

# Simulink® Report Generator™

User's Guide



# MATLAB® & SIMULINK®

R2020b



## How to Contact MathWorks



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



Phone: 508-647-7000



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

*Simulink® Report Generator™ User's Guide*

© COPYRIGHT 1999–2020 by The MathWorks, Inc.

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

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

### Trademarks

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

### Patents

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

## Revision History

January 1999	First printing	New (Release 11)
December 2000	Second printing	Revised (Release 12)
June 2004	Third printing	Revised for Version 2.02 (Release 14)
August 2004	Online only	Revised for Version 2.1
October 2004	Online only	Revised for Version 2.1.1 (Release 14SP1)
December 2004	Online only	Revised for Version 2.2 (Release 14SP1+)
April 2005	Online only	Revised for Version 2.2.1 (Release 14SP2+)
September 2005	Online only	Revised for Version 2.3.1 (Release 14SP3)
March 2006	Online only	Revised for Version 3.0 (Release 2006a)
September 2006	Online only	Revised for Version 3.1 (Release 2006b)
March 2007	Fourth printing	Revised for Version 3.2 (Release 2007a)
September 2007	Fifth printing	Revised for Version 3.2.1 (Release 2007b)
		This publication was previously for MATLAB® and Simulink®. It is now for Simulink® only.
March 2008	Online only	Revised for Version 3.3 (Release 2008a)
October 2008	Online only	Revised for Version 3.4 (Release 2008b)
October 2008	Online only	Revised for Version 3.5 (Release 2008b+)
March 2009	Online only	Revised for Version 3.6 (Release 2009a)
September 2009	Online only	Revised for Version 3.7 (Release 2009b)
March 2010	Online only	Revised for Version 3.8 (Release 2010a)
September 2010	Online only	Revised for Version 3.9 (Release 2010b)
April 2011	Online only	Revised for Version 3.10 (Release 2011a)
September 2011	Online only	Revised for Version 3.11 (Release 2011b)
March 2012	Online only	Revised for Version 3.12 (Release 2012a)
September 2012	Online only	Revised for Version 3.13 (Release 2012b)
March 2013	Online only	Revised for Version 3.14 (Release 2013a)
September 2013	Online only	Revised for Version 3.15 (Release 2013b)
March 2014	Online only	Revised for Version 3.16 (Release 2014a)
October 2014	Online only	Revised for Version 4.0 (Release 2014b)
March 2015	Online only	Revised for Version 4.1 (Release 2015a)
September 2015	Online only	Revised for Version 4.2 (Release 2015b)
October 2015	Online only	Rereleased for Version 4.1.1 (Release 2015aSP1)
March 2016	Online only	Revised for Version 5.0 (Release 2016a)
September 2016	Online only	Revised for Version 5.1 (Release 2016b)
March 2017	Online only	Revised for Version 5.2 (Release 2017a)
September 2017	Online only	Revised for Version 5.3 (Release 2017b)
March 2018	Online only	Revised for Version 5.4 (Release 2018a)
September 2018	Online only	Revised for Version 5.5 (Release 2018b)
March 2019	Online only	Revised for Version 5.6 (Release 2019a)
September 2019	Online only	Revised for Version 5.7 (Release 2019b)
March 2020	Online only	Revised for Version 5.8 (Release 2020a)
September 2020	Online only	Revised for Version 5.9 (Release 2020b)



<b>Simulink Report Generator Product Description</b> .....	<b>1-2</b>
Key Features .....	1-2
<b>Relationship Between Simulink Report Generator and MATLAB Report Generator</b> .....	<b>1-3</b>
Finders and Reporters .....	1-3
Web Views and Embedded Web Views .....	1-4
<b>System Design Documentation and Results Reporting</b> .....	<b>1-7</b>
Types of Reports .....	1-7
System Design Documentation .....	1-7
Results Reporting .....	1-7
<b>Report Generation for Simulink and Stateflow Elements</b> .....	<b>1-9</b>
Simulink Report API Classes .....	1-10
Find and Report on Blocks in a Model .....	1-12
Use Specific Finders and Reporters for Different Block Types .....	1-14
Find and Report on Stateflow Elements .....	1-17
<b>Generate Reports Without Customizing</b> .....	<b>1-21</b>
Predefined Standard Reports .....	1-21
Report API .....	1-21
Web View .....	1-21
<b>Report Creation Workflow</b> .....	<b>1-23</b>
<b>Report Components</b> .....	<b>1-24</b>
About Report Components .....	1-24
Report Structure Components .....	1-24
System-Based Components .....	1-24
User-Supplied Information Components .....	1-25
Dynamic Reporting Components .....	1-26
Format Control at the Component Level .....	1-26
<b>Working with the Report Explorer</b> .....	<b>1-27</b>
About the Report Explorer .....	1-27
<b>Acknowledgments</b> .....	<b>1-28</b>

## Generate System Design Description Reports

### 2

<b>System Design Description</b> .....	2-2
Predefined Standard Reports .....	2-2
What Is the System Design Description? .....	2-2
What You Can Do with the Report .....	2-2
Report Contents .....	2-3
<b>Generate a System Design Description Report</b> .....	2-5
<b>Customize the System Design Description</b> .....	2-6
Using the Report Explorer to Customize the Report .....	2-6
Building a Dialog Box for a Custom Report Setup File .....	2-7
<b>Generate a System Design Report with the Report API</b> .....	2-8

## System Design Description

### 3

<b>System Design Description Dialog Box</b> .....	3-2
System Design Description Overview .....	3-2
Title .....	3-2
Subtitle .....	3-3
Authors .....	3-3
Image .....	3-4
Legal Notice .....	3-4
Design details .....	3-4
Model references .....	3-5
Subsystems from custom libraries .....	3-5
Requirements traceability .....	3-6
Glossary and report explanation .....	3-6
File format .....	3-6
Stylesheet or Template .....	3-7
File name .....	3-9
Folder .....	3-10
If report exists, increment name to prevent overwriting .....	3-10
Package type .....	3-10

## Creating Simulink Reports

### 4

<b>Create a Simulink Report Generator Report</b> .....	4-2
<b>Report on MATLAB Function</b> .....	4-13
Find and Report on MATLAB Function Blocks .....	4-13
Find and Report on Stateflow MATLAB Functions .....	4-16
Customize MATLAB Function Reporter Output .....	4-17

<b>Use Simulink Report Explorer Components in a Report API Report . . .</b>	<b>4-20</b>
Create the Report Explorer Setup File . . . . .	4-20
Create a Report Generator Program . . . . .	4-21
<b>Report Systems Hierarchically . . . . .</b>	<b>4-25</b>
<b>Customize Simulink Diagram Hyperlinks in HTML and PDF Reports . .</b>	<b>4-27</b>
<b>Tile Simulink Diagrams . . . . .</b>	<b>4-29</b>
<b>Create a Simulink Bus Object Report . . . . .</b>	<b>4-32</b>
<b>Report System Inputs and Outputs . . . . .</b>	<b>4-34</b>
<b>Reporting on DocBlock Blocks . . . . .</b>	<b>4-37</b>
<b>Report Model Notes . . . . .</b>	<b>4-39</b>
<b>Report Execution Order of Tasks and Blocks in a Simulink System . . . .</b>	<b>4-41</b>
<b>Create a Simulink Report Generator Report Interactively . . . . .</b>	<b>4-50</b>
Specify Report Options in the Setup File . . . . .	4-50
Add Report Content with Components . . . . .	4-51
Generate the Report . . . . .	4-82
<b>Generate a Report Associated with a Model . . . . .</b>	<b>4-85</b>
<b>Logical and Looping Components . . . . .</b>	<b>4-86</b>
<b>Filter with Loop Context Functions . . . . .</b>	<b>4-87</b>
Create and Save the Setup File . . . . .	4-87
Add Components . . . . .	4-87
Run the Report . . . . .	4-88
<b>Loop Context Functions . . . . .</b>	<b>4-89</b>
For Simulink Modeling Elements . . . . .	4-89
For Stateflow Modeling Elements . . . . .	4-89

## Export Simulink Models to Web Views

# 5

<b>Web Views . . . . .</b>	<b>5-2</b>
What Is a Web View? . . . . .	5-2
System Requirements . . . . .	5-2
Web View Files . . . . .	5-2
<b>Export Models to Web View Files . . . . .</b>	<b>5-4</b>
<b>Display and Navigate a Web View . . . . .</b>	<b>5-5</b>
Display a Web View When You Export It . . . . .	5-5
Open a Web View File in a Web Browser . . . . .	5-5

View Contents of a System .....	5-6
View Block Parameters and Signal Properties .....	5-7
Access Optional Web View Information .....	5-7
<b>Search a Web View .....</b>	<b>5-8</b>
Perform a Search .....	5-8
Sort Search Results .....	5-9
Navigate Between Search Results and Model Elements .....	5-10
<b>Create and Use a Web View .....</b>	<b>5-11</b>
About This Tutorial .....	5-11
Export Specific Systems .....	5-11
Navigate the Web View .....	5-13
Display Parameters and Properties of Blocks and Signals .....	5-14
Open the Web View .....	5-16
<b>Include Model Requirements and Coverage Data in a Web View .....</b>	<b>5-17</b>
Prepare the Model for an Optional Web View .....	5-17
Add Optional Views to a Web View .....	5-17
Open an Optional Web View .....	5-17
<b>Embedded Web View Reports .....</b>	<b>5-19</b>
What Is Embedded Web View? .....	5-19
Navigating an Embedded Web View Report .....	5-20
Embedded Web View Packaging .....	5-22
View Embedded Web View Reports .....	5-22
<b>Create an Embedded Web View Report Generator .....</b>	<b>5-23</b>
Create an Embedded Web View Report Generator Class .....	5-23
<b>Specify Export Options for Embedded Web View Report .....</b>	<b>5-24</b>
<b>Specify Document Content for Embedded Web View Report .....</b>	<b>5-25</b>
<b>Generate Table of Contents for Embedded Web View Report .....</b>	<b>5-26</b>
<b>Get Model Objects for Embedded Web View Report .....</b>	<b>5-27</b>
<b>Create Hyperlinks for Embedded Web View Report .....</b>	<b>5-28</b>
<b>Suppress Link Warning Messages for Embedded Web View Report .....</b>	<b>5-33</b>
<b>Generate an Embedded Web View Report .....</b>	<b>5-34</b>
Class Definition File for an Embedded Web View .....	5-34
<b>Web View .....</b>	<b>5-37</b>
Web View Export Dialog Box Overview .....	5-37
Systems to Export .....	5-37
Referenced Models .....	5-38
Library Links .....	5-38
MathWorks Library Links .....	5-38
Masked Subsystems .....	5-38
Package name .....	5-38
Folder .....	5-38
If package exists, increment name to prevent overwriting .....	5-39



Package Type .....	5-39
Include Model Coverage view .....	5-39
Include Embedded Coder view .....	5-39
Include Requirements view .....	5-39
Include Coverage view .....	5-40

## **Components**

**6**

## **Classes**

**7**

## **Functions**

**8**



# Getting Started

---

- “Simulink Report Generator Product Description” on page 1-2
- “Relationship Between Simulink Report Generator and MATLAB Report Generator” on page 1-3
- “System Design Documentation and Results Reporting” on page 1-7
- “Report Generation for Simulink and Stateflow Elements” on page 1-9
- “Generate Reports Without Customizing” on page 1-21
- “Report Creation Workflow” on page 1-23
- “Report Components” on page 1-24
- “Working with the Report Explorer” on page 1-27
- “Acknowledgments” on page 1-28

## **Simulink Report Generator Product Description**

### **Design and automatically generate reports from Simulink models and Stateflow charts**

Simulink Report Generator provides functions and APIs that enable you to include block diagrams, Stateflow® charts, MATLAB® Function blocks, truth tables, data dictionaries, and other model elements in your reports. You can design and generate reports in PDF, Microsoft® Word, Microsoft PowerPoint®, and HTML. You can generate standard reports such as system design descriptions, as well as custom reports containing design artifacts such as generated code, requirements traceability, documentation, and test results. Artifacts for DO-178, ISO 26262, IEC 61508, and related industry standards can also be included.

Simulink Report Generator enables you to create web views that let you view, navigate, and share Simulink models from a web browser without a Simulink license. You can embed model web views in HTML code generation, requirements, coverage, and other types of reports.

### **Key Features**

- Automated reporting from Simulink and Stateflow
- PDF, Microsoft Word, Microsoft PowerPoint, and HTML formats
- Templates for programmatic and forms-based reporting
- Automatic capture of simulation results and model specifications
- Web views for viewing and navigating models in web browsers
- Artifacts for DO-178 and IEC 61508

# Relationship Between Simulink Report Generator and MATLAB Report Generator

Simulink Report Generator extends MATLAB Report Generator by adding the ability to find and report on Simulink block diagrams and elements and Stateflow charts and elements. Simulink Report Generator also provides Web Views and Embedded Web Views of Simulink models.

## Finders and Reporters

The Simulink Report API has finder classes that find and report on Simulink diagrams, subsystems, blocks, annotations, and other elements. It also has finders for Stateflow, which find and report on charts, states, transitions, and other elements. All of the finders are derived from the `mreportgen.finder.Finder` base class. In addition to finders, the Report API has reporter classes, which you use to customize reporting on finder results. See “Report Generation for Simulink and Stateflow Elements” on page 1-9 for more information and a full list of available finders and reporters. In addition the finders and reporters, the MATLAB Report API and DOM API provide many features useful for creating Simulink reports, such as title page, table of contents, and chapter reporters.

This example shows results reported for Simulink blocks by the Report API `BlockFinder` class.

Chapter 1. Blocks in Simulink vdp model	
1.1. Constant	
Table 1.1. vdp/Constant Properties	
Property	Value
Type	Block
Block Type	Constant
Constant value	1
Interpret vector parameters as 1-D	on
Output data type	Inherit: Inherit from 'Constant value'
Lock output data type setting against changes by the fixed-point tools	off
Sample time	inf
Frame period	inf
1.2. More Info	
Table 1.2. vdp/More Info Properties	
Property	Value
Type	System

This example shows results reported for Stateflow transitions by the Report API StateflowDiagramElementFinder class.

**Chapter 2. Transitions**

**Table 2.1. sldemo\_fuelsys/fuel\_rate\_control/control\_logic/Fail/? Properties**

Property	Value
Type	Transition
Destination	None
ExecutionOrder	1
Label	?

**Table 2.2. sldemo\_fuelsys/fuel\_rate\_control/control\_logic/Fail/INC Properties**

Property	Value
Type	Transition
Source	None
Destination	One
ExecutionOrder	1
Label	INC

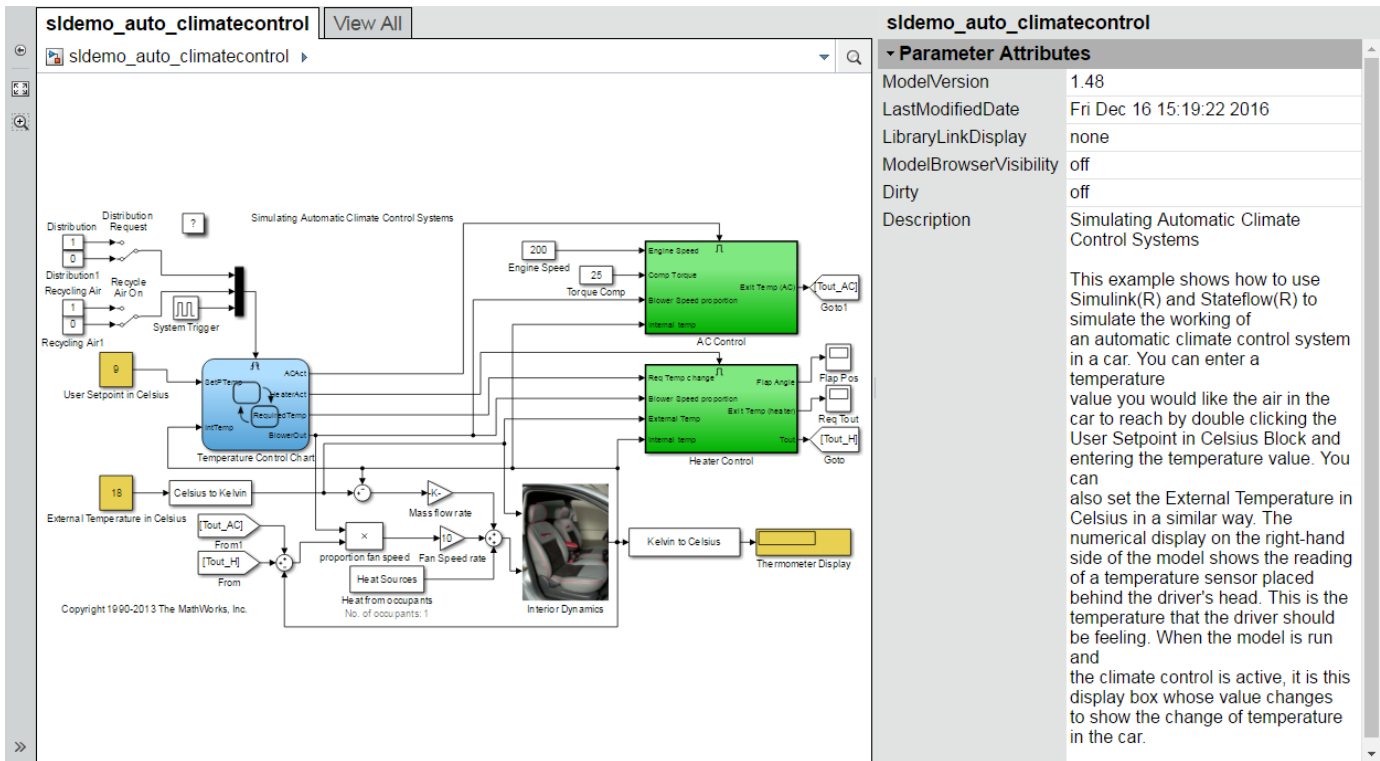
**Table 2.3. sldemo\_fuelsys/fuel\_rate\_control/control\_logic/Fail/INC Properties**

