments

cvdo — Coverage data object
`cvdata`

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

See `cvdata` for the object structure.

`simOut` — Simulation data object
`Simulink.SimulationOutput`

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

Version History

Introduced before R2006a

See Also

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

cvtest

Create model coverage test specification object

Description

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

Creation

Syntax

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

Description

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

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

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

Note Coverage metric settings specified in the `cvtest` object override coverage metric settings set in the model configuration parameters.

Input Arguments

`root` — Name or handle of model or path to subsystem
`character array | string array`

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

Properties

`id` — Internal Model ID
`scalar`

This property is read-only.

Internal model ID, returned as a scalar.

`modelcov` — Internal Coverage Configuration ID
`scalar`

This property is read-only.

Internal coverage configuration ID, returned as a scalar.

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

This property is read-only.

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

Data Types: `char` | `string`

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

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

Data Types: `char` | `string`

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

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

The setup command is executed before each simulation.

Data Types: `char` | `string`

`settings` — Coverage settings
`structure`

Types of coverage to collect, specified as a structure.

`settings` includes the following fields:

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

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

`options` — Advanced coverage options
`structure`

Advanced coverage options, specified as a structure.

`options` includes the following fields:

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

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

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

`filter` — Coverage filter
`structure`

Coverage filter, specified as a structure.

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

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

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

Model reference settings, specified as a structure.

`modelRefSettings` includes the following fields:

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

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

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

`emlSettings` has one field, `emlSettings.enableExternal`. Set `emlSettings.enableExternal` to `1` if you want to collect coverage for external program files called by MATLAB functions, and `0` if you do not want to collect coverage for external program files called by MATLAB functions.

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

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

`sfcnSettings` has one field, `sfcnSettings.enableSfcn`. Set `sfcnSettings.enableSfcn` to `1` if you want to collect coverage for S-Functions in your model, and `0` if you do not want to collect coverage for S-Functions in your model.

For more information, see S-Function.

Examples

Analyze Coverage Using a `cvtest` Object

This example shows how to run coverage analysis using a `cvtest` object.

Open the `slvnvdemo_ratelim_harness` model and define the test object using `cvtest`. When you create the `cvtest` object, specify the Adjustable Rate Limiter block as the model object to analyze.

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

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

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

To collect decision coverage and saturation on integer overflow coverage, enable the `decision` and `overflowsaturation` properties in `testObj` object by setting the properties to `true` or `1`.

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

Finally, perform the coverage analysis by calling `cvsim` with `testObj`.

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

Version History

Introduced before R2006a

See Also

`cvsim` | `cv.cvdatagroup`

Topics

"Automating Model Coverage Tasks"

decisioninfo

Retrieve decision coverage information from `cvdata` object

Syntax

```
covInfo = decisioninfo(cvdo, modelObject)
covInfo = decisioninfo(cvdo, modelObject, simMode)
covInfo = decisioninfo(cvdo, modelObject, ignoreDescendants)
[covInfo, description] = decisioninfo(cvdo, modelObject)
```

Description

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

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

`covInfo = decisioninfo(cvdo, modelObject, ignoreDescendants)` returns decision coverage results for `modelObject`, depending on the value of `ignoreDescendants`.

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

Examples

View Decision Coverage Data

This example shows how to view decision coverage data for a block in your model.

Load the model.

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

Configure the coverage settings for the model by using a `Simulink.SimulationInput` object.

```
simIn = Simulink.SimulationInput(modelName);
simIn = setModelParameter(simIn,'CovEnable','on');
simIn = setModelParameter(simIn,'CovMetricStructuralLevel','Decision');
simIn = setModelParameter(simIn,'CovSaveSingleToWorkspaceVar','on');
simIn = setModelParameter(simIn,'CovSaveName','covData');
```

Simulate the model by passing `simIn` as the input to `sim`.

```
simOut = sim(simIn);
```

Extract the coverage data as a `cvdata` object from the `SimulationOutput` object, `simOut`.

```
covData = simOut.covData;
```

View the decision coverage results for the Saturation block by calling `decisioninfo` with the block path.

```
blockPath = [modelName,'/Saturation'];
decisionCov = decisioninfo(covData,blockPath)
```

```
decisionCov =

     3     4
```

`decisioninfo` returns an array with two scalars. The first value is the number of satisfied decision outcomes, and the second value is the number of total decision outcomes. Use these to determine the percentage of satisfied decision outcomes for the Saturation block.

```
percentDecisionCov = 100 * decisionCov(1) / decisionCov(2)
```

```
percentDecisionCov =

    75
```

Input Arguments

cvdo — Coverage data
`cvdata` object

Coverage data, specified as a `cvdata` object.

Data Types: `cvdata`

`modelObject` — Model object
`character array` | `string array` | `Simulink handle` | `Stateflow ID` | `cell array`

Model object, specified as a character array, string array, Simulink handle, Stateflow ID, or cell array.

To specify a model object, such as a block or a Stateflow chart, use one of these formats:

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

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

To specify an S-Function block or its contents, use one of these formats:

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

To specify a code coverage result, such as coverage data collected during software-in-the-loop (SIL) or processor-in-the-loop (PIL) analysis, use one of these formats:

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

Data Types: `char` | `string` | `cell` | `Stateflow.State` | `Stateflow.Transition`

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

Simulation mode during coverage analysis, specified as one of these options:

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 SIL or PIL simulation mode.
`"ModelRefTopSIL"` or `"ModelRefTopPIL"`	Model reference in SIL or PIL simulation mode with the code interface set to top model.

Data Types: `char` | `string`

`ignoreDescendants` — Whether to ignore descendants in coverage results
`false` or `0` (default) | `true` or `1`

Whether to ignore descendants in coverage results, specified as a numeric or logical `1 (true)` or `0 (false)`, where:

- `0 (false)` includes coverage results of descendant objects.
- `1 (true)` ignores coverage results of descendant objects.

Data Types: `single` | `double` | `logical`

Output Arguments

`covInfo` — Coverage information
scalar

Coverage information, returned as a two-element array of the form `[covered_outcomes,total_outcomes]` if `cvdo` contains decision coverage data, or an empty array if it does not.

`covered_outcomes`	Number of decision outcomes satisfied for `modelObject`
`total_outcomes`	Total number of decision outcomes for `modelObject`

Data Types: `double`

`description` — Decision coverage description
`structure`

Decision coverage description, returned as a structure array with the following fields:

`isFiltered` — Block exclusion flag
`0` | `1`

Block exclusion flag, returned as `1` if the block is excluded and `0` if it is not.

Data Types: `double`

`filterRationale` — Block coverage filter rationale
`character array`

Block coverage filter rationale, returned as a character array.

Data Types: `char`

`justifiedCoverage` — Number of justified coverage objective outcomes
`scalar`

Number of justified coverage objective outcomes, returned as a scalar double.

Data Types: `double`

`isJustified` — Block justification flag
`0` | `1`

Block justification flag, returned as `1` if the block is justified or `0` if it is not.

Data Types: `double`

`decision` — Information for individual decisions
`structure array`

Information for individual decisions, returned as a structure with the following fields:

`isFiltered` — Decision exclusion flag
`0` | `1`

Decision exclusion flag, returned as `1` if the decision is excluded or `0` if it is not.

Data Types: `double`

`isJustified` — Decision justification flag
`0` | `1`

Decision justification flag, returned as `1` if the decision is justified or `0` if it is not.

Data Types: `double`

`filterRationale` — Coverage filter rationale
`character array`

Coverage filter rationale, returned as a character array.

Data Types: `char`

`text` — Description of decision
`character array`

Description of the decision, returned as a character array.

Data Types: `char`

`outcome` — Information for individual decision outcomes
`structure`

Information for individual decision outcomes, returned as a structure array with the following fields:

`text`	Outcome of the decision, returned as `'true'` or `'false'`.
`executionCount`	The number of time steps the decision had this outcome, returned as a scalar double.
`isFiltered`	Decision outcome exclusion flag, returned as `1` if the outcome is excluded or `0` if it is not.
`isJustified`	Decision outcome justification flag, returned as `1` if the outcome is justified or `0` if it is not.
`filterRationale`	Coverage filter rationale, returned as a character array.

`executedIn`	Aggregated coverage test case traceability information, returned as a structure array. If your coverage data does not contain aggregated results from multiple simulations, `executedIn` is an empty array. If your coverage data contains aggregated results, `executedIn` indicates which test runs executed the decision outcome.
`executedIn.uniqueId`	Unique identifier for the `cvdata` object created by the test case that executed this condition outcome, returned as a character array.
`executedIn.analyzedModel`	The name of the model analyzed, returned as a character array.
`executedIn.description`	The test case description, returned as a character array.
`executedIn.date`	The date and time of the simulation that created the `cvdata` object, returned as a character array.
`executedIn.traceLabel`	The short name of the test case, returned as a character array.
`executedIn.testRunInfo`	Test details, returned as a structure array.
`executedIn.testRunInfo.runid`	The identifier of the run that generated the coverage result, returned as a scalar double.
`executedIn.testRunInfo.runName`	The name of the test case that generated the coverage result, returned as a character array.
`executedIn.testRunInfo.testId`	Identifying data for the test case that generated the coverage result, returned as a structure array.
`executedIn.testRunInfo.testId.uuid`	The unique identifier for the test case that generated the coverage result, returned as a character array.
`executedIn.testRunInfo.testId.contextType`	The test context in which the coverage result was obtained, returned as one of these values: `'RE'` for the Coverage Results Explorer `'ST'` for the Test Manager in Simulink Test

Data Types: `struct`

Data Types: `struct`

Data Types: `struct`

Alternatives

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

1. Open the model.
2. In the Model Editor, select **Model Settings** on the **Modeling** tab.

3. On the **Coverage** pane of the Configuration Parameters dialog box, select **Enable coverage analysis**.
4. Under **Coverage metrics**, select **Decision** as the structural coverage level.
5. Click **OK** to close the Configuration Parameters dialog box and save your changes.
6. Simulate the model by clicking the **Run** button and review the results.

Version History

Introduced in R2006b

See Also

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

Topics

"Decision Coverage (DC)"

executioninfo

Retrieve execution coverage information from `cvdata` object

Syntax

```
covInfo = executioninfo(cvdo,modelObject)
covInfo = executioninfo(cvdo,modelObject,simMode)
covInfo = executioninfo(cvdo,modelObject,ignoreDescendants)
[covInfo,description] = executioninfo(cvdo,modelObject)
```

Description

`covInfo = executioninfo(cvdo,modelObject)` returns execution coverage results from the `cvdata` object `cvdo` for the model component specified by `modelObject`. If `cvdo` contains code coverage data, `executioninfo` returns the sum of statement coverage, function coverage, and function call coverage metrics.

`covInfo = executioninfo(cvdo,modelObject,simMode)` returns execution coverage results for the simulation mode `simMode`.

`covInfo = executioninfo(cvdo,modelObject,ignoreDescendants)` returns execution coverage results for `modelObject` with or without its descendants, depending on the value of `ignoreDescendants`.

`[covInfo,description] = executioninfo(cvdo,modelObject)` returns description structure arrays for each execution point associated with `modelObject`. If `cvdo` contains coverage data for the code coverage metrics statement coverage, function coverage, or function call coverage, `description` contains structure arrays for each relevant metric.

Examples

Retrieve Execution Coverage Results for a Block

This example shows how to extract execution coverage results for a block from a `cvdata` object.

Load the `slvnvdemo_cv_small_controller` model.

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

Configure the coverage settings for the model by using a `Simulink.SimulationInput` object.

```
simIn = Simulink.SimulationInput(modelName);
simIn = setModelParameter(simIn,"CovEnable","on");
simIn = setModelParameter(simIn,"CovMetricStructuralLevel",...
                                "BlockExecution");
simIn = setModelParameter(simIn,"CovSaveSingleToWorkspaceVar","on");
simIn = setModelParameter(simIn,"CovSaveName","covData");
```

Simulate the model by passing `simIn` as the input to `sim`.

```
simOut = sim(simIn);
covData = simOut.covData;
```

Retrieve the handle for the Saturation block by using `get_param`.

```
blockHandle = get_param(modelName+"/Saturation","Handle");
```

Extract the execution coverage information by using `executioninfo`.

```
executionCov = executioninfo(covData,blockHandle)
```

```
executionCov =

     1     1
```

Compute the percentage of execution outcomes satisfied.

```
percentCoverage = 100*executionCov(1) / executionCov(2)
```

```
percentCoverage =

   100
```

Retrieve Statement and Function Coverage

This example shows how to extract statement, function, and function call coverage for a model simulated in software-in-the-loop (SIL) mode.

Load the model.

```
modelName = "SILTopModel";
load_system(modelName);
```

Configure the coverage settings for the model by using a `Simulink.SimulationInput` object.

```
simIn = Simulink.SimulationInput(modelName);
simIn = setModelParameter(simIn,"CovEnable","on");
simIn = setModelParameter(simIn,"CovMetricStructuralLevel","MCDC");
simIn = setModelParameter(simIn,"CovSaveSingleToWorkspaceVar","on");
simIn = setModelParameter(simIn,"CovSaveName","covData");
simIn = setModelParameter(simIn,"SimulationMode","software-in-the-loop (sil)");
```

Simulate the model by passing `simIn` as the input to `sim`.

```
simOut = sim(simIn);
covDataSIL = simOut.covData;
```

```
### Starting build procedure for: SILTopModel
### Successful completion of build procedure for: SILTopModel

Build Summary

Top model targets built:
```

```
Model        Action                        Rebuild Reason
=====================================================================================
SILTopModel  Code generated and compiled.  Code generation information file does not exist.

1 of 1 models built (0 models already up to date)
Build duration: 0h 0m 17.111s
### Preparing to start SIL simulation ...
Building with 'Microsoft Visual C++ 2019 (C)'.
MEX completed successfully.
### Updating code generation report with SIL files ...
### Starting SIL simulation for component: SILTopModel
### Stopping SIL simulation for component: SILTopModel
### Completed code coverage analysis
```

Pass `covDataSIL` and the file name `SILTopModel.c` as inputs to `executioninfo`.

```
[covMetricsSIL,covMetricsSILdesc] = executioninfo(covDataSIL,"SILTopModel.c")
```

```
covMetricsSIL =

    21    24


covMetricsSILdesc = 

  struct with fields:

             isFiltered: 0
      justifiedCoverage: 0
            isJustified: 0
        filterRationale: ''
               function: [1x6 struct]
           functionCall: [1x4 struct]
    executableStatement: [1x18 struct]
               decision: [1x24 struct]
```

When you analyze coverage on a model in SIL mode, `executioninfo` returns the data for these code coverage metrics:

- Statement coverage
- Function coverage
- Function call coverage

`covMetricsSIL`, the first output argument, contains two values: the total satisfied code coverage objectives and the total number of code coverage objectives. Determine the percentage of satisfied statement, function, and function call coverage by dividing the numbers.

```
percentCodeCov = 100 * covMetricsSIL(1) / covMetricsSIL(2)
```

```
percentCodeCov =

   87.5000
```

To retrieve the details about the individual metrics, use the second output argument, `covMetricsSILdesc`. The analyzed code is missing coverage for three statements. To determine which statements are missing coverage, you can search for which fields of the statement coverage description object have an execution count of zero.

```
execCounts = [covMetricsSILdesc.executableStatement.executionCount];
missingCovIdxs = execCounts == 0;
missingCov = covMetricsSILdesc.executableStatement(missingCovIdxs);
```

The `missingCov` array contains the location of the three missing statement objectives in the structure array. Look at the structure for the first missing statement.

```
disp(missingCov(1))
```

```
           isFiltered: 0
    justifiedCoverage: 0
          isJustified: 0
      filterRationale: ''
                 text: 'Statement executed'
       executionCount: 0
             fileName: 'SILTopModel.c'
         functionName: 'CounterTypeB'
       sourceLocation: [1x1 struct]
                 kind: 'if'
        modelElements: {1x2 cell}
```

You can see that it is an if statement located in the function `counterTypeB`, which is in the file `SILTopModel.c`. Look at its `sourceLocation` field, to see the exact line numbers that represent the missing statement objective.

```
disp(missingCov(1).sourceLocation)
```

```
    startLine: 111
     startCol: 5
      endLine: 113
       endCol: 5
```

In `SILTopModel.c`, lines 114 through 116 are missing statement coverage.

Input Arguments

cvdo — Coverage data
`cvdata` object

Coverage data, specified as a `cvdata` object.

Data Types: `cvdata`

`modelObject` — Model object
`character array` | `string array` | `Simulink handle` | `Stateflow ID` | `cell array`

Model object, specified as a character array, string array, Simulink handle, Stateflow ID, or cell array.

To specify a model object, such as a block or a Stateflow chart, use one of these formats:

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

To specify an S-Function block or its contents, use one of these formats:

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

To specify a code coverage result, such as coverage data collected during software-in-the-loop (SIL) or processor-in-the-loop (PIL) analysis, use one of these formats:

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

Data Types: `char` | `string` | `cell` | `Stateflow.State` | `Stateflow.Transition`

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

Simulation mode during coverage analysis, specified as one of these options:

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 SIL or PIL simulation mode.
`"ModelRefTopSIL"` or `"ModelRefTopPIL"`	Model reference in SIL or PIL simulation mode with the code interface set to top model.

Data Types: `char` | `string`

`ignoreDescendants` — Whether to ignore descendants in coverage results
`false` or `0` (default) | `true` or `1`

Whether to ignore descendants in coverage results, specified as a numeric or logical `1 (true)` or `0 (false)`, where:

- `0 (false)` includes coverage results of descendant objects.
- `1 (true)` ignores coverage results of descendant objects.

Data Types: `single` | `double` | `logical`

Output Arguments

`covInfo` — Coverage information
scalar

Coverage information, returned as a two-element scalar array of the form `[covered_outcomes,total_outcomes]` if `cvdo` contains execution coverage data, or an empty array if it does not. The elements in the array are:

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

Data Types: `double`

`description` — Execution coverage description
structure

Execution coverage description, returned as a structure with these fields:

`isFiltered` — Block exclusion flag
`0` | `1`

Block exclusion flag, returned as `1` if the block is excluded and `0` if it is not.

Data Types: `double`

`filterRationale` — Block coverage filter rationale
`character array`

Block coverage filter rationale, returned as a character array.

Data Types: `char`

`justifiedCoverage` — Number of justified coverage objective outcomes
`scalar`

Number of justified coverage objective outcomes, returned as a scalar double.

Data Types: `double`

`isJustified` — Block justification flag
`0` | `1`

Block justification flag, returned as `1` if the block is justified or `0` if it is not.

Data Types: `double`

`decision` — Information for individual decisions
`structure array`

Information for individual decisions, returned as a structure array with these fields:

`text` — Block execution text
`'Block executed'`

Block execution text, returned as the character array `'Block executed'`. The `text` field does not change even if the block has 0% execution coverage.

Data Types: `char`

`executionCount` — Number of time steps the model object executed
scalar double

Number of time steps the model object executed, returned as a scalar double.

Data Types: `double`

Data Types: `struct`

`function` — Function coverage information
`structure array`

Function coverage information, returned as N_f structure arrays, where N_f is the number of functions analyzed. `function` appears when `cvdo` contains code coverage data for the function coverage metric. Each structure contains these fields:

`isFiltered` — Function exclusion flag
`0` | `1`

Function exclusion flag, returned as `1` if the function is excluded and `0` if it is not.

Data Types: `double`

`justifiedCoverage` — Number of justified coverage outcomes
`scalar`

Number of justified coverage outcomes, returned as a scalar double.

Data Types: `double`

`isJustified` — Function justification flag
`0` | `1`

Function justification flag, returned as `1` if the function is justified and `0` if it is not.

Data Types: `double`

`filterRationale` — Function coverage filter rationale
`[]` (default) | `character array`

Function coverage filter rationale, returned as a character array. If the function is not filtered or the filter rationale is not set, `filterRationale` returns an empty array.

Data Types: `char`

`text` — Function coverage text
`'Function entry'`

Function coverage text, returned as `'Function entry'`.

Data Types: `char`

`executionCount` — Execution count of function
`integer`

Execution count of the function, returned as a 64-bit integer.

Data Types: `int64`

`fileName` — Name of file containing function
`character array`

Name of the file containing the function, returned as a character array.

Data Types: `char`

`functionName` — Name of analyzed function
`character array`

Name of the analyzed function, returned as a character array.

Data Types: `char`

`sourceLocation` — Location of analyzed function in source code
`structure array`

Location of the analyzed function in the source code, returned as a structure array with these fields:

Field Name	Description	Datatype
`startLine`	Line of the source code where the function begins	`int64`
`startCol`	Column of the source code where the function begins	`int64`
`endLine`	Line of the source code where the function ends	`int64`

Field Name	Description	Datatype
`endCol`	Column of the source code where the function ends	`int64`

Data Types: `struct`

`modelElements` — Model elements that correspond to function
`character array`

Model elements that correspond to the function, returned as a character array. The `modelElements` field appears when you analyze a model in SIL or PIL mode.

Data Types: `char`

Data Types: `struct`

`functionCall` — Function call coverage information
`structure array`

Function call coverage information, returned as a N_c structure arrays, where N_c is the number of function calls analyzed. `functionCall` appears when `cvdo` contains code coverage data for the function call coverage metric. Each structure contains these fields:

`isFiltered` — Function call exclusion flag
`0` | `1`

Function call exclusion flag, returned as `1` if the function call is excluded and `0` if it is not.

Data Types: `double`

`justifiedCoverage` — Number of justified coverage outcomes
`scalar`

Number of justified coverage outcomes, returned as a scalar double.

Data Types: `double`

`isJustified` — Function call outcome justification flag
`0` | `1`

Function call outcome justification flag, returned as `1` if the function call outcome is justified and `0` if it is not.

Data Types: `double`

`filterRationale` — Function call coverage filter rationale
`[]` (default) | `character array`

Function call coverage filter rationale, returned as a character array. If the function call is not filtered or the filter rationale is not set, `filterRationale` returns an empty array.

Data Types: `char`

`text` — Function call coverage text
`'Function called'`

Function call coverage text, returned as `'Function called'`.

Data Types: `char`

`executionCount` — Execution count of function call
`integer`

Execution count of the function call, returned as a 64-bit integer.

Data Types: `int64`

`fileName` — Name of file containing function call
`character array`

Name of the file containing the function call, returned as a character array.

Data Types: `char`

`functionName` — Name of analyzed function call
`character array`

Name of the analyzed function call, returned as a character array.

Data Types: `char`

`sourceLocation` — Location of analyzed function call in source code
`structure array`

Location of the analyzed function call in the source code, returned as a structure array with these fields:

Field Name	Description	Datatype
`startLine`	Line of the source code where the function call begins	`int64`
`startCol`	Column of the source code where the function call begins	`int64`
`endLine`	Line of the source code where the function call ends	`int64`
`endCol`	Column of the source code where the function call ends	`int64`

Data Types: `struct`

`expression` — Function call expression
`character array`

Function call expression, returned as a character array.

Data Types: `char`

`modelElements` — Model element that corresponds to function call
`character array`

Model element that corresponds to the function call, returned as a character array. The `modelElements` field appears when you analyze a model in SIL or PIL mode.

Data Types: `char`

Data Types: `struct`

`executableStatement` — Statement coverage information
`structure array`

Statement coverage information, returned as N_s structure arrays, where N_s is the number of executable statements. `executableStatement` appears when `cvdo` contains code coverage data for the statement coverage metric. Each structure array contains these fields:

`isFiltered` — Statement exclusion flag
`0 | 1`

Statement exclusion flag, returned as `1` if the statement is excluded and `0` if it is not.

Data Types: `double`

`justifiedCoverage` — Number of justified statement outcomes
`scalar`

Number of justified statement outcomes, returned a scalar double.

Data Types: `double`

`isJustified` — Statement outcome justification flag
`0 | 1`

Statement outcome justification flag, returned as `1` if the statement outcome is justified and `0` if it is not.

Data Types: `double`

`filterRationale` — Statement coverage filter rationale
`character array`

Statement coverage filter rationale, returned as a character array. If the statement is not filtered or the filter rationale is not set, `filterRationale` returns an empty array.

Data Types: `char`

`text` — Statement coverage text
`'Statement executed'`

Statement coverage text, returned as `'Statement executed'`.

Data Types: `char`

`executionCount` — Statement execution count
`scalar`

Statement execution count, returned as a 64-bit integer.

Data Types: `int64`

`fileName` — Name of file containing statement
`character array`

Name of the file containing the statement, returned as a character array.

Data Types: `char`

`functionName` — Name of function containing statement
`character array`

Name of the function containing the statement, returned as a character array.

Data Types: `char`

`sourceLocation` — Location of analyzed statement in source code
`structure array`

Location of the analyzed statement in the source code, returned as a structure array with these fields:

Field Name	Description	Datatype
`startLine`	Line of the source code where the statement begins	`int64`
`startCol`	Column of the source code where the statement begins	`int64`
`endLine`	Line of the source code where the statement ends	`int64`
`endCol`	Column of the source code where the statement ends	`int64`

Data Types: `struct`

`kind` — Type of statement analyzed
`character array`

Type of statement analyzed, returned as a character array.

Example: `'stmt'`, `'if'`

Data Types: `char`

`modelElements` — Model element that corresponds to statement
`character array`

Model element that corresponds to the statement, returned as a character array. The `modelElements` field appears when you analyze a model in SIL or PIL mode.

Data Types: `char`

Data Types: `struct`

Alternatives

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

1. Open the model.
2. In the Model Editor, in the **Modeling** tab, select **Model Settings**.
3. On the **Coverage** pane of the Configuration Parameters dialog box, select **Enable coverage analysis**.
4. Under **Coverage metrics**, set **Structural coverage level** to `Block Execution` .
5. Click **OK** to close the Configuration Parameters dialog box and save your changes.

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

Version History

Introduced in R2006b

See Also

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

Topics

"Execution Coverage (EC)"

get

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

Get specified `cvdata` object in `cvdatagroup`

Syntax

```
covData = get(cvdg,dataName)
covData = get(cvdg,dataName,simMode)
```

Description

`covData = get(cvdg,dataName)` returns one or more `cvdata` objects in the `cv.cvdatagroup` object `cvdg` that correspond to the model or file specified in `dataName`. `get` returns multiple `cvdata` objects if you simulate the same model reference with multiple different simulation modes.

`covData = get(cvdg,dataName,simMode)` returns the `cvdata` object or objects that have the simulation mode `simMode`.

Input Arguments

cvdg — Coverage data group
`cv.cvdatagroup` object

Coverage data group, specified as a `cv.cvdatagroup` object.

Data Types: `cv.cvdatagroup`

dataName — Name of coverage data
character array | string array

Name of coverage data, specified as a character array or string array. `dataName` can be the name of a model, subsystem, or external code file that corresponds to one or more `cvdata` objects contained in the `cv.cvdatagroup`. If the function does not find a match, it returns an empty array. If the function finds multiple matches, it returns a cell array of `cvdata` objects.

Data Types: `char` | `string`

simMode — Simulation mode
`"Normal"` | `"SIL"` | `"PIL"` | `"ModelRefSIL"` | `"ModelRefPIL"` | `"ModelRefTopSIL"` | `"ModelRefTopPIL"`

Simulation mode that corresponds to one or more `cvdata` objects in the `cv.cvdatagroup` object, specified as one of the following:

Object Specification	Description
`"Normal"`	Model in normal simulation mode.

Object Specification	Description
`"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 the code interface set to the top model.

If the function does not find a match, it returns an empty array. If the function finds multiple matches, it returns a cell array.

Data Types: `char` | `string`

Output Arguments

covData — Coverage data

`cvdata` object | cell array

Coverage data, returned as a `cvdata` object, or a cell array of `cvdata` objects. If you specified `simMode`, `covDataArray` only contains coverage data with the matching `simMode`.

Data Types: `cell`

Examples

Get Coverage Data from cvdatagroup Objects

This example shows how to retrieve a specified coverage data object, all coverage data objects, or all coverage data object names from a `cv.cvdatagroup` object.

Load the example model.

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

Use the `Simulink.SimulationInput` class to set up the simulation scenario with coverage enabled.

```
simIn = Simulink.SimulationInput(modelName);
simIn = setModelParameter(simIn,'CovEnable','on');
simIn = setModelParameter(simIn,'CovMetricStructuralLevel','MCDC');
simIn = setModelParameter(simIn,'CovSaveSingleToWorkspaceVar','on');
simIn = setModelParameter(simIn,'CovSaveName','covData');
simIn = setModelParameter(simIn,'CovSaveOutputData','off');
```

Simulate the model by passing the `SimulationInput` object to the `sim` function, and then retrieve the coverage data from the `SimulationOutput` object.

```
simOut = sim(simIn);
covData = simOut.covData
```

```
covData = ... cv.cvdatagroup
```

```
  slcoverageExternalFile (simulation mode: Normal)
  slvnvdemo_eml_model_coverage_demo (simulation mode: Normal)
```

The example model contains a MATLAB® Function block that calls an external MATLAB function. Because Simulink Coverage returns the coverage data for the MATLAB function in a separate `cvdata` object, `sim` returns a `cv.cvdatagroup` object that contains the coverage results for both the model and the MATLAB function.

To determine the name associated with each `cvdata` object inside the `cv.cvdatagroup`, use the class method `allNames`.

```
covDataNames = allNames(covData)
```

```
covDataNames = 

  2x1 cell array

    {'slcoverageExternalFile'           }
    {'slvnvdemo_eml_model_coverage_demo'}
```

The `cv.cvdatagroup` contains the coverage results for the model as well as the MATLAB function.

To retrieve the coverage data for the model, use the `get` class method.

```
modelCovData = get(covData,modelName)
```

```
modelCovData = ... cvdata
            version: (R2023a)
                 id: 1314
               type: TEST_DATA
               test: cvtest object
             rootID: 1316
           checksum: [1x1 struct]
          modelinfo: [1x1 struct]
          startTime: 03-Mar-2023 13:40:11
           stopTime: 03-Mar-2023 13:40:11
  intervalStartTime: 0
   intervalStopTime: 0
simulationStartTime: 0
 simulationStopTime: 1
             filter: 
            simMode: Normal
```

To retrieve coverage data for both the model and the external file, use the `getAll` class method.

```
covDataArray = getAll(covData)
```

```
covDataArray = 

  2x1 cell array
```

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

Version History

Introduced in R2007b

See Also

`cvdata` | `cv.cvdatagroup`

getAll

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

Get all `cvdata` objects in `cvdatagroup` object

Syntax

```
covDataArray = getAll(cvdg)
covDataArray = getAll(cvdg,simMode)
```

Description

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

`covDataArray = getAll(cvdg,simMode)` returns the `cvdata` objects that have simulation mode `simMode`.

Input Arguments

cvdg — Coverage data group
`cv.cvdatagroup` object

Coverage data group, specified as a `cvdatagroup` object.

Data Types: `cv.cvdatagroup`

simMode — Simulation mode
`"Normal"` | `"SIL"` | `"PIL"` | `"ModelRefSIL"` | `"ModelRefPIL"` | `"ModelRefTopSIL"` | `"ModelRefTopPIL"`

Simulation mode that corresponds to one or more `cvdata` objects in the `cv.cvdatagroup` object, specified as one of 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 the code interface set to the top model.

If the function does not find a match, it returns an empty array. If the function finds multiple matches, it returns a cell array.

Data Types: `char` | `string`

Output Arguments

covDataArray — Coverage data
cell array

Coverage data, returned as a cell array of cvdata objects. If you specified simMode, covDataArray contains only coverage data that matches simMode.

Data Types: cell

Examples

Get Coverage Data from cvdatagroup Objects

This example shows how to retrieve a specified coverage data object, all coverage data objects, or all coverage data object names from a cv.cvdatagroup object.

Load the example model.

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

Use the Simulink.SimulationInput class to set up the simulation scenario with coverage enabled.

```
simIn = Simulink.SimulationInput(modelName);
simIn = setModelParameter(simIn,'CovEnable','on');
simIn = setModelParameter(simIn,'CovMetricStructuralLevel','MCDC');
simIn = setModelParameter(simIn,'CovSaveSingleToWorkspaceVar','on');
simIn = setModelParameter(simIn,'CovSaveName','covData');
simIn = setModelParameter(simIn,'CovSaveOutputData','off');
```

Simulate the model by passing the SimulationInput object to the sim function, and then retrieve the coverage data from the SimulationOutput object.

```
simOut = sim(simIn);
covData = simOut.covData
```

```
covData = ... cv.cvdatagroup

  slcoverageExternalFile (simulation mode: Normal)
  slvnvdemo_eml_model_coverage_demo (simulation mode: Normal)
```

The example model contains a MATLAB® Function block that calls an external MATLAB function. Because Simulink Coverage returns the coverage data for the MATLAB function in a separate cvdata object, sim returns a cv.cvdatagroup object that contains the coverage results for both the model and the MATLAB function.

To determine the name associated with each cvdata object inside the cv.cvdatagroup, use the class method allNames.

```
covDataNames = allNames(covData)
covDataNames =

  2x1 cell array

    {'slcoverageExternalFile'           }
    {'slvnvdemo_eml_model_coverage_demo'}
```

The `cv.cvdatagroup` contains the coverage results for the model as well as the MATLAB function.

To retrieve the coverage data for the model, use the `get` class method.

```
modelCovData = get(covData,modelName)
```

```
modelCovData = ... cvdata
            version: (R2023a)
                 id: 1314
               type: TEST_DATA
               test: cvtest object
             rootID: 1316
           checksum: [1x1 struct]
          modelinfo: [1x1 struct]
          startTime: 03-Mar-2023 13:40:11
           stopTime: 03-Mar-2023 13:40:11
  intervalStartTime: 0
   intervalStopTime: 0
simulationStartTime: 0
 simulationStopTime: 1
             filter: 
            simMode: Normal
```

To retrieve coverage data for both the model and the external file, use the `getAll` class method.

```
covDataArray = getAll(covData)
```

```
covDataArray =

  2x1 cell array

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

Version History

Introduced in R2007b

extract

Extract subsystem coverage data from system-level coverage data

Syntax

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

Description

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

Examples

Create HTML Coverage Report for a Subsystem from Model Coverage Data

Record coverage data for the `sf_car` model.

```
model = 'sf_car'
openExample(model)
load_system(model)
cvdo = cvsim(model);
```

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

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

Create a coverage report from the extracted coverage data.

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

Input Arguments

cvdo — System-level coverage data object
object (default)

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

Data Types: `object`

`subsystem` — Subsystem in a Simulink model
character vector (default)

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

Data Types: `character vector`

Output Arguments

excvd — Extracted coverage data object
object

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

Version History

Introduced in R2019b

R2019b: Extract subsystem coverage data from system-level coverage data

You can use `extract` to extract the coverage data for a subsystem.

See Also

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

Topics
"Automating Model Coverage Tasks"

getCoverageInfo

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

Syntax

```
[covInfo,description] = getCoverageInfo(cvdo,modelObject)
[covInfo,description] = getCoverageInfo(cvdo,modelObject,metric)
[covInfo,description] = getCoverageInfo(cvdo,modelObject,metric,
ignoreDescendants)
```

Description

`[covInfo,description] = getCoverageInfo(cvdo,modelObject)` 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.

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

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

Examples

Get Coverage for a Block Type

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 Block

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 for All Metrics

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.

Input Arguments

cvdo — Coverage data
`cvdata` object

Coverage data, specified as a `cvdata` object.

Data Types: `cvdata`

`modelObject` — Model object
`character array | string array | Simulink handle | Stateflow ID | cell array`

Model object, specified as a character array, string array, Simulink handle, Stateflow ID, or cell array.

To specify a model object, such as a block or a Stateflow chart, use one of these formats:

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

To specify an S-Function block or its contents, use one of these formats:

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

To specify a code coverage result, such as coverage data collected during software-in-the-loop (SIL) or processor-in-the-loop (PIL) analysis, use one of these formats:

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

Data Types: `char` | `string` | `cell` | `Stateflow.State` | `Stateflow.Transition`

`metric` — Coverage metric to retrieve
`cvmetric.Sldv` enumeration

Coverage metric to retrieve, specified as a `cvmetric.Sldv` enumeration, or a cell array of enumerations, with values that correspond to Simulink Design Verifier library blocks. If you do not specify a metric, `getCoverageInfo` returns coverage information for all available metrics for the specified `modelObject`.

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

Data Types: `cvmetric.Sldv`

`ignoreDescendants` — Whether to ignore descendants in coverage results
`false` or `0` (default) | `true` or `1`

Whether to ignore descendants in coverage results, specified as a numeric or logical `1 (true)` or `0 (false)`, where:

- `0 (false)` includes coverage results of descendant objects.
- `1 (true)` ignores coverage results of descendant objects.

Data Types: `single` | `double` | `logical`

Output Arguments

`covInfo` — Coverage Information
scalar | cell array

Coverage information, returned as a two-element array of the form `[covered_outcomes,total_outcomes]` if `cvdo` contains decision coverage data, or an empty array if it does not. If `modelObject` receives coverage for more than one metric, `covInfo` returns a cell array of two-element arrays where each cell corresponds to one metric.

`covered_outcomes`	Number of decision outcomes satisfied for `modelObject`
`total_outcomes`	Total number of decision outcomes for `modelObject`

Data Types: `double`

`description` — Coverage description
structure | cell array

Coverage description, returned as a structure array with the following fields:

Note If `modelObject` receives coverage for more than one metric, `description` returns a cell array of structures where each cell contains a `description` structure for one metric.

`isFiltered` — Block exclusion flag
`0 | 1`

Block exclusion flag, returned as `1` if the block is excluded and `0` if it is not.

Data Types: `double`

`filterRationale` — Block coverage filter rationale
`character array`

Block coverage filter rationale, returned as a character array.

Data Types: `char`

`justifiedCoverage` — Number of justified coverage objective outcomes
`scalar`

Number of justified coverage objective outcomes, returned as a scalar double.

Data Types: `double`

`isJustified` — Block justification flag
`0 | 1`

Block justification flag, returned as `1` if the block is justified or `0` if it is not.

Data Types: `double`

`testobjects` — Test objective information
structure

Test objective information, returned as a structure with the following fields:

`text` — Text
`'T'`

Text, returned as the character array `'T'`.

Data Types: `char`

`executionCount` — Number of time steps the objective is true
scalar double

Number of time steps the objective is true, returned as a scalar double.

Data Types: `double`

Data Types: `struct`

Data Types: `struct`

Alternatives

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

1 Open the model.

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

Version History

Introduced in R2009b

See Also

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

Topics

"Simulink Design Verifier Coverage"

mcdcinfo

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

Syntax

```
covInfo = mcdcinfo(cvdo,modelObject)
covInfo = mcdcinfo(cvdo, modelObject, simMode)
covInfo = mcdcinfo(cvdo,modelObject,ignoreDescendants)
[covInfo, description] = mcdcinfo(cvdo,modelObject)
```

Description

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

`covInfo = mcdcinfo(cvdo, modelObject, simMode)` returns MCDC results from the `cvdata` object `cvdo` for the model component specified by `modelObject` for the simulation mode `simMode`.

`covInfo = mcdcinfo(cvdo,modelObject,ignoreDescendants)` returns MCDC results for `modelObject`, depending on the value of `ignoreDescendants`.

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

Examples

Retrieve MCDC Coverage Information

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

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

Input Arguments

cvdo — Coverage data
`cvdata` object

Coverage data, specified as a `cvdata` object.

Data Types: `cvdata`

`modelObject` — Model object
`character array | string array | Simulink handle | Stateflow ID | cell array`

Model object, specified as a character array, string array, Simulink handle, Stateflow ID, or cell array.

To specify a model object, such as a block or a Stateflow chart, use one of these formats:

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

To specify an S-Function block or its contents, use one of these formats:

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

To specify a code coverage result, such as coverage data collected during software-in-the-loop (SIL) or processor-in-the-loop (PIL) analysis, use one of these formats:

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

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

Data Types: `char` | `string` | `cell` | `Stateflow.State` | `Stateflow.Transition`

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

Simulation mode during coverage analysis, specified as one of these options:

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 SIL or PIL simulation mode.
`"ModelRefTopSIL"` or `"ModelRefTopPIL"`	Model reference in SIL or PIL simulation mode with the code interface set to top model.

Data Types: `char` | `string`

`ignoreDescendants` — Whether to ignore descendants in coverage results
`false` or `0` (default) | `true` or `1`

Whether to ignore descendants in coverage results, specified as a numeric or logical `1 (true)` or `0 (false)`, where:

- `0 (false)` includes coverage results of descendant objects.
- `1 (true)` ignores coverage results of descendant objects.

Data Types: `single` | `double` | `logical`

Output Arguments

`covInfo` — Coverage information
scalar

Coverage information, returned as a two-element array of the form `[covered_outcomes,total_outcomes]` if `cvdo` contains MCDC data, or an empty array if it does not.

`covered_outcomes`	Number of MCDC outcomes satisfied for `modelObject`
`total_outcomes`	Total number of MCDC outcomes for `modelObject`

Data Types: `double`

`description` — MCDC coverage description
`structure`

MCDC coverage description, returned as a structure array with the following fields:

`isFiltered` — Block exclusion flag
`0 | 1`

Block exclusion flag, returned as `1` if the block is excluded and `0` if it is not.

Data Types: `double`

`filterRationale` — Block coverage filter rationale
`character array`

Block coverage filter rationale, returned as a character array.

Data Types: `char`

`justifiedCoverage` — Number of justified coverage objective outcomes
`scalar`

Number of justified coverage objective outcomes, returned as a scalar double.

Data Types: `double`

`text` — Description of the measured objective
`character array`

Description of the measured objective, returned as a character array.

Data Types: `char`

`condition` — Information for individual MCDC outcomes
`structure`

Information for individual MCDC outcomes, returned as a structure with the following fields:

`isFiltered` — MCDC outcome exclusion flag
`0 | 1`

MCDC outcome exclusion flag, returned as `1` if the MCDC outcome is excluded or `0` if it is not.

Data Types: `double`

`isJustified` — MCDC outcome justification flag
`0 | 1`

MCDC outcome justification flag, returned as `1` if the MCDC outcome is justified or `0` if it is not.

Data Types: `double`

`filterRationale` — Coverage filter rationale
`character array`

Coverage filter rationale, returned as a character array. If a filter is not applied, or the applied filter does not contain filter rules, `filterRationale` is returned as an empty array.

Data Types: `char`

`text` — Description of MCDC outcome
`character array`

Description of the MCDC outcome, returned as a character array.

Data Types: `char`

`achieved` — MCDC outcome flag
`0` | `1`

MCDC outcome flag, returned as `1` if the MCDC outcome is fully satisfied, and `0` if it is not. The MCDC outcome is fully satisfied if each condition independently affects the decision outcome.

Data Types: `double`

`trueResult` — Condition values that achieve the `true` MCDC result
`scalar`

Condition values that achieve the `true` MCDC result, returned as a

Data Types: `double`

`falseResult` — Number of time steps condition is `false`
`scalar`

Number of time steps the condition is `false`, returned as a scalar.

Data Types: `double`

`trueExecutedIn` — Aggregated coverage test case traceability information
`structure array` | `[]`

Aggregated coverage test case traceability information, returned as a structure array. If your coverage data does not contain aggregated results from multiple simulations, `trueExecutedIn` is an empty array. If your coverage data contains aggregated results, `trueExecutedIn` indicates which test runs executed the `true` condition outcome and has the following fields:

`uniqueId`	Unique identifier for the `cvdata` object created by the test case that executed this condition outcome, returned as a character array.
`analyzedModel`	The name of the model analyzed, returned as a character array.
`description`	The test case description, returned as a character array.
`date`	The date and time of the simulation that created the `cvdata` object, returned as a character array.
`traceLabel`	The short name of the test case, returned as a character array.
`testRunInfo`	Test details, returned as a structure array.
`testRunInfo.runid`	The identifier of the run that generated the coverage result, returned as a scalar double.
`testRunInfo.runName`	The name of the test case that generated the coverage result, returned as a character array.
`testRunInfo.testId`	Identifying data for the test case that generated the coverage result, returned as a structure array.

`testRunInfo.testId.uuid`	The unique identifier for the test case that generated the coverage result, returned as a character array.
`testRunInfo.testId.contextType`	The test context in which the coverage result was obtained, returned as one of these values: `'RE'` for the Coverage Results Explorer `'ST'` for the Test Manager in Simulink Test

Data Types: `struct`

`falseExecutedIn` — Aggregated coverage test case traceability information
`structure array` | `[]`

Aggregated coverage test case traceability information, returned as a structure array. If your coverage data does not contain aggregated results from multiple simulations, `falseExecutedIn` is an empty array. If your coverage data contains aggregated results, `falseExecutedIn` indicates which test runs executed the `false` condition outcome and has the following fields:

`uniqueId`	Unique identifier for the `cvdata` object created by the test case that executed this condition outcome, returned as a character array.
`analyzedModel`	The name of the model analyzed, returned as a character array.
`description`	The test case description, returned as a character array.
`date`	The date and time of the simulation that created the `cvdata` object, returned as a character array.
`traceLabel`	The short name of the test case, returned as a character array.
`testRunInfo`	Test details, returned as a structure array.
`testRunInfo.runid`	The identifier of the run that generated the coverage result, returned as a scalar double.
`testRunInfo.runName`	The name of the test case that generated the coverage result, returned as a character array.
`testRunInfo.testId`	Identifying data for the test case that generated the coverage result, returned as a structure array.
`testRunInfo.testId.uuid`	The unique identifier for the test case that generated the coverage result, returned as a character array.
`testRunInfo.testId.contextType`	The test context in which the coverage result was obtained, returned as one of these values: `'RE'` for the Coverage Results Explorer `'ST'` for the Test Manager in Simulink Test

Data Types: `struct`

Data Types: `double`

Data Types: `struct`

Alternatives

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

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

Version History

Introduced in R2006b

See Also

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

Topics

"Modified Condition/Decision Coverage (MCDC)"
"MCDC Analysis"

overflowsaturationinfo

Retrieve saturation on integer overflow coverage from `cvdata` object

Syntax

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

Description

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

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

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

Examples

Extract Saturation on Integer Overflow Data

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

Generate coverage data

Open the model and set coverage settings.

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

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

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

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

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

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

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

Extract saturation on integer overflow results

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

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


saturationResults =

     1     2


percentSaturationCoverage =

    50
```

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

Determine Individual Integer Overflow Outcomes

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

Generate coverage data

Load the `slvnvdemo_saturation_on_overflow_coverage` example model.

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

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

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

Extract saturation on integer overflow results

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

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

```
covResults =

     1     2


covDesc = 

  struct with fields:

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


percentSaturation =

    50
```

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

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

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

```
outcome1 = 

  struct with fields:

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


outcome2 = 

  struct with fields:

          execCount: 0
     executionCount: 0
```

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

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

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

Input Arguments

`covdata` — Coverage results data

`cvdata` object

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

`object` — Model or model component

full path | handle

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

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

Example: `'slvnvdemo_saturation_on_overflow_coverage'`

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

`ignoreDescendants` — Whether to ignore descendants in coverage results

`false` or `0` (default) | `true` or `1`

Whether to ignore descendants in coverage results, specified as a numeric or logical `1 (true)` or `0 (false)`, where:

- `0 (false)` includes coverage results of descendant objects.
- `1 (true)` ignores coverage results of descendant objects.

Data Types: `single` | `double` | `logical`

Output Arguments

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

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

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

Data Types: `double`

`description` — Textual description of coverage outcomes
structure array

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

`isFiltered`	`0` if the model component specified by `object` is not excluded from coverage recording. `1` if the model component specified by `object` is excluded from coverage recording. For more information about excluding objects from coverage, see “Coverage Filtering”.	
`decision.text`	`'Saturate on integer overflow'`	
`decision.outcome`	Structure array containing two fields for each coverage outcome:	
	`executionCount`	Number of times saturation on integer overflow for `object` evaluated to the outcome described by `text`.
	`text`	`'true'` or `'false'`
	Saturation on integer overflow has two possible outcomes, `'true'` and `'false'`.	