Property	Value
Type	Transition
Source	One
Destination	Two
ExecutionOrder	1
Label	INC

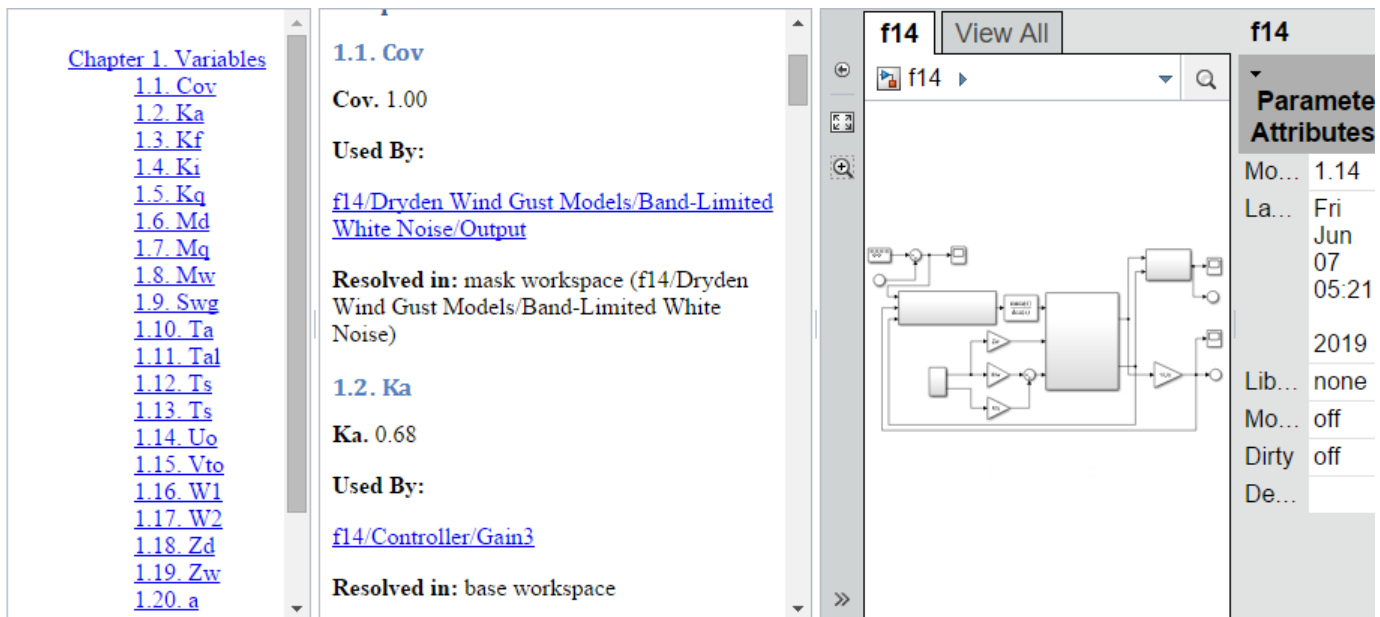
## Web Views and Embedded Web Views

Simulink Report Generator also provides Web Views and Embedded Web Views. A Web View is an interactive rendition of a Simulink model that you can view in a Web browser. An Embedded Web View is an HTML report that contains one or more Web Views.

This example shows a Web View.



This example shows a portion of an Embedded Web View report.



**See Also**  
mlreportgen.finder.Finder

## **More About**

- “Report Generation for Simulink and Stateflow Elements” on page 1-9
- “What Is a Reporter?”
- “Create a Report Generator”
- “Create a Simulink Report Generator Report” on page 4-2
- “Create and Use a Web View” on page 5-11
- “Create an Embedded Web View Report Generator” on page 5-23



# System Design Documentation and Results Reporting

## Types of Reports

Two common goals for creating reports are:

- “System Design Documentation” on page 1-7 — Capture information about the design decisions, structure, implementation, and operational details of a system.
- Results reporting on page 1-7 — Present results of running a system.

You use a similar workflow for creating and generating reports for both goals. However, some components are particularly useful for each use case.

## System Design Documentation

System documentation helps you to:

- Capture design decisions
- Record implementation details
- Communicate the system design and interfaces among groups

When you create a Simulink Report Generator report to provide system design documentation, the report captures information about the system design from the model. Each time that you generate the report, you see up-to-date documentation for the design.

The table includes examples of components that are useful for system design documentation reports.

System Information	Examples of Components to Use
Requirements	Requirements Summary Table (for requirements specified with Simulink Requirements™)
System layout	System Hierarchy, System Snapshot
Model configuration	Model Configuration Set, Model Advisor Report
Block parameter settings	Simulink Dialog Snapshot, Block Loop
Properties	Simulink Property Table, Simulink Summary Table
Variables	Variable Table, Simulink Workspace Variable
System documentation included in a model	Documentation, Simulink Name

## Results Reporting

Capturing results from simulating a model is useful for:

- Model regression testing
- Verifying and validating designs
- Exploring design alternatives
- Optimizing designs

The table includes examples of components that are useful in results reports.

<b>Results Information</b>	<b>Examples of Components to Use</b>
Signal values	Scope Snapshot, Block Loop
Simulation processing	Model Simulation, Model Configuration Set, Fixed Point Logging Options
Figures generated with MATLAB	Figure Snapshot, To Workspace Plot
Generated code	Code Generation Summary, Import Generated Code

You can use components such as the `Model Simulation` component to control how the model simulates. Other components, such as the `Scope Snapshot`, show the results of the simulation.

## Report Generation for Simulink and Stateflow Elements

Finders and reporters reduce the amount of time and complexity required to write code to find and report on Simulink model elements, such as diagrams and blocks, and on Stateflow charts and transitions, etc. The Simulink Report Generator Report API is a layer on top of the DOM API. Its finders and reporters are based on the Simulink and Stateflow find base class. You specify the container in which to find specific elements, such as blocks in a subsystem or states in a chart. Finder objects return their results in a corresponding array of finder result objects.

The Simulink Report API also includes reporter classes, which return an image of the container. This image is the top level of a model.

### Chapter 1. RootSystem

Copyright 1997-2016 The MathWorks, Inc.

### 1.1. Engine

**Table 1.1. sf\_car/Engine Properties**

Property	Value
Type	System

Every result object returned by a finder has an associated reporter object, which reports on those results. The reporter object holds the content and formats the content, such as tables of properties and data plots. You add the reporter objects to your reports. Use the MATLAB Report API reporters to define common report elements. See “What Is a Reporter?” for information.

All finders and reporters have these features:

- Default behaviors and values
- Allow overriding and customizing their output

All finders have `find`, `hasNext`, and `next` methods. The `find` method finds and returns in an array of result objects all elements for each found element of the specified type. The `hasNext` and `next` methods find and return one element at a time and are used to iterate over a list of results. The `hasNext` method checks whether the container has at least one of the element of the specified type. If the container has one or more of the elements, then the `hasNext` method queues it for the `next` method to find and return as a result object.

All reporters have predefined templates. The template for each reporter defines its formatting, layout, and content holes. You do not need to change the template or specify any formats, layouts, or holes unless you want a customized report. You can customize your report by copying and editing its default template or by using a new template. Editing a copy of the default template gives you a starting point and structure to follow to customize your template. Using a new template lets you completely define your template starting with a blank file. To change the order of the report content, reorder the holes in the template. Finders do not use templates. Another way to customize a reporter class is by subclassing it.

The default reporter templates for each output type are in a template library, which is at

```
matlab\toolbox\shared\slreportgen\rpt\rpt\+slreportgen\
+report\<reporter>\resources\templates\<output>
```

For example, the path to the default template for the `DiagramReporter` for PDF output is

```
matlab\toolbox\shared\slreportgen\rpt\rpt\+slreportgen\
+report\@DiagramReporter\resources\templates\pdf\default.pdfxtx
```

For a detailed example of editing a template, see the "Customize a Report API Template" section of "What Is a Reporter?"

## Simulink Report API Classes

The Simulink Report API provides these finder, result, and reporter classes. To use these classes in a report generator program, you must create a container of type `slreportgen.report.Report` to hold the report.

### Finder and Result Classes

Report API Class	Description
<code>slreportgen.finder.AnnotationFinder</code>	Finds Simulink block diagram annotations.
<code>slreportgen.finder.BlockFinder</code>	Finds blocks in a Simulink block diagram.
<code>slreportgen.finder.BlockResult</code>	Contains a block found by a <code>BlockFinder</code> object.
<code>slreportgen.finder.ChartDiagramFinder</code>	Finds Stateflow charts in a model.
<code>slreportgen.finder.DataDictionaryFinder</code>	Finds Simulink data dictionaries.
<code>slreportgen.finder.DataDictionaryResult</code>	Contains a data dictionary found by a <code>DataDictionary</code> object.
<code>slreportgen.finder.DiagramElementFinder</code>	Finds elements of a Simulink block diagram or Stateflow chart.

Report API Class	Description
<code>slreportgen.finder.DiagramElementResult</code>	Contains a diagram element found by a <code>DiagramElementFinder</code> object.
<code>slreportgen.finder.DiagramFinder</code>	Finds block diagrams and charts in a Simulink model.
<code>slreportgen.finder.DiagramResult</code>	Contains a diagram found by a <code>DiagramFinder</code> object.
<code>slreportgen.finder.ModelVariableFinder</code>	Finds variables used by a Simulink model.
<code>slreportgen.finder.ModelVariableResult</code>	Contains a model variable found by a <code>ModelVariableFinder</code> object.
<code>slreportgen.finder.StateFinder</code>	Finds states in a Stateflow chart.
<code>slreportgen.finder.StateflowDiagramElementFinder</code>	Finds elements of a Stateflow chart.
<code>slreportgen.finder.SystemDiagramFinder</code>	Finds system block diagrams in a Simulink model.

### Reporter Classes

Report API Class	Description
<code>slreportgen.report.BusObject</code>	Reports on <code>Simulink.Bus</code> objects use by a model.
<code>slreportgen.report.DataDictionary</code>	Reports on a Simulink data dictionary.
<code>slreportgen.report.Diagram</code>	Creates a snapshot of a Simulink block diagram or a Stateflow chart.
<code>slreportgen.report.DocBlock</code>	Reports on a Simulink <code>DocBlock</code> .
<code>slreportgen.report.ExecutionOrder</code>	Reports on the tasks of a model or nonvirtual subsystem and the blocks in each task, sorted by execution order.
<code>slreportgen.report.LookupTable</code>	Reports on breakpoints and output points of a Simulink lookup table block.
<code>slreportgen.report.MATLABFunction</code>	Reports on a MATLAB Function block or a Stateflow MATLAB function.
<code>slreportgen.report.ModelConfiguration</code>	Reports on the active configuration set of a model.
<code>slreportgen.report.ModelVariable</code>	Reports on a model variable.
<code>slreportgen.report.Notes</code>	Reports on Simulink or Stateflow diagram notes.
<code>slreportgen.report.SimulinkObjectProperties</code>	Creates a table of the properties of a Simulink object.
<code>slreportgen.report.StateflowObjectProperties</code>	Creates a table of the properties of a Stateflow object.
<code>slreportgen.report.SystemHierarchy</code>	Creates a nested list of the subsystems of a Simulink model or subsystem.
<code>slreportgen.report.SystemIO</code>	Reports on Simulink system input and output signals.

Report API Class	Description
<code>slreportgen.report.TestSequence</code>	Reports on a Simulink Test Sequence block.
<code>slreportgen.report.TruthTable</code>	Reports on a Simulink truth table block or a Stateflow truth table object.

## Find and Report on Blocks in a Model

This example shows how to find and report on all Simulink blocks in the `vdp` model using the `BlockFinder` class. The resulting HTML report includes default information and uses default formatting for each block.

- 1 Import the Report API package, which let you use class names without including their package names. For example, you can use `BlockFinder` instead of `slreportgen.finder.BlockFinder`. In addition to importing the Simulink Report API base classes, import the MATLAB Report API base class. A typical report includes a title page, table of contents, chapters, and sections, which you include as reporter classes in the MATLAB Report API.

```
import slreportgen.finder.*
import slreportgen.report.*
import mlreportgen.report.*
```

- 2 Load the `vdp` model.

```
model_name = 'vdp';
load_system(model_name)
```

- 3 Create the container object to hold the report and open the report. In this case, the output report is saved in zipped `vdp_model.htmx` HTML report. You can use any output name you want. If you run the report generator more than once using the same output file name, the output file is overwritten. To use Simulink Report API finders and reporters in your report generator program, you must use the fully qualified name to create the container object.

```
rpt = slreportgen.report.Report('vdp_model', 'html');
open(rpt)
```

- 4 Add a chapter and specify its title.

```
ch = Chapter('Blocks in Simulink vdp model');
```

- 5 Use the `BlockFinder` class to create a finder. In this case, the `BlockFinder` finds all the blocks in the model. Use the `find` method to find the blocks specified by the finder.

```
finder = BlockFinder(model_name);
results = find(finder);
```

- 6 Loop through the results of the `find` method and create a section for each block, and add the block property table to the section. Then, add each section to the chapter. After all blocks have been added, add the chapter to the report.

```
for result = results
    sect = Section('Title', result.Name);
    append(sect, result)
    append(ch, sect)
end
append(rpt, ch);
```

- 7 Close the report and model, and view the report.

```
close(rpt);
close_system(model_name);
rptview(rpt);
```

The full program is

```
import slreportgen.finder.*
import slreportgen.report.*
import mlreportgen.report.*

model_name = 'vdp';
load_system(model_name);

rpt = slreportgen.report.Report('vdp_model','html');
open(rpt)

ch = Chapter('Blocks in Simulink vdp model');
finder = BlockFinder(model_name);
results = find(finder);
for result = results
    sect = Section('Title',result.Name);
    append(sect,result)
    append(ch,sect);
end
append(rpt,ch);

close(rpt)
close_system(model_name)
rptview(rpt)
```

The chapter heading and the section headings and property tables of the first two blocks of the resulting report are shown.

## Chapter 1. Blocks in Simulink vdp model

### 1.1. Constant

**Table 1.1. vdp/Constant Properties**

Property	Value
Type	Block
Block Type	Constant
Constant value	1
Interpret vector parameters as 1-D	on
Output data type	Inherit: Inherit from 'Constant value'
Lock output data type setting against changes by the fixed-point tools	off
Sample time	inf
Frame period	inf

### 1.2. More Info

**Table 1.2. vdp/More Info Properties**

Property	Value
Type	System

## Use Specific Finders and Reporters for Different Block Types

Create a PDF report generator that finds all blocks in the `sldemo_radar_eml` model.

To find all blocks, use the `BlockFinder`. The `if` statement shows how to test for MATLAB Function blocks. Use the `MATLABFunction` reporter to report MATLAB Function block details. The `else` statement shows how blocks other than MATLAB Function blocks use the `BlockFinder` `find` method results.



```

blkfinder = BlockFinder(model_name);
blks = find(blkfinder);

if slreportgen.utils.isMATLABFunction(blks(i).Object)
    rptr = MATLABFunction(blks(i).Object);
    sec = Section(blks(i).Name);
    append(sec,rptr)
    append(ch,sec)

else
    sec = Section(blks(i).Name);
    append(sec,blks(i))
    append(ch,sec)

```

```

blkfinder = BlockFinder(model_name);
blks = find(blkfinder);

```

The full program is

```

import slreportgen.report.*
import slreportgen.finder.*
import mlreportgen.report.*

model_name = 'sldemo_radar_eml';
load_system(model_name)

rpt = slreportgen.report.Report('radar','pdf');
open(rpt)

blkfinder = BlockFinder(model_name);
blks = find(blkfinder);
ch = Chapter('Blocks in sldemo_radar_eml Model');

for i=1:length(blks)
    if slreportgen.utils.isMATLABFunction(blks(i).Object)
        rptr = MATLABFunction(blks(i).Object);
        sec = Section(blks(i).Name);
        append(sec,rptr)
        append(ch,sec)
    else
        sec = Section(blks(i).Name);
        append(sec,blks(i))
        append(ch,sec)
    end
end
append(rpt,ch)

close(rpt)
close_system(model_name)
rptview(rpt)

```

An example of the information reported for a MATLAB Function block by the MATLABFunction reporter is

## Chapter 1. Blocks in sldemo\_radar\_eml Model

### 1.1. MATLAB Function

**Table 1.1. MATLAB Function Object Properties**

Property	Value
Update Method	INHERITED
Sample Time	-1
Support variable-size arrays	0
Saturate on integer overflow	1
Treat these inherited Simulink signal types as fi objects	Fixed-point
MATLAB Function block fimath	Other:UserSpecified
Input fi math	fimath(... 'RoundMode', 'floor',... 'OverflowMode', 'wrap',... 'ProductMode', 'KeepLSB', 'ProductWordLength', 32,... 'SumMode', 'KeepLSB', 'SumWordLength', 32,... 'CastBeforeSum', true)
Description	

**Table 1.2. MATLAB Function Argument Summary**

Name	Scope	Port	Data Type	Size
meas	Input	1	double	2
residual	Output	1	double	2
xhatOut	Output	2	double	4
deltat	Parameter	NaN	double	1

#### MATLAB Function Function Script

```
function [residual, xhatOut] = EXTKALMAN(meas, deltat)
%EXTKALMAN Radar Data Processing Tracker Using an Extended Kalman Filter
```

An example of the information reported by the find method of the BlockFinder is

## 1.2. Meas. noise intensity

**Table 1.3. sldemo\_radar\_eml/Meas. noise intensity Properties**

Property	Value
Type	Block

2

Chapter 1. Blocks in sldemo\_radar\_eml Model

Property	Value
Block Type	Gain
Gain	[300 0; 0 .01]
Multiplication	Matrix(K*u) (u vector)
Parameter data type	Inherit: Inherit via internal rule
Output data type	Inherit: Inherit via internal rule
Lock output data type setting against changes by the fixed-point tools	off
Integer rounding mode	Floor
Saturate on integer overflow	off
Sample time (-1 for inherited)	-1

## 1.3. Radar Measurement Noise

**Table 1.4. sldemo\_radar\_eml/Radar Measurement Noise Properties**

Property	Value
Type	Block
Block Type	SubSystem
Noise power	[1 1]
Sample time	0.1
Seed	[52341 62341]
Interpret vector parameters as 1-D	on

## Find and Report on Stateflow Elements

This example describes how to find and report on Stateflow states, transitions, and junctions. It reports on the control\_logic chart of the sldemo\_fuelsys model.