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

Data Types: `struct`

Version History

Introduced in R2013a

See Also

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

Topics

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

relationalboundaryinfo

Retrieve relational boundary coverage from `cvdata` object

Syntax

```
covInfo = relationalboundaryinfo(cvdo,modelObject)
covInfo = relationalboundaryinfo(cvdo,modelObject,simMode)
covInfo = relationalboundaryinfo(cvdo,modelObject,ignoreDescendants)
[covInfo,description] = relationalboundaryinfo(cvdo,modelObject)
```

Description

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

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

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

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

Examples

Collect Relational Boundary Coverage for Supported Block in Model

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

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

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

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

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

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

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

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

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

```
covResults =

     2     4


covDesc =

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

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

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

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

```
ans =

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

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

View results for the first relational operation in the block.

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

```
ans =

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


ans =

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

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

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

```
ans = 

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

View results for the second relational operation in the block.

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

```
ans = 

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


ans = 

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

Input Arguments

cvdo — Coverage data

`cvdata` object

Coverage data, specified as a `cvdata` object.

Data Types: `cvdata`

modelObject — Model object

`character array` | `string array` | `Simulink handle` | `Stateflow ID` | `cell array`

Model object, specified as a character array, string array, Simulink handle, Stateflow ID, or cell array.

To specify a model object, such as a block or a Stateflow chart, use one of these formats:

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

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

To specify an S-Function block or its contents, use one of these formats:

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

To specify a code coverage result, such as coverage data collected during software-in-the-loop (SIL) or processor-in-the-loop (PIL) analysis, use one of these formats:

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

Data Types: `char` | `string` | `cell` | `Stateflow.State` | `Stateflow.Transition`

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

Simulation mode during coverage analysis, specified as one of these options:

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 SIL or PIL simulation mode.
`"ModelRefTopSIL"` or `"ModelRefTopPIL"`	Model reference in SIL or PIL simulation mode with the code interface set to top model.

Data Types: `char` | `string`

`ignoreDescendants` — Whether to ignore descendants in coverage results
`false` or `0` (default) | `true` or `1`

Whether to ignore descendants in coverage results, specified as a numeric or logical `1 (true)` or `0 (false)`, where:

- `0 (false)` includes coverage results of descendant objects.
- `1 (true)` ignores coverage results of descendant objects.

Data Types: `single` | `double` | `logical`

Output Arguments

`covInfo` — Coverage information
scalar | `[]`

Coverage information, returned as a two-element array of the form `[covered_outcomes, total_outcomes]` if `cvdo` contains relational boundary coverage data, or an empty array if it does not.

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

Data Types: `double`

`description` — Textual description of coverage outcomes
structure array

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

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

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

Data Types: `struct`

Version History

Introduced in R2014b

See Also

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

Topics

“Command Line Verification Tutorial”
“Relational Boundary Coverage”

sigrangeinfo

Retrieve signal range coverage information from `cvdata` object

Syntax

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

Description

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

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

Examples

Collect Signal Range Data for a Block

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

Load the model and set up coverage options

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

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

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

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

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

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

Get signal range data

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

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

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

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

```
minVal =

     0


maxVal =

   25.0000
```

Input Arguments

cvdo — Coverage data
`cvdata` object

Coverage data, specified as a `cvdata` object.

Data Types: `cvdata`

`modelObject` — Model object
`character array` | `string array` | `Simulink handle` | `Stateflow ID` | `cell array`

Model object, specified as a character array, string array, Simulink handle, Stateflow ID, or cell array.

To specify a model object, such as a block or a Stateflow chart, use one of these formats:

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

To specify an S-Function block or its contents, use one of these formats:

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

Object Specification	Description
`{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

To specify a code coverage result, such as coverage data collected during software-in-the-loop (SIL) or processor-in-the-loop (PIL) analysis, use one of these formats:

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

Data Types: `char` | `string` | `cell` | `Stateflow.State` | `Stateflow.Transition`

`portID` — Output port of the block
character array | string array

Output port of the block, specified as a character array or a string array.

Data Types: `char` | `string`

Output Arguments

`max` — Maximum signal value output by model object
scalar (default) | `-Inf`

Maximum signal value output by the model object, returned as a scalar. If `modelObject` output is a vector, `max` is also a vector.

If a block is not executed, `sigrangeinfo` returns `-Inf` for the `max` value of that block.

Data Types: `double`

`min` — Minimum signal value output by model object
scalar (default) | `+Inf`

Minimum signal value output by the model object, returned as a scalar. If `modelObject` output is a vector, `max` is also a vector.

If a block is not executed, `sigrangeinfo` returns `+Inf` for the `min` value of that block.

Data Types: `double`

Alternatives

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

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

Version History

Introduced in R2006b

See Also

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

sigsizeinfo

Retrieve signal size coverage information from `cvdata` object

Syntax

```
[min,max,allocated] = sigsizeinfo(cvdo,modelObject)
[min,max,allocated] = sigsizeinfo(cvdo,modelObject,portID)
```

Description

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

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

Examples

Retrieve Signal Size Information

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);
```

Input Arguments

cvdo — Coverage data
`cvdata` object

Coverage data, specified as a `cvdata` object.

Data Types: `cvdata`

`modelObject` — Model object
`character array` | `string array` | `Simulink handle` | `Stateflow ID` | `cell array`

Model object, specified as a character array, string array, Simulink handle, Stateflow ID, or cell array.

To specify a model object, such as a block or a Stateflow chart, use one of these formats:

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

To specify an S-Function block or its contents, use one of these formats:

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

To specify a code coverage result, such as coverage data collected during software-in-the-loop (SIL) or processor-in-the-loop (PIL) analysis, use one of these formats:

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

Data Types: `char` | `string` | `cell` | `Stateflow.State` | `Stateflow.Transition`

`portID` — Output port of the block
character array | string array

Output port of the block, specified as a character array or a string array.

Data Types: `char` | `string`

Output Arguments

`max` — Maximum signal value output by model object
double

Maximum signal value output by the model object, returned as a scalar. If `modelObject` output is a vector, `max` is also a vector.

Data Types: `double`

`min` — Minimum signal value output by model object
double

Minimum signal value output by the model object, returned as a scalar. If `modelObject` output is a vector, `max` is also a vector.

Data Types: `double`

`allocated` — Allocated signal size output by model object
double

Allocated signal size output by the model object, returned as a scalar. If `modelObject` has multiple outputs, `max` is a vector that contains the allocated signal size for each output.

Data Types: `double`

Alternatives

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

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

Version History

Introduced in R2010b

See Also

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

slvnvextract

Extract atomic subsystem or subchart contents into new model

Syntax

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

newModel = slvnvextract(modelmissingslfunctiondef)

newModel = slvnvextract(modelmissingslfunctiondef, 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`.

`newModel = slvnvextract(modelmissingslfunctiondef)` extracts a stub model for the model `modelmissingslfunctiondef` which has missing Simulink function definitions for the Simulink Design Verifier to analyze. Returns the name of the new model in `newModel`. `sldvextract` uses the input model name with a suffix `SldvStub` for the extracted model name, appending a numeral to the model name if that model name already exists.

`newModel = slvnvextract(modelmissingslfunctiondef, showModel)` opens the extracted model if you set `showModel` to `true`. Extracted model will be only loaded if `showModel` is set to `false`.

Examples

Extract Atomic Subsystem to New Model

This example shows how to extract an atomic subsystem from a model and copy it to a new model using `slvnvextract`.

Load the `sldemo_lct_bus` model.

```
modelName = 'slcoverage_lct_bus';
load_system(modelName);
load slcoverage_lct_data.mat
```

The `slcoverage_lct_bus` model has an S-Function. Build the S-Function by using `legacy_code`.

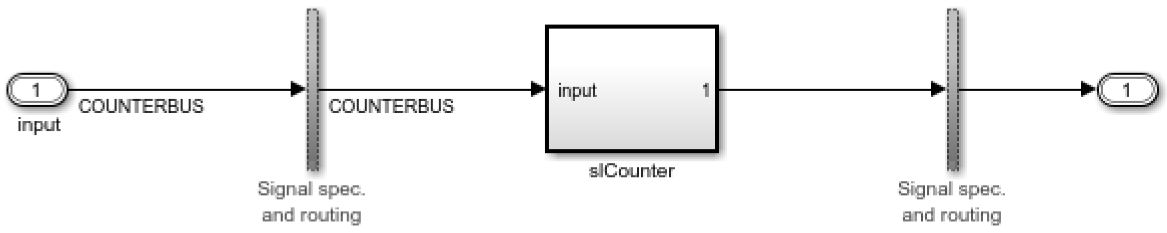
```
def = legacy_code('initialize');
def.SFunctionName = 'slcoverage_sfun_counterbus';
def.OutputFcnSpec = 'void counterbusFcn(COUNTERBUS u1[1], int32 u2, COUNTERBUS y1[1], int32 y2[1
def.HeaderFiles   = {'counterbus.h'};
def.SourceFiles   = {'counterbus.c'};
def.Options.supportCoverage = true;
legacy_code('generate_for_sim', def);
```

```
### Start Compiling slcoverage_sfun_counterbus
mex -IC:\TEMP\Bdoc23a_2213998_3568\ib570499\38\tpefa5766f\slcoverage-ex66778043 -c C:\TEMP\Bdoc23
Building with 'Microsoft Visual C++ 2019 (C)'.
MEX completed successfully.
mex -IC:\TEMP\Bdoc23a_2213998_3568\ib570499\38\tpefa5766f\slcoverage-ex66778043 C:\TEMP\Bdoc23a_2
Building with 'Microsoft Visual C++ 2019 (C)'.
MEX completed successfully.
mex -IC:\TEMP\Bdoc23a_2213998_3568\ib570499\38\tpefa5766f\slcoverage-ex66778043 -c C:\TEMP\Bdoc23
Building with 'Microsoft Visual C++ 2019 (C)'.
MEX completed successfully.
mex -IC:\TEMP\Bdoc23a_2213998_3568\ib570499\38\tpefa5766f\slcoverage-ex66778043 C:\TEMP\Bdoc23a_2
Building with 'Microsoft Visual C++ 2019 (C)'.
MEX completed successfully.
### Finish Compiling slcoverage_sfun_counterbus
### Exit
```

Extract the slCounter subsystem from the model and copy it to a new model.

```
newModel = slvnvextract([modelName,'/slCounter'],true)
```

```
newModel =

    'slCounter'
```



Extract Subchart and Copy to a New Model

This example shows how to extract an atomic Stateflow® chart from a model and copy it to a new model using `slvnvextract`.

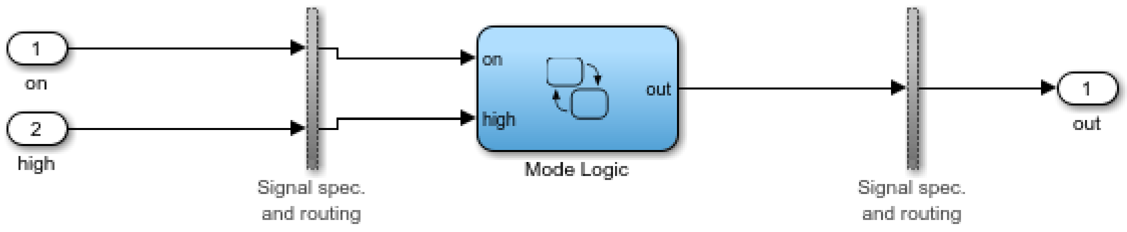
Load the `slvnvdemo_covfilt` model.

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

Extract the Mode Logic atomic subchart from the model and copy it into a new model.

```
newModel = slvnvextract([modelName,'/Mode Logic'],true)
```

```
newModel =

    'ModeLogic'
```



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.

modelmissingslfunctiondef — Missing Simulink function
character vector | string scalar

Full path to the missing Simulink function model, specified as a character vector or string scalar.

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`.

Version History

Introduced in R2010b

slvnvharnessopts

Generate default options for `slvnvmakeharness`

Syntax

`harnessopts = slvnvharnessopts`

Description

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

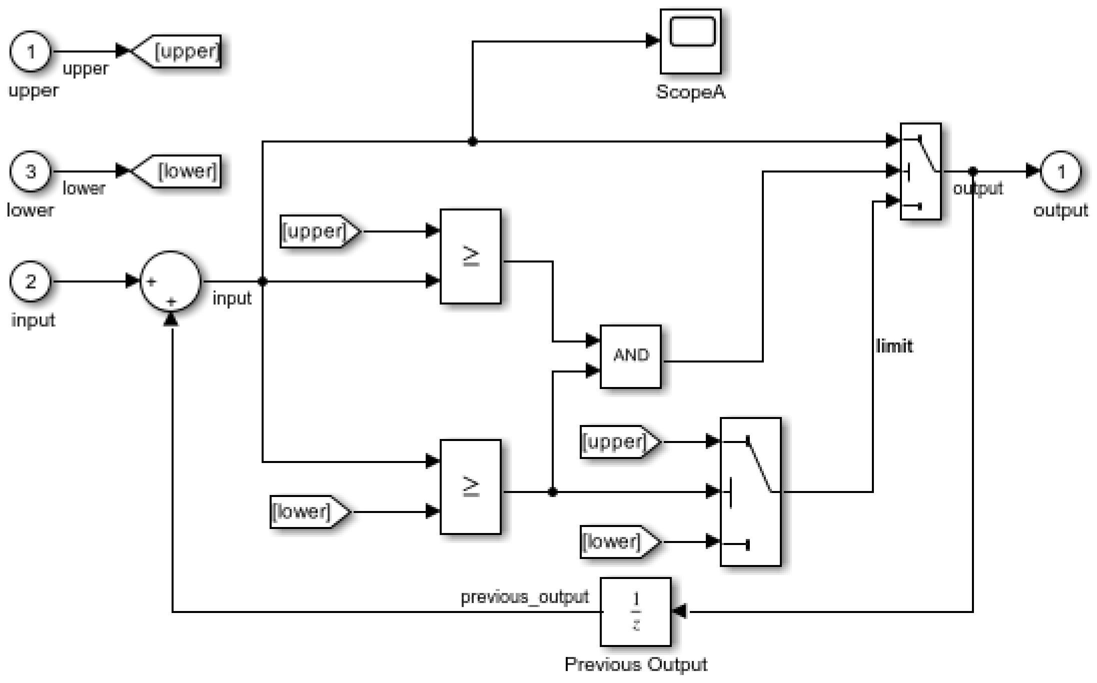
Examples

Create a Coverage Harness with Default Options

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

Open the model using `open_system`.

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



Copyright 2015 The MathWorks, Inc.

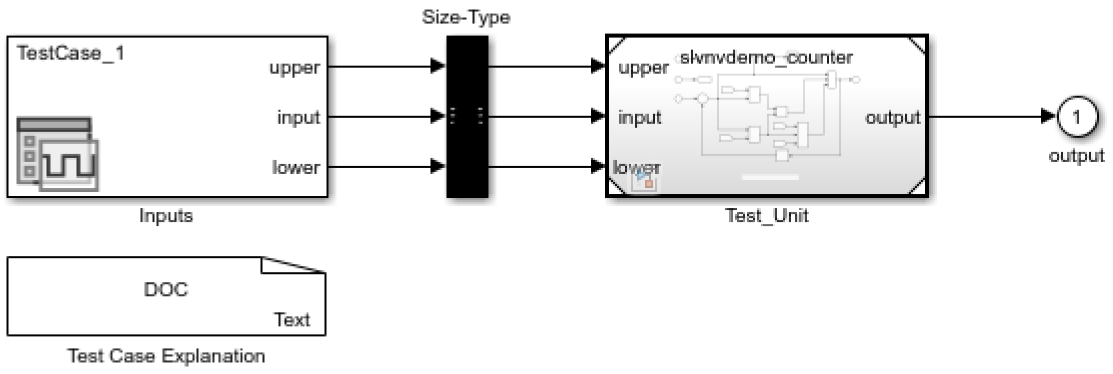
Create the harness options object using `slvnvharnessopts`.

```
harnessOpts = slvnvharnessopts
```

```
harnessOpts = 

  struct with fields:

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

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

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



Output Arguments

`harnessopts` — Default harness options

structure

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

Field	Description
`harnessFilePath`	Specifies the file path for creating the harness model. If an invalid path is specified, `slvnvmakeharness` does not save the harness model, but it creates and opens the harness model. If you do not specify this option, `slvnvmakeharness` generates a new harness model and saves it in the MATLAB current folder. Default: `''`
`modelRefHarness`	Generates the test harness model that includes `model` in a Model block. When `false`, the test harness model includes a copy of `model`. Default: `true`

Field	Description
`usedSignalsOnly`	When `true`, the Signal Editor block in the harness model has signals only for input signals in the model. You must have the Simulink Design Verifier software and `model` must be compatible with that software to detect the input signals. Default: `false`

Version History
Introduced in R2010b

See Also
`slvnvmakeharness`

slvnvlogsignals

Log test data for component or model during simulation

Syntax

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

Description

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

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

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

Examples

Log and Visualize Simulation Data

This example shows how to log simulation data for a Model block and use the logged data to create a harness model and visualize the data in the referenced model.

Open the directory that contains the example files.

```
openExample('ComponentBasedModelingWithModelReferenceExample')
```

Open the model and log the signals for the CounterB block.

```
open_system('sldemo_mdlref_basic');
data = slvnvlogsignals('sldemo_mdlref_basic/CounterB');
```

Create a harness model for `sldemo_mdlref_counter` using the logged data and the default harness options.

```
load_system('sldemo_mdlref_counter');
harnessOpts = slvnvharnessopts
[harnessFilePath]  = ...
```

```
    slvnvmakeharness('sldemo_mdlref_counter', data, ...
    harnessOpts);
```

Input Arguments

`model_block` — Component or model
character vector or string | handle

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

`harness_model` — Harness name
character vector or string | handle

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

`test_case_index` — Indices of test cases to be simulated
scalar

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

Output Arguments

`data` — Output data
structure

Structure that contains the logged data.

Version History

Introduced in R2010b

See Also

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

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`. `slvnvmakeharness` creates a harness model containing the Model block, a Signal Editor 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 harness creation options structure `harnessOpts`. 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.

Examples

Create a Test Harness Using the Default Options

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

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

Input Arguments

`model` — Simulink model
character array | string array | handle

Simulink model, specified as a character array, string array, or model handle.

`dataFile` — Data structure
`''` (default) | structure | character array | string array

Data structure, specified as a structure, or as a character array or string array containing the name of the `.mat` file. Use `slvnvlogsignals` or `slvnvmergedata` to create this structure. The structure contains information about the model, its input and output ports, and any preexisting test signals. Use this input when you have previously logged test data that you want to import into a new test harness.

`harnessOpts` — Harness options
structure

Harness options, specified as a structure. Use `slvnvharnessopts` to create this structure.

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 Editor 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`

Output Arguments

`harnessFilePath` — Path and file name of harness model
character array

Path and file name of the harness model, returned as a character array.

Version History

Introduced in R2010b

See Also

`slvnvharnessopts` | `slvnvmergeharness`

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`.

Examples

Log Signals and Merge Logged Data

This example shows how to merge logged data from multiple test cases.

Open the directory that contains the example files.

```
openExample('ComponentBasedModelingWithModelReferenceExample')
```

Open the `sldemo_mdlref_basic` model, which contains three Model blocks that reference the `sldemo_mdlref_counter` model.

```
open_system('sldemo_mdlref_basic');
```

Log the input signals to the three Model blocks.

```
data1 = slvnvlogsignals('sldemo_mdlref_basic/CounterA');
data2 = slvnvlogsignals('sldemo_mdlref_basic/CounterB');
data3 = slvnvlogsignals('sldemo_mdlref_basic/CounterC');
```

Merge the logged data.

```
merged_data = slvnvmergedata(data1, data2, data3);
```

Simulate the referenced model, `sldemo_mdlref_counter`, with coverage enabled with the merged data.

```
open_system('sldemo_mdlref_counter');
runOpts = slvnvruntestopts;
runOpts.coverageEnabled = true;
[ outData, initialCov ] = slvnvruntest('sldemo_mdlref_counter', ...
     merged_data, runOpts);
```

Generate a coverage report to view the results.

```
cvhtml('Initial coverage', initialCov);
```

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.

Version History

Introduced in R2011a

See Also

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

slvnvmergeharness

Combine test data from harness models

Syntax

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

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.

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

Examples

Log Signals and Merge Test Harnesses

This example shows how to merge multiple test harnesses.

Open the directory that contains the example files.

```
openExample('ComponentBasedModelingWithModelReferenceExample')
```

Log the input signals to the three Model blocks in the `sldemo_mdlref_basic` example model.

```
open_system('sldemo_mdlref_basic');
data1 = slvnvlogsignals('sldemo_mdlref_basic/CounterA');
data2 = slvnvlogsignals('sldemo_mdlref_basic/CounterB');
data3 = slvnvlogsignals('sldemo_mdlref_basic/CounterC');
open_system('sldemo_mdlref_counter');
```

Create three test harnesses using the logged signals.

```
harness1FilePath = slvnvmakeharness('sldemo_mdlref_counter', data1);
harness2FilePath = slvnvmakeharness('sldemo_mdlref_counter', data2);
harness3FilePath = slvnvmakeharness('sldemo_mdlref_counter', data3)
[~, harness1] = fileparts(harness1FilePath);
[~, harness2] = fileparts(harness2FilePath);
[~, harness3] = fileparts(harness3FilePath);
```

Merge the three test harnesses.

```
slvnvmergeharness('new_harness_model',{harness1, harness2, harness3});
```

Input Arguments

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

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

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

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

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

Parameter settings for the test cases of each test harness model, specified as 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.

Output Arguments

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

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

Version History

Introduced in R2010b

See Also

`slvnvlogsignals` | `slvnvmakeharness`

slvnvruncgvtest

Use Code Generation Verification (CGV) API to 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`.

Examples

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

This example shows how to log signals, run tests, and compare results for generated code using the code generation API.

Open the directory that contains the example files.

```
openExample('ComponentBasedModelingWithModelReferenceExample')
```

Open the `sldemo_mdlref_basic` example model and log the input signals to the CounterA model block.

```
open_system('sldemo_mdlref_basic');
load_system('sldemo_mdlref_counter');
loggedData = slvnvlogsignals('sldemo_mdlref_basic/CounterA');
```

Create the default configuration object for `slvnvruncgvtest` and allow the model to be configured to execute test cases with the CGV API.

```
runOpts = slvnvruntestopts('cgv');
runOpts.allowCopyModel = true;
```

Execute the test using the logged signals first in normal mode, and then in software-in-the-loop (SIL) mode.

```
cgvObjectSim = slvnvruncgvtest('sldemo_mdlref_counter', loggedData, runOpts);
runOpts.cgvConn = 'sil';
cgvObjectSil = slvnvruncgvtest('sldemo_mdlref_counter', loggedData, runOpts);
```

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));
```

Input Arguments

`model` — Model name

character vector or string

Model name, specified as a character array or string.

`dataFile` — Input data file name

structure | character array | string

Input data file name, specified as a structure, a character array, or a string. Generate data by either:

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

`runOpts` — Simulation run options

structure

Simulation run options, specified as a structure array. The fields of `runOpts` specify the configuration of `slvnvruncgvtest` .

Field Name	Description
`testIdx`	Test case index array to simulate from `dataFile`. If `testIdx` = `[]` (the default), `slvnvruncgvtest` simulates all test cases.
`allowCopyModel`	If you have not configured your model to execute test cases with the CGV API, this field specifies creating and configuring the model. If `true` and you have not configured your `model` to execute test cases with the CGV API, `slvnvruncgvtest` copies the model, fixes the configuration, and executes the test cases on the copied model. If `false` (the default), an error occurs if the tests cannot execute with the CGV API. **Note** If you have not configured the top-level model or any referenced models to execute test cases, `slvnvruncgvtest` does not copy the model, even if `allowCopyModel` is `true`. An error occurs.
`cgvCompType`	Defines the software-in-the-loop (SIL) or processor-in-the-loop (PIL) approach for CGV: • `'topmodel'` (default) • `'modelblock'`

Field Name	Description
`cgvConn`	Specifies mode of execution for CGV: • `'sim'` (default) • `'sil'` • `'pil'`

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

Output Arguments

`cgvObject` — Code generation verification analysis results
`cgv.CGV` object

Code generation verification analysis results, returned as a `cgv.CGV` object.