This portion of the code uses the StateFinder and its find method to find and report on states in the chart. It loops through the array of found states and adds each one to the chapter.

```

stFinder = StateFinder(subsys);
states = find(stFinder);
for state = states
    append(chapter, state)
end
append(rpt, chapter)

```

Chapter 1. States	
<b>Table 1.1. sldemo_fuelsys/fuel_rate_control/control_logic/O2 Properties</b>	
Property	Value
Type	AND State
Description	This state determines the validity of the exhaust gas oxygen sensor (EGO) data.
Label	O2
<b>Table 1.2. sldemo_fuelsys/fuel_rate_control/control_logic/Pressure Properties</b>	
Property	Value
Type	AND State
Description	This state assesses the validity of the manifold absolute pressure (MAP) sensor.
Label	Pressure
<b>Table 1.3. sldemo_fuelsys/fuel_rate_control/control_logic/Throttle Properties</b>	
Property	Value
Type	AND State
Description	This state determines the validity of the throttle sensor signal.
Label	Throttle
<b>Table 1.4. sldemo_fuelsys/fuel_rate_control/control_logic/Speed Properties</b>	
Property	Value
Type	AND State
Description	This state infers the validity of the speed sensor data. A failure is indicated by the presence of manifold vacuum at zero speed.
Label	Speed

To report on the transitions, use the `StateflowDiagramElementFinder` and its `find` method. To show the property table with a narrower width than the default, customize the output. First, obtain the reporter for the result. To set the width, use the `TableWidth` property of the reporter.

```

chapter = Chapter('Title','Transitions');
trFinder = StateflowDiagramElementFinder...
    ('Container',subsys,'Types','transition');
transitions = find(trFinder);
for transition = transitions
    rptr = transition.getReporter;
    rptr.PropertyTable.TableWidth = '3in';
    append(chapter, rptr)
end
append(rpt, chapter)

```

**Chapter 2. Transitions****Table 2.1. sldemo\_fuelsys/fuel\_rate\_control/control\_logic/Fail/? Properties**

Property	Value
Type	Transition
Destination	None
ExecutionOrder	1
Label	?

**Table 2.2. sldemo\_fuelsys/fuel\_rate\_control/control\_logic/Fail/INC Properties**

Property	Value
Type	Transition
Source	None
Destination	One
ExecutionOrder	1
Label	INC

**Table 2.3. sldemo\_fuelsys/fuel\_rate\_control/control\_logic/Fail/INC Properties**

Property	Value
Type	Transition
Source	One
Destination	Two
ExecutionOrder	1
Label	INC

The complete program is

```
import mlreportgen.report.*
import slreportgen.report.*
import slreportgen.finder.*

model_name = 'sldemo_fuelsys';
load_system(model_name);
subsys = strcat(model_name,...
    '/fuel_rate_control/control_logic');

rpt = slreportgen.report.Report('output','pdf');
open(rpt)

tp = TitlePage('Title',...
    'Control Logic Chart of sldemo_fuelsys');
append(rpt,tp)
append(rpt,TableOfContents);

chapter = Chapter('Title','States');
stFinder = StateFinder(subsys);
states = find(stFinder);
for state = states
    append(chapter,state)
end
append(rpt,chapter)
```

```
chapter = Chapter('Title','Transitions');
trFinder = StateflowDiagramElementFinder...
('Container',subsys,'Types','transition');
transitions = find(trFinder);
for transition = transitions
    rptr = transition.getReporter;
    rptr.PropertyTable.TableWidth = '3in';
    append(chapter,rptr)
end
append(rpt,chapter)

close(rpt)
close_system(model_name)
rptview(rpt)
```

## See Also

slreportgen.report.Report

## More About

- “Relationship Between Simulink Report Generator and MATLAB Report Generator” on page 1-3
- “What Is a Reporter?”
- “Templates”

## Generate Reports Without Customizing

### In this section...

“Predefined Standard Reports” on page 1-21

“Report API” on page 1-21

“Web View” on page 1-21

You can use Simulink Report Generator without customizing reports by using:

- Predefined standard reports
- Report API objects
- Web view

### Predefined Standard Reports

Simulink Report Generator comes with two predefined, standard reports for Simulink:

- System Design Description Report
- System Requirements (requires Simulink Requirements)

The System Design Description Report provides summary or detailed information about a system design represented by a model. You can choose report options by using the report dialog box, or you can create a customized version using the Report Explorer. You can use the System Design Description report setup file as a starting point for creating a setup file for your own report. For details, see “Generate a System Design Description Report” on page 2-5.

The System Requirements report includes information about all the requirements associated with the model and its objects. You must have Simulink Requirements installed to use the System Requirements report.

To generate the System Requirements report from the Simulink Toolstrip:

- 1 If the **Requirements** tab is not available, on the **Apps** tab, in the **Model Verification, Validation, and Test** section, click **Requirements Manager**.
- 2 On the **Requirements** tab, in the **Share** section, click **Share > Generate Model Traceability Report**.

### Report API

The Report API, which is based on the DOM API, is a set of document objects, such as title page, table of contents, chapter, and figure objects, that do the work of dozens of lines of code based on DOM objects. As a result, the Report API greatly reduces the amount of code needed to generate reports. As with the DOM API, you can add content to your report in the form of built-in MATLAB objects, such as strings, number and character arrays, and cell arrays. The Report API converts these to DOM objects before adding them to your report. You can also use text, paragraph and other DOM objects directly to format the content that you add to your report.

### Web View

A web view is a view of a model that you can explore in a web browser. Web views are useful for presenting models to audiences and for sharing models with colleagues who do not have MathWorks®

products installed. You can use web views to navigate subsystems and see properties of blocks and signals. For details, see “Export Models to Web View Files” on page 5-4.



## Report Creation Workflow

Use this general approach for creating reports interactively.

- 1 Open the Report Explorer. In the Simulink Toolstrip, on the **Apps** tab, under **Simulation Graphics and Reporting**, click **Report Generator**.
- 2 Create a report setup file for your new report design.
- 3 Add components to the report setup file. Components determine the behavior and contents of your report. You can use the supplied components and you can create your own custom components.
- 4 Choose a Microsoft Word, HTML, or PDF template or a Report Explorer style sheet to associate styles with the report setup file.
- 5 Generate the report.

To create reports programmatically using the Report API and DOM API, see “Report Generator Development” and “Create Report Programs”.

### See Also

### Related Examples

- “Generate a Report Using a Template”
- “Add Report Content with Components” on page 4-51

### More About

- “Report Setup”
- “Layout Style Sheets”

## Report Components

In this section...
“About Report Components” on page 1-24
“Report Structure Components” on page 1-24
“System-Based Components” on page 1-24
“User-Supplied Information Components” on page 1-25
“Dynamic Reporting Components” on page 1-26
“Format Control at the Component Level” on page 1-26

### About Report Components

Include components in a report setup file to specify report behavior and insert content, such as tables, lists, and figures, into a report. Use the Report Explorer on page 1-27 to add components to a report and to specify their behavior.

Use a combination of these types of components in your report setup file.

Component Type	Description
“Report Structure Components” on page 1-24	Include a title page, sections, and other components to organize the content of a report.
“System-Based Components” on page 1-24	Include components that obtain information directly from a model to include in a report.
“User-Supplied Information Components” on page 1-25	Include text and graphics that you supply, independent of a model.
“Dynamic Reporting Components” on page 1-26	Set up dynamic control for when to include components and what information to report on for a component, based on data from a model or on other conditions that you specify.

### Report Structure Components

To add a title page, use a **Title Page** component. You can include an abstract and legal notice information. For an example, see “Add a Title Page” on page 4-56.

To organize a report into sections, use **Chapter/Subsection** components. For an example, see “Create a Section for Each Iteration” on page 4-68.

### System-Based Components

The Simulink Report Generator includes components that get information from a model to include in a report. Using system-based components allows your report to describe the current state of a model. Once the setup file contains these components, you can generate the report whenever you want to capture the latest version of a model.

Property table components display property name/property value pairs for objects in tables. Summary table components insert tables that include specified properties for objects into generated reports. The tables contain one object per row, with each object property appearing in a column.

To use descriptive information from DocBlock blocks, use the `Documentation` component.

A few examples of system-based components include:

- MATLAB Property Table
- Simulink Workspace Variable
- System Hierarchy
- Simulink Summary Table
- Simulink Dialog Snapshot
- Block Execution Order List
- Model Loop
- Model Configuration Set
- Scope Snapshot

For examples of using system-based components, see:

- “Property Table Components”
- “Summary Table Components”
- “Create the Body of the Report” on page 4-62

The Simulink Report Generator also includes system-based components that contain model elements from the following Simulink products:

- Stateflow
- Fixed-Point Designer™
- Simulink Coder™
- Simulink Check™
- Simulink Requirements

## User-Supplied Information Components

In addition to using system-based components to extract data from a system and insert that information into a report, you can also add content that you, or others, supply. For example, to include text, use the `Paragraph` and `Text` components.

To insert a graphic from a file, use an `Image` component. To insert ASCII text, use an `Import File` component.

To include notes about the report source files, use a `Comment` component.

For an example, see “Add Introductory Text to the First Chapter”.

## Dynamic Reporting Components

Dynamic reporting components execute conditionally, enabling you to decide when a child component executes or how many times a child component executes. To control the report generation flow, use logical and flow components such as `Logical If`, `Logical Then`, `While Loop`, or `For Loop`.

A looping component runs its child components a specified number of times. There are several looping components, including logical loops, Handle Graphics® loops, and model and chart loops. For model and chart loops, you can control aspects such as the order in which the report sorts blocks.

For examples, see:

- “Logical and Looping Components”
- “Add Logical Then and Logical Else Components” on page 4-59
- “Create the Body of the Report” on page 4-62
- “Filter with Loop Context Functions” on page 4-87

## Format Control at the Component Level

The output format and stylesheet that you select for a report determines most aspects of the generated report formatting. For details, see “Report Output Format”.

In addition to stylesheets that control the format and layout of the report, for some components you can set properties to specify formatting details for that specific instance of a component. For example, for the `Simulink Property Table`, you can specify whether to display table borders or specify the alignment of text in table cells.

## Working with the Report Explorer

### About the Report Explorer

The *Report Explorer* is the MATLAB Report Generator and Simulink Report Generator graphical interface. It allows you to:

- Create and modify report setup files.
- Apply style sheets to format the generated report.
- Specify the report file format.
- Generate reports.

To open the Report Explorer, enter `report` in the MATLAB Command Window.

- The *Outline pane* on the left shows the hierarchy of components in currently opened report setup files. Report components can reside within other report components, creating parent, child, and sibling relationships.
- The *Library pane* in the middle lists the objects available in the context of the Outline pane.

Outline Pane Context	Library Pane Contents
No report setup file is open.	Reports
Report setup file is open.	Components
Style sheet is open.	Style sheet attributes

- The *Properties pane* contents depend on the Outline pane context. If no report setup file is open, on the right displays tasks the Report Explorer can perform. If a report setup file is open, the Properties pane displays the properties for the item that is currently selected in the Library pane.

Outline Pane Context	Properties Pane Contents
No report setup file is open.	Tasks that the Report Explorer can perform
Report setup file is open.	Properties for the item that is currently selected  After you create a report setup file, the Properties pane initially displays properties for the report setup file as a whole.

---

**Tip** If the Report Explorer window opens with only two panes, one of the panes is hidden. You can move the vertical boundaries between the panes to reveal any hidden pane, or to make visible panes wider or narrower.

---

## **Acknowledgments**

Simulink Report Generator uses Antenna House® XSL Formatter. Antenna House is a trademark of Antenna House, Inc.

Antenna House XSL Formatter© 2009-2019 Copyright Antenna House, Inc.

# Generate System Design Description Reports

---

- “System Design Description” on page 2-2
- “Generate a System Design Description Report” on page 2-5
- “Customize the System Design Description” on page 2-6
- “Generate a System Design Report with the Report API” on page 2-8

## System Design Description

In this section...
“Predefined Standard Reports” on page 2-2
“What Is the System Design Description?” on page 2-2
“What You Can Do with the Report” on page 2-2
“Report Contents” on page 2-3

### Predefined Standard Reports

From the Simulink Toolstrip, you can generate two predefined, standard Simulink Report Generator reports called:

- System Design Description
- System Requirements Traceability

The System Design Description report provides summary or detailed information about a system design represented by a model. You can choose report options using the report dialog, or you can create a customized version using the Report Explorer. For details, see “Generate a System Design Description Report” on page 2-5.

You can use the System Design Description report setup file as a starting point for creating a setup file for your own report.

You can also generate an HTML model report for Stateflow charts. For details, see “Generate a Model Report” (Stateflow).

The System Requirements Traceability report requires that you have Simulink Requirements installed. The System Requirements Traceability report includes information about all the requirements associated with the model and its objects.

Follow these steps to generate the System Requirements Traceability report from the Simulink Toolstrip:

- 1 If the **Requirements** tab is not available, on the **Apps** tab, in the **Model Verification, Validation, and Test** section, click **Requirements Manager**.
- 2 On the **Requirements** tab, in the **Share** section, click **Share > Generate Model Traceability Report**.

### What Is the System Design Description?

The System Design Description is a prebuilt Simulink Report Generator report that describes the system design represented by a Simulink model.

By default, the Simulink Report Generator generates the report for the model from which you invoke the System Design Description report option.

### What You Can Do with the Report

You can use the System Design Description to



- Review a system design without having the model open
- Generate summary and detailed descriptions of the design
- Assess compliance with design requirements
- Archive the system design in a format independent of the modeling environment
- Build a customized version of the report, using the Report Explorer

## Report Contents

You can specify what kinds of information to include in the report, in terms of:

- What elements of a model to include in the report (for example, whether to include subsystems from custom libraries)
- Whether to generate a summary version or a detailed version of the System Design Description report.

For details, see “Generate a System Design Description Report” on page 2-5.

### Summary Version

Section	Information
Report Overview	Model version
Root System	<ul style="list-style-type: none"> <li>• Block diagram representing the algorithms that compute root system outputs</li> <li>• Description (if available from model)</li> <li>• Interface: name, data type, and other properties of the system input and output signals</li> <li>• Subsystems: the path and a block diagram for each subsystem</li> <li>• State charts</li> <li>• Requirements (optional)</li> </ul>
Subsystems	<ul style="list-style-type: none"> <li>• Path</li> <li>• Block diagram</li> </ul>
System Design Variables	<ul style="list-style-type: none"> <li>• Design variables</li> <li>• Functions in design variable expressions</li> </ul>

### Detailed Version

The detailed version of the report includes all the information that is in the summary form of the report, as well as more information about the system components. The atomic subsystem information is more detailed than virtual subsystem information.

Section	Information
Report Overview	Model version

Section	Information
Root system	<ul style="list-style-type: none"> <li>• Block diagram representing the algorithms that compute root system outputs</li> <li>• Description (if available from model)</li> <li>• Interface: name, data type, and other properties of the root system input and output signals</li> <li>• Block parameters                             <ul style="list-style-type: none"> <li>• Includes detailed information about MATLAB Function blocks</li> </ul> </li> <li>• Block execution order for root system and atomic subsystems</li> <li>• Look-up tables</li> <li>• Simulink workspace variables</li> <li>• Model configuration sets</li> <li>• State charts</li> <li>• Requirements (optional)</li> </ul>
Subsystems	<p>The same type of information as the information for the root system, as well as:</p> <ul style="list-style-type: none"> <li>• Path of the subsystem in the model</li> <li>• (For atomic subsystems) Checksum that indicates whether the version of an atomic subsystem that generates the report differs from other versions of the subsystem</li> <li>• Referenced models (optional)</li> <li>• Subsystems from custom libraries (optional)</li> </ul>
State Charts	<ul style="list-style-type: none"> <li>• State chart</li> <li>• States</li> <li>• Transitions between the states</li> <li>• Junctions</li> <li>• Events that trigger state transitions</li> <li>• Data types</li> <li>• Targets</li> <li>• Truth tables</li> </ul>

**Report Captures Documentation Included in a Model**

The System Design Description reports documentation included in a model, including:

- The model description (from the model properties)
- The block property Description
- DocBlock model documentation blocks

To enrich the generated System Design Description, add corresponding information in the model.

## Generate a System Design Description Report

Generate a system design description report to create a standard report of your model from the Simulink Editor. The System Design Description report provides summary or detailed information about a system design represented by a model.

When you generate the report, you can specify layout and content options for:

- Title page contents
- Report content
- Report file format and storage location

---

**Tip** For faster report generation, set **File format** to one of the `from template` options. For example, select `Direct PDF (from template)` to output to PDF.

---

- 1 Open the model or subsystem for which you want to generate a report. The model must compile without error for the report to generate.
- 2 On the **Modeling** tab, in the **Design** section, click **System Design Report**.
- 3 In the Design Description dialog box, specify layout and content options for the report. To display detailed information about each option, right-click the label and select **What's This**.
- 4 Click **Generate**.

To create a customized version of the report, click **Customize Content**. This option creates a copy of the report setup file and opens the copy in the Report Explorer. See “Customize the System Design Description” on page 2-6.

### See Also

### More About

- “System Design Description” on page 2-2

## Customize the System Design Description

### In this section...

“Using the Report Explorer to Customize the Report” on page 2-6

“Building a Dialog Box for a Custom Report Setup File” on page 2-7

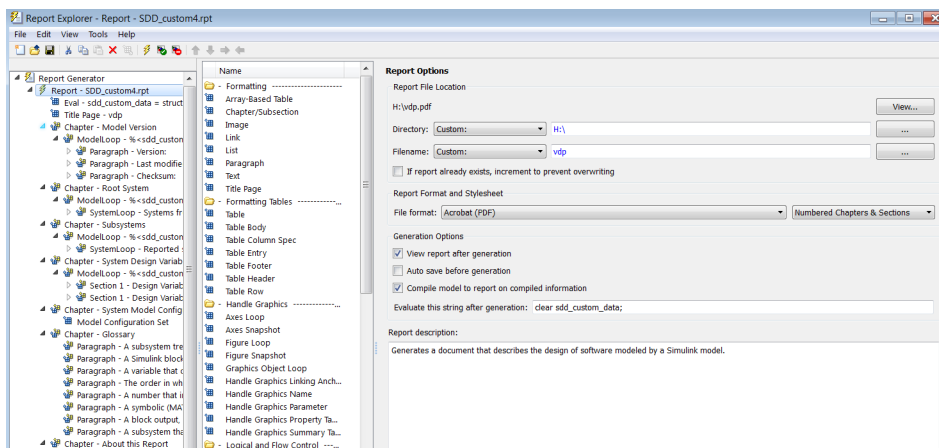
You can create customized versions of the System Design Description report by using the Report Explorer and, optionally the MATLAB tools for building graphical user interfaces.

By default, when you open a customized version of the report, the System Design Description dialog box does not open.

### Using the Report Explorer to Customize the Report

To customize the System Design Description setup file in the Simulink Report Generator using the Report Explorer:

- 1 In the System Design Description dialog box, click the **Customize Content** button to open the Report Explorer.



The Report Explorer reflects any changes (for example, a different report name) that you made in the System Design Description dialog box.

- 2 In the Report Explorer, add or modify components. See “Add Report Content with Components” on page 4-51 and “Information Components”.

- Do not remove the `sdd_custom_data` structure, which is defined as:

```
sdd_custom_data = struct('model',bdroot,'rootSystem',gcs);
```

You can modify the `model` argument, which is the model for which you generated the report and the `rootSystem` argument, which is the system-level in the model at which, and below which, you want to use to generate the report.

- Do not remove or modify functions that begin with `StdRpt`, such as `%StdRpt.getChecksum`
- 3 Optionally modify a style sheet (see “Layout Style Sheets”).
  - 4 Save the customized report with a name other than `SDD_custom.rpt`.

## **Building a Dialog Box for a Custom Report Setup File**

To provide options for your custom report, you can create a dialog box, like the System Design Description dialog box. The dialog box that you create for your custom report can allow others to adapt the report to meet their needs, without their having to use the Report Explorer.

## Generate a System Design Report with the Report API

This example shows how to generate a Report API-based system design report. The system design report is a description of the design of a dynamic system generated from the Simulink® model of the system.

A Report API-based system design report uses objects of Report API classes to report on system components. See “Report Generation for Simulink and Stateflow Elements” on page 1-9.

### The gensdd Function

The gensdd function, which is included with this example, generates a report that includes these sections:

- Title Page
- Table of Contents
- List of Figures
- List of Tables
- System Hierarchy
- Root System chapter that contains the root block diagram, task and block execution order, and properties of each block in the root diagram
- SubSystems chapter that contains the diagram, block execution order, and block properties of the model subsystems
- Charts chapter that contains the charts and chart object properties of each of the model charts
- Design Data chapter that contains the model variables
- System Model Configuration chapter that contains details about the active configuration set for the model

The complete gensdd function is listed at the end of this example. You can modify the gensdd.m file to create a custom system design report.

### Generate a PDF System Design Report for the sf\_car Model

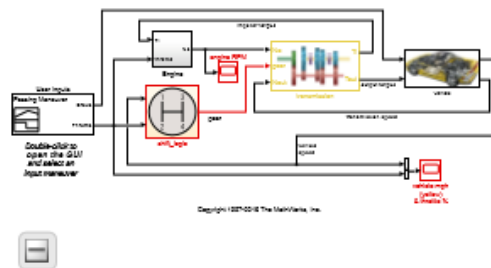
Generate a system design report for the sf\_car model and specify that the document is a PDF.

```
gensdd('sf_car', 'PDF');
```

Here is the first page of the report:

# SF\_CAR

## System Design Description



**John Doe**

01-Jan-2020

A copy of the report is included with this example in the file `sdd_sf_car_copy.pdf`.

### Customize the Report

The `gensdd` function uses objects of these Report API classes to find system design elements and report on the system design:





- `slreportgen.report.SystemHierarchy`
- `slreportgen.report.Diagram`
- `slreportgen.report.SystemIO`
- `slreportgen.report.ExecutionOrder`
- `slreportgen.report.ModelConfiguration`
- `slreportgen.finder.SystemDiagramFinder`
- `slreportgen.finder.ChartDiagramFinder`
- `slreportgen.finder.DiagramResult`
- `slreportgen.finder.StateflowDiagramElementFinder`
- `slreportgen.finder.DiagramElementResult`
- `slreportgen.report.LookupTable`
- `slreportgen.report.StateflowObjectProperties`
- `slreportgen.report.TruthTable`
- `slreportgen.finder.ModelVariableFinder`

- `slreportgen.finder.ModelVariableResult`
- `slreportgen.report.ModelVariable`
- `slreportgen.finder.BlockFinder`
- `slreportgen.finder.BlockResult`
- `slreportgen.report.DocBlock`
- `slreportgen.report.MATLABFunction`
- `slreportgen.report.SimulinkObjectProperties`
- `slreportgen.report.TestSequence`

You can use the properties of these objects to filter and format the information that is reported. For example, in the `makeSystemHierarchy` function in the `gensdd.m` file, you can change the system hierarchy list format to an ordered list by setting the `ListFormatter` property of the `slreportgen.report.SystemHierarchy` reporter.

```
function makeSystemHierarchy(rpt, hModel)
% Create a chapter reporting on the system hierarchy
import mlreportgen.report.*
import slreportgen.report.*
ch = Chapter("Title", "System Hierarchy");
ol = mlreportgen.dom.OrderedList();
add(ch, SystemHierarchy("Source", hModel, "ListFormatter", ol));
add(rpt, ch);
end
```

### Chapter 1. System Hierarchy

1.  [sf car](#)
  1.  [Engine](#)
  2.  [transmission](#)
    1.  [transmission ratio](#)

The report also uses objects of these Report API classes to create and format the sections of the report:

- `mlreportgen.report.TitlePage`
- `mlreportgen.report.TableOfContents`
- `mlreportgen.report.ListOfFigures`
- `mlreportgen.report.ListOfTables`
- `mlreportgen.report.Chapter`
- `mlreportgen.report.Section`

You can customize the appearance of the report and sections. See “Report Formatting Approaches”.

#### The Complete `gensdd` Function

type [gensdd.m](#)



```

function gensdd(model,doctype)
%GENSDD Generates a system design description from the system's model
% gensdd() generates a PDF system design description for the sf_car
% model.
%
% gensdd(model) generates a PDF system design description for the
% specified model.
%
% gensdd(model, doctype) generates a system design description document
% from the specified model and document type: 'html', 'docx', or 'pdf'.
%
% The generated document is a description of a dynamic system's design
% generated from its Simulink model. The description contains the
% following sections:
%
% * Title Page
% * Table of Contents
% * List of Figures
% * List of Tables
% * System Hierarchy
% * Root System Chapter -- Contains root block diagram, task and block
% execution order, and properties of each block in the root diagram.
% * Subsystems Chapter -- Contains diagram, block execution order, and
% block properties of model's subsystems.
% * Charts Chapter -- Contains charts and chart object properties of each
% of the model's charts.
% * Design Data Chapter -- Contains the model variables.
% * System Model Configuration Chapter -- Contains details about the
% active configuration set for the model.

import mlreportgen.dom.*
import mlreportgen.report.*
import slreportgen.report.*

if nargin < 1
    model = 'sf_car';
    doctype = 'pdf';
end

if nargin < 2
    doctype = 'pdf';
end

hModel = load_system(model);
rpt = slreportgen.report.Report(['sdd_' get_param(model, 'Name')], doctype);
open(rpt);

makeTitlePage(rpt, hModel);
add(rpt, TableOfContents);
add(rpt, ListOfFigures);
add(rpt, ListOfTables);
makeSystemHierarchy(rpt, hModel);
makeRootSystemChapter(rpt, hModel);
makeSubsystemsChapter(rpt, hModel);
makeChartsChapter(rpt, hModel);
makeDesignDataChapter(rpt, hModel);
makeModelConfigurationChapter(rpt, hModel);

```

```
close(rpt);
rptview(rpt);

close_system(model);

end

function makeTitlePage(rpt, hModel)
import mlreportgen.report.*
import slreportgen.report.*

tp = TitlePage;
tp.Title = upper(get_param(hModel, 'Name'));
tp.Subtitle = 'System Design Description';
tp.Author = 'John Doe';

diag = Diagram(hModel);
diag.Scaling = 'custom';
diag.Height = '2in';
diag.Width = '3in';
tp.Image = getSnapshotImage(diag, rpt);
add(rpt, tp);
end

function makeSystemHierarchy(rpt, hModel)
% Create a chapter reporting on the system hierarchy
import mlreportgen.report.*
import slreportgen.report.*
ch = Chapter("Title", "System Hierarchy");
add(ch, SystemHierarchy(hModel));
add(rpt, ch);
end

function makeRootSystemChapter(rpt, hModel)
% Create a chapter reporting on the root system diagram and its blocks
% and add the chapter to the main report.
import mlreportgen.report.*
import slreportgen.report.*
import slreportgen.finder.*

ch = Chapter("Title", "Root System");
diag = Diagram(hModel);
add(ch, diag);

add(ch, SystemIO(hModel));

% Add block execution order section
makeExecutionOrderSection(ch, hModel);

% Add subsections containing the properties for each block in the
% subsystem diagram.
makeBlockSections(ch, hModel);

add(rpt, ch);
end

function makeSubsystemsChapter(rpt, hModel)
% Create a chapter reporting on a model's subsystems and the blocks that
```

```

% they contain and add the chapter to the main report.
import mlreportgen.report.*
import slreportgen.report.*
import slreportgen.finder.*

% Create a chapter to hold the subsystems.
ch = Chapter("Title", "Subsystems");

% Use a finder to find all the subsystem diagrams in the model. The finder
% returns an array of SystemDiagramResult objects, each of which
% contains a Diagram reporter that creates a snapshot of the subsystem
% diagram
finder = SystemDiagramFinder(hModel);
finder.IncludeRoot = false;
systems = find(finder);

% Add the subsystem diagram results to the chapter.
for system = systems

    % Create a subsection to contain the subsystem diagram.
    section = Section("Title", system.Name);

    % Add the subsystem diagram reporter to the diagram subsection.
    % Add the subsystem diagram results to the chapter.
    diag = getReporter(system);
    diag.MaskedSystemLinkPolicy = 'system';
    add(section, diag);

    ioSect = Section('Title', 'System Interface');
    add(ioSect, SystemIO('Object', system, 'ShowDetails', false));
    add(section, ioSect);

    % If the subsystem is nonvirtual, add a subsection detailing the block
    % execution order
    if strcmp(get_param(system.Object, "IsSubsystemVirtual"), "off")
        makeExecutionOrderSection(section, system);
    end

    % Add subsections containing the properties for each block in the
    % subsystem diagram.
    makeBlockSections(section, system);

    % Add the subsystem diagram section to the chapter.
    add(ch, section);
end

% Add the subsystems chapter to the main report.
add(rpt, ch);

end

function makeChartsChapter(rpt, hModel)
% Create a chapter reporting on a model's Stateflow charts and the objects
% that they contain and add the chapter to the main report.

import mlreportgen.report.*
import slreportgen.report.*
import slreportgen.finder.*

```

```
finder = ChartDiagramFinder(hModel);
charts = find(finder);

if ~isempty(charts)
    ch = Chapter("Title", "Charts");
    for chart = charts
        section = Section("Title", chart.Name);
        diag = getReporter(chart);
        add(section, diag);

        % Report the objects in this chart
        objFinder = StateflowDiagramElementFinder(chart);
        sfObjects = find(objFinder);
        for sfObj = sfObjects
            objSection = Section("Title", sfObj.Name);
            add(objSection, sfObj);
            add(section, objSection);
        end

        add(ch, section);
    end
    add(rpt, ch);
end

end

function makeDesignDataChapter(rpt, hModel)
% Create a chapter reporting on the model variables

import mlreportgen.dom.*
import mlreportgen.report.*
import slreportgen.report.*
import slreportgen.finder.*

ch = Chapter("Title", "Design Data");

finder = ModelVariableFinder(hModel);

results = find(finder);
n = numel(results);

if n > 0
    s = Section("Title", "Design Data Summary");

    vars = cell(n, 4);
    for i = 1:n
        result = results(i);
        % Get link target for variable reporter
        lt = getVariableID(result);
        value = getVariableValue(results(i));
        vars{i, 1} = InternalLink(lt, results(i).Name);
        vars{i, 2} = class(value);
        vars{i, 3} = results(i).Source;
        vars{i, 4} = results(i).SourceType;
    end

    t = FormalTable(["Name", "Type", "Source", "Source Type"], vars);
end
```

```

% Set styles for table header
t.Header.TableEntriesStyle = {Bold, BackgroundColor("lightgrey")};
% Set styles for entire table
t.Width = "100%";
t.Border = "solid";
t.RowSep = "solid";
t.ColSep = "solid";

add(s, t);
add(ch, s);

s = Section("Title", "Design Data Details");
% Separate multiple variable details be a horizontal rule
if n > 1
    for result = results(1:end-1)
        add(s, result);
        add(s, HorizontalRule);
    end
end
add(s, results(end));
add(ch, s);

add(rpt, ch);
end

end

function makeModelConfigurationChapter(rpt, hModel)
% Create a chapter reporting on the active configuration set of the
% reported model.

import mlreportgen.report.*
import slreportgen.report.*

ch = Chapter("Title", "System Model Configuration");

modelConfig = ModelConfiguration(hModel);

% Add the reporter to the chapter and chapter to the report
add(ch,modelConfig);
add(rpt,ch);
end

function section = makeBlockSections(parent, system)
% Create subsections containing the properties of each block in the
% system and add it to the parent chapter or subsection.
import mlreportgen.report.*
import slreportgen.finder.*

blocksSection = Section("Title", "Blocks");
finder = BlockFinder(system);
elems = find(finder);
for elem = elems
    section = Section("Title", strep(elem.Name, newline, ' '));
    add(section, elem);
    add(blocksSection, section);
end
add(parent, blocksSection);

```

```
end

function makeExecutionOrderSection(parent, system)
% Create a section to display a list of blocks in the system in order of
% execution. If the system is a top-level model, display information about
% all tasks in the model.
import mlreportgen.report.*
import slreportgen.report.*
section = Section('Title', 'Block Execution Order');

eo = ExecutionOrder(system);
if ~slreportgen.utils.isModel(system)
    % Only show task details for top-level models
    eo.ShowTaskDetails = false;
end

add(section, eo)
add(parent, section);
end
```

### See Also

slreportgen.finder.BlockFinder | slreportgen.finder.BlockResult |  
slreportgen.finder.ChartDiagramFinder | slreportgen.finder.DiagramElementResult  
| slreportgen.finder.DiagramResult | slreportgen.finder.ModelVariableFinder |  
slreportgen.finder.ModelVariableResult |  
slreportgen.finder.StateflowDiagramElementFinder |  
slreportgen.finder.SystemDiagramFinder | slreportgen.report.Diagram |  
slreportgen.report.ModelVariable | slreportgen.report.SimulinkObjectProperties |  
slreportgen.report.SystemHierarchy | slreportgen.report.SystemIO

### More About

- “Report Generation for Simulink and Stateflow Elements” on page 1-9

# System Design Description

---

## System Design Description Dialog Box

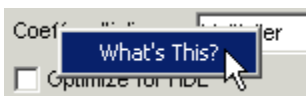
In this section...
"System Design Description Overview" on page 3-2
"Title" on page 3-2
"Subtitle" on page 3-3
"Authors" on page 3-3
"Image" on page 3-4
"Legal Notice" on page 3-4
"Design details" on page 3-4
"Model references" on page 3-5
"Subsystems from custom libraries" on page 3-5
"Requirements traceability" on page 3-6
"Glossary and report explanation" on page 3-6
"File format" on page 3-6
"Stylesheet or Template" on page 3-7
"File name" on page 3-9
"Folder" on page 3-10
"If report exists, increment name to prevent overwriting" on page 3-10
"Package type" on page 3-10

### System Design Description Overview

Choose options for the content, format, and location of the generated System Design Description report. To customize the report template, click the **Customize** button to open the report in the Report Explorer.

#### To get help on an option

- 1 Right-click the option's text label.
- 2 Select **What's This** from the popup menu.



#### See Also

"Generate Standard Reports"

### Title

Provide the title text. For PDF, Word, and RTF format reports, the title is on the title page. For HTML format reports, the title is at the top of the report.



## Settings

**Default:** <Model name>

- Title can include letters, numbers, and special characters.
- Length is unlimited.

## See Also

“Generate Standard Reports”

## Subtitle

Provide the subtitle text. For PDF, Word, and RTF format reports, the subtitle is under the title on the title page. For HTML format reports, the subtitle is under the title at the top of the report.

## Settings

**Default:** System Design Description

- If you do not want a subtitle, delete the default setting and leave the field blank.
- Subtitle can include letters, numbers, and special characters.
- Length is unlimited.

## Tip

If you generate both summary and detailed versions of the report, consider reflecting the type of report in the subtitle.

## See Also

“Generate Standard Reports”

## Authors

List the names of the creators of the system for which you are generating the design description. List of authors is under the subtitle.

## Settings

**Default:** Value of the ModifiedBy parameter for the model. The ModifiedBy parameter indicates the last person who updated the model.

## Tip

To find the creator of the system, in the Simulink Toolstrip, on the **Modeling** tab, in the **Setup** section, click **Model Settings** and then select **Model Properties**. Click the **History** tab.

## See Also

“Generate Standard Reports”

### Image

Include an image on the title page (for PDF, Word, and RTF format reports) or near the top of the report (for the HTML format).