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

Field	Description
`tout_slvnvruncgvtest`	Simulation time
`xout_slvnvruncgvtest`	State data
`yout_slvnvruncgvtest`	Output signal data
`logsout_slvnvruncgvtest`	Signal logging data for: • Signals connected to outports • Signals that are configured for logging data on the model

Tips

To run `slvnvruncgvtest`, you must have an Embedded Coder® license.

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

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

You can not run `slvnvruncgvtest` on a model that uses configuration set references. Consider changing your model to contain a copy of the configuration set instead of a reference to it.

Version History

Introduced in R2010b

See Also

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

Topics

"Programmatic Code Generation Verification" (Embedded Coder)

slvnvruntest

Simulate model by using input data

Syntax

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

Description

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

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

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

Examples

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

This example shows how to analyze a model for coverage and example the output data.

Open the directory that contains the example files.

```
openExample('ComponentBasedModelingWithModelReferenceExample')
```

Analyze the `sldemo_mdlref_basic` model and log the input signals to the CounterA model block.

```
open_system('sldemo_mdlref_basic');
loggedData = slvnvlogsignals('sldemo_mdlref_basic/CounterA');
```

Using the logged signals, simulate the model referenced in the Counter block.

```
runOpts = slvnvruntestopts;
runOpts.coverageEnabled = true;
load_system('sldemo_mdlref_counter');
[outData] = slvnvruntest('sldemo_mdlref_counter',...
    loggedData, runOpts);
```

Examine the output data from the first test case using the Simulation Data Inspector.

```
Simulink.sdi.createRun('Test Case 1 Output', 'namevalue',...
    {'output'}, {outData(1).find('logsout_slvnvruntest')});
Simulink.sdi.view;
```

Input Arguments

`model` — Simulink model that you simulate
character vector or string | handle

The Simulink model to simulate.

`dataFile` — Input data
character vector or string | structure

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

`runOpts` — Configuration specification
structure

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

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

Output Arguments

`outData` — Output objects obtained after simulating the test cases
array of `Simulink.SimulationOutput` objects

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

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

`covData` — Object that contains model coverage data
`covdata` object

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

Note `covdata` might reference a file containing the coverage results. The coverage data from the referenced file is automatically loaded into memory when `covdata` is used by a coverage function. This file gets stored in the `sldv_covoutput` folder generated within `sldv_output` folder.

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.

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

Version History

Introduced in R2010b

See Also

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

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')`es a `runOpts` structure for `slvnvruncgvtest`.

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')
```

Output Arguments

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

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

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

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

Field Name	Description
`cgvConn`	Available only for `slvnvruncgvtest`. Specifies mode of execution for CGV: • `'sim'` • `'sil'` • `'pil'` **Default:**`'sim'`
`fastRestart`	Available only for `slvnvruntest`. If `true`, Simulink Coverage uses fast restart mode for model simulation. **Default:** `true`
`useParallel`	Available only for `slvnvruntest`. If `true`, Simulink Coverage simulates test cases with parallel computing. This option requires a Parallel Computing Toolbox license. **Default:** `false`

Version History

Introduced in R2010b

See Also

`slvnvruncgvtest` | `slvnvruntest`

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

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

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

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

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

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

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

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

Data Types: `char`

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

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

`'all'` exports all masked blocks.

Data Types: `char`

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

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

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

Data Types: `char`

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

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

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

Data Types: `char`

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

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

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

Data Types: `char`

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

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

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

Data Types: `char`

`CovData` — `cvdata` objects to use
cvdata

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

Example: `'CovData'`, `covdata`

Output Arguments

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

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

Tips

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

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

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

Version History

Introduced in R2015a

See Also

`slwebview_req`

tableinfo

Retrieve lookup table coverage information from `cvdata` object

Syntax

```
covInfo = tableinfo(cvdo,modelObject)
covInfo = tableinfo(cvdo,modelObject,ignoreDescendants)
[covInfo,exeCounts] = tableinfo(cvdo,modelObject)
[covInfo,exeCounts,brkEquality] = tableinfo(cvdo,modelObject)
```

Description

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

`covInfo = tableinfo(cvdo,modelObject,ignoreDescendants)` returns lookup table coverage results for `modelObject`, depending on the value of `ignoreDescendants`.

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

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

Examples

Retrieve Lookup Table Coverage Information

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

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

```
%Percent MCDC outcomes covered
percent_cov = 100 * cov(1) / cov(2)
```

Input Arguments

cvdo — Coverage data
`cvdata` object

Coverage data, specified as a `cvdata` object.

Data Types: `cvdata`

modelObject — Model object
`character array` | `string array` | `Simulink handle` | `Stateflow ID` | `cell array`

Model object, specified as a character array, string array, Simulink handle, Stateflow ID, or cell array.

To specify a model object, such as a block or a Stateflow chart, use one of these formats:

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

To specify an S-Function block or its contents, use one of these formats:

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

To specify a code coverage result, such as coverage data collected during software-in-the-loop (SIL) or processor-in-the-loop (PIL) analysis, use one of these formats:

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

Data Types: `char` | `string` | `cell` | `Stateflow.State` | `Stateflow.Transition`

`ignoreDescendants` — Whether to ignore descendants in coverage results
`false` or `0` (default) | `true` or `1`

Whether to ignore descendants in coverage results, specified as a numeric or logical `1 (true)` or `0 (false)`, where:

- `0 (false)` includes coverage results of descendant objects.
- `1 (true)` ignores coverage results of descendant objects.

Data Types: `single` | `double` | `logical`

Output Arguments

`covInfo` — Coverage information
`scalar`

Coverage information, returned as a two-element array of the form `[covered_intervals,total_intervals]` if `cvdo` contains lookup table coverage data, or an empty array if it does not.

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

Data Types: `double`

`exeCounts` — Execution count
double array

Execution count, returned as an array of doubles. `exeCounts` contains execution counts for each interpolation and extrapolation interval in the lookup table block. `exeCounts` has one more entry in each dimension than the lookup table block.

Data Types: `double`

`brkEquality` — Breakpoint equality flag
cell array

Breakpoint equality flag, returned as a cell array. Each cell contains a vector which identifies the number of times during simulation the lookup table block input is equivalent to a breakpoint value. Each vector represents the breakpoints along a different lookup table dimension.

Data Types: `cell`

Alternatives

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

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

Version History

Introduced in R2006b

See Also

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

Topics

"Lookup Table Coverage"

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

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

Example: 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\'`

Version History

Introduced in R2015a

See Also

Topics

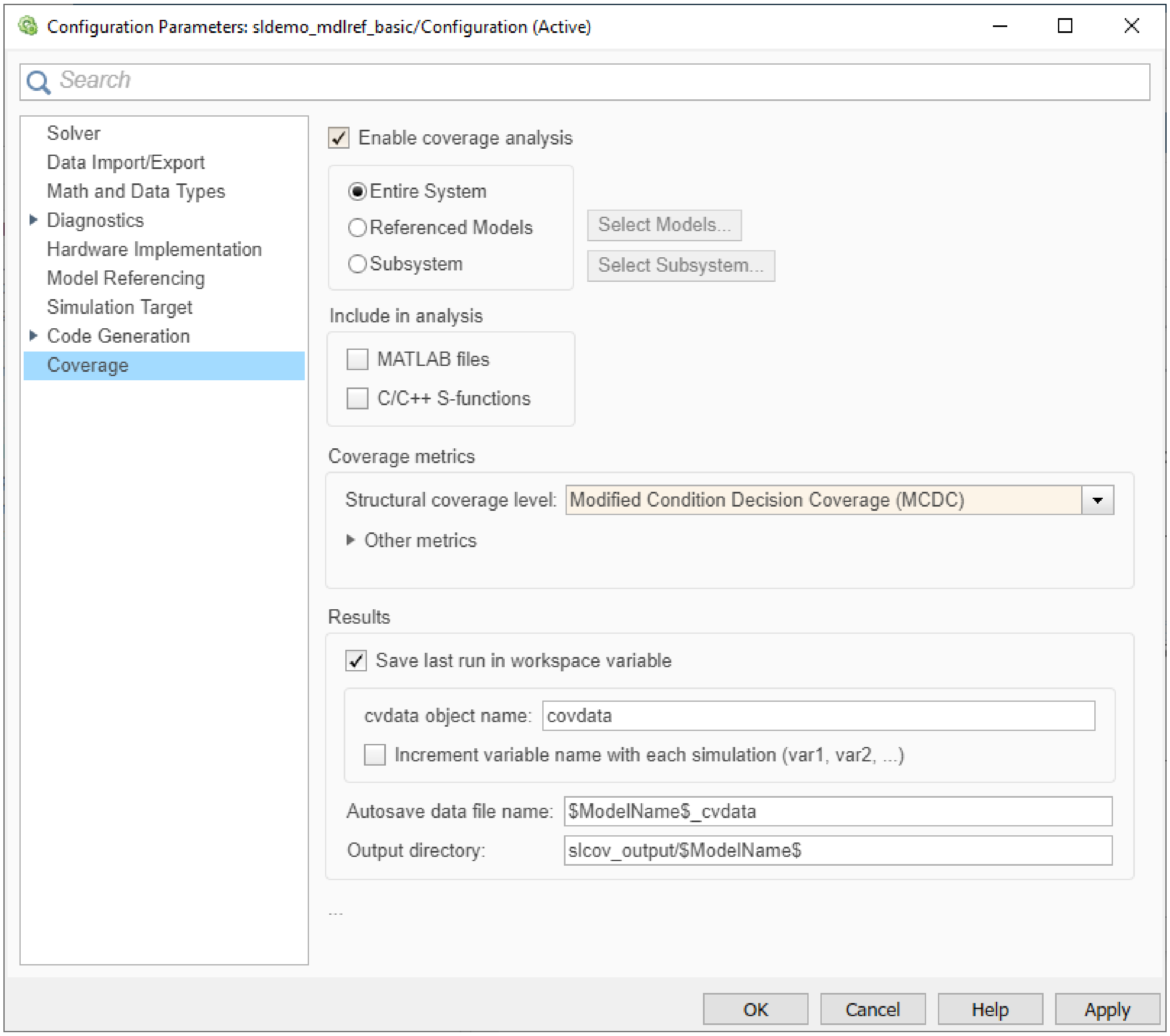
"Create a Basic C MEX S-Function"
"Templates for C S-Functions"
"Coverage for Custom C/C++ Code in Simulink Models"

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

Simulink Coverage Settings

Coverage Settings

Basic Coverage Settings


Configuration Parameters: sldemo_mdlref_basic/Configuration (Active)

Search

- Solver
- Data Import/Export
- Math and Data Types
- Diagnostics
- Hardware Implementation
- Model Referencing
- Simulation Target
- Code Generation
- Coverage

✓ Enable coverage analysis

● Entire System
○ Referenced Models — Select Models...
○ Subsystem — Select Subsystem...

Include in analysis

☐ MATLAB files
☐ C/C++ S-functions

Coverage metrics

Structural coverage level: Modified Condition Decision Coverage (MCDC)

▸ Other metrics

Results

✓ Save last run in workspace variable

cvdata object name: covdata

☐ Increment variable name with each simulation (var1, var2, ...)

Autosave data file name: $ModelName$_cvdata

Output directory: slcov_output/$ModelName$

...

OK Cancel Help Apply

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

- “Record coverage for C/C++ S-functions” on page 2-6
- “Structural coverage level” on page 2-6
- “Lookup table” on page 2-7
- “Signal range” on page 2-8
- “Signal size” on page 2-8
- “Objectives and constraints” on page 2-9
- “Saturation on integer overflow” on page 2-9
- “Relational boundary” on page 2-10
- “Relational boundary coverage absolute tolerance” on page 2-10
- “Relational boundary coverage relative tolerance” on page 2-10
- “Save last run in workspace variable” on page 2-11
- “Last coverage run variable name” on page 2-12
- “Increment variable name with each simulation” on page 2-12
- “Autosave data file name” on page 2-13
- “Output directory” on page 2-13

Coverage Pane Overview

Specify the Simulink Coverage analysis options.

Enable coverage analysis

Enable coverage analysis. See “Specify Coverage Options”.

Settings

☑ On

Coverage data is collected during simulation.

☐ Off (default)

Coverage data is not collected during simulation.

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

Scope of coverage analysis

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

Settings

Entire System (default)

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

Referenced Models

Coverage data is collected for one or more referenced models. To specify the referenced models, use the parameter “Select Models” on page 2-4. 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 and Observer models for which you want coverage. You can also select the top-level model. The icon next to the model name indicates the simulation mode: `Normal`, `SIL`, or `PIL`. Only Observer models in `Normal` mode are analyzed for coverage.

Dependency

To enable this parameter, select:

- “Enable coverage analysis” on page 2-3
- Specify referenced models for “Scope of coverage analysis” on page 2-3.

Command-Line Information

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

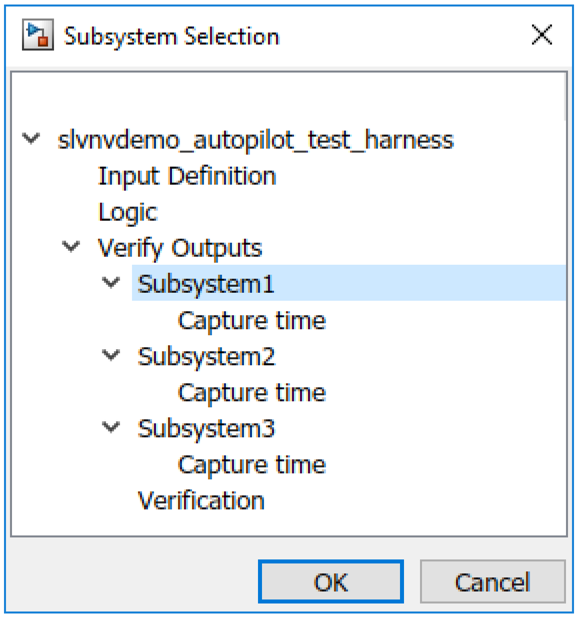
Parameter: `CovModelRefExcluded`
Type: Character vector or string
Value: Comma-separated list of model names, for instance, `'mRefA, mRefB, mRefC'`. If the same model is referenced in two simulation modes, you can distinguish between them using `:`, for instance, `'mRefA:normal, mRefA:sil'`.
Default: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

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 2-3

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”.

If your model contains C/C++ S-functions, selecting or clearing **Record coverage for C/C++ S-functions** can cause the coverage checksum to change.

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”.

☐ 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”.

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 2-10

Command-Line Information

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

Relational boundary coverage relative tolerance

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

Settings

Enter a number less than 1.

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

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

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

Dependency

To enable this parameter, select:

- “Enable coverage analysis” on page 2-3
- “Relational boundary” on page 2-10

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

Save last run in workspace variable

Save the coverage data from simulation in a MATLAB variable.

You can retrieve coverage information from this variable later. For instance, to retrieve decision coverage information, use the `decisioninfo` function. For the full list of functions, see “Manage Coverage Data”.

Settings

☑ On

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

☐ 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 2-12. 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 2-11

Command-Line Information

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

Increment variable name with each simulation

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

Settings

✓ On

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

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

☐ Off (default)

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

Dependency

To enable this parameter, select:

- "Enable coverage analysis" on page 2-3
- "Save last run in workspace variable" on page 2-11

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 2-24

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.

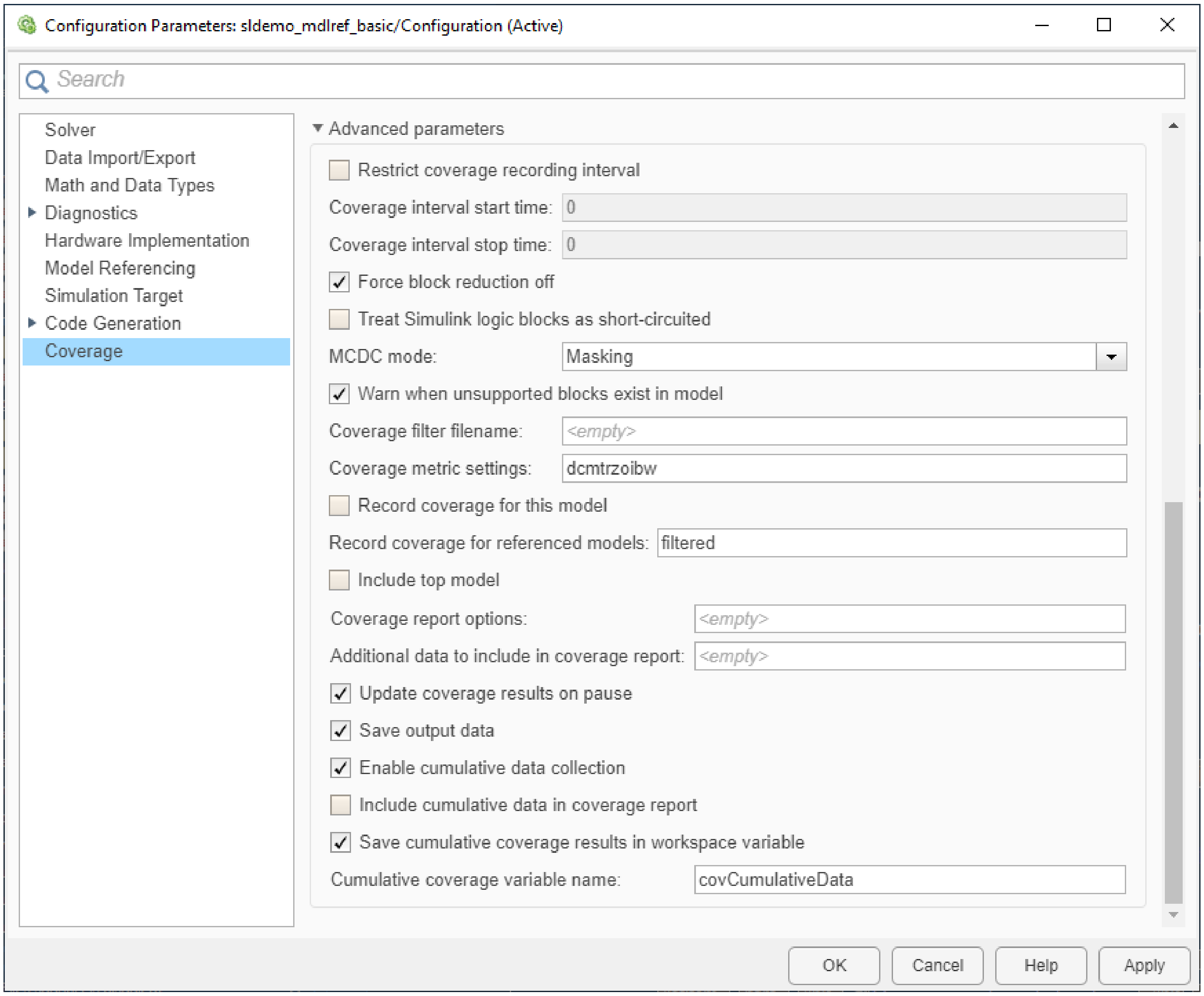
By default, the files are saved in a subfolder `slcov_output/$ModelName$` relative to the current working folder. Here `$ModelName$` is the model name.

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/$ModelName$'`

Advanced Coverage Settings



- [ ] Restrict coverage recording interval
- Coverage interval start time: 0
- Coverage interval stop time: 0
- [✓] Force block reduction off
- [ ] Treat Simulink logic blocks as short-circuited
- MCDC mode: Masking
- [✓] Warn when unsupported blocks exist in model
- Coverage filter filename: <empty>
- Coverage metric settings: dcmtrzoibw
- [ ] Record coverage for this model
- Record coverage for referenced models: filtered
- [ ] Include top model
- Coverage report options: <empty>
- Additional data to include in coverage report: <empty>
- [✓] Update coverage results on pause
- [✓] Save output data
- [✓] Enable cumulative data collection
- [ ] Include cumulative data in coverage report
- [✓] Save cumulative coverage results in workspace variable
- Cumulative coverage variable name: covCumulativeData

- "Restrict coverage recording interval" on page 2-15
- "Coverage interval start time" on page 2-15
- "Coverage interval stop time" on page 2-16
- "Force block reduction off" on page 2-16
- "Treat Simulink logic blocks as short-circuited" on page 2-17
- "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 2-21
- “Coverage report options” on page 2-22
- “Additional data to include in coverage report” on page 2-23
- “Exclude inactive choices of variants” on page 2-24
- “Update coverage results on pause” on page 2-24
- “Save output data” on page 2-24
- “Enable cumulative data collection” on page 2-25
- “Include cumulative data in coverage report” on page 2-25
- “Save cumulative coverage results in workspace variable” on page 2-26
- “Cumulative coverage variable name” on page 2-27

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 2-16

☐ 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 2-15

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 2-15

Command-Line Information

Parameter: `CovStopTime`
Type: Floating-point number
Value: Time in seconds, for instance, 4
Default: 0

Force block reduction off

Report coverage for every block in the model that is supported for coverage.

Settings

☑ On (default)

Coverage is recorded for every supported block in the model. The value of the configuration parameter **Block reduction** is ignored. See "Block reduction".

☐ Off

Coverage is not recorded for blocks that are effectively removed from the model because of block reduction. For instance, coverage is not recorded for a block that is reduced by dead code elimination.

Dependency

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

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 example, 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.

`Unique-Cause`

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 2-6.

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 before simulation, or to justify certain coverage outcomes.

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.

Note If you apply a model coverage filter that contains exclusion rules using the `CovFilter` parameter, collect coverage, and then remove the filter after simulation, you do not see coverage that the block would receive from that simulation without the filter applied. It is recommended to apply your coverage filter after simulation instead. This limitation does not apply to code coverage.

For an example of applying coverage filters after simulation, see:

- “Creating and Using Coverage Filters” for filtering in the Simulink UI.

- “Filter Coverage Results Using a Script” for filtering programmatically.

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 example, enter `'dc'` to collect decision and condition coverage.

To specify what coverage type(s) to collect, enter one or more of the letters in the following table as a single character array.

Note Using `CovMetricSettings` is not recommended. Use the alternatives in the table instead.

Letter	Effect on Coverage Analysis	Preferred Alternative
`d`	Collect decision coverage	Use the “Structural coverage level” on page 2-6 parameter
`c`	Collect condition coverage	Use the “Structural coverage level” on page 2-6 parameter
`m`	Collect MCDC coverage	Use the “Structural coverage level” on page 2-6 parameter
`t`	Collect lookup table coverage	Use the “Lookup table” on page 2-7 parameter
`r`	Collect signal range coverage	Use the “Signal range” on page 2-8 parameter
`z`	Collect signal size coverage	Use the “Signal size” on page 2-8 parameter
`o`	Collect coverage for Simulink Design Verifier blocks	Use the “Objectives and constraints” on page 2-9 parameter
`i`	Collect saturation on integer overflow coverage	Use the “Saturation on integer overflow” on page 2-9 parameter

Letter	Effect on Coverage Analysis	Preferred Alternative
`b`	Collect relational boundary coverage	Use the “Relational boundary” on page 2-10 parameter
`s`	Specify that coverage must take into account the order of operands in blocks that perform a logical operation	Use the “Treat Simulink logic blocks as short-circuited” on page 2-17 parameter.
`w`	Provide a warning when blocks in the model are not supported for coverage analysis	Use the “Warn when unsupported blocks exist in model” on page 2-18 parameter
`e`	Disables display coverage results using model coloring	**Note** `CovHighlightResults` has been removed. Include `e` in `CovMetricSettings` to avoid a warning message.

Dependency

To enable this parameter, select:

- “Enable coverage analysis” on page 2-3
- 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 2-5

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 2-3.

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 2-3.
- To specify the referenced models, use “Select Models” on page 2-4.

Settings

Enter one of these:

- `on`: Coverage data is collected for all referenced models and Observer models.
- `off`: Coverage data is not collected for referenced models.
- `filtered`: Coverage data is collected for all referenced models and Observer models except those excluded using the parameter “Select Models” on page 2-4.

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 2-3.
- To include or exclude the top-level model, use “Select Models” on page 2-4.

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 2-3.

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

Coverage report options

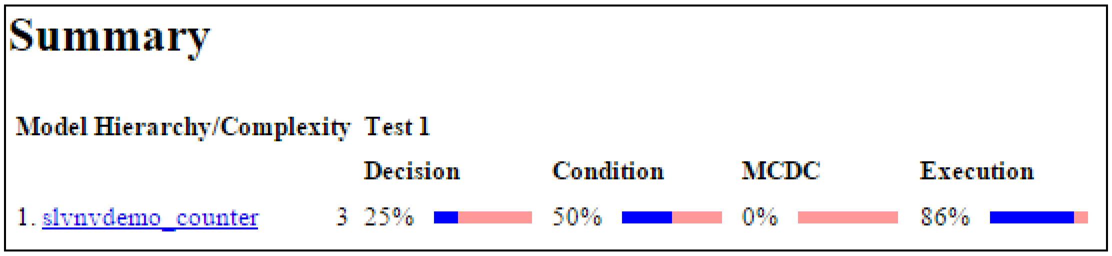
Specify the formatting of certain aspects of the coverage report (HTML).

Note For an easier way to specify report formatting, see Report from Results Explorer.

Settings

Enter a space-separated list of flags. The available flags are:

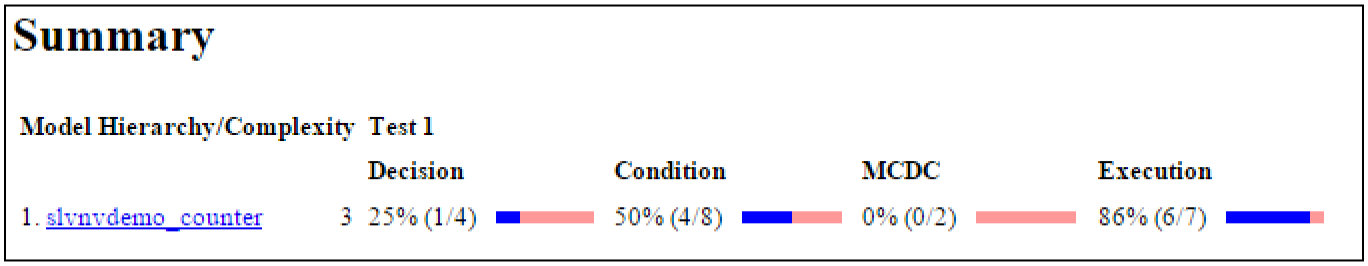
- `'-sRT=0'` — Do not show report
- `'-sVT=1'` — Open a web view of the report in a browser. See also "Export Model Coverage Web View".
- `'-aTS=1'` — Show each test in the model summary.
- `'-bRG=1'` — Show bar graphs in the model summary.



Summary

Model Hierarchy/Complexity		Test 1			
		Decision	Condition	MCDC	Execution
1. slvnvdemo_counter	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. slvnvdemo_counter	3	25% (1/4)	50% (4/8)	0% (0/2)	86% (6/7)

- `'-nFC=0'` — Do not report fully covered model objects
- `'-nFD=0'` — Exclude fully covered model object details from report.
- `'-scm=1'` — Include cyclomatic complexity numbers in summary. See also “Cyclomatic Complexity”.
- `'-bcm=1'` — Include cyclomatic complexity numbers in block details.
- `'-xEv=0'` — Filter Stateflow events from report.
- `'-agT=1'` — Show aggregated tests information in coverage report.
- `'-xEM=1'` — Filter execution metric from coverage report.
- `'-req=1'` — Include linked requirements in coverage report.

Dependency

To enable this parameter, select:

- “Enable coverage analysis” on page 2-3

Command-Line Information
Parameter: `CovHTMLOptions`
Type: Character vector or string
Value:
Default:

Additional data to include in coverage report

Include additional model coverage data from `cvdata` objects in the model coverage report.

Settings

Enter the name of a `cvdata` object associated with a simulation.

You get a `cvdata` object when you record coverage and save coverage data in a workspace variable. See:

- “Last coverage run variable name” on page 2-12
- “Cumulative coverage variable name” on page 2-27

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

Command-Line Information

Parameter: `CovCompData`
Type: Character vector or string
Value: Name of `cvdata` object.
Default: No default

Exclude inactive choices of variants

Exclude inactive choices of Simulink variants with startup activation time and inactive variant configurations in Stateflow charts from coverage data results.

Settings

✓ On

Exclude inactive choices of variants from the coverage report.

☐ Off (default)

Include inactive choices of variants in the coverage report.

You can also change the variant reporting behavior after your simulation completes by changing the value of the “`excludeInactiveVariants`” on page 3-0 property in the `cvdata` object before generating the coverage report.

Command-Line Information

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

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 2-13.

☐ 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 2-25. To save the data in one workspace variable, use the parameters “Save cumulative coverage results in workspace variable” on page 2-26 and “Cumulative coverage variable name” on page 2-27.

☐ 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
- “Enable cumulative data collection” on page 2-25

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 “Manage Coverage Data”.

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 2-27.

☐ 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 2-25

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 2-26
- “Enable cumulative data collection” on page 2-25

Command-Line Information

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

3

Classes

slcoverage.BlockSelector class

Package: `slcoverage`

Select blocks for coverage filter

Description

Specify block selection criteria for a filter rule.

The `slcoverage.BlockSelector` class is a `handle` class.

Creation

Description

`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 or an external MATLAB function called by a MATLAB function 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: `'slcoverage_lct_bus:18'`, `'RelationalOperator'`

Attributes:

SetAccess

  protected

Data Types: `char` | `string` | `handle` | `integer`

Properties

`ConstructorCode` — Code used to create this selector object

character array

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

Attributes:

`GetAccess`	`public`
`SetAccess`	`protected`

`Description` — Description of the selector

character vector

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

Attributes:

`GetAccess`	`public`
`SetAccess`	`protected`

Id — Model element identifier

Simulink ID (default) | property | handle

Model element identifier, specified as the property name of the element, the handle to an element, or the Simulink identifier of the element. Use a handle or ID for selector types that select an instance. Use a property name, such as the value of the `'BlockType'` property of a block, to select multiple model elements.

Attributes

`SetAccess`	`protected`

Data Types: `char` | `string` | `handle` | `integer`

Type — Block selector type

`slcoverage.BlockSelectorType` value

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

- `slcoverage.BlockSelectorType.BlockInstance` — An instance of a block or an external MATLAB function called by a MATLAB function 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.

Attributes:

`GetAccess`	`public`
`SetAccess`	`protected`

Methods

Public Methods

allSelectors	Selectors for model or code element

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 = 'slcoverage_lct_bus';
open_system(modelName);
set_param(modelName,'CovMetricStructuralLevel','MCDC','RecordCoverage','on');
```

Select blocks that have the same block type as the `upper GE input` block to add a filter rule for.

```
type = get_param('slcoverage_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('slcoverage_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.

Get a Selector for a MATLAB Function

This example shows how to get a selector for a MATLAB® function using the `slcoverage.BlockSelector` class.

The example model `slvnvdemo_eml_model_coverage_demo` contains a series of MATLAB function blocks. One of the MATLAB function blocks calls an external function called `slcoverageExternalFile.m`.

Start by loading the model and generating coverage results.

```
modelName = 'slvnvdemo_eml_model_coverage_demo';
load_system(modelName)
covData = cvsim(modelName);
```

Since the model uses an external function, the coverage results are returned in a `cv.cvdatagroup` object. Use the `cv.cvdatagroup.get` class method to extract the `cvdata` object that contains the results for the `slcoverageExternalFile` function.

```
functionCov = get(covData,'slcoverageExternalFile');
```

To exclude the entire function from the coverage report, use an exclusion filter. First, use the `slcoverage.BlockSelector` class to create a selector for the external MATLAB function `slcoverageExternalFile.m`. The correct `BlockSelectorType` enumeration to use for this is `BlockInstance`.

```
sel = slcoverage.BlockSelector(slcoverage.BlockSelectorType.BlockInstance,'slcoverageExternalFile
```

Create an empty `slcoverage.Filter` object.

```
filt = slcoverage.Filter;
```

Create the exclusion filter rule using the `slcoverage.FilterRule` class. Use the selector you created, the rationale for filtering the function, and the filter type of `slcoverage.FilterMode.Exclude`.

```
rule = slcoverage.FilterRule(sel,'Exclude function demo',slcoverage.FilterMode.Exclude);
```

Add the filter rule to the filter object, name the filter, and save it to a file.

```
addRule(filt,rule);
setFilterName(filt,'mfileFilter');
setFilterDescription(filt,'Demo exclusion filter for external m-file');
save(filt,'externalFileFilter')
```

Apply the filter file to the cvdata object which contains coverage results for the function `slcoverageExternalFile.m`.

```
functionCov.filter = 'externalFileFilter';
```

Create the coverage report and observe that the function `slcoverageExternalFile` now has no coverage data reported in the summary, and if you click on the function name to open the function report, you see `slcoverageExternalFile` listed under **Objects Filtered from Coverage Analysis**.

```
cvhtml('covReport',covData)
```

Version History

Introduced in R2017b

See Also

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

Topics

“Top-Level Model Coverage Report”
“Create, Edit, and View Coverage Filter Rules”

slcoverage.CodeSelector class

Package: `slcoverage`

Select custom C or C++ code for coverage filter

Description

Use objects of the `slcoverage.CodeSelector` class to specify custom C or C++ code selection criteria for a filter rule.

The `slcoverage.CodeSelector` class is a `handle` class.

Creation

`sel = slcoverage.CodeSelector(type,fileName)` creates `CodeSelector` object of the specified `type` based on the specified `fileName` and sets the `Type` and `FileName` properties.

`sel = slcoverage.CodeSelector(type,fileName,functionName)` creates a `CodeSelector` object based on the specified C or C++ `functionName` in the file and sets the `FunctionName` property.

`sel = slcoverage.CodeSelector(type,fileName,functionName,expr,exprIndex)` creates a `CodeSelector` object for the specified expression and expression index and sets the `Expr` and `ExprIndex` properties.

`sel = slcoverage.CodeSelector(type,fileName,functionName,expr,exprIndex,outcomeIndex)` creates a `CodeSelector` object based on the specified coverage outcome and sets the `OutcomeIndex` property.

`sel = slcoverage.CodeSelector(type,fileName,functionName,expr,exprIndex,outcomeIndex,parentExprIndex)` creates a `CodeSelector` object based on the specified coverage outcome that belongs to an expression owned by `parentExprIndex` and sets the `DecOrCondIndex` property to `parentExprIndex`.

Properties

Type — Type of custom C or C++ code to select
`slcoverage.CodeSelectorType` value

Type of custom C or C++ code to select, specified as an enumeration of the `slcoverage.CodeSelectorType` class:

- `slcoverage.CodeSelectorType.File` — A custom C or C++ code file name.
- `slcoverage.CodeSelectorType.Function` — A custom C or C++ code function name.
- `slcoverage.CodeSelectorType.Decision` — A custom C or C++ code decision.

- `slcoverage.CodeSelectorType.Condition` — A custom C or C++ code condition.
- `slcoverage.CodeSelectorType.DecisionOutcome` — A custom C or C++ code decision outcome.
- `slcoverage.CodeSelectorType.ConditionOutcome` — A custom C or C++ code condition outcome.
- `slcoverage.CodeSelectorType.MCDCOutcome` — A custom C or C++ code MCDC outcome.
- `slcoverage.CodeSelectorType.RelationalBoundaryOutcome` — A custom C or C++ code relational boundary outcome.

Example: `slcoverage.CodeSelectorType.Function`

Attributes

`SetAccess`	`protected`

Data Types: `slcoverage.CodeSelectorType`

`FileName` — C or C++ file to select
character array | string array

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

Example: `'myfile.c'`

Attributes

`SetAccess`	`protected`

Data Types: `char` | `string`

`FunctionName` — C or C++ function to select
character array | string array

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

Example: `'counterbusFcn'`

Attributes

`SetAccess`	`protected`

Data Types: `char` | `string`

`Expr` — Decision or condition expression to select
character array | string array

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

Example: `'x | y'`

Attributes

`SetAccess`	`protected`

Data Types: `char` | `string`

`ExprIndex` — Expression index
integer

Expression index, specified as an integer. If you are filtering an outcome, this property is the index of the expression that owns that outcome. If you are filtering an expression, this property is the index of that expression inside the body of the function.

Example: 2

Attributes

`SetAccess`	`protected`

Data Types: `single` | `double` | `int`

`OutcomeIndex` — Index of outcome to select
integer

Index of outcome to select, specified as an integer:

- For a Boolean expression, enter `1` for the `F` outcome or `2` for the `T` outcome.
- For a switch/case statement, enter `1` for the first case, `2` for the second case, and so on.
- For relational boundary
  - Integer type:
    - enter `1` for type `-1`.
    - enter `2` for type `+1`.
    - enter `3` for type `0`.
  - Float type:
    - enter `1` for `[-tol 0]` or `[-tol 0)`.
    - enter `2` for `(0 tol]` or `[0 tol]` is outcome `2`

  For more information about relational boundary coverage, see “Relational Boundary Coverage”.

Example: 2

Attributes

`SetAccess`	`protected`

Data Types: `single` | `double` | `int`

`DecOrCondIndex` — Parent expression index
integer

Parent expression index, specified as an integer. Use this input when you are filtering an expression owned by a parent decision or condition. This property is the index of the parent decision or condition relative to the function.

Example: 2

Attributes

`SetAccess`	`protected`

Data Types: `single` | `double` | `int`

`ConstructorCode` — Code used to create this selector object

character array

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

Attributes:

`GetAccess`	`public`
`SetAccess`	`protected`

`Description` — Description of the selector

character vector

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

Attributes:

`GetAccess`	`public`
`SetAccess`	`protected`

Id — Model element identifier

Simulink ID (default) | property | handle

This property is empty for the `slcoverage.CodeSelector` class.

Attributes

`SetAccess`	`protected`

Data Types: `char` | `string` | `handle` | `integer`

Methods

Public Methods

allSelectors	Selectors for model or code element

Examples

Add Code Selector Rules to a Filter

This example shows how to select custom C or C++ code for which you want to add a filter rule.

Load the model.

```
modelName = 'slcovCCallerExample';
Simulink.importExternalCTypes('my_func.h','EnumClass','dynamic');
load_system(modelName)
```

Configure coverage settings using a `Simulink.SimulationInput` object.

```
covSet = Simulink.SimulationInput(modelName);
covSet = covSet.setModelParameter('CovEnable','on');
covSet = covSet.setModelParameter('CovMetricStructuralLevel','MCDC');
covSet = covSet.setModelParameter('CovSFcnEnable','on');
covSet = covSet.setModelParameter('CovSaveSingleToWorkspaceVar','on');
covSet = covSet.setModelParameter('CovSaveName','covData');
covSet = covSet.setModelParameter('SimAnalyzeCustomCode','on');
```

Simulate the model using `covSet` object as the input.

```
simOut = sim(covSet);
covData = simOut.covData;
```

Create a selector object to filter the custom C function `timesK`.

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

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

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

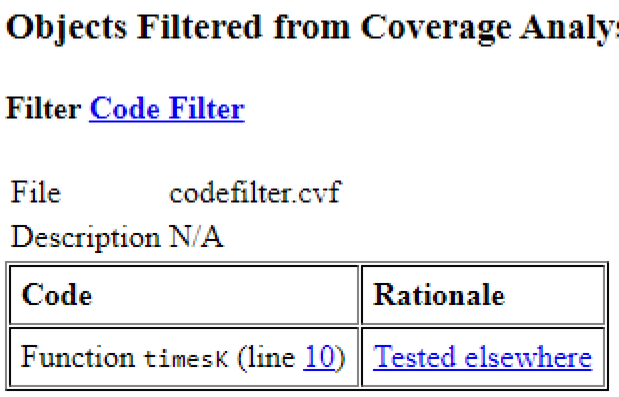
Save the filter as `codefilter` and add it to the `cvdata` object for `my_func.c`. Because the coverage data is stored in a `cv.cvdatagroup` object, use the `get` method to set the property.

```
save(filt,'codefilter');
covData.get('my_func.c').filter = 'codefilter';
```

Generate a coverage report.

```
cvhtml('codeCovReport',covData)
```

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



Objects Filtered from Coverage Analysis

Filter Code Filter

File	codefilter.cvf
Description	N/A

Code	Rationale
Function `timesK` (line 10)	Tested elsewhere

Create a C Code Outcome Selector

This example shows how to use an `slcoverage.CodeSelector` object to filter a code outcome in a custom C or C++ program called by a C Caller block.

Open the Model and Enable Coverage Analysis

Open the model.

```
modelName = 'slcovCCallerExample';
Simulink.importExternalCTypes('my_func.h','EnumClass','dynamic');
load_system(modelName)
```

Configure coverage settings using a `Simulink.SimulationInput` object.

```
covSet = Simulink.SimulationInput(modelName);
covSet = covSet.setModelParameter('CovEnable','on');
covSet = covSet.setModelParameter('CovMetricStructuralLevel','ConditionDecision');
covSet = covSet.setModelParameter('CovSFcnEnable','on');
covSet = covSet.setModelParameter('CovSaveSingleToWorkspaceVar','on');
covSet = covSet.setModelParameter('CovSaveName','covData');
```

Simulate the model using `covSet` as the input.

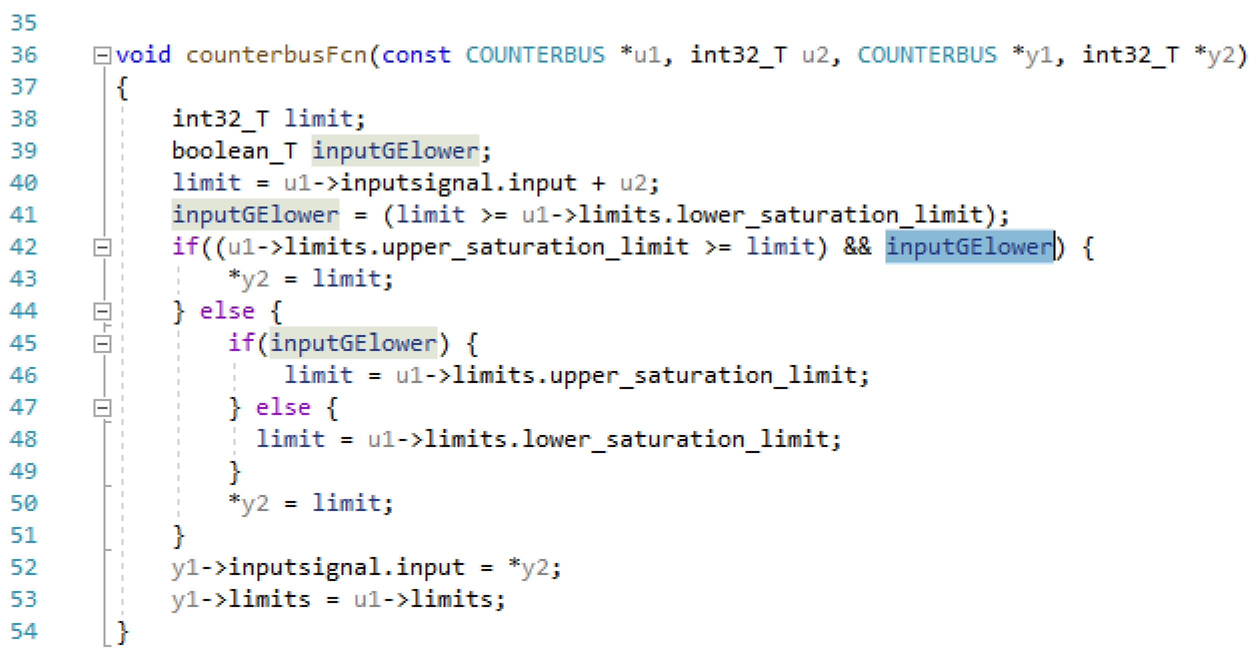
```
simOut = sim(covSet);
covData = simOut.covData;
```

The simulation returns the coverage data as a `cv.cvdatagroup` object when both the model and custom code are analyzed for coverage. To extract the code coverage data, use the `get` method of the `cvdatagroup` class.

```
codeCovData = get(covData,'my_func.c');
```

Justify the Missing Outcome

In this example, you justify the `F` outcome of the `inputGElower` condition in the `(u1->limits.upper_saturation_limit >= limit) && inputGElower` decision, which is located inside the `counterbusFcn` function in the `my_func.c` source file.



```
void counterbusFcn(const COUNTERBUS *u1, int32_T u2, COUNTERBUS *y1, int32_T *y2)
{
    int32_T limit;
    boolean_T inputGElower;
    limit = u1->inputsignal.input + u2;
    inputGElower = (limit >= u1->limits.lower_saturation_limit);
    if((u1->limits.upper_saturation_limit >= limit) && inputGElower) {
        *y2 = limit;
    } else {
        if(inputGElower) {
            limit = u1->limits.upper_saturation_limit;
        } else {
          limit = u1->limits.lower_saturation_limit;
        }
        *y2 = limit;
    }
    y1->inputsignal.input = *y2;
    y1->limits = u1->limits;
}
```

Create a selector object using `slcoverage.CodeSelector`. The first input is a `CodeSelectorType` enumeration. To justify a condition outcome, use a `ConditionOutcome` enumeration. The second input is the code source file, `my_func.c`. The third input is the name of the function that contains the outcome, `counterbusFcn`. The fourth input is the expression which contains the outcome, `(u1-`

>limits.upper_saturation_limit >= limit) && inputGElower. The fifth input is the index of the expression that owns the outcome. In this case, inputGElower is the second condition within its parent condition, so this input is 2. The sixth input is the condition outcome index, which is 1 for the F outcome of a Boolean expression. The seventh input is the index of the parent decision or condition, which is 1 for (u1->limits.upper_saturation_limit >= limit) && inputGElower because it is the first decision in the function.

```
enum = slcoverage.CodeSelectorType.ConditionOutcome;
sel = slcoverage.CodeSelector(enum,'my_func.c','counterbusFcn','(u1->limits.upper_saturation_lim
```

Create a Filter object and a FilterRule object and apply the rule to the filter.

```
filt = slcoverage.Filter;
rule = slcoverage.FilterRule(sel,'condition does not apply');
addRule(filt,rule);
```

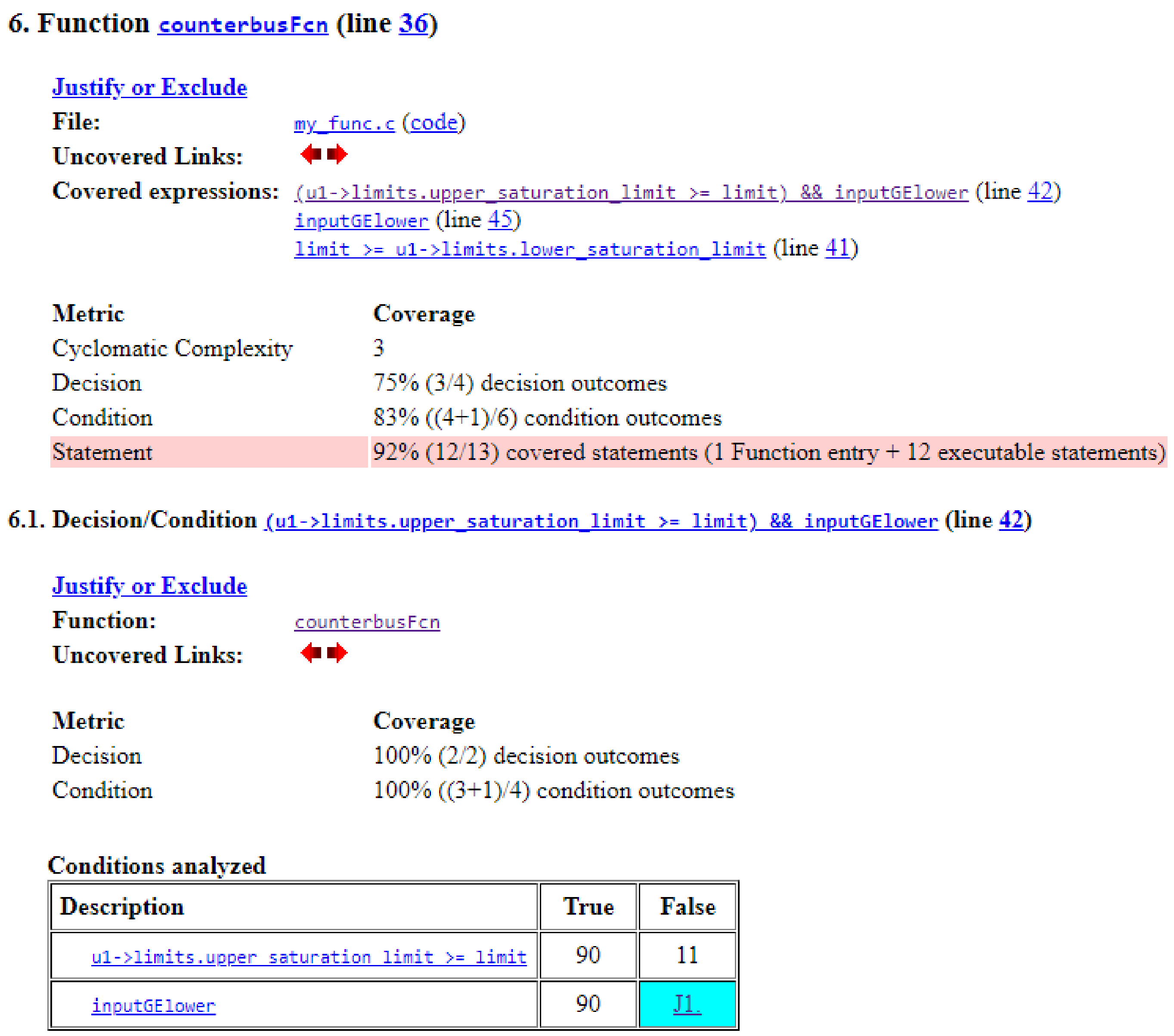
Save the filter to a filter file and then apply the filter to the cvdata object.