#### Settings

##### No Default

- Specify the full path to the image file that you want to include in the report or click the **Select Image** button to browse to the image file.
- Supported image file formats include:
  - GIF
  - JPEG
  - BMP
  - PNG
  - TIF

#### Tip

An example of an image you might want to use is a logo or other graphic for a company, division, or project involved in the system design.

#### See Also

“Generate Standard Reports”

### Legal Notice

Provide legal notification text. For PDF, Word, and RTF format reports, the legal notice appears at the bottom of the title page (second page ). For HTML format reports, the legal notice appears near the top of the report.

#### Settings

**Default:** For Internal Distribution Only

- Length is unlimited.
- The Legal Notice field does not support text formatting (such as bold or italics).

#### See Also

“Generate Standard Reports”

### Design details

To generate a detailed system design description report, use the default (enabled). To generate a summary description, disable this option.

## Settings

**Default:** Enabled (generate a detailed report)

- The summary report provides system design information about the root system and block diagrams for the subsystems in the model.
  - The information about the root system includes:
    - Block diagram
    - Interface: name, data type, and other properties of the system input and output signals
    - Look-up tables
    - State charts
    - Requirements (optional)
- The detailed version of the report includes all the information that is in the summary form of the report. The detailed version includes the following information, in addition to the summary information. Atomic subsystem information is more detailed than virtual subsystem information.
  - Block parameters
  - Block execution order for root system and atomic subsystems
  - (For atomic subsystems) Checksum that indicates whether the version of an atomic subsystem used to generate the report differs from other versions of the subsystem

## See Also

“Generate Standard Reports”

## Model references

To include model references, use the default (enabled). To exclude model references, disable this option.

## Settings

**Default:** Enabled

## See Also

- “Generate Standard Reports”
- “Model References”

## Subsystems from custom libraries

Include library links to subsystems defined in custom (user-created) libraries.

## Settings

**Default:** Enabled

**See Also**

- “Generate Standard Reports”

**Requirements traceability**

Include links from blocks to the requirements that the blocks meet.

**Settings**

**Default:** Enabled

- To capture requirements links in the report, the model must include requirements links. Use Simulink Requirements to establish requirements links.
- If you use the default (enabled) and there are no requirements links in the model, the generated report omits the Requirements section.

**See Also**

- “Generate Standard Reports”

**Glossary and report explanation**

Include a glossary of terms in the report and a description of the report contents. The glossary includes definitions of Simulink terms such as “atomic subsystem,” “block diagram,” “signal.” The glossary helps readers who are unfamiliar with Simulink to understand the system design description. The glossary and report explanation sections appear at the end of the report and are three pages long (in PDF).

**Settings**

**Default:** Enabled

**Tip**

The report explanation (“About this Report”) describes the information in each report section.

**See Also**

“Generate Standard Reports”

**File format**

Specify the output format for the report. Generating a report formatted by a template is generally faster than generating a report formatted by a Report Explorer stylesheet.

**Settings**

Direct PDF (from template)

Generate a PDF report from a template.

PDF (from Word template)

Generate a PDF report using a Word template.

**HTML (from template)**

Generate an HTML report from a template. You can choose a **Package type** option.

**Single-File HTML (from template)**

Generate an HTML report from a template as a single file.

**Word (from template)**

Generate a Word report using a template.

**Acrobat (PDF)**

Generate a PDF report using an XSL stylesheet.

**Web (HTML)**

Generate an HTML report using an XSL stylesheet.

**Word Document**

Generate a Rich Text Format (RTF) document using a DSSSL stylesheet and convert the RTF document to a Word .doc file. Available only on Microsoft Windows®.

**Rich Text Format**

Generate a Rich Text Format (RTF) document using a DSSSL stylesheet.

**See Also**

“Generate Standard Reports”

**Stylesheet or Template**

Specify the stylesheet or template to use for the report.

**Settings**

The settings depend on the **File format** option you choose.

When you use a `from template` option, you can use:

- The default template for the output type
- The `Default Numbered` template, which numbers sections in chapters with numbers like 1.1, 1.2, and 1.1.1
- A customized template provided at your site

When you use one of the other output options, you can choose from several standard style sheets or any customized versions provided at your site.

For the `Acrobat (PDF)` file format, these are the stylesheet options:

**Default:** Default print stylesheet

## Default print stylesheet

Displays title page, table of contents, list of titles

## Standard Print

Displays title page, table of contents, list of titles

**Simple Print**

Suppresses title page, table of contents, list of titles

**Compact Simple Print**

Minimizes page count, suppresses title, table of contents, list of titles

**Large Type Print**

Uses 12-point font (slightly larger than Standard Print)

**Very Large Type Print**

Uses 24-point font and landscape paper orientation

**Compact Print**

Minimizes white space to reduce page count

**Unnumbered Chapters & Sections**

Chapters and sections are not numbered

**Numbered Chapters & Sections**

Chapters and sections are both numbered

**Paginated Sections**

Sections are printed with page breaks

**Custom Header**

Lets you specify custom headers and footers

**Custom Titlepage**

Lets you specify custom title page content and presentation

**Verbose Print**

Lets you specify advanced print options

For the **Web (HTML)** file format, these are the stylesheet options:

**Default for Web (HTML):** Default HTML stylesheet

**Default HTML stylesheet**

HTML on a single page

**Simulink book HTML stylesheet**

HTML on multiple pages; suppresses chapter headings and table of contents

**Truth Table HTML stylesheet**

HTML on multiple pages; suppresses chapter headings and table of contents

**Multi-page Web**

HTML, with each chapter on a separate page

**Single-page Web**

HTML on a single page

**Single-page Unnumbered Chapters & Sections**

HTML on a single page; chapters and sections are not numbered

**Single-page Numbered Chapters & Sections**

HTML on a single page; chapters and sections are numbered

**Single-page Simple**

HTML on a single page; suppresses title page and table of contents

**Multi-page Simple**

HTML on multiple pages; suppresses title page and table of contents

**Multi-page Unnumbered Chapters & Sections**

HTML on multiple pages; chapters and sections are not numbered

**Multi-page Numbered Chapters & Sections**

HTML on multiple pages; chapters and sections are not numbered

For the **Rich Text Format** and **Word** file formats, these are the stylesheet options:

**Default for Rich Text Format and Word file formats: Standard Print****Standard Print**

Displays title page, table of contents, list of titles

**Simple Print**

Suppresses title page, table of contents, list of titles

**Compact Simple Print**

Minimizes page count, suppresses title, table of contents, list of titles

**Large Type Print**

Uses 12-point font (slightly larger than Standard Print)

**Very Large Type Print**

Uses 24-point font and landscape paper orientation

**Compact Print**

Minimizes white space to reduce page count

**Unnumbered Chapters & Sections**

Chapters and sections are not numbered

**Numbered Chapters & Sections**

Chapters and sections are both numbered

**See Also**

“Generate Standard Reports”

**File name**

Provide a name for the generated report file.

**Settings**

**Default:** <Name of model>

Do not include the file format extension. (For example, enter *MyReport*, but *not MyReport.pdf*.)

### Tip

If you generate both summary and detailed versions of the report, consider reflecting the type of report in the file name.

### See Also

“Generate Standard Reports”

### Folder

Provide a path to the folder in which to store the generated report file.

### Settings

#### No Default

- Use a full path name
- Click the **Select Folder** button to browse to the folder where you want to store the generated report.

### See Also

“Generate Standard Reports”

### If report exists, increment name to prevent overwriting

Increment the file name to preserve an existing report.

### Settings

**Default:** Enabled (increments the file name to prevent overwriting of an existing report file)

### Tips

If you generate both summary and detailed versions of the report, consider reflecting the type of report in the file name.

### See Also

“Generate Standard Reports”

### Package type

Packaging to use for reports generated using an HTML template

### Settings

**Default:** Zipped

#### Zipped

Package report files in a single compressed file that has the report name, with a .zip extension.



**Unzipped**

Generate the report files in a subfolder of the current folder. The subfolder has the report name.

**Both zipped and unzipped**

Package the report files as both zipped and unzipped.

**Dependency**

To use the **Packaging type options**, set **File format** to HTML (from template), choose a packaging options for the output files.

**See Also**

“Generate Standard Reports”



# Creating Simulink Reports

---

- “Create a Simulink Report Generator Report” on page 4-2
- “Report on MATLAB Function” on page 4-13
- “Use Simulink Report Explorer Components in a Report API Report” on page 4-20
- “Report Systems Hierarchically” on page 4-25
- “Customize Simulink Diagram Hyperlinks in HTML and PDF Reports” on page 4-27
- “Tile Simulink Diagrams” on page 4-29
- “Create a Simulink Bus Object Report” on page 4-32
- “Report System Inputs and Outputs” on page 4-34
- “Reporting on DocBlock Blocks” on page 4-37
- “Report Model Notes” on page 4-39
- “Report Execution Order of Tasks and Blocks in a Simulink System” on page 4-41
- “Create a Simulink Report Generator Report Interactively” on page 4-50
- “Generate a Report Associated with a Model” on page 4-85
- “Logical and Looping Components” on page 4-86
- “Filter with Loop Context Functions” on page 4-87
- “Loop Context Functions” on page 4-89

## Create a Simulink Report Generator Report

The Simulink Report Generator Report API comprises a set of objects designed to find and format model and simulation data. You can use these objects with MATLAB Report API and DOM API objects to create MATLAB programs that generate reports on Simulink models and simulations. The following example illustrates using the Simulink Report API and the MATLAB Report API to create a MATLAB program. This program generates a report on the contents of a Simulink model. The report contains these sections:

- Title Page
- Table of Contents
- Root System Chapter — Contains the root block diagram and properties of each block in the root diagram
- Subsystems Chapter -- Contains the diagram and block properties of each subsystem of the model
- Stateflow Charts Chapter -- Contains charts and chart object properties of each chart in the model

### 1 Import the API functions.

To eliminate the need to use fully qualified names of Report, Finder, and DOM API functions, use these statements. For example, instead of using `slreportgen.finder.BlockFinder`, you can use `BlockFinder`.

```
import slreportgen.report.*
import slreportgen.finder.*
import mlreportgen.report.*
```

### 2 Load the `sf_car` model.

```
model = load_system('sf_car');
```

### 3 Create a report object.

Use a Simulink report constructor (`slreportgen.report.Report`) to create a report object to hold the contents of the report. You must fully qualify the name of the constructor to distinguish it from the MATLAB report constructor (`mlreportgen.report.Report`). Set the name of the report to `sdd_` followed by the value of the Name property of the model.

```
rpt = slreportgen.report.Report(['sdd_' ...
    get_param('sf_car', 'Name')], 'pdf');
```

To customize properties that apply to the whole report, see `slreportgen.report.Report`.

### 4 Add a title page.

Use a title page reporter constructor (`mlreportgen.report.TitlePage`) to create a title page reporter. This reporter generates a title page based on its properties. Set the `Title`, `Subtitle`, and `Author` properties to character arrays that specify the report title, subtitle, and author, respectively.

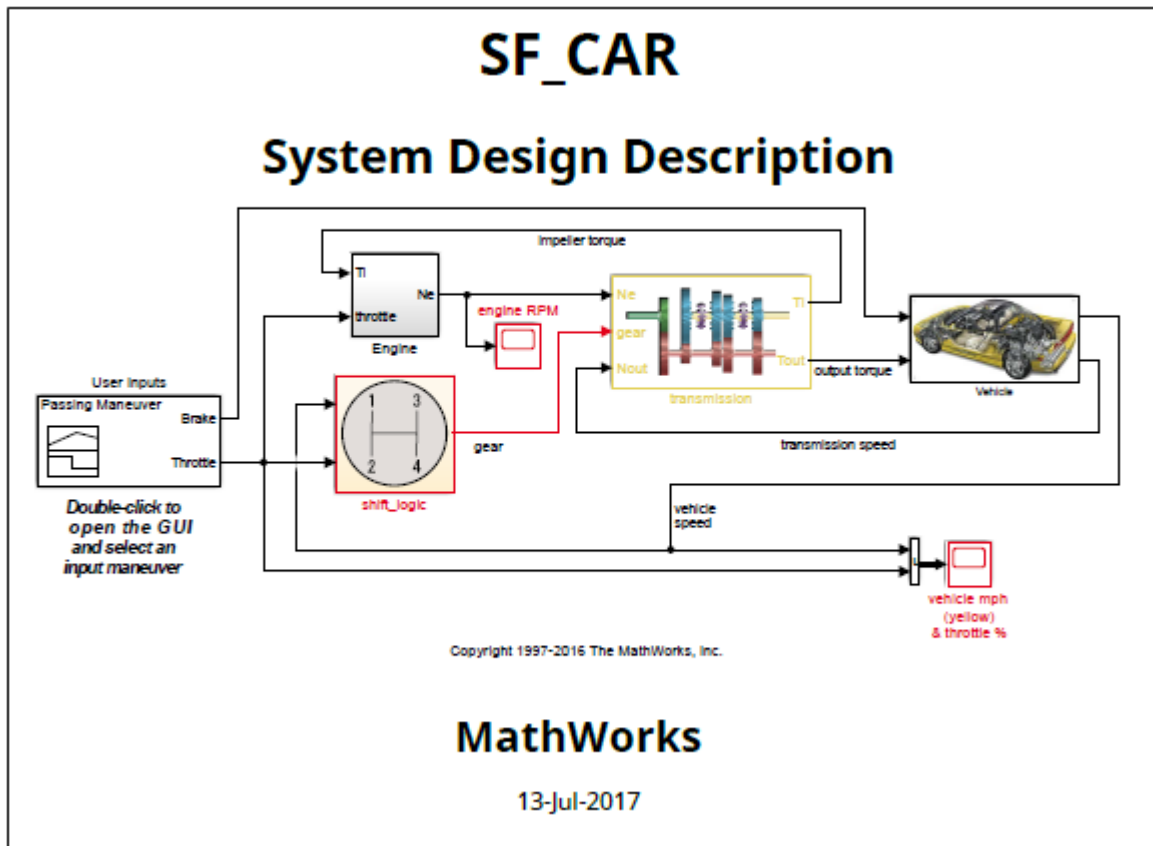
Use a diagram reporter constructor (`slreportgen.report.Diagram`) to create a diagram reporter for this model. This reporter generates an image of the block diagram of the model. To include this image on the report title page, assign the diagram reporter to the `Image` property of the title page reporter. Then, add the title page to the report.

```
tp = TitlePage;
tp.Title = upper(get_param(model, 'Name'));
```

```

tp.Subtitle = 'System Design Description';
tp.Author = 'MathWorks';
tp.Image = Diagram(model);
append(rpt,tp);

```



To customize additional title page properties, see `mlreportgen.report.TitlePage`.

##### 5 Add a table of contents.

Use a table of contents (TOC) reporter constructor to create a TOC reporter. This reporter generates a TOC for the report. Add the TOC reporter to the report.

```

toc = TableOfContents;
append(rpt,toc);

```

Table of Contents	
<a href="#">Chapter 1. RootSystem</a> .....	1
<a href="#">1.1. Engine</a> .....	1
<a href="#">1.2. Mux</a> .....	1
<a href="#">1.3. User Inputs</a> .....	1
<a href="#">1.4. Vehicle</a> .....	2
<a href="#">1.5. engine RPM</a> .....	2
<a href="#">1.6. shift logic</a> .....	2
<a href="#">1.7. transmission</a> .....	3
<a href="#">1.8. vehicle mph (yellow) &amp; throttle %</a> .....	3
<a href="#">Chapter 2. Subsystems</a> .....	4
<a href="#">2.1. Engine</a> .....	4
<a href="#">2.2. Mux</a> .....	4
<a href="#">2.3. User Inputs</a> .....	4
<a href="#">2.4. Vehicle</a> .....	4
<a href="#">2.5. engine RPM</a> .....	5
<a href="#">2.6. shift logic</a> .....	5
<a href="#">2.7. transmission</a> .....	5
<a href="#">2.8. vehicle mph (yellow) &amp; throttle %</a> .....	5

To customize the table of contents, see `mlreportgen.report.TableOfContents`.

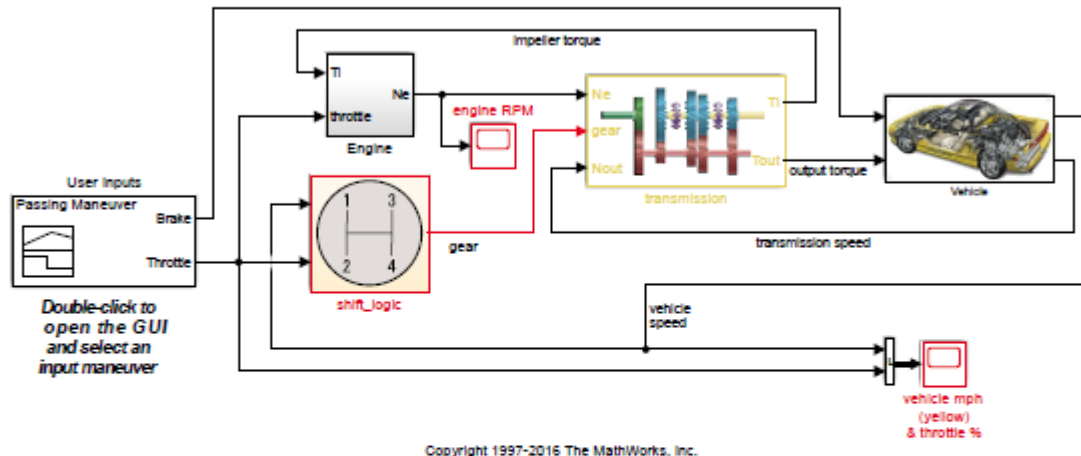
#### 6 Add a chapter for the root system.

Use a chapter constructor (`mlreportgen.report.Chapter`) to create a chapter reporter. This reporter generates a chapter based on its `Title` and `Content` properties. The reporter automatically numbers the chapter title. The chapter reporter also generates the chapter page headers and footers and its page numbers.

Add a model diagram reporter to the chapter. This reporter returns an image of the block diagram of the model that you add to the chapter.

```
ch = Chapter("Title", "RootSystem");
append(ch, Diagram(model));
```

# Chapter 1. RootSystem



## 1.1. Engine

**Table 1.1. sf\_car/Engine Properties**

Property	Value
Type	System

For information on customizing chapters, see `mlreportgen.report.Chapter`.

- 7 Add chapter sections for each root system block.

Use the block finder constructor (`slreportgen.report.BlockFinder`) to create a block finder for the root diagram. Then, use the `find` function of the block finder. The `find` function returns an array of block result objects (`slreportgen.report.BlockResult`), each of which contains a block.

Loop through the block result objects. For each result, construct a section reporter (`mlreportgen.report.Section`). This reporter generates a numbered report section based on its `Title` and `Content` properties. Set the section `Title` property to the name of the block on which it reports.

Add the current block result to the section reporter. Adding the result sets the section reporter `Content` property to a `simulink.report.SimulinkObjectProperties` reporter. This `SimulinkObjectProperties` reporter generates a table of the properties of the current block, which is then added to the section. Add each subsection to the parent chapter. Then add the chapter to the report.

```
blkFinder = BlockFinder(model);
blocks = find(blkFinder);
for block = blocks
```

```

    section = Section("Title", ...
        strrep(block.Name, newline, ' '));
    append(section,block);
    append(ch,section);
end
append(rpt,ch);

```

## 1.1. Engine

**Table 1.1. sf\_car/Engine Properties**

Property	Value
Type	System

## 1.2. Mux

**Table 1.2. sf\_car/Mux Properties**

Property	Value
Type	Block
Block Type	Mux
Number of inputs	2
Display option	none

For information finding blocks and how to customize sections, see `slreportgen.finder.BlockFinder` and `mlreportgen.report.Section`, respectively.

- 8 Add a chapter for subsystems.

Create a chapter for the subsystems of the model and the blocks in each subsystem.

```
ch = Chapter("Title", "Subsystems");
```

- 9 Find subsystem diagrams in the model.

Find all subsystem diagrams in the model. The finder returns an array of `DiagramResult` objects, each of which contains a `Diagram` reporter that creates a snapshot of the subsystem diagram.

```
sysdiagFinder = SystemDiagramFinder(model);
sysdiagFinder.IncludeRoot = false;
```



# Chapter 2. Subsystems

## 2.1. Engine

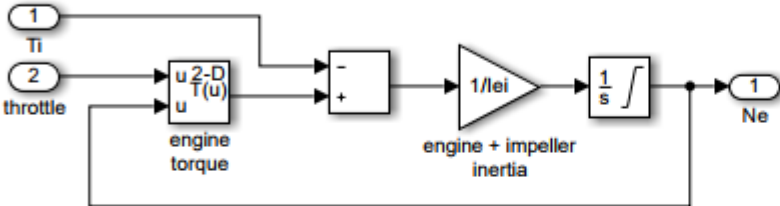


Figure 2.1. Engine

### 2.1.1. Ti

Table 2.1. sf\_car/Engine/Ti Properties

Property	Value
Type	Block
Block Type	Inport
Port number	1
Port dimensions (-1 for inherited)	-1
Sample time (-1 for inherited)	-1
Data type	Inherit: auto

For more information, see `slreportgen.finder.SystemDiagramFinder` and `slreportgen.finder.DiagramResult`

10 Add results to chapter sections.

Using loops, create a chapter section for each subsystem. Find the blocks and block elements in each subsystem. Add a table of block elements to each chapter section and add each section to the chapter. Then, add the chapter to the report.

```

while hasNext(sysdiagFinder)
    system = next(sysdiagFinder);
    section1 = Section("Title",system.Name);
    append(section1,system);

    blkFinder1 = BlockFinder(system);
    elems = find(blkFinder1);
    for elem = elems

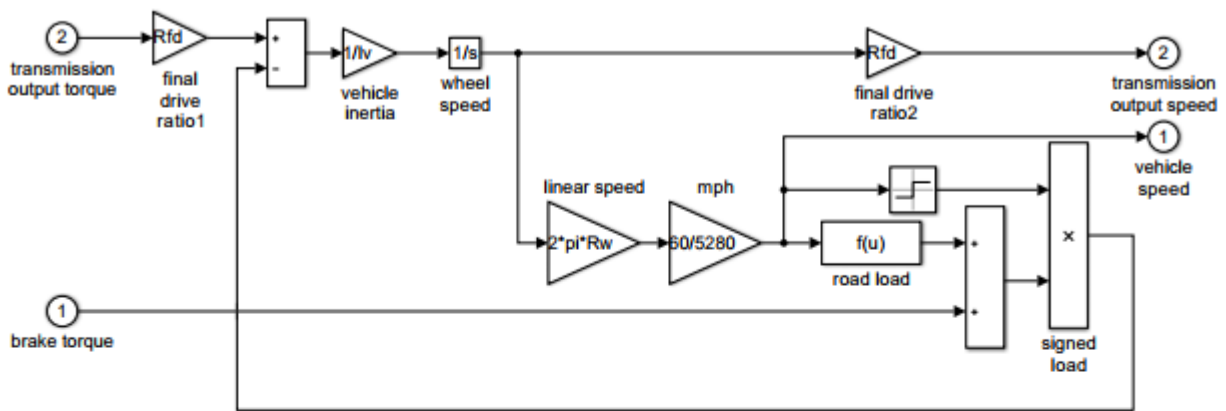
```

```

        section2 = Section("Title",...
            strep(elem.Name,newline,' '));
        append(section2,elem);
        append(section1,section2);
    end
    append(ch,section1);
end
append(rpt,ch);

```

## 2.2. Vehicle



Vehicle

Figure 2.3. Vehicle

### 2.2.1. brake torque

Table 2.9. sf\_car/Vehicle/brake torque Properties

Property	Value
Type	Block
Block Type	Inport
Port number	1
Port dimensions (-1 for inherited)	-1
Sample time (-1 for inherited)	-1
Data type	Inherit: auto

### 2.2.2. transmission output torque

---

**Note** Simulink finders can operate in either array or iterator mode. In array mode, use the finder `find` function to return the search results as an array of results. In iterator mode, use the finder `hasNext` and `next` functions to return the search results one-by-one. Use iterator mode when searching for diagrams in models that have many model references. Iterator mode closes a model after compiling and searching it, whereas `find` mode keeps all the models that it searches open. Having many open models can potentially consume all system memory and slow report generation. Although the model used in this example does not contain model references, the example uses iterator mode to illustrate its syntax.

---

**11** Add a chapter for Stateflow charts and objects.

Find all Stateflow charts in the model. Create a chapter. Using loops, add subsections for each chart. Find all the elements in each chart and add them to the subsections. Then, add the section to the chapter and the chapter to the report.

```
ch = Chapter("Title", "Stateflow Charts");

chdiagFinder = ChartDiagramFinder(model);
while hasNext(chdiagFinder)
    chart = next(chdiagFinder);
    section = Section("Title", chart.Name);
    append(section, chart);

    objFinder = StateflowDiagramElementFinder(chart);
    sfObjects = find(objFinder);
    for sfObj = sfObjects
        title = sfObj.Name;
        if isempty(title)
            title = sfObj.Type;
        end
        objSection = Section("Title", title);
        append(objSection, sfObj);
        append(section, objSection);
    end
    append(ch, section);
end
append(rpt, ch);
```

For information on chart and diagram element finders, see `slreportgen.finder.ChartDiagramFinder` and `slreportgen.finder.StateflowDiagramElementFinder`.

# Chapter 3. Stateflow Charts

## 3.1. shift\_logic

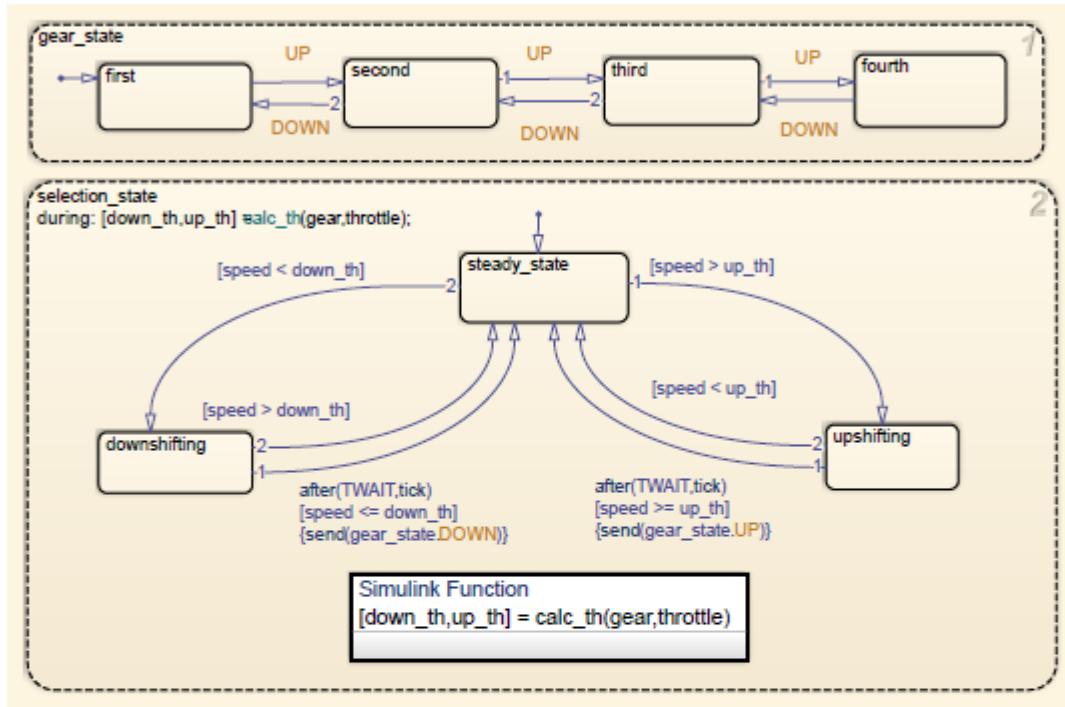


Figure 3.1. shift\_logic

### 3.1.1. gear\_state

Table 3.1. sf\_car/shift\_logic/gear\_state Properties

Property	Value
Type	AND State
Label	gear_state
Events	DOWN UP

### 3.1.2. selection\_state

- 12 Close the report, run the report, and close the model.

```

close(rpt);
rptview(rpt);
close_system(model);

```

The complete code is:

```

import slreportgen.report.*
import slreportgen.finder.*
import mlreportgen.report.*
model = load_system('sf_car');
rpt = slreportgen.report.Report(['sdd_'...
    get_param('sf_car', 'Name')], 'pdf');

tp = TitlePage;
tp.Title = upper(get_param(model, 'Name'));
tp.Subtitle = 'System Design Description';
tp.Author = 'MathWorks';
tp.Image = Diagram(model);
append(rpt, tp);
toc = TableOfContents;
append(rpt, toc);

ch = Chapter("Title", "RootSystem");
append(ch, Diagram(model));
blkFinder = BlockFinder(model);
blocks = find(blkFinder);
for block = blocks
    section = Section("Title", ...
        strep(block.Name, newline, ' '));
    append(section, block);
    append(ch, section);
end
append(rpt, ch);

ch = Chapter("Title", "Subsystems");
sysdiagFinder = SystemDiagramFinder(model);
sysdiagFinder.IncludeRoot = false;

while hasNext(sysdiagFinder)
    system = next(sysdiagFinder);
    section1 = Section("Title", system.Name);
    append(section1, system);

    blkFinder1 = BlockFinder(system);
    elems = find(blkFinder1);
    for elem = elems
        section2 = Section("Title", ...
            strep(elem.Name, newline, ' '));
        append(section2, elem);
        append(section1, section2);
    end
    append(ch, section1);
end
append(rpt, ch);

ch = Chapter("Title", "Stateflow Charts");
chdiagFinder = ChartDiagramFinder(model);
while hasNext(chdiagFinder)

```

```
chart = next(chdiagFinder);
section = Section("Title",chart.Name);
append(section,chart);

objFinder = StateflowDiagramElementFinder(chart);
sfObjects = find(objFinder);
for sfObj = sfObjects
    title = sfObj.Name;
    if isempty(title)
        title = sfObj.Type;
    end
    objSection = Section("Title",title);
    append(objSection,sfObj);
    append(section,objSection);
end
append(ch,section);
end
append(rpt,ch);

close(rpt);
rptview(rpt);
close_system(model);
```

### **See Also**

rptview

## Report on MATLAB Function

The Report API provides multiple ways to report on Simulink MATLAB Function blocks and Stateflow MATLAB functions. To report detailed information, use the `slreportgen.report.MATLABFunction` reporter. This reporter reports properties, arguments, function code, function symbols, and supporting functions.

Other ways to report on MATLAB Function blocks or Stateflow MATLAB functions are by using the `SimulinkObjectProperties` or `StateflowObjectProperties` reporter, respectively. These reporters, however, do not provide function code formatting or report function symbols, supporting functions, or arguments. Use these reporters if you want only property information.

The Report API provides finders for finding blocks and Stateflow elements, including MATLAB Functions, throughout a model or chart. These finders are the `BlockFinder`, `DiagramElementFinder`, and `StateflowDiagramElementFinder`.

These examples show how to use a finder in your report generator program. For cases where you know the handle or path to a MATLAB Function, you do not need to use a finder (see `slreportgen.report.MATLABFunction` for examples).

### Find and Report on MATLAB Function Blocks

Use Report API finders and the `slreportgen.report.MATLABFunction` reporter to report on MATLAB Function blocks.

Use the `BlockFinder` to find all blocks of type `SubSystem`, which includes MATLAB Function blocks. If you search for all block types, the `BlockFinder` can take more time to return results than if you limit the search to `SubSystem` block types.

```
blkfinder = slreportgen.finder.BlockFinder(model_name);
blkfinder.BlockTypes = "SubSystem";
blks = find(blkfinder);
```

Then, loop through the returned `SubSystem` blocks to test whether the block is a MATLAB Function block. Create a `MATLABFunction` reporter for each MATLAB Function block, set desired properties, and add each result to a report.

```
for i=1:length(blks)
    block = blks(i).Object;
    if slreportgen.utils.isMATLABFunction(block)
        rptr = MATLABFunction(block);
        rptr.IncludeArgumentProperties = true;
        add(rpt,rptr);
    end
end
```

This code is an example of a report generator program that finds and reports MATLAB Function blocks.

```
import slreportgen.report.*
import slreportgen.finder.*

model_name = 'sldemo_eml_galaxy';
load_system(model_name);
rpt = slreportgen.report.Report;
```

```
blkfinder = BlockFinder(model_name);
blkfinder.BlockTypes = "SubSystem";
blks = find(blkfinder);

for i=1:length(blks)
    block = blks(i).Object;
    if slreportgen.utils.isMATLABFunction(block)
        rptr = MATLABFunction(block);
        rptr.IncludeArgumentProperties = true;
        add(rpt,rptr);
    end
end

close(rpt);
close_system(model_name);
rptview(rpt);
```

This image shows a section of the report output for one of the MATLAB Function blocks. It shows the block properties table, the summary table for one of the arguments, and a portion of the function script. In the actual output, all of the argument tables appear before the function script.



**Table 0.1. Apply Newtonian gravitation Object Properties**

Property	Value
Update Method	INHERITED
Sample Time	
Support variable-size arrays	0
Saturate on integer overflow	1
Treat these inherited Simulink signal types as fixed-point objects	Fixed-point
MATLAB Function block fimath	Other:UserSpecified
Input fimath	fimath(... 'RoundMode', 'floor',... 'OverflowMode', 'wrap',... 'ProductMode', 'KeepLSB', 'ProductWordLength', 32,... 'SumMode', 'KeepLSB', 'SumWordLength', 32,... 'CastBeforeSum', true)
Description	

**Table 0.2. Apply Newtonian gravitation Argument Summary**

Name	Scope	Port	Data Type	Size
heavy1	Output	1	double	[700, 8]
light1	Output	2	double	[700, 8]
light	Input	1	double	[700, 8]
heavy	Input	2	double	[700, 8]

**Apply Newtonian gravitation Function Script**

```
function [heavy1,light1] = ApplyGravity(light,heavy)

G = 6.672E-11; % Nm^2/kg^2 (Gravitational constant)

YearInSeconds = 365*24*60*60;
timeStep = 2000000*YearInSeconds;

n = size(heavy,1);

heavy1 = heavy;
light1 = light;

for i = 1:n,
    mi = heavv(i,1):
```

## Find and Report on Stateflow MATLAB Functions

Use the `StateflowDiagramElementFinder` and the `slreportgen.report.MATLABFunction` reporter to find and report on Stateflow MATLAB functions.

```
elemfinder = StateflowDiagramElementFinder(chart_name);  
elemfinder.Types = "emfunction";  
elems = find(elemfinder);
```

Then, loop through the returned MATLAB functions. Create a `MATLABFunction` reporter for each MATLAB function and add it to a report.

```
for i = 1:length(elems)  
    rptr = MATLABFunction(elems(i).Object);  
    add(rpt,rptr);  
end
```

This code is an example of a report generator program that finds and reports MATLAB functions in Stateflow charts.

```
import slreportgen.report.*  
import slreportgen.finder.*  
  
model_name = 'sf_server';  
load_system(model_name);  
chart_name = 'sf_server/transmitter';  
rpt = slreportgen.report.Report;  
  
elemfinder = StateflowDiagramElementFinder(chart_name);  
elemfinder.Types = "emfunction";  
elems = find(elemfinder);  
  
for i = 1:length(elems)  
    rptr = MATLABFunction(elems(i).Object);  
    add(rpt,rptr);  
end  
  
close(rpt);  
close_system(model_name);  
rptview(rpt);
```

This image shows a section of the report output for one of the MATLAB Function blocks. It shows the object properties table and a portion of the function script.