```
save(filt,'codeOutcomeFilter');
codeCovData.filter = 'codeOutcomeFilter';
```

Review the Coverage Report

Verify the outcome is filtered by generating a coverage report using cvhtml.

```
cvhtml('filteredCodeCovReport',codeCovData)
```



6. Function counterbusFcn (line 36)

Justify or Exclude

File:	my_func.c (code)
Uncovered Links:	⬅ ➡
Covered expressions:	(u1->limits.upper_saturation_limit >= limit) && inputGElower (line 42) inputGElower (line 45) limit >= u1->limits.lower_saturation_limit (line 41)

Metric	**Coverage**
Cyclomatic Complexity	3
Decision	75% (3/4) decision outcomes
Condition	83% ((4+1)/6) condition outcomes
Statement	92% (12/13) covered statements (1 Function entry + 12 executable statements)

6.1. Decision/Condition (u1->limits.upper_saturation_limit >= limit) && inputGElower (line 42)

Justify or Exclude

Function:	counterbusFcn
Uncovered Links:	⬅ ➡

Metric	**Coverage**
Decision	100% (2/2) decision outcomes
Condition	100% ((3+1)/4) condition outcomes

Conditions analyzed

Description	**True**	**False**
u1->limits.upper_saturation_limit >= limit	90	11
inputGElower	90	J1

Version History

Introduced in R2018b

R2021a: Justify unsatisfied code coverage outcomes

You can use the new `slcoverage.CodeSelectorType` enumerations to justify code coverage outcomes without excluding the entire expression:

- `slcoverage.CodeSelectorType.DecisionOutcome`
- `slcoverage.CodeSelectorType.ConditionOutcome`

- `slcoverage.CodeSelectorType.MCDCOutcome`
- `slcoverage.CodeSelectorType.RelationalBoundaryOutcome`

When using the new `slcoverage.CodeSelectorType` enumerations, use these new properties as optional input arguments:

- `outcomeIndex`
- `DecOrCondIndex`

See Also

`cv.cvdatagroup` | `slcoverage.Filter` | `slcoverage.FilterRule` | `slcoverage.MetricSelector` | `slcoverage.SFcnSelector` | `cvdata`

Topics

"Top-Level Model Coverage Report"
"Create, Edit, and View Coverage Filter Rules"

slcoverage.Filter class

Package: `slcoverage`

Coverage filter set

Description

Use an object of the `slcoverage.Filter` class to filter out unsatisfied coverage objectives by creating rules using the `slcoverage.FilterRule` class.

The `slcoverage.Filter` class is a `handle` class.

Creation

Description

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

`filt = slcoverage.Filter(filterFile)` creates an `slcoverage.Filter` object that contains the filter rules saved in `filterFile`.

Input Arguments

`filterFile` — Filter file
path name

Filter file in CVF format, specified as a character array of the path name to the file, or a cell array of character arrays. You do not need to include the file extension.

Example: `'myfilt'`, `'filters\myfilt'`, `{'myfilt1', 'myfilt2'}`

Methods

Public Methods

addRule	Add coverage filtering rule to filter
removeRule	Remove rule from filter rule set
rules	Rules for filter
setFilterName	Set name of coverage filter object
filterName	Get name of coverage filter object
setFilterDescription	Set description of coverage filter object
filterDescription	Get description of coverage filter object
save	Save coverage filter object to coverage filter file

Examples

Add Rule to a Filter File

This example shows how to add a rule to a coverage filter file.

Create a filter object and use the `BlockSelector` class to create a `BlockSelector` object for the Saturation block in the `slvnvdemo_covfilt` model.

```
filt = slcoverage.Filter;
blockSel = slcoverage.BlockSelector(slcoverage.BlockSelectorType.BlockInstance,...
           'slvnvdemo_covfilt/Saturation');
```

Create a rule to filter the Saturation block using `slcoverage.FilterRule` with the selector as the first input and the rationale as the second input.

```
rule = slcoverage.FilterRule(blockSel,'Edge case');
```

Use the `addRule` method of the `slcoverage.Filter` class to add the rule to the filter object.

```
addRule(filt,rule);
```

Save the filter with the new rule to a filter file using the `save` method of the `slcoverage.Filter` class.

```
save(filt,'myFilterFile')
```

Create and Use a Coverage Filter Object

This example shows how to create a filter object and add a rule to exclude a subsystem from coverage analysis.

Open the `slvnvdemo_covfilt` model. Use a `SimulationInput` object to enable coverage recording and specify coverage settings.

```
modelName = 'slvnvdemo_covfilt';
load_system(modelName)
simIn = Simulink.SimulationInput(modelName);
simIn = setModelParameter(simIn,'CovEnable','on');
simIn = setModelParameter(simIn,'CovMetricStructuralLevel','MCDC');
simIn = setModelParameter(simIn,'CovSaveSingleToWorkspaceVar','on');
simIn = setModelParameter(simIn,'CovSaveName','covData');
```

Simulate the model. The coverage data is stored in the value supplied for the `CovSaveName` property.

```
simOut = sim(simIn);
covData = simOut.covData;
```

You can filter a block by using the `slcoverage.BlockSelector` class. To exclude the Switchable config subsystem, use `slcoverage.BlockSelectorType.SubsystemAllContent` as the first input.

```
subsysSel = slcoverage.BlockSelector(...
                      slcoverage.BlockSelectorType.SubsystemAllContent,...
                      'slvnvdemo_covfilt/Switchable config');
```

Create the filter rule by passing the selector, rationale, and the exclude filter mode as inputs.

```
rule = slcoverage.FilterRule(subsysSel,...
                    'Unused configuration',...
                    slcoverage.FilterMode.Exclude);
```

Create an `slcoverage.Filter` object and then add the rule to it.

```
filt = slcoverage.Filter;
addRule(filt,rule);
```

Save the filter to a file called `blockFilter.cvf`. To create a report that uses this coverage filter, add the filter file as the value to the `filter` property of `covData`, and create a report called `coverageData.html` using `covData`.

```
save(filt,'blockFilter')
covData.filter = 'blockFilter';
cvhtml('coverageData',covData)
```

Use Multiple Coverage Filter Files for a Simulation

If you have multiple filter files that each contain their own set of rules, you can apply them to a coverage result set by creating a cell array of the filter file names or path names. In this example, you apply two filter files to a single `cvdata` object and then view the report to see that the filters are applied.

Load the `slvnvdemo_covfilt` model into memory.

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

Set the coverage settings by using a `Simulink.SimulationInput` object and simulate the model using `sim`.

```
simIn = Simulink.SimulationInput(modelName);
simIn = setModelParameter(simIn,'CovEnable','on');
simIn = setModelParameter(simIn,'CovMetricStructuralLevel','MCDC');
simIn = setModelParameter(simIn,'CovSaveSingleToWorkspaceVar','on');
simIn = setModelParameter(simIn,'CovSaveName','covData');
simOut = sim(simIn);
```

Apply the two filters to the `cvdata` object by assigning them to the filter property as a cell array.

```
covData.filter = {'filter_1','filter_2'};
cvhtml('twoFiltersCovData',covData);
```

You can see the applied coverage filters in the **Objects Filtered from Coverage Analysis** section of the coverage report.

Version History

Introduced in R2017b

See Also

`slcoverage.MetricSelector` | `slcoverage.BlockSelector` | `slcoverage.SFcnSelector` | `slcoverage.FilterRule`

Topics

“Create, Edit, and View Coverage Filter Rules”

slcoverage.FilterRule class

Package: `slcoverage`

Create coverage filter rule

Description

Use an object of the `slcoverage.FilterRule` class to specify a coverage filtering rule that includes the selector and the rationale for filtering.

The `slcoverage.FilterRule` class is a `handle` class.

Creation

`rule = slcoverage.FilterRule(selector,rationale)` creates the filter rule object `rule` using the specified `selector` and `rationale` and sets the `selector` and `rationale` properties.

`rule = slcoverage.FilterRule(selector,rationale,mode)` creates a `FilterRule` object with the specified filter `mode` and sets the `mode` property. You can use only justify with metric selectors.

Properties

`selector` — Selector for the rule
`slcoverage.BlockSelector` object | `slcoverage.MetricSelector` object | `slcoverage.SFcnSelector` object | `slcoverage.CodeSelector` object

Selector for the rule, specified as an `slcoverage.BlockSelector` object, `slcoverage.MetricSelector` object, `slcoverage.SFcnSelector` object, or `slcoverage.CodeSelector` object. The selector specifies the model object or coverage outcome to which the rule applies.

Attributes:

`GetAccess`	`public`
`SetAccess`	`public`

Data Types: `slcoverage.BlockSelector` | `slcoverage.MetricSelector` | `slcoverage.SFcnSelector` | `slcoverage.CodeSelector`

`rationale` — Rationale for filter rule
character array | string array

Rationale for filter rule, specified as a character array or string array.

Attributes:

`GetAccess`	`public`
`SetAccess`	`public`

Data Types: `char` | `string`

mode — Filter mode
`slcoverage.FilterMode.Justify` (default) | `slcoverage.FilterMode.Exclude`

Filter mode, specified as `slcoverage.FilterMode.Justify` or `slcoverage.FilterMode.Exclude`.

Attributes:

GetAccess	public
SetAccess	public

Data Types: `slcoverage.FilterMode`

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 = 'slcoverage_lct_bus';
open_system(modelName);
set_param(modelName,'CovMetricStructuralLevel','MCDC','RecordCoverage','on');
```

Select blocks with block type `'RelationalOperator'` to add a filter rule for.

```
bl = slcoverage.BlockSelector(slcoverage.BlockSelectorType.BlockType,'RelationalOperator');
```

Create a filter object, create a rule, and add the rule to the filter. This rule excludes the selection from coverage analysis.

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

Version History
Introduced in R2017b

See Also
`slcoverage.Filter` | `slcoverage.MetricSelector` | `slcoverage.BlockSelector` | `slcoverage.SFcnSelector`

Topics
“Coverage Filter Rules and Files”
“Create, Edit, and View Coverage Filter Rules”

slcoverage.MetricSelector class

Package: `slcoverage`

Select metric criterion for coverage filter

Description

Use an object of the `slcoverage.MetricSelector` class to specify metric selection criteria for a coverage filter rule.

The `slcoverage.MetricSelector` class is a `handle` class.

Creation

Description

`sel = slcoverage.MetricSelector(type,element,objIndex,outIndex)` creates a metric selector object of type `type` for the specified model element `element` at the objective index `objIndex` and outcome index `outIndex`.

You can only create a justify rule for a metric selector. For more information about the difference between justification and exclusion, see "Coverage Filtering".

For more information on the condition and decision coverage tables produced in the report, see "Top-Level Model Coverage Report".

Input Arguments

type — Metric selector type
`slcoverage.MetricSelectorType.ConditionOutcome` | `slcoverage.MetricSelectorType.DecisionOutcome` | `slcoverage.MetricSelectorType.MCDCOutcome` | `slcoverage.MetricSelectorType.RelationalBoundaryOutcome` | `slcoverage.MetricSelectorType.SaturationOverflowOutcome`

Metric selector type, specified as:

- `slcoverage.MetricSelectorType.ConditionOutcome` objects select condition metric objective outcomes.
- `slcoverage.MetricSelectorType.DecisionOutcome` objects select decision metric objective outcomes.
- `slcoverage.MetricSelectorType.MCDCOutcome` objects select MCDC metric objective outcomes.
- `slcoverage.MetricSelectorType.RelationalBoundaryOutcome` objects select outcome metrics related to relational boundary outcomes.
- `slcoverage.MetricSelectorType.SaturationOverflowOutcome` objects select outcome metrics related to saturation on integer overflow outcomes.

`element` — Model element to select
handle | Simulink ID

Model element to select, specified as a handle or the Simulink identifier of the model element.

Example: `'slcoverage_lct_bus:18'`

`objIndex` — Index of objective
integer

Index of the objective that you want to filter, specified as an integer.

Example: 1

`outIndex` — Index of outcome
integer

Index of the outcome that you want to filter, specified as an integer.

Example: 2

Properties

`ConstructorCode` — Code used to create this selector object
character array

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

Attributes:

`GetAccess`	`public`
`SetAccess`	`protected`

`Description` — Description of the selector
character vector

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

Attributes:

`GetAccess`	`public`
`SetAccess`	`protected`

`Id` — Element identifier
Simulink ID (default) | handle

Identifier of the model element, returned as character vector of the Simulink ID or a handle.

Attributes:

`GetAccess`	`public`
`SetAccess`	`protected`

`ObjectiveIndex` — Index of objective
integer

Index of the objective for this selector, returned as an integer.

Attributes:

`GetAccess`	`public`
`SetAccess`	`protected`

`OutcomeIndex` — Index of outcome
integer

Index of the outcome for this selector, returned as an integer.

Attributes:

`GetAccess`	`public`
`SetAccess`	`protected`

Type — Metric selector type
`ConditionOutcome` | `DecisionOutcome` | `MCDCOutcome` | `RelationalBoundaryOutcome` | `SaturationOverflowOutcome`

Selector type, returned as `ConditionOutcome`, `DecisionOutcome`, `MCDCOutcome`, `RelationalBoundaryOutcome`, or `SaturationOverflowOutcome`.

Attributes:

`GetAccess`	`public`
`SetAccess`	`protected`

Methods

Public Methods

allSelectors	Selectors for model or code element

Examples

Add Metric Selector Rule to a Filter

This example shows how to select a metric and add a rule that uses that metric. In this example, you create a rule to justify an unsatisfied decision for a Saturation block.

Open the Model and Enable Coverage Analysis

Load the model into memory.

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

Use a `Simulink.SimulationInput` object to configure coverage for the model.

```
covSet = Simulink.SimulationInput(modelName);
covSet = covSet.setModelParameter('CovEnable','on');
covSet = covSet.setModelParameter('CovMetricStructuralLevel','MCDC');
covSet = covSet.setModelParameter('CovSFcnEnable','on');
covSet = covSet.setModelParameter('StopTime','20');
covSet = covSet.setModelParameter('CovSaveSingleToWorkspaceVar','on');
covSet = covSet.setModelParameter('CovSaveName','covData');
```

Simulate the model using the `SimulationInput` object as the input.

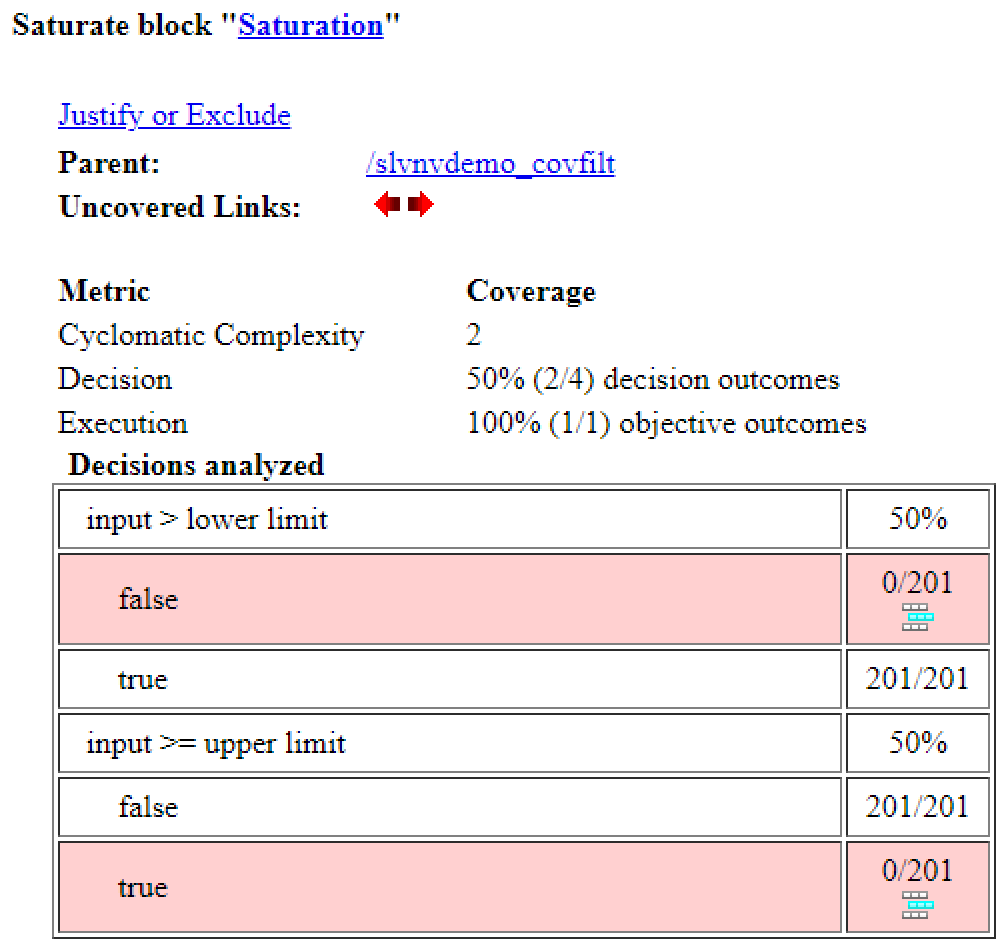
```
simOut = sim(covSet);
```

View the coverage results before applying a filter. You can access the coverage using `decisioninfo`, or you can view the HTML report using `cvhtml`.

```
covInitial = decisioninfo(covData,[modelName,'/Saturation']);
percentInitial = 100 * covInitial(1)/covInitial(2)
```

```
percentInitial =

    50
```

```
cvhtml('covReportInitial',covData)
```



Saturate block "Saturation"

Justify or Exclude

Parent:	/slvnvdemo_covfilt
Uncovered Links:	⬅ ➡

Metric	**Coverage**
Cyclomatic Complexity	2
Decision	50% (2/4) decision outcomes
Execution	100% (1/1) objective outcomes

Decisions analyzed

input > lower limit	50%
false	0/201
true	201/201
input >= upper limit	50%
false	201/201
true	0/201

Both `decisioninfo` and `cvhtml` show the same result of 50% decision coverage. If you don't intend your current tests to exercise this outcome, you can justify the outcome so it is no longer reported as missing coverage.

In this example, we justify the `false` decision outcome of the `input > lower limit` decision objective in the Saturation block.

Justify the Missing Condition Objective

`MetricSelector` objects accept the block path or the block handle as the second input. Get the block handle of the Saturation block by using `getSimulinkBlockHandle`.

```
id = getSimulinkBlockHandle([modelName,'/Saturation']);
```

Because the objective being justified is a decision outcome, the first input to the metric selector constructor is `slcoverage.MetricSelectorType.DecisionOutcome`. The second input is the block handle. The last two are the index of the objective to justify and the index of the outcome of that objective, respectively.

Because the `input > lower limit` decision objective is the first objective for the Saturation block, its objective index is `1`. Because the `false` outcome of this objective is the first outcome, its outcome index is also `1`. Therefore, the last two inputs are `1,1`.

```
metr = slcoverage.MetricSelector(slcoverage.MetricSelectorType.DecisionOutcome,id,1,1);
```

Create a filter and rule. In this case, we use the default filter mode of justify. Then add the rule to the filter using the `addRule` method.

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

Save the filter to a filter file using the `save` method. Then apply the filter file to the `cvdata` object by assigning the `filter` property to the new filter file.

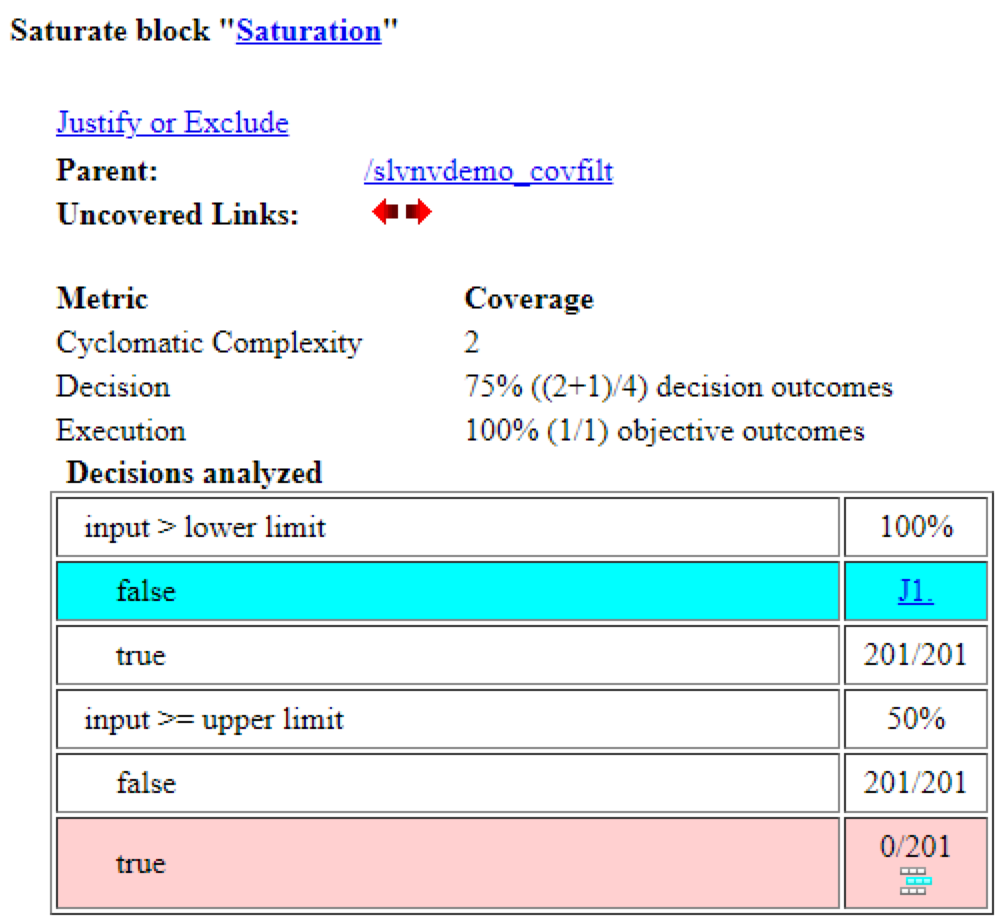
```
filt.save('metrfilter');
covData.filter = 'metrfilter';
```

Re-generate the coverage results for the Saturation block using the new filtered `cvdata` object.

```
covFiltered = decisioninfo(covData,[modelName,'/Saturation']);
percentInitial = 100 * covFiltered(1)/covFiltered(2)
```

```
percentInitial =

    75
```

```
cvhtml('covReportFiltered',covData)
```



Saturate block "Saturation"

Justify or Exclude

Parent: /slvnvdemo_covfilt

Uncovered Links:

Metric	**Coverage**
Cyclomatic Complexity	2
Decision	75% ((2+1)/4) decision outcomes
Execution	100% (1/1) objective outcomes

Decisions analyzed

input > lower limit	100%
false	J1
true	201/201
input >= upper limit	50%
false	201/201
true	0/201

In the HTML report, the missing decision outcome is highlighted to indicate that it is justified. Decision coverage for the Saturation block is now 75%.