**Table 0.1. init\_gui Object Properties**

Property	Value
Saturate on integer overflow	1
MATLAB Function block filename	Other:UserSpecified
Input filename	fimath(... 'RoundMode', 'floor',... 'OverflowMode', 'wrap',... 'ProductMode', 'KeepLSB', 'ProductWordLength', 32,... 'SumMode', 'KeepLSB', 'SumWordLength', 32,... 'CastBeforeSum', true)
Description	

**init\_gui Function Script**

```
function init_gui

coder.extrinsic('set', 'findall', 'figure', 'delete', 'axes', 'text', 'rectangle')
fig = sf_server_handle_map('fig');

if is_invalid_gui_obj_handle(fig)
    % see if there is an existing figure
    figTemp = findall(0, 'type', 'figure', 'tag', 'SF_SERVER');
    if ~isempty(figTemp)
        deleteFig = 0;

        fig = figTemp;
        lastObj = fig;
        sf_server_handle_map('fig', fig);
    end
end
```

**Customize MATLAB Function Reporter Output**

You can customize the output of a MATLAB Function reporter in the same way that you customize any report or reporter:

- Use DOM classes — Specify formats using DOM classes, such as `mlreportgen.dom.Paragraph`, and use them in your program. For example, this code sets the appearance of the function script.

```
rptr = slreportgen.report.MATLABFunction;
paraScript = mlreportgen.dom.Paragraph;
paraScript.FontFamilyName = 'Arial';
paraScript.FontSize = '12pt';
paraScript.Color = 'blue';
rptr.FunctionScript = paraScript;
```

- Edit a copy of the default template — The advantage of saving customizations in a new template is that you can easily reuse those customizations by using that template for another report generator program. The template and style sheets for the MATLABFunction reporter are located in the `matlab\toolbox\shared\slreportgen\rpt\rpt\+slreportgen\+report\@MATLABFunction\resources\templates` folder.

This example shows the steps for copying and editing a MATLABFunction reporter html template.

- 1 Create a copy of the default html template. In this example, the template package is saved as `myHTMLTemplate.htm` in the current working folder.

```
mfunction = slreportgen.report.MATLABFunction;
mfunction.createTemplate('myHTMLTemplate', 'html');
```

- 2 Unzip the template package. The unzipped template package is a folder of document, style sheet, and image files. In this example, the template package folder is saved to the current working folder.

```
unzipTemplate('myHTMLTemplate.htm');
```

- 3 From the `stylesheets` folder, open the `root.css` file in a text editor. The `root.css` file contains the default styles for the MATLABFunction reporter. The beginning of the file and the first style are:

```
/******
```

```
* MATLABFunction Reporter
```

```
*****/
```

```
/* Default style for the MATLAB function script title */
```

```
.MATLABFunctionFunctionScriptTitle {
```

```
font-family: 'Noto Sans', 'Noto Sans CJK JP', 'Noto Sans CJK SC', 'Noto Sans CJK KR';
```

```
font-weight: bold;
```

```
margin-top: 10pt;
```

```
color: black;
```

```
white-space: pre;
```

```
}
```

- 4 Edit the styles as desired. In this example, the top margin above the function script title is increased to 20 points and the color of the title to blue.

```
/******
```

```
MATLABFunction Reporter
```

```
*****/
```

```
/* Default style for the MATLAB function script title */
```

```
.MATLABFunctionFunctionScriptTitle {
```

```
font-family: 'Noto Sans', 'Noto Sans CJK JP', 'Noto Sans CJK SC', 'Noto Sans CJK KR';  
font-weight: bold;  
margin-top: 12pt;  
color: blue;  
white-space: pre;  
}
```

**5** Save the file.

**6** Zip the template folder into a template package. For this example, the template package is `myHTMLTemplate.htmx`.

```
zipTemplate('myHTMLTemplate');
```

**7** In your report generator program, to use the saved template, specify the template source.

```
mfunction.TemplateSrc = 'myHTMLTemplate';
```

See “Templates for DOM API Report Programs” for additional information.

## See Also

MATLAB Function | `slreportgen.finder.BlockFinder` |  
`slreportgen.finder.DiagramElementFinder` |  
`slreportgen.finder.StateflowDiagramElementFinder` |  
`slreportgen.report.MATLABFunction` | `unzipTemplate` | `zipTemplate`

## Use Simulink Report Explorer Components in a Report API Report

The `RptFile` reporter lets you use Simulink Report Explorer components in a Report API-based report program. This reporter is useful if your report program needs to generate content for which a Report Explorer component exists but for which no Report API reporter is available. For example, the Report Explorer includes a component named Block Type Count that generates the number of each type of block that a model contains. No equivalent Report API reporter exists.

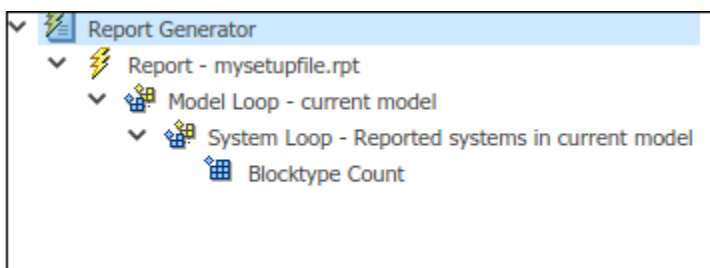
This example shows how to use the `RptFile` reporter to include a count of the types of blocks used in the `f14` Simulink model. This information is obtained from the Report Explorer Block Type Count component and is presented in tables in the generated Report API report.

### Create the Report Explorer Setup File

Create a Report Explorer setup file that includes a Block Type Count component. For information about creating a report setup file, see “Report Setup”.

- 1 Type report to open the Report Explorer.
- 2 In the panel on the right, click **Create and edit a Report file**. Save the file as `mysetupfile.rpt`.
- 3 From the Simulink folder in the middle panel, add a Model Loop component to your report. Set the **Model name** to `Current block diagram`.
- 4 From the Simulink folder in the middle panel, add a System Loop component as a child of the Model Loop. Set these options:
  - **Loop on Systems** — Select systems automatically
  - **Include subsystems in Simulink functions** — selected
  - **Sort Systems** — By system depth
- 5 From the Simulink folder in the middle panel, add a Block Type Count component as a child of the System Loop. Set these options:
  - **Table title** — Block Type Count
  - **Show block names in title** — selected
  - **Sort table** — Alphabetically by block type
- 6 Save the file.

The `mysetupfile.rpt` hierarchy is



## Create a Report Generator Program

These steps describe how to create a Report Generator program that includes a `RptFile` reporter for the `mysetupfile.rpt` Report Explorer setup file.

---

**Note** The full program is listed after the steps.

---

- 1 To eliminate the need to use fully-qualified names of the report, finder, and utility functions, import the API functions. For example, instead of using `mlreportgen.report.TitlePage`, you can use `TitlePage`.

```
import slreportgen.report.*
import slreportgen.finder.*
import mlreportgen.report.*
import mlreportgen.utils.*
```

- 2 Load the `f14` model.

```
model = "f14";
load_system(model);
```

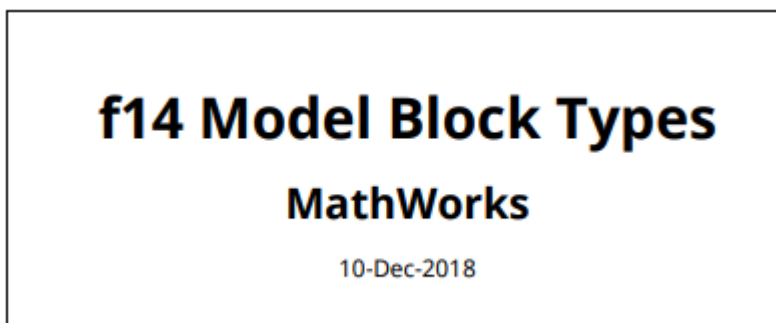
- 3 Create a report object to hold the contents of the report. Use a Simulink report constructor (`slreportgen.report.Report`) to create a report object. You must fully qualify the name of the constructor to distinguish it from the MATLAB report constructor (`mlreportgen.report.Report`). Specify the name of the report as "My Report" and the output type as PDF.

```
rpt = slreportgen.report.Report("MyReport","pdf");
```

- 4 Add a title page and table of contents to the report.

```
titlepg = TitlePage();
titlepg.Title = "f14 Model Block Types ";
titlepg.Author = "MathWorks";
add(rpt,titlepg);
```

```
toc = TableOfContents;
add(rpt,toc);
```



## Table of Contents

Chapter 1. f14.....	1
Chapter 2. f14/Aircraft Dynamics Model.....	2
Chapter 3. f14/Controller.....	3
Chapter 4. f14/Dryden Wind Gust Models.....	4
Chapter 5. f14/Dryden Wind Gust Models/Band-Limited White Noise.....	5
Chapter 6. f14/Nz pilot calculation.....	6

- 5 Find all systems in the model.

```
sysdiag_finder = SystemDiagramFinder(model);
found_diags = find(sysdiag_finder);
```

- 6 Use a for loop to create a separate chapter for each system and include a system snapshot with a single-line caption.

Create a RptFile reporter based on `mysetupfile.rpt`. The reporter generates a table of block type counts for the current system. Add the RptFile reporter to the chapter and add the chapter to the report.

```
for sysdiag = found_diags
    chap = Chapter(sysdiag.Path);

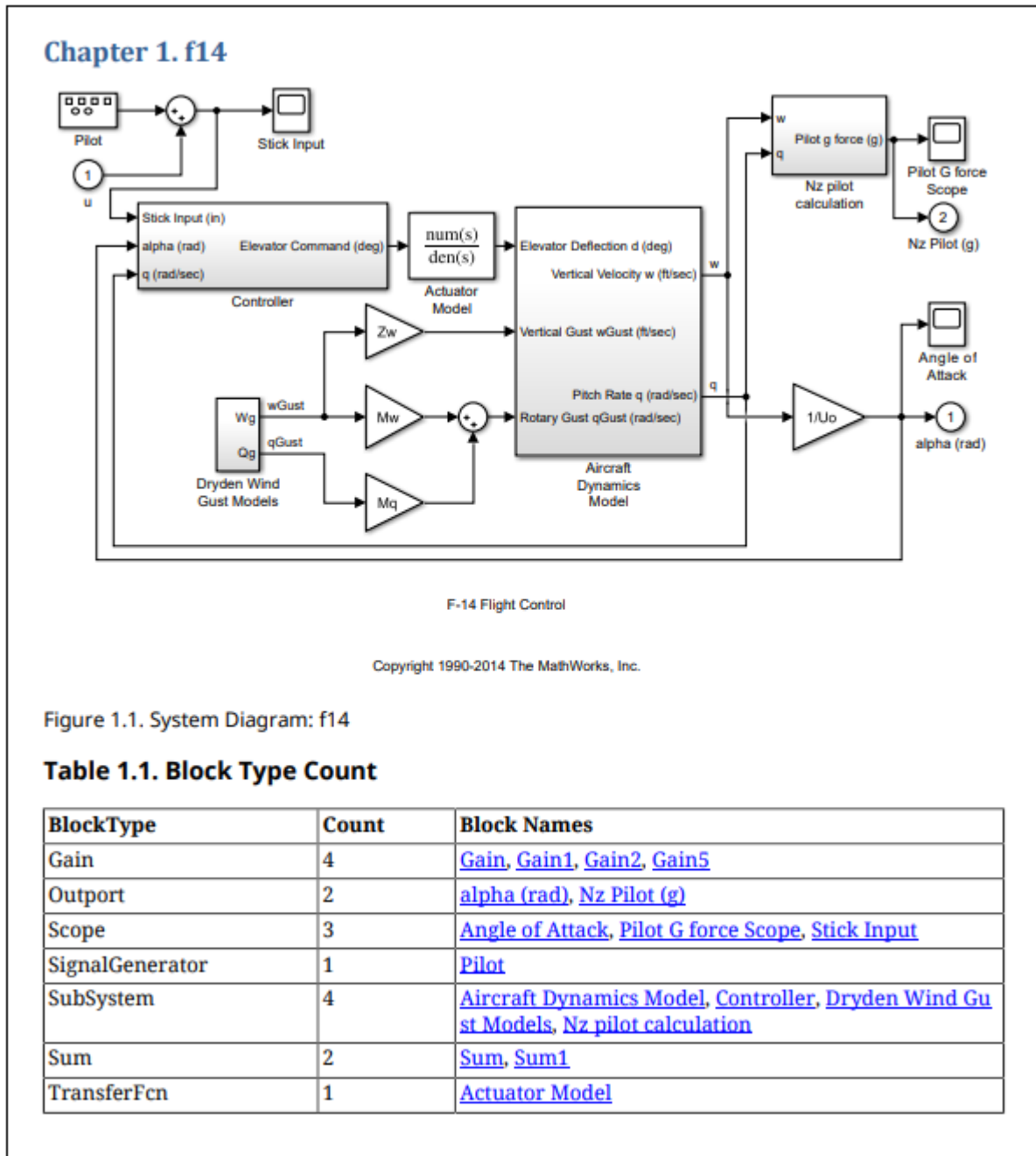
    snapshot = Diagram(sysdiag.Path);
    oneline = makeSingleLineText(sysdiag.Name);
    snapshot.Snapshot.Caption = strcat...
        ("System Diagram: ", oneline);
    add(chap, snapshot);

    rptFile = RptFile("mysetupfile.rpt");
    rptFile.Model = model;
    rptFile.System = sysdiag.Path;

    add(chap, rptFile);
    add(rpt, chap);
end
```

The first time this loop runs during report generation, a snapshot and block count of the top-level system of the model is added to the report.





7 Close and view the report.

```
close(rpt);
rptview(rpt);
```

The full program is

```
import slreportgen.report.*
import slreportgen.finder.*
import mlreportgen.report.*
import mlreportgen.utils.*
```

```
model = "f14";
load_system(model);

rpt = slreportgen.report.Report("MyReport","pdf");

titlepg = TitlePage();
titlepg.Title = "f14 Model Block Types ";
titlepg.Author = "MathWorks";
add(rpt,titlepg);

toc = TableOfContents;
add(rpt,toc);

sysdiag_finder = SystemDiagramFinder(model);
found_diags = find(sysdiag_finder);
for sysdiag = found_diags
    chap = Chapter(sysdiag.Path);

    snapshot = Diagram(sysdiag.Path);
    oneline = makeSingleLineText(sysdiag.Name);
    snapshot.Snapshot.Caption = strcat...
        ("System Diagram: ",oneline);
    add(chap,snapshot);

    rptFile = RptFile("mysetupfile.rpt");
    rptFile.Model = model;
    rptFile.System = sysdiag.Path;
    add(chap,rptFile);

    add(rpt,chap);
end

close(rpt);
rptview(rpt);
```

## Report Systems Hierarchically

This example shows how to create a report with sections that are numbered according to system hierarchy. Each section contains a system snapshot and subsections that contain subsystem snapshots. To create such a section, create a section object, add a diagram snapshot, and then add subsystem sections. To create the subsystem sections, again create a section, add a subsystem diagram snapshot and then add its subsystem sections. The algorithm to create the sections is recursive. This example creates and uses a local function called `createSystemSection`, which implements the recursive algorithm.

### Create Hierarchical Report Using `createSystemSection` Function

Open a model.

```
model = "sf_car";
open_system(model);
```

Create and open a report object.

```
% Change the output type from "pdf" to "docx" or "html" to create a
% Word or HTML report, respectively.
rpt = slreportgen.report.Report("myreport", "pdf");
open(rpt);
```

Add a title page.

```
titlepage = mlreportgen.report.TitlePage();
titlepage.Title = "Hierarchical Report";
add(rpt, titlepage);
```

Add a table of contents with the number of levels set to 6, which is the maximum.

```
toc = mlreportgen.report.TableOfContents();
toc.TOCObj.NumberOfLevels = 6;
add(rpt, toc);
```

Create system sections for the model by calling the `createSystemSection` local function (see below). This function recursively calls itself to create sections for the subsystems.

```
section = createSystemSection(model);
add(rpt, section);
```

Generate and display the report.

```
close(rpt);
rptview(rpt);
```

### Define `createSystemSection` Local Function

A system section is composed of a system snapshot and its subsystems in subsections. To create a system section, find all systems one level deep by using an `slreportgen.finder.DiagramFinder` with a `SearchDepth` of 1.

```
function section = createSystemSection(sys)
    df = slreportgen.finder.DiagramFinder(sys);
    df.SearchDepth = 1;
```

```
% Use the finder in iterator mode. The next function returns search results
% one-by-one and the hasNext function determines when there are no more
% search results. To obtain the current system, call the next function
% once.
sysResult = next(df);

% Now, create a section using mlreportgen.report.Section with the system
% name as the title.
section = mlreportgen.report.Section( ...
    "Title", mlreportgen.utils.normalizeString(sysResult.Name));

% Add a system snapshot and a caption that shows the full diagram path.
% To include additional information about the system, add it to the
% section object.
diag = slreportgen.report.Diagram(sysResult.Object);
diag.Snapshot.appendCaption(sysResult.Path);
add(section, diag);

% To create subsections, loop through all subsystems and recursively call
% createSystemSection. Before calling createSystemSection, add a page break
% so each system starts on a new page. Note that adding a page break right
% after the system snapshot would add a blank page at the end of the report.
while hasNext(df)
    childSysResult = next(df);
    add(section, mlreportgen.dom.PageBreak());
    subSection = createSystemSection(childSysResult.Object);
    add(section, subSection);
end
end
```

## Customize Simulink Diagram Hyperlinks in HTML and PDF Reports

This example shows, for PDF and HTML reports, how to customize navigation hyperlinks of Simulink diagrams embedded in reports. By default, clicking on a diagram element navigates to the section of the report that documents that element. To specify a different destination for the hyperlinks, follow the procedure in this example.