Version History

Introduced in R2017b

R2020b: Justify missing coverage outcomes for individual MCDC objectives

You can use `slcoverage.MetricSelectorType.MCDCOutcome` to justify individual modified condition/decision coverage (MCDC) outcomes without excluding the entire expression.

See Also

`slcoverage.BlockSelector` | `getSimulinkBlockHandle` | `slcoverage.Filter` | `slcoverage.FilterRule` | `slcoverage.SFcnSelector`

Topics

"Coverage Filtering"
"Top-Level Model Coverage Report"
"Create, Edit, and View Coverage Filter Rules"

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 model object.

The `slcoverage.Selector` class is a `handle` class.

Properties

`ConstructorCode` — Code used to create this selector object

character array

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

Attributes:

`GetAccess`	`public`
`SetAccess`	`protected`

`Description` — Description of the selector

character vector

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

Attributes:

`GetAccess`	`public`
`SetAccess`	`protected`

Id — Element identifier

Simulink ID (default) | handle

Identifier of the model element, returned as character vector of the Simulink ID or a handle.

Attributes:

`GetAccess`	`public`
`SetAccess`	`protected`

Type — Selector type

selector type value

Selector type, returned as a selector type of the corresponding selector.

Attributes:

GetAccess	public
SetAccess	protected

Methods

Public Methods

allSelectors	Selectors for model or code element

Examples

Get All Selectors

This example shows how to get all the selectors for an And block and then add a rule to justify or exclude a selector. Metric selectors can only be justified.

Load the model and set coverage settings

```
modelName = 'slvnvdemo_covfilt';
load_system(modelName);
set_param(modelName,'CovEnable','on','CovMetricStructuralLevel','MCDC');
```

First, get the block handle for the And block.

```
id = getSimulinkBlockHandle([modelName,'/Saturation']);
```

Get the selectors using the block handle.

```
sel = slcoverage.Selector.allSelectors(id)
```

```
sel = 

  1x10 heterogeneous Selector (BlockSelector, MetricSelector) array with properties:

    Description
    Type
    Id
    ConstructorCode
```

The block has ten selectors. You can index into each one to see their contents. In this example, you want to justify the sixth selector.

```
sel(6)
```

```
ans = 

  MetricSelector with properties:

      ObjectiveIndex: 2
        OutcomeIndex: 2
         Description: 'T outcome of input > upper limit in Saturate block "Saturation"'
                Type: DecisionOutcome
                Id: 'slvnvdemo_covfilt:5'
   ConstructorCode: 'slcoverage.MetricSelector(slcoverage.MetricSelectorType.DecisionOutcome, 's
```

Create a justify rule, then create a filter object and add the rule to it.

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

Save the filter and generate a coverage report.

```
filt.save('metrfilter');
csim = cvsim(modelName);
csim.filter = 'metrfilter';
cvhtml('cov',csim,'-sRT=0');
```

Version History

Introduced in R2017b

See Also

`slcoverage.MetricSelector` | `slcoverage.BlockSelector` | `slcoverage.CodeSelector` | `slcoverage.SFcnSelector`

Topics

"Create, Edit, and View Coverage Filter Rules"

slcoverage.SFcnSelector class

Package: `slcoverage`

Select S-function criterion for filtering rule

Description

Use objects of the `slcoverage.SFcnSelector` class to specify the S-function selection criteria for a filter rule.

The `slcoverage.SFcnSelector` class is a `handle` class.

Creation

`sel = slcoverage.SFcnSelector(type,id)` creates an `SFcnSelector` object of the specified `type` based on S-Function block `id` and sets the `Type` and `Id` properties.

`sel = slcoverage.SFcnSelector(type,id,fileName)` creates an `SFcnSelector` object based on the specified C or C++ `fileName`, and sets the `FileName` property.

`sel = slcoverage.SFcnSelector(type,id,fileName,functionName)` creates an `SFcnSelector` object based on the specified C or C++ `functionName` in the specified file and sets the `FunctionName` property.

`sel = slcoverage.SFcnSelector(type,id,fileName,functionName,expr,exprIndex)` creates an `SFcnSelector` object based on the specified expression and `exprIndex` and sets the `Expr` and `ExprIndex` properties.

`sel = slcoverage.SFcnSelector(type,id,fileName,functionName,expr,exprIndex,outcomeIndex)` creates an `SFcnSelector` object based on the specified coverage outcome and sets the `OutcomeIndex` property.

`sel = slcoverage.SFcnSelector(type,id,fileName,functionName,expr,exprIndex,outcomeIndex,parentExprIndex)` creates an `SFcnSelector` object based on the specified coverage outcome whose parent is another expression specified by `parentExprIndex` and sets the `DecOrCondIndex` property.

Properties

Type — Type of S-function
`slcoverage.SFcnSelectorType` enumeration

Type of S-function to select, specified as an enumeration of the `slcoverage.SFcnSelectorType` class:

- `slcoverage.SFcnSelectorType.SFcnName` selects the specified S-function.

- `slcoverage.SFcnSelectorType.SFcnInstanceCppFileName` selects the generated code file for this block. Use this input argument with the `fileName` argument.
- `slcoverage.SFcnSelectorType.SFcnInstanceCppFunction` selects an instance of a C or C ++ function. Use this input argument with the `fileName` and `functionName` arguments.
- `slcoverage.SFcnSelectorType.SFcnInstanceCppCondition` selects a condition in the specified code. Use this input argument with `fileName`, `functionName`, `expr`, and `exprIndex` arguments.
- `slcoverage.SFcnSelectorType.SFcnInstanceCppDecision` selects a decision in the specified code. Use this input argument with `fileName`, `functionName`, `expr`, and `exprIndex` arguments.
- `slcoverage.SFcnSelectorType.SFcnInstanceCppDecisionOutcome` selects a decision outcome in the specified code. Use this input argument with `fileName`, `functionName`, `expr`, `exprIndex`, and `outcomeIndex` arguments.
- `slcoverage.SFcnSelectorType.SFcnInstanceCppConditionOutcome` selects a condition outcome in the specified code. Use this input argument with `fileName`, `functionName`, `expr`, `exprIndex`, and `outcomeIndex` arguments. If the expression that owns the outcome is owned by a parent decision, also use the `parentExprIndex` argument.
- `slcoverage.SFcnSelectorType.SFcnInstanceCppMCDCOutcome` selects an MCDC outcome in the specified code. Use this input argument with `fileName`, `functionName`, `expr`, `exprIndex`, and `outcomeIndex` arguments.
- `slcoverage.SFcnSelectorType.SFcnInstanceCppRelationalBoundaryOutcome` selects a relational boundary outcome in the specified code. Use this input argument with the `fileName`, `functionName`, `expr`, `exprIndex`, and `outcomeIndex` arguments. If the relational operator outcome is owned by a condition that is inside a decision, also use the `parentExprIndex` argument.

Data Types: `slcoverage.SFcnSelectorType`

Id — Model element identifier
Simulink ID (default) | property | handle

Model element identifier, specified as the property name of the element, the handle to an element, or the Simulink identifier of the element. Use a handle or ID for selector types that select an instance. Use a property name, such as the value of the `'BlockType'` property of a block, to select multiple model elements.

Attributes

`SetAccess`	`protected`

Data Types: `char` | `string` | `handle` | `integer`

`FileName` — C or C++ file to select
`character array` | `string array`

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

Example: `'myfile.c'`

Attributes:

`SetAccess`	`protected`

Data Types: `char` | `string`

`FunctionName` — C or C++ function to select
`character array` | `string array`

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

Example: `'counterbusFcn'`

Attributes:

`SetAccess`	`protected`

Data Types: `char` | `string`

`Expr` — Expression to select
`character array` | `string array`

Expression to select, specified as a character array or string array.

Example: `'inputGElower'`

Attributes:

`SetAccess`	`protected`

Data Types: `char` | `string`

`ExprIndex` — Expression index
scalar

Expression index, specified as an integer. If you are filtering an outcome, this property is the index of the expression that owns that outcome. If you are filtering an expression, this property is the index of that expression inside the body of the function.

Example: 2

Attributes:

`SetAccess`	`protected`

Data Types: `single` | `double` | `int`

`OutcomeIndex` — Index of outcome to select
integer

Index of outcome to select, specified as an integer:

- For a Boolean expression, enter `1` for the `F` outcome or `2` for the `T` outcome.
- For a switch/case statement, enter `1` for the first case, `2` for the second case, and so on.
- For relational boundary
  - Integer type:
    - enter `1` for type `-1`.
    - enter `2` for type `+1`.

    - enter `3` for type `0`.
- Float type:
    - enter `1` for `[-tol 0]` or `[-tol 0)`.
    - enter `2` for `(0 tol]` or `[0 tol]` is outcome `2`

For more information about relational boundary coverage, see “Relational Boundary Coverage”.

Example: 2

Attributes

`SetAccess`	`protected`

Data Types: `single` | `double` | `int`

`DecOrCondIndex` — Parent expression index
integer

Parent expression index, specified as an integer. Use this input to filter an expression that is owned by a parent decision or condition; this is the index of the parent decision or condition relative to the function.

Example: 2

Attributes

`SetAccess`	`protected`

Data Types: `single` | `double` | `int`

`Description` — Description of the selector
character vector

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

Attributes:

`GetAccess`	`public`
`SetAccess`	`protected`

`ConstructorCode` — Code used to create this selector object
character array

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

Attributes:

`GetAccess`	`public`
`SetAccess`	`protected`

Methods

Public Methods

allSelectors	Selectors for model or code element

Examples

Create an S-Function Selector

This example shows how to create an S-Function selector.

Load the model by using `load_system`.

```
load_system('slvnvdemo_covfilt');
```

Create an S-Function selector by using `slcoverage.SFcnSelector`. To select the S-Function based on its name, enter `slcoverage.SFcnSelectorType.SFcnName` as the first input. The second input is the path to the S-Function.

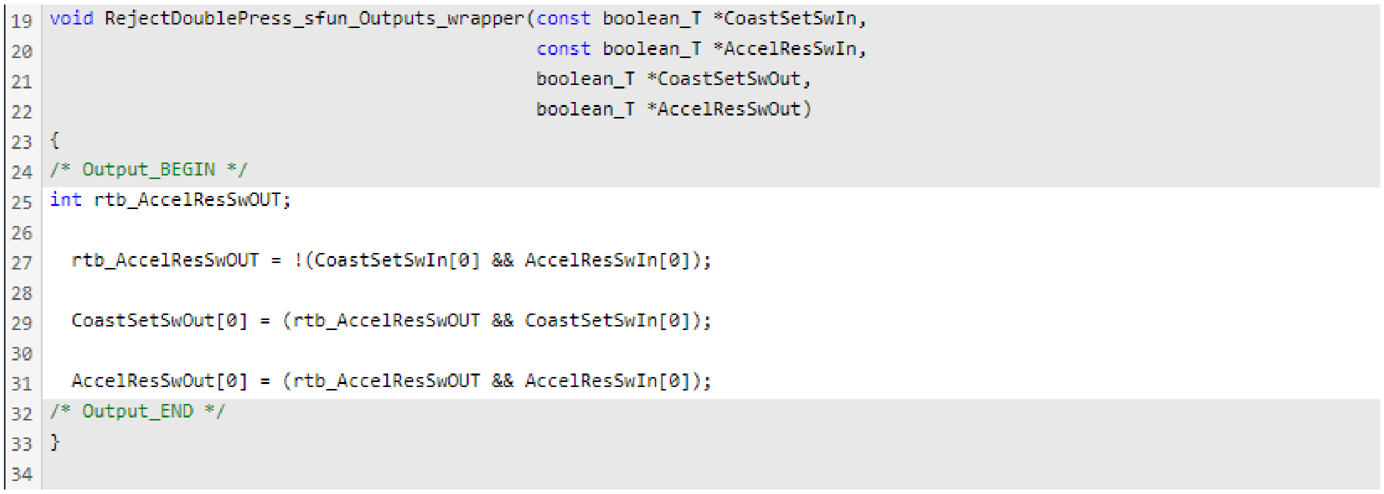
```
sel = slcoverage.SFcnSelector(slcoverage.SFcnSelectorType.SFcnName,...
                         'slvnvdemo_covfilt/Mode Logic/ SFunction')
```

```
sel = 

  SFcnSelector with properties:

           FileName: ''
       FunctionName: ''
               Expr: ''
       OutcomeIndex: []
     DecOrCondIndex: []
        Description: 'N/A'
               Type: SFcnName
                 Id: 'slvnvdemo_covfilt/Mode Logic/ SFunction'
    ConstructorCode: 'slcoverage.SFcnSelector(slcoverage.SFcnSelectorType.SFcnName, 'slvnvdemo_c
```

Create an S-Function Outcome Selector

This example shows how to create a selector for an S-Function using the `slcoverage.SFcnSelector` class.

In this example, your model has an S-Function named `RejectDoublePress_sfun`. Inside the S-Function, suppose that the condition `rtb_AccelResSwOUT` is never false, and you want to justify this false outcome in the coverage report.



```
void RejectDoublePress_sfun_Outputs_wrapper(const boolean_T *CoastSetSwIn,
                                            const boolean_T *AccelResSwIn,
                                            boolean_T *CoastSetSwOut,
                                            boolean_T *AccelResSwOut)
{
/* Output_BEGIN */
int rtb_AccelResSwOUT;

  rtb_AccelResSwOUT = !(CoastSetSwIn[0] && AccelResSwIn[0]);

  CoastSetSwOut[0] = (rtb_AccelResSwOUT && CoastSetSwIn[0]);

  AccelResSwOut[0] = (rtb_AccelResSwOUT && AccelResSwIn[0]);
/* Output_END */
}
```

Load the model.

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

Configure coverage settings using a `Simulink.SimulationInput` object.

```
covSet = Simulink.SimulationInput(modelName);
covSet = covSet.setModelParameter('CovEnable','on');
covSet = covSet.setModelParameter('CovMetricStructuralLevel','ConditionDecision');
covSet = covSet.setModelParameter('CovSFcnEnable','on');
covSet = covSet.setModelParameter('CovSaveSingleToWorkspaceVar','on');
covSet = covSet.setModelParameter('CovSaveName','covData');
covSet = covSet.setModelParameter('SimAnalyzeCustomCode','on');
```

Simulate the model using `covSet` as the input.

```
simOut = sim(covSet);
covData = simOut.covData;
```

```
Generating 'RejectDoublePress_sfun.c' ....Please wait

### Building S-function 'RejectDoublePress_sfun.c' for ex_cc_cruise_control_doublepress_sfun/Reje
mex -R2018a -IC:\TEMP\Bdoc23a_2213998_3568\ib570499\38\tpefa5766f\slcoverage-ex97019589 -DUSE_PUE
Building with 'Microsoft Visual C++ 2019 (C)'.
MEX completed successfully.
mex -R2018a -IC:\TEMP\Bdoc23a_2213998_3568\ib570499\38\tpefa5766f\slcoverage-ex97019589 -DUSE_PUE
Building with 'Microsoft Visual C++ 2019 (C)'.
MEX completed successfully.
```

Create a selector object for the outcome missing coverage by using the `slcoverage.SFcnSelector` class. To select a condition outcome inside an S-Function, use the `SFcnInstanceCppConditionOutcome` enumeration as the first input. The second input is the block ID or block path to the S-Function. The third input is the file name of the source file that contains the expression, `RejectDoublePress_sfun_wrapper.c`. The fourth input is the name of the function that contains the condition, `RejectDoublePress_sfun_Outputs_wrapper`. The fifth input is the expression which owns the outcome, which is `rtb_AccelResSwOUT && CoastSetSwIn[0]`, because this parent decision owns the condition `rtb_AccelResSwOUT`.

The sixth input is the index of the expression that owns the outcome relative to its parent, and because `rtb_AccelResSwOUT` is the first condition in its parent decision, this input is `1`. The seventh input is the outcome index, which is `1` because this is the `F` case of a Boolean expression. The final input is the index of the parent expression relative to the function, and because `rtb_AccelResSwOUT && CoastSetSwIn[0]` is the second decision in the function, this input is `2`.

```
enum = slcoverage.SFcnSelectorType.SFcnInstanceCppConditionOutcome;
SFunID = Simulink.ID.getSID([modelName, '/RejectDoublePress']);
sel = slcoverage.SFcnSelector(enum,SFunID,'RejectDoublePress_sfun_wrapper.c',...
   'RejectDoublePress_sfun_Outputs_wrapper','rtb_AccelResSwOUT && CoastSetSwIn[0]',1,1,2);
```

Create a `Filter` object and a rule based on the selector, then add the rule to the filter.

```
filt = slcoverage.Filter;
rule = slcoverage.FilterRule(sel,'Tested elsewhere',...
           slcoverage.FilterMode.Justify);
addRule(filt,rule);
setFilterName(filt,'S-Function Filter')
```

Save the filter as `sfunfilter` and add it to the cvdata object for `my_func.c` by setting the `filter` property to the filter file name.

```
save(filt,'sfunfilter');
covData.filter = 'sfunfilter';
```

Generate a coverage report.

```
cvhtml('codeCovReport',covData)
```

Review the report. Click the `RejectDoublePress_sfun` link under S-Function Code Coverage Results to see the filtered outcome under Objects Filtered from Coverage Analysis.

Version History

Introduced in R2017b

R2021a: Justify unsatisfied code coverage outcomes

You can use the new `slcoverage.SFcnSelectorType` enumerations to justify code coverage outcomes without excluding the entire expression:

- `slcoverage.SFcnSelectorType.SFcnInstanceCppDecisionOutcome`
- `slcoverage.SFcnSelectorType.SFcnInstanceCppConditionOutcome`
- `slcoverage.SFcnSelectorType.SFcnInstanceCppMCDCOutcome`
- `slcoverage.SFcnSelectorType.SFcnInstanceCppRelationalBoundaryOutcome`

When using the new `slcoverage.SFcnSelectorType` enumerations, use these new properties as optional input arguments:

- `outcomeIndex`

- `DecOrCondIndex`

See Also

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

Topics

"Top-Level Model Coverage Report"
"Create, Edit, and View Coverage Filter Rules"

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 = 'slcoverage_lct_bus';
open_system(modelName);
set_param(modelName,'CovMetricStructuralLevel','MCDC','RecordCoverage','on');
```

Create a `BlockSelector` object, `bl`. This block selector selects all blocks in the model with the property `'RelationalOperator'`.

```
bl = slcoverage.BlockSelector(slcoverage.BlockSelectorType.BlockType,'RelationalOperator');
```

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

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

Version History

Introduced in R2017b

See Also

`slcoverage.Filter` | `slcoverage.FilterRule` | `slcoverage.MetricSelector` | `slcoverage.BlockSelector` | `slcoverage.SFcnSelector` | `removeRule`

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.

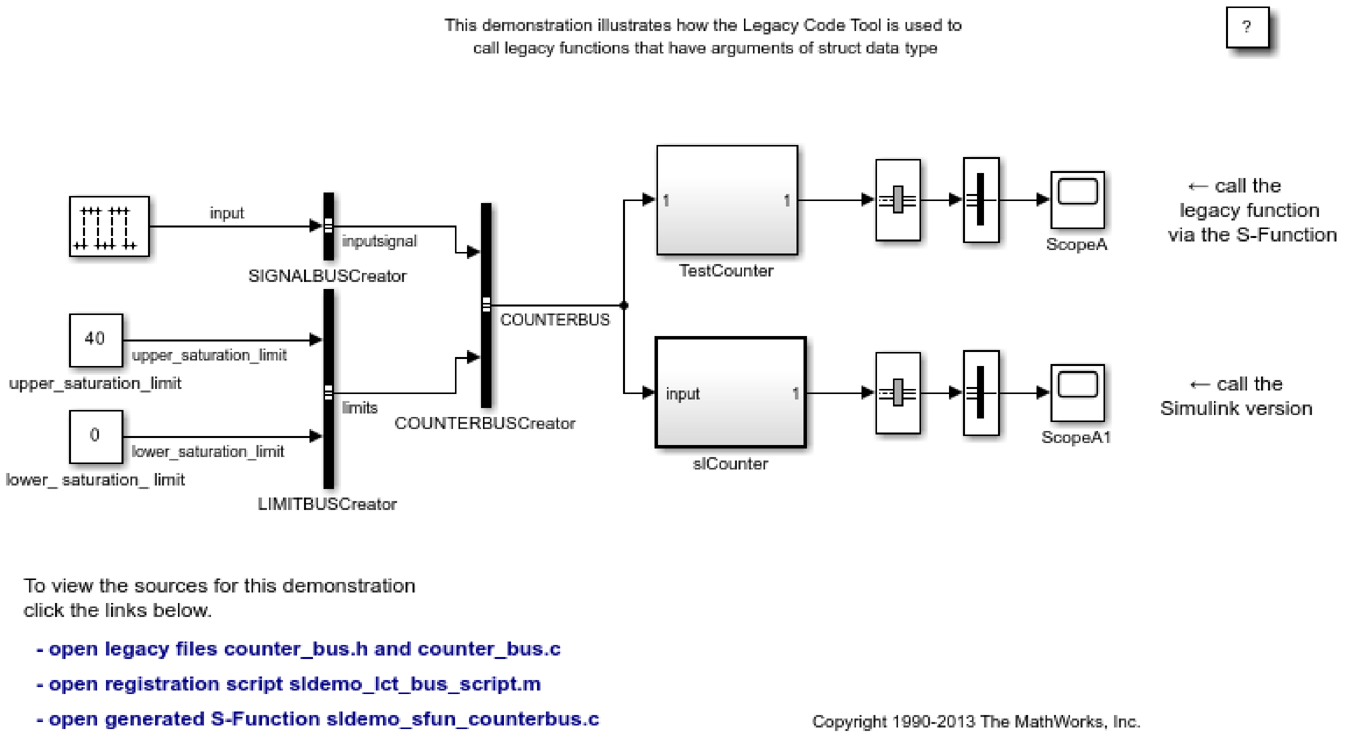
Examples

Remove Rules from Filter Objects

This example shows how to remove a rule from a coverage filter programmatically.

Open the model and specify coverage settings.

```
modelName = 'slcoverage_lct_bus';
open_system(modelName);
set_param(modelName,'CovMetricStructuralLevel','MCDC',...
                    'RecordCoverage','on');
```


This demonstration illustrates how the Legacy Code Tool is used to
call legacy functions that have arguments of struct data type
?
input
inputsignal
SIGNALBUSCreator
40
upper_saturation_limit
upper_saturation_limit
0
lower_saturation_limit
lower_ saturation_ limit
limits
LIMITBUSCreator
COUNTERBUS
COUNTERBUSCreator
TestCounter
input
slCounter
ScopeA
ScopeA1
← call the
legacy function
via the S-Function
← call the
Simulink version
To view the sources for this demonstration
click the links below.
- open legacy files counter_bus.h and counter_bus.c
- open registration script sldemo_lct_bus_script.m
- open generated S-Function sldemo_sfun_counterbus.c
Copyright 1990-2013 The MathWorks, Inc.

Create two `BlockSelector` objects, `blockSel1` and `blockSel2`.

```
blockSel1 = slcoverage.BlockSelector(...
            slcoverage.BlockSelectorType.BlockType,'RelationalOperator');
blockSel2 = slcoverage.BlockSelector(...
            slcoverage.BlockSelectorType.BlockInstance,...
            'slcoverage_lct_bus/slCounter/And');
```

Create a filter object, create two rule objects, and add each rule to the filter object.

```
filt = slcoverage.Filter;
rule1 = slcoverage.FilterRule(blockSel1,'Tested elsewhere',...
                              slcoverage.FilterMode.Exclude);
rule2 = slcoverage.FilterRule(blockSel2,'Value is never greater than 0');
filt.addRule(rule1);
filt.addRule(rule2);
```

Review the rules. Look the first rule in the array.

```
filtRules = filt.rules
filtRules(1)
```

```
filtRules = 

  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(rule1);
filtRules = filt.rules
```

```
filtRules = 

  FilterRule with properties:

     Selector: [1x1 slcoverage.BlockSelector]
         Mode: Justify
    Rationale: 'Value is never greater than 0'
```

Version History
Introduced in R2017b

See Also
`rules` | `slcoverage.Filter` | `addRule` | `slcoverage.FilterRule`

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 = 'slcoverage_lct_bus';
open_system(modelName);
set_param(modelName,'CovMetricStructuralLevel','MCDC','RecordCoverage','on');
```

Create a `BlockSelector` object, `bl`. Create a filter object, create a rule, and add the rule to the filter.

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

Create another rule and add it to the filter object.

```
id = Simulink.ID.getSID('slcoverage_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'
```

Version History

Introduced in R2017b

See Also

`slcoverage.Filter` | `addRule` | `slcoverage.FilterRule` | `removeRule`

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

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

Example: `'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.BlockSelectorTypeslcoverage.CodeSelectorType`, `slcoverage.MetricSelectorType`, or `slcoverage.SFcnSelectorType` values. Possible values:

- Block selector types:
  - `slcoverage.BlockSelectorType.BlockInstance` — An instance of a block or an external MATLAB function called by a MATLAB function block.
  - `slcoverage.BlockSelectorType.BlockType` — All blocks of the specified block type.
  - `slcoverage.BlockSelectorType.Chart` — A Stateflow chart.
  - `slcoverage.BlockSelectorType.MaskType` — Blocks that use the specified mask type.
  - `slcoverage.BlockSelectorType.State` — A Stateflow state.
  - `slcoverage.BlockSelectorType.StateAllContent` — Stateflow state and its contents.
  - `slcoverage.BlockSelectorType.StateflowFunction` — A Stateflow function.
  - `slcoverage.BlockSelectorType.Subsystem` — A subsystem block.
  - `slcoverage.BlockSelectorType.SubsystemAllContent` — A subsystem and its contents.
  - `slcoverage.BlockSelectorType.TemporalEvent` — A Stateflow temporal event.
  - `slcoverage.BlockSelectorType.Transition` — A Stateflow transition.
- Code selector types:
  - `slcoverage.CodeSelectorType.File` — A custom C or C++ code file name.
  - `slcoverage.CodeSelectorType.Function` — A custom C or C++ code function name.
  - `slcoverage.CodeSelectorType.Decision` — A custom C or C++ code decision.
  - `slcoverage.CodeSelectorType.Condition` — A custom C or C++ code condition.
  - `slcoverage.CodeSelectorType.DecisionOutcome` — A custom C or C++ code decision outcome.
  - `slcoverage.CodeSelectorType.ConditionOutcome` — A custom C or C++ code condition outcome.
  - `slcoverage.CodeSelectorType.MCDCOutcome` — A custom C or C++ code MCDC outcome.
  - `slcoverage.CodeSelectorType.RelationalBoundaryOutcome` — A custom C or C++ code relational boundary outcome.

- Metric selector types:
  - `slcoverage.MetricSelectorType.ConditionOutcome` objects select condition metric objective outcomes.
  - `slcoverage.MetricSelectorType.DecisionOutcome` objects select decision metric objective outcomes.
  - `slcoverage.MetricSelectorType.MCDCOutcome` objects select MCDC metric objective outcomes.
  - `slcoverage.MetricSelectorType.RelationalBoundaryOutcome` objects select outcome metrics related to relational boundary outcomes.
  - `slcoverage.MetricSelectorType.SaturationOverflowOutcome` objects select outcome metrics related to saturation on integer overflow outcomes.
- S-function selector types:
  - `slcoverage.SFcnSelectorType.SFcnName` selects the specified S-function.
  - `slcoverage.SFcnSelectorType.SFcnInstanceCppFileName` selects the coverage data in the generated code file for this block.
  - `slcoverage.SFcnSelectorType.SFcnInstanceCppFunction` selects a function.
  - `slcoverage.SFcnSelectorType.SFcnInstanceCppCondition` selects a condition outcome of the S-function block.
  - `slcoverage.SFcnSelectorType.SFcnInstanceCppDecision` selects a decision outcome of the S-function block.

`Description` — Description text to match

character vector or string

Description text to match for the selector that you want to return, specified as a character vector or string. For example, if you want to return only the selectors that include the text `F outcome` in the description, use this syntax:

```
s = slcoverage.Selector.allSelectors(id,'Description','F outcome')
```

`SimulationMode` — Simulation mode

character vector or string

Simulation mode to run when selecting code filters, entered as one of the following:

Object Specification	Description
`'normal'` (default)	Extract code selectors for custom code in normal simulation, such as custom code called from a C Caller block or a Stateflow chart.
`'sil'`	Extract code selectors for code generated in Simulation-in-the-Loop (SIL) mode and code selectors for the top model code interface
`'pil'`	Extract code selectors for code generated in Processor-in-the-Loop (PIL) mode and code selectors for the top model code interface

Object Specification	Description
`'xil'`	If SIL-mode code exists, extract code selectors for code generated in SIL mode and extract code selectors for the top model code interface; otherwise, extract code selectors for code generated in PIL mode and extract code selectors for the top model code interface
`'modelrefsil'`	Extract code selectors for the model reference code interface in SIL mode
`'modelrefpil'`	Extract code selectors for the model reference code interface in PIL mode
`'modelrefxil'`	If SIL-mode code exists, extract code selectors for the model reference code interface in SIL mode, if the model is in SIL mode; otherwise,extract code selectors for the model reference code interface in PIL mode

Output Arguments

`sel` — Selectors for the model or code element
array of `Selector` objects

Selectors for the model or code element, returned as an array of `Selector` objects.

Examples

Get All Selectors

This example shows how to get all the selectors for an And block and then add a rule to justify or exclude a selector. Metric selectors can only be justified.

Load the model and set coverage settings

```
modelName = 'slvnvdemo_covfilt';
load_system(modelName);
set_param(modelName,'CovEnable','on','CovMetricStructuralLevel','MCDC');
```

First, get the block handle for the And block.

```
id = getSimulinkBlockHandle([modelName,'/Saturation']);
```

Get the selectors using the block handle.

```
sel = slcoverage.Selector.allSelectors(id)
```

```
sel = 

  1x10 heterogeneous Selector (BlockSelector, MetricSelector) array with properties:

    Description
    Type
    Id
    ConstructorCode
```

The block has ten selectors. You can index into each one to see their contents. In this example, you want to justify the sixth selector.

```
sel(6)
```

```
ans = 

  MetricSelector with properties:

     ObjectiveIndex: 2
       OutcomeIndex: 2
        Description: 'T outcome of input > upper limit in Saturate block "Saturation"'
               Type: DecisionOutcome
                 Id: 'slvnvdemo_covfilt:5'
    ConstructorCode: 'slcoverage.MetricSelector(slcoverage.MetricSelectorType.DecisionOutcome, '
```

Create a justify rule, then create a filter object and add the rule to it.

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

Save the filter and generate a coverage report.

```
filt.save('metrfilter');
csim = cvsim(modelName);
csim.filter = 'metrfilter';
cvhtml('cov',csim,'-sRT=0');
```

Get Selector by Type and Description

This example shows how to get a selector by type and description. In this example, you get all selectors for the False outcomes of the And block whose type is `ConditionOutcome`.

Load the model into memory.

```
modelName = 'slcoverage_lct_bus';
load_system(modelName);
load slcoverage_lct_data.mat
```

The slcoverage_lct_bus model has an S-Function. Build the S-Function by using legacy_code.

```
def = legacy_code('initialize');
def.SFunctionName = 'slcoverage_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.Options.supportCoverage = true;
legacy_code('generate_for_sim', def);
```

```
### Start Compiling slcoverage_sfun_counterbus
```

```
mex -IC:\TEMP\Bdoc23a_2213998_3568\ib570499\38\tpefa5766f\slcoverage-ex13162396 -c C:\TEMP\Bdoc23
Building with 'Microsoft Visual C++ 2019 (C)'.
MEX completed successfully.
mex -IC:\TEMP\Bdoc23a_2213998_3568\ib570499\38\tpefa5766f\slcoverage-ex13162396 C:\TEMP\Bdoc23a_2
Building with 'Microsoft Visual C++ 2019 (C)'.
MEX completed successfully.
mex -IC:\TEMP\Bdoc23a_2213998_3568\ib570499\38\tpefa5766f\slcoverage-ex13162396 -c C:\TEMP\Bdoc23
Building with 'Microsoft Visual C++ 2019 (C)'.
MEX completed successfully.
mex -IC:\TEMP\Bdoc23a_2213998_3568\ib570499\38\tpefa5766f\slcoverage-ex13162396 C:\TEMP\Bdoc23a_2
Building with 'Microsoft Visual C++ 2019 (C)'.
MEX completed successfully.
### Finish Compiling slcoverage_sfun_counterbus
### Exit
```

Get the false outcome condition selectors for the And block by searching for descriptions that include F.

```
id = getSimulinkBlockHandle([modelName,'/slCounter/And']);
sel = slcoverage.Selector.allSelectors(id, ...
         'Type',slcoverage.MetricSelectorType.ConditionOutcome, ...
         'Description','F')
```

```
sel = 

  1x2 MetricSelector array with properties:

    ObjectiveIndex
    OutcomeIndex
    Description
    Type
    Id
    ConstructorCode
```

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

```
sel.ConstructorCode
```

```
ans = 

    'slcoverage.MetricSelector(slcoverage.MetricSelectorType.ConditionOutcome, 'slcoverage_lct_bu


ans = 

    'slcoverage.MetricSelector(slcoverage.MetricSelectorType.ConditionOutcome, 'slcoverage_lct_bu
```

Version History

Introduced in R2017b

See Also

`slcoverage.MetricSelector` | `slcoverage.BlockSelector` | `slcoverage.CodeSelector` | `slcoverage.SFcnSelector` | `slcoverage.Selector`

setFilterName

Class: `slcoverage.Filter`
Package: `slcoverage`

Set name of coverage filter object

Syntax

```
setFilterName(filterObj,filterName)
```

Description

`setFilterName(filterObj,filterName)` sets the name of the filter to the specified value.

Input Arguments

`filterObj` — Coverage filter
`slcoverage.Filter` object

Coverage filter, specified as an `slcoverage.Filter` object.

Data Types: `slcoverage.Filter`

`filterName` — Coverage filter name
`character array` | `string array`

Coverage filter name, specified as a character array or string array.

Data Types: `char` | `string`

Examples

Edit and View Coverage Filter Details

This example shows how to use the `slcoverage.Filter` methods to set and get filter names and descriptions.

Create a new filter object by using the `slcoverage.Filter` class.

```
filt = slcoverage.Filter;
```

Set the filter name and description by using `setFilterName` and `setFilterDescription`, respectively.

```
setFilterName(filt,'myCovFilter');
setFilterDescription(filt,'Justify missing coverage for unreachable outcomes');
```

Get the filter name and description by using `filterName` and `filterDescription` with the filter object as the input.

```
filtName = filterName(filt)
filtDescr = filterDescription(filt)
```

```
filtName =

    'myCovFilter'


filtDescr =

    'Justify missing coverage for unreachable outcomes'
```

Alternatives

You can also create, edit, and view filters in Simulink. See “Create, Edit, and View Coverage Filter Rules” for more information.

Version History

Introduced in R2020a

See Also

`slcoverage.Filter` | `addRule` | `removeRule` | `rules` | `filterName` | `setFilterDescription` | `filterDescription` | `save`

Topics

“Filter Coverage Results Using a Script”

filterName

Class: `slcoverage.Filter`
Package: `slcoverage`

Get name of coverage filter object

Syntax

```
filtName = filterName(filterObj)
```

Description

`filtName = filterName(filterObj)` returns the name of the specified coverage filter.

Input Arguments

`filterObj` — Coverage filter
`slcoverage.Filter` object

Coverage filter, specified as an `slcoverage.Filter` object.

Data Types: `slcoverage.Filter`

Output Arguments

`filtName` — Filter name
`character array` | `string array`

Filter name, returned as a character array or string array.

Data Types: `char` | `string`

Examples

Edit and View Coverage Filter Details

This example shows how to use the `slcoverage.Filter` methods to set and get filter names and descriptions.

Create a new filter object by using the `slcoverage.Filter` class.

```
filt = slcoverage.Filter;
```

Set the filter name and description by using `setFilterName` and `setFilterDescription`, respectively.

```
setFilterName(filt,'myCovFilter');
setFilterDescription(filt,'Justify missing coverage for unreachable outcomes');
```

Get the filter name and description by using `filterName` and `filterDescription` with the filter object as the input.

```
filtName = filterName(filt)
filtDescr = filterDescription(filt)
```

```
filtName =

    'myCovFilter'


filtDescr =

    'Justify missing coverage for unreachable outcomes'
```

Alternatives

You can also create, edit, and view filters in Simulink. See “Create, Edit, and View Coverage Filter Rules” for more information.

Version History

Introduced in R2020a

See Also

`slcoverage.Filter` | `addRule` | `removeRule` | `rules` | `setFilterName` | `setFilterDescription` | `filterDescription` | `save`

Topics

“Filter Coverage Results Using a Script”

setFilterDescription

Class: `slcoverage.Filter`
Package: `slcoverage`

Set description of coverage filter object

Syntax

```
setFilterDescription(filterObj,descr)
```

Description

`setFilterDescription(filterObj,descr)` sets the description of the filter to the specified value.

Input Arguments

`filterObj` — Coverage filter
`slcoverage.Filter` object

Coverage filter, specified as an `slcoverage.Filter` object.

Data Types: `slcoverage.Filter`

`descr` — Coverage filter description
`character array` | `string array`

Coverage filter description, specified as a character array or string array.

Data Types: `char` | `string`

Examples

Edit and View Coverage Filter Details

This example shows how to use the `slcoverage.Filter` methods to set and get filter names and descriptions.

Create a new filter object by using the `slcoverage.Filter` class.

```
filt = slcoverage.Filter;
```

Set the filter name and description by using `setFilterName` and `setFilterDescription`, respectively.

```
setFilterName(filt,'myCovFilter');
setFilterDescription(filt,'Justify missing coverage for unreachable outcomes');
```

Get the filter name and description by using `filterName` and `filterDescription` with the filter object as the input.

```
filtName = filterName(filt)
filtDescr = filterDescription(filt)
```

```
filtName =

    'myCovFilter'


filtDescr =

    'Justify missing coverage for unreachable outcomes'
```

Alternatives

You can also create, edit, and view filters in Simulink. See “Create, Edit, and View Coverage Filter Rules” for more information.

Version History

Introduced in R2020a

See Also

`slcoverage.Filter` | `addRule` | `removeRule` | `rules` | `setFilterName` | `filterName` | `filterDescription` | `save`

Topics

“Filter Coverage Results Using a Script”

filterDescription

Class: `slcoverage.Filter`
Package: `slcoverage`

Get description of coverage filter object

Syntax

```
descr = filterDescription(filterObj)
```

Description

`descr = filterDescription(filterObj)` returns the description of the specified coverage filter.

Input Arguments

`filterObj` — Coverage filter
`slcoverage.Filter`

Coverage filter, specified as an `slcoverage.Filter` object.

Data Types: `slcoverage.Filter`

Output Arguments

`descr` — Filter description
`character array` | `string array`

Filter description, returned as a character array or string array.

Data Types: `char` | `string`

Examples

Edit and View Coverage Filter Details

This example shows how to use the `slcoverage.Filter` methods to set and get filter names and descriptions.

Create a new filter object by using the `slcoverage.Filter` class.

```
filt = slcoverage.Filter;
```

Set the filter name and description by using `setFilterName` and `setFilterDescription`, respectively.

```
setFilterName(filt,'myCovFilter');
setFilterDescription(filt,'Justify missing coverage for unreachable outcomes');
```

Get the filter name and description by using `filterName` and `filterDescription` with the filter object as the input.

```
filtName = filterName(filt)
filtDescr = filterDescription(filt)
```

```
filtName =

    'myCovFilter'


filtDescr =

    'Justify missing coverage for unreachable outcomes'
```

Alternatives

You can also create, edit, and view filters in Simulink. See “Create, Edit, and View Coverage Filter Rules” for more information.

Version History

Introduced in R2020a

See Also

`slcoverage.Filter` | `addRule` | `removeRule` | `rules` | `setFilterName` | `filterName` | `setFilterDescription` | `save`

Topics

“Filter Coverage Results Using a Script”

save

Class: `slcoverage.Filter`
Package: `slcoverage`

Save coverage filter object to coverage filter file

Syntax

```
save(filterObj,fileName)
```

Description

`save(filterObj,fileName)` saves the specified filter object with the specified file name. The generated file will have the `.cvf` extension.

Input Arguments

`filterObj` — Coverage filter
`slcoverage.Filter` object

Coverage filter, specified as an `slcoverage.Filter` object.

Data Types: `slcoverage.Filter`

`fileName` — File name
`character array` | `string array`

File name, specified as a character array or string array.

Data Types: `char` | `string`

Examples

Create and Save a Coverage Filter

Create a filter object by using the `slcoverage.Filter` class, then set the filter name to `myCovFilter` using `setFilterName`.

```
filt = slcoverage.Filter;
setFilterName(filt,'myCovFilter')
```

Save the filter as `myCovFilter.cvf` by using `save`.

```
save(filt,'myCovFilter')
```

Alternatives

You can also create and save filters in Simulink. See “Create, Edit, and View Coverage Filter Rules” for more information.

Version History

Introduced in R2020a

See Also

`slcoverage.Filter` | `addRule` | `removeRule` | `rules` | `setFilterName` | `filterName` | `setFilterDescription` | `filterDescription`

Topics

"Filter Coverage Results Using a Script"

cvdata

Access Simulink Coverage data in the MATLAB workspace

Description

`cvdata` objects store model coverage data.

Creation

A `cvdata` object is generated automatically when you simulate a model that has coverage enabled. You can access the coverage data by using coverage functions. The `cvdata` object becomes invalid if you close or modify its parent model.

Properties

`dbVersion` — Coverage data origin release
`character array`

This property is read-only.

Coverage data origin release, returned as a character array.

Data Types: `char`

`id` — Internal coverage data ID
`scalar`

This property is read-only.

Internal coverage data ID, returned as a scalar.

Data Types: `double`

`type` — Internal coverage data type
`TEST_DATA` | `DERIVED_DATA`

This property is read-only.

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

Data Types: `char`

`test` — Test data
`cvtest` object

This property is read-only.

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

Data Types: `cvtest`

`rootID` — Internal root ID
`scalar`

This property is read-only.

Internal root ID, returned as a scalar.

Data Types: `double`

`checksum` — Coverage data checksum
`struct`

This property is read-only.

Coverage data checksum, returned as a structure array. Some changes to the model cause the `checksum` to change, for example, editing the model structure like removing or adding blocks, or changing some configuration parameters. To aggregate coverage results, the `cvdata.checksum` properties must match.

Data Types: `struct`

`modelinfo` — Model information
`struct`

This property is read-only.

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

`cvdata.modelinfo` has the following fields:

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

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

Data Types: `struct`

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

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

Data Types: `char`

`stopTime` — System time at simulation stop
`scalar`

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

Data Types: `char`

`intervalStartTime` — Coverage interval start time
`scalar`

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

Data Types: `double`

`intervalStopTime` — Coverage interval stop time
`scalar`

Coverage interval stop time, returned as a scalar. This value comes from the `CovStopTime` parameter. For more information, see “Coverage interval stop time” on page 2-16.

Data Types: `double`

`filter` — Coverage filter file name
`character array` | `cell array`

Coverage filter file name, returned as a character array or a cell array of character arrays. This property contains the coverage filter file name. If a coverage filter is not applied, this field is empty. You can apply a coverage filter after simulation by assigning the name of a valid filter file to this property.

Data Types: `char` | `cell`

`simMode` — Simulation mode
`character array`

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

Data Types: `char`

`excludeInactiveVariants` — Whether to exclude inactive variants in coverage report
`0` (default) | `1`

Whether to exclude inactive variants in coverage report, specified as `0` or `1`. Set this property to `0` to report inactive variants in the coverage report, and set this property to `1` to exclude inactive variants from the coverage report.

This property affects only Simulink variants with startup activation time and inactive variant configurations in Stateflow charts.

If you aggregate two or more `cvdata` objects with different values of `excludeInactiveVariants`, the resulting aggregated `cvdata` object has an `excludeInactiveVariants` property value of `0`.

Example: `covData.excludeInactiveVariants = 1;`

Data Types: `double`

Object Functions

View and Save Coverage Results

cvhtml	Create HTML coverage report from model coverage objects
cvmodelview	Display model coverage results with model highlighting
cvresults	Returns active coverage data, clears and loads active coverage data from a file
cvsave	Save coverage settings and results to file

Extract Coverage Results from `cvdata` Objects

extract	Extract subsystem coverage data from system-level coverage data
complexityinfo	Retrieve cyclomatic complexity coverage information from cvdata object
conditioninfo	Retrieve condition coverage information from cvdata object
decisioninfo	Retrieve decision coverage information from cvdata object
executioninfo	Retrieve execution coverage information from cvdata object
getCoverageInfo	Retrieve coverage information for Simulink Design Verifier blocks from cvdata object
mcdcinfo	Retrieve modified condition/decision coverage information from cvdata object
overflowsaturationinfo	Retrieve saturation on integer overflow coverage from cvdata object
relationalboundaryinfo	Retrieve relational boundary coverage from cvdata object
sigrangeinfo	Retrieve signal range coverage information from cvdata object
sigsizeinfo	Retrieve signal size coverage information from cvdata object
tableinfo	Retrieve lookup table coverage information from cvdata object

You can use these specialized MATLAB operators with `cvdata` objects.

`+`	Returns the union of two `cvdata` objects. The returned `cvdata` object contains the total aggregated coverage for two simulations or tests.
`-`	Returns the set difference between the left and right operands. The returned `cvdata` object contains the coverage objective outcomes that are satisfied by the left operand but not the right operand. Use `-` to determine how much additional coverage is attributed to a specific simulation or test.
`* and .*`	Returns the intersection between two `cvdata` objects. The returned `cvdata` object contains only the coverage satisfied by both operands. Use `*` to detect overlapping coverage between two simulations or tests.

Examples

View Decision Coverage Data

This example shows how to view decision coverage data for a block in your model.

Load the model.

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

Configure the coverage settings for the model by using a `Simulink.SimulationInput` object.

```
simIn = Simulink.SimulationInput(modelName);
simIn = setModelParameter(simIn,'CovEnable','on');
simIn = setModelParameter(simIn,'CovMetricStructuralLevel','Decision');
simIn = setModelParameter(simIn,'CovSaveSingleToWorkspaceVar','on');
simIn = setModelParameter(simIn,'CovSaveName','covData');
```

Simulate the model by passing `simIn` as the input to `sim`.

```
simOut = sim(simIn);
```

Extract the coverage data as a `cvdata` object from the `SimulationOutput` object, `simOut`.

```
covData = simOut.covData;
```

View the decision coverage results for the Saturation block by calling `decisioninfo` with the block path.

```
blockPath = [modelName,'/Saturation'];
decisionCov = decisioninfo(covData,blockPath)
```

```
decisionCov =

     3     4
```

`decisioninfo` returns an array with two scalars. The first value is the number of satisfied decision outcomes, and the second value is the number of total decision outcomes. Use these to determine the percentage of satisfied decision outcomes for the Saturation block.

```
percentDecisionCov = 100 * decisionCov(1) / decisionCov(2)
```

```
percentDecisionCov =

    75
```

Version History

Introduced before R2006a

See Also

`cvsim` | `cv.cvdatagroup` | `sim`

Topics

"Automating Model Coverage Tasks"
"Perform Operations on Coverage Data"
"Analyze Coverage Data Using A Script"
"Record Coverage in Parallel Simulations by Using Parsim"