### Set up the report and load a Simulink model

Import the DOM and Report API packages so you do not have to use long, fully-qualified class names.

```
import mlreportgen.dom.*
import slreportgen.report.*
```

Create and open a Simulink report.

```
rpt = Report("myreport", "pdf");
open(rpt);
```

Load a Simulink model.

```
model = "sf_car";
load_system(model);
```

### Include the sf\_car root system diagram using the Diagram reporter

The Diagram reporter overlays each element of the sf\_car diagram snapshot with a hyperlink to navigate to a section of the report that describes that element. The hyperlink and its ID are created using the element's path in the model. For example, a subsystem block, such as Engine or transmission, includes a hyperlink used for navigating to the corresponding subsystem diagram snapshot in the report.

```
diag1 = Diagram(model);
diag1.Snapshot.Caption = strcat("Diagram snapshot for root system: ",model);
add(rpt,diag1);
add(rpt,PageBreak);
```

### Include the sf\_car/Engine subsystem diagram using the Diagram reporter

This reporter prefaces the report object that it creates with a hyperlink target whose ID is also based on the reported element's path in the model. The Diagram reporter (diag1) for root system sf\_car also uses the same ID to create the hyperlink on the Engine block in the snapshot. So, clicking on the Engine block automatically targets to this subsystem diagram snapshot in the report.

```
engine = strcat(model, "/", "Engine");

diag2 = Diagram(engine);
diag2.Snapshot.Caption = strcat("Diagram snapshot for subsystem: ",engine);
add(rpt,diag2);
add(rpt,PageBreak);
```

### Include the sf\_car/transmission subsystem diagram using the Diagram reporter

Clicking on transmission block in the sf\_car root system diagram snapshot navigates to the transmission subsystem diagram snapshot in the report.

To customize the target for the hyperlink, remove the link target for this reporter by setting the `LinkTarget` property of the reporter to an empty string. This ensures that clicking on the transmission block in the `sf_car` root system diagram snapshot does not navigate to the transmission subsystem diagram. Then create a custom target for the hyperlink as described in the next section.

```
transmission = strcat(model, "/", "transmission");

diag3 = Diagram(transmission);
diag3.LinkTarget = "";
diag3.Snapshot.Caption = strcat("Diagram snapshot for subsystem: ", transmission);
add(rpt, diag3);
add(rpt, PageBreak);
```

### Create custom target for the `sf_car/transmission` block hyperlink

To set a new target for the hyperlink, first use the `slreportgen.utils.getObjectID` function to obtain the same ID that the `Diagram` reporter uses. Use the `SimulinkObjectProperties` reporter to generate a property table for the `transmission` block. Change the `LinkTarget` property of the reporter to the ID obtained with `slreportgen.utils.getObjectID`. The `Diagram` reporter (`diag1`) for root system `sf_car` also uses the same ID to create the hyperlink on the `transmission` block in the snapshot, so clicking on the block now targets this block property table.

```
id = slreportgen.utils.getObjectID(transmission);

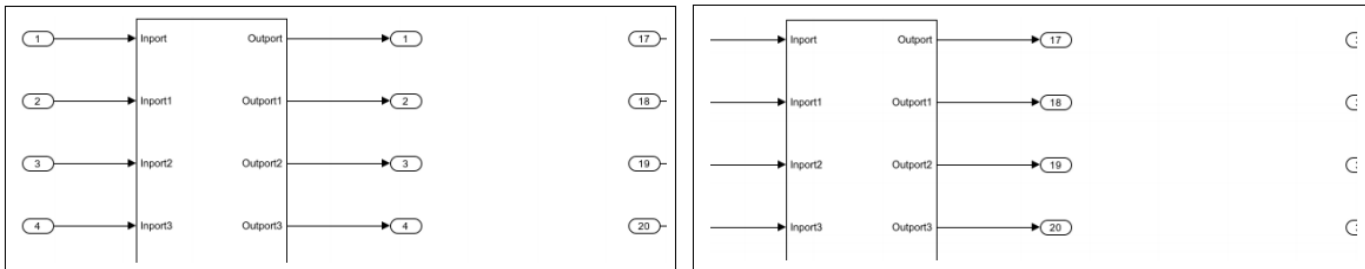
props = SimulinkObjectProperties(transmission);
props.LinkTarget = id;
add(rpt, props);
```

### Close and view the report

```
close(rpt);
rptview(rpt);
```

## Tile Simulink Diagrams

This example shows how to create a report with a large diagram that spans across multiple pages.



### Create Report with Image Tiles

Open a model with a large diagram.

```
model = 'slreportgen_demo_big_diagram';
open_system(model);
```

Create large image file to split into tiles.

```
imgFile = [model '.png'];
print('-dpng', ['-s' model], imgFile);
```

Create and open a report.

```
% To create a Word report, change the output type from "pdf" to "docx".
% To create an HTML report, change "pdf" to "html" or "html-file" for
% a multifile or single-file report, respectively.
rpt = slreportgen.report.Report('myreport2', 'pdf');
open(rpt);
```

Get the page layout information.

```
pageLayout = rpt.Document.CurrentPageLayout;
pageSize = pageLayout.PageSize;
pageMargins = pageLayout.PageMargins;
```

Set the page header and footer to 0 inches to maximize space.

```
pageMargins.Header = '0in';
pageMargins.Footer = '0in';
```

Determine the image tile size that fits onto the page. The optimal tile size is the page size minus the page margins, gutters, headers and footers. Also, adjust the tile height to allow 0.5 inches for the caption. Note that for PDF documents, MATLAB Report Generator defines one inch as equal to 96 pixels.

```
dpi = 96;
units = mlreportgen.utils.units;

tileHeight = units.toPixels(pageSize.Height, 'Resolution', dpi) ...
    - units.toPixels(pageMargins.Top, 'Resolution', dpi) ...
    - units.toPixels(pageMargins.Bottom, 'Resolution', dpi) ...
```

```
- units.toPixels(pageMargins.Header, 'Resolution', dpi) ...
- units.toPixels(pageMargins.Footer, 'Resolution', dpi) ...
- units.toPixels('0.5in', 'Resolution', dpi);

tileWidth = units.toPixels(pageSize.Width, 'Resolution', dpi) ...
- units.toPixels(pageMargins.Left, 'Resolution', dpi) ...
- units.toPixels(pageMargins.Right, 'Resolution', dpi) ...
- units.toPixels(pageMargins.Gutter, 'Resolution', dpi);

tileSize = [tileWidth tileHeight];
```

Call the `sliceImage` local function (see below) to slice the large image file into image tiles.

```
tiles = sliceImage(imgFile, [tileWidth tileHeight]);
```

Add the tile images to the report. Also, also add a caption to indicate where the tile image belongs in relation to the overall image.

```
for i = 1:numel(tiles)
    tile = tiles{i};
    formalImage = mlreportgen.report.FormalImage(tile.File);
    formalImage.ScaleToFit = false;
    formalImage.Caption = sprintf('row: %d, col: %d', tile.Row, tile.Col);
    add(rpt, formalImage);
end
```

Generate and display the report.

```
close(rpt);
rptview(rpt);
```

### Define `slicelImage` Local Function

To slice an image file into tiles, read in the image file and copy tile-size parts into multiple image files.

```
function tiles = sliceImage(imgFile, tileSize)
    % Read in the image file and determine the number of row and column
    % tiles. Note that the image data is row-major, where the rows are
    % specified first and the columns are second.
    img = imread(imgFile);
    imgSize = size(img);

    imgRows = imgSize(1); % image height
    imgCols = imgSize(2); % image width

    tileNumRows = tileSize(2); % tile height
    tileNumCols = tileSize(1); % tile width

    numCols = ceil(imgCols / tileNumCols);
    numRows = ceil(imgRows / tileNumRows);

    % Preallocate the tile data structures.
    tiles = cell(1, numCols*numRows);

    % Determine the base filename to create the tile image filenames.
    [fPath, fName, fExt] = fileparts(imgFile);
    tileName = fullfile(fPath, fName);
```



```
% Iterate through all rows and columns.
count = 0;
for rowIdx = 1:numRows
    for colIdx = 1:numCols
        % Determine the starting and ending image data indices to copy
        % into the tile image. At the edges, the ending indices are
        % the number of rows and number of columns.
        rowStart = (rowIdx - 1) * tileNumRows + 1;
        rowEnd = rowStart + tileNumRows - 1;

        colStart = (colIdx - 1) * tileNumCols + 1;
        colEnd = colStart + tileNumCols - 1;

        if (rowEnd >= imgRows)
            rowEnd = imgRows;
        end
        nTileRows = rowEnd - rowStart + 1;

        if (colEnd >= imgCols)
            colEnd = imgCols;
        end
        nTileCols = colEnd - colStart + 1;

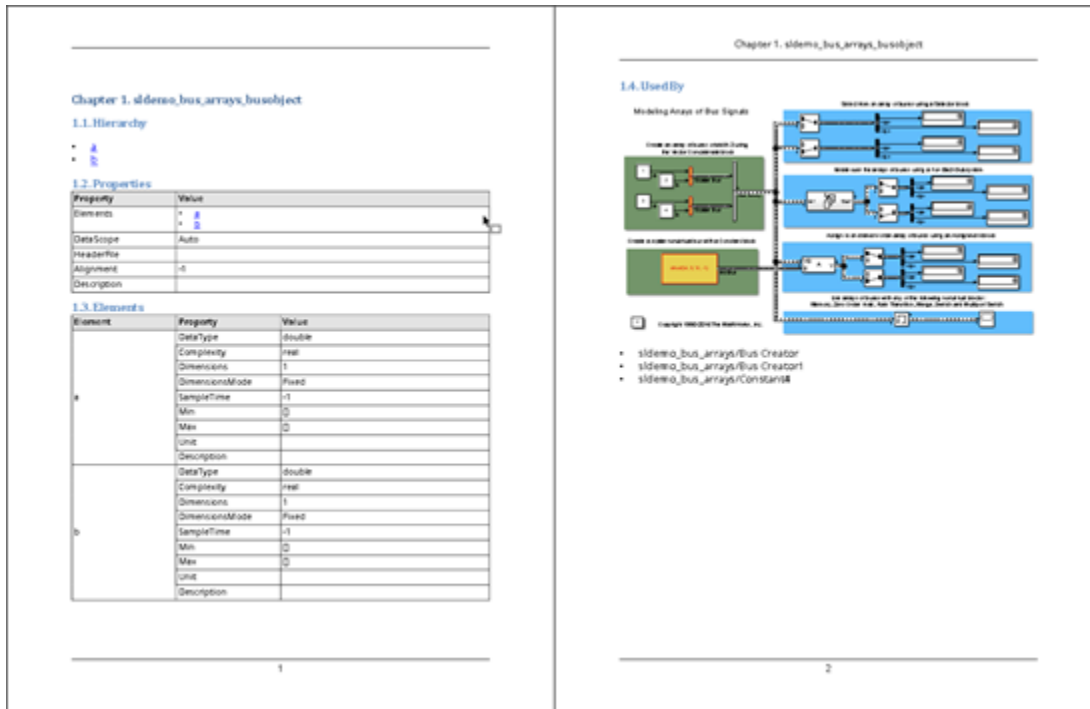
        % Copy the tile image data onto a white image tile.
        tileImg = uint8(255 * ones(tileNumRows, tileNumCols, 3));
        tileImg(1:nTileRows, 1:nTileCols, :) = img(rowStart:rowEnd,...
            colStart:colEnd, :);

        % Write out the image tile.
        outFile = sprintf('%s_%d_%d.%s', tileName, rowIdx, colIdx, fExt);
        imwrite(tileImg, outFile);

        % Create the tile data structure to describe the tile.
        count = count + 1;
        tiles{count} = struct( ...
            'File', outFile, ...
            'Row', rowIdx, ...
            'Col', colIdx);
    end
end
end
```

## Create a Simulink Bus Object Report

This example shows how to create a report that describes all of the bus objects used by a Simulink® model. This report creates a chapter for each bus object. Each chapter has a section for the bus object hierarchy, bus object properties table, bus elements properties table, and list of blocks that use the bus.



### Import Packages

Import the Report API packages so that you do not have to use long, fully qualified class names.

```
import mlreportgen.report.*
import slreportgen.finder.*
import slreportgen.report.*
```

### Open Model

Open a model that has bus objects.

```
model = "sldemo_bus_arrays";
open_system(model);
```

### Create Report

Create and open a report object. To create a Microsoft® Word, HTML, or single-file HTML report, change "pdf" to "docx", "html", or "html-file", respectively.

```
rpt = slreportgen.report.Report(model + "_bus_object_report", "pdf");
open(rpt);
```

Add a title page and a table of contents.

```

titlepage = TitlePage("Title", model + ": Bus Object Report", "Author", "John Doe");
add(rpt, titlepage);
toc = TableOfContents();
add(rpt, toc);

```

### Find and Report on Bus Objects

Find all the variables used in the model.

```
finder = ModelVariableFinder(model);
```

Loop through the variable finder results to find the bus objects and report on them. Use the `getVariableValue` method to identify which variables are bus objects. Use the `slreportgen.report.BusObject` reporter to report on the bus objects.

```

while hasNext(finder)
    result = next(finder);
    if isa(getVariableValue(result), "Simulink.Bus")
        % Create a Bus object reporter
        busReporter = BusObject(result);
        % Create a Chapter
        chapter = Chapter(busReporter.Name);
        % Add bus to chapter
        add(chapter, busReporter)
        % Add chapter to the report
        add(rpt, chapter);
    end
end

```

### Close Report

Close and view the report.

```

close(rpt);
rptview(rpt);

```

### View Sample Report

To see a more comprehensive bus object report, view the `asbhl20_bus_object_report.pdf` that is available with this example. You must have Aerospace Blockset™ to open the `asbhl20` model.

```
rptview asbhl20_bus_object_report.pdf
```

### See Also

```

getVariableValue | slreportgen.finder.ModelVariableFinder |
slreportgen.finder.ModelVariableResult | slreportgen.report.BusObject |
slreportgen.report.ModelVariable

```

### More About

- “Report Generation for Simulink and Stateflow Elements” on page 1-9
- “What Is a Reporter?”

## Report System Inputs and Outputs

This example shows how to create a report that describes the inputs and outputs of a model or subsystem. The report includes a chapter for the top-level model and each subsystem in the model. Each chapter includes a section for the inputs and outputs and a section for the blocks in the system.

This image shows the input and output summaries included in the report.

### 1.1. Inputs and Outputs

**Table 1.1. slreportgen\_demo\_SystemIO Input Summary**

Block	Source	Name	DataType
slreportgen_demo_SystemIO/Input1	mappedIO.getElement(1)	signal_1	double
slreportgen_demo_SystemIO/Input2	mappedIO.getElement(2)	signal_2	double
slreportgen_demo_SystemIO/Input	mappedIO.getElement(3)	signal_3	double

**Table 1.2. slreportgen\_demo\_SystemIO Output Summary**

Block	Destination	DataType
slreportgen_demo_SystemIO/Output	yout{1}	double

### 2.1. Inputs and Outputs

**Table 2.1. Compute Result Input Summary**

Port	Source	Name	DataType
<a href="#">Inport 1</a>	<a href="#">slreportgen_demo_SystemIO/Input1</a>	signal_1	double
<a href="#">Inport 2</a>	<a href="#">slreportgen_demo_SystemIO/Math Function</a>		double

**Table 2.2. Compute Result Output Summary**

Port	Destination	DataType
<a href="#">Outport 1</a>	<a href="#">slreportgen_demo_SystemIO/Sum (Port 1)</a>	double

### Open Model

Open a model. This example uses a model that has top-level input and output blocks and a subsystem with inputs and outputs. The top-level input signals are stored in a variable, `mappedIO`, which is created when the model is opened.

```
model = "slreportgen_demo_SystemIO";
open_system(model);
```

### Report Setup

Import the Report Generator API packages so you do not have to use long, fully-qualified class names.

```
import mlreportgen.report.*
import slreportgen.report.*
import slreportgen.finder.*
```

Create and open a Simulink report object. To create a Microsoft® Word, HTML, or single-file HTML report, change "pdf" to "docx", "html", or "html-file", respectively.

```
rpt = slreportgen.report.Report(model + "_SystemIO_Report", "pdf");
open(rpt);
```

Add a title page and table of contents.

```
titlepage = TitlePage("Title", model + ": System I/O Report", "Author", "Jane Doe");
add(rpt, titlepage);
toc = TableOfContents();
add(rpt, toc);
```

## Report on Inputs and Outputs

Find and loop through all systems in the model.

```
finder = SystemDiagramFinder(model);
while hasNext(finder)
    system = next(finder);
```

Create a new chapter and add the diagram result.

```
ch = Chapter("Title", sprintf("System %s", system.Name));
add(ch, system);
```

Create an "Inputs and Outputs" section and a SystemIO reporter.

```
ioSect = Section("Inputs and Outputs");
ioRptr = SystemIO(system);
```

For subsystem inputs and outputs, the SystemIO reporter by default includes details about the input and output ports of the subsystem. For model inputs and outputs, the reporter includes details about inport and outport blocks. If the system is a model, set the SystemIO options to omit these block details because this report includes the same information in the "Blocks" section of the chapter.

```
if strcmp(system.Type, "Simulink.BlockDiagram")
    ioRptr.ShowDetails = false;
end
add(ioSect, ioRptr);
add(ch, ioSect);
```

Create a section to include details about each block in the system. Source and destination blocks included in SystemIO summary tables link to the corresponding block details in this section.

```
blkSect = Section("Blocks");
blkFinder = BlockFinder(system);
results = find(blkFinder);
add(blkSect, results);
add(ch, blkSect);
```

Add the chapter to the report.

```
add(rpt, ch);
end
```

### **Close Report**

Close and view the report.

```
close(rpt);  
rptview(rpt);
```

## Reporting on DocBlock Blocks

This example shows how to include the contents of Simulink DocBlock blocks in a Microsoft® Word report generated by the Report API. The example model, `slreportgen_demo_docblock`, contains only DocBlock blocks, with one block for each kind of DocBlock document type:

- RTF
- HTML
- Text

In the generated report, the contents of the DocBlock blocks look like this:

### Chapter 1. System `slreportgen_demo_docblock` your Word documentation

#### your HTML documentation

Plain text documentation

Import the API packages so that you can refer to API classes by their unqualified names, that is, without the names of the class packages in which they reside.

```
import mlreportgen.report.*
import slreportgen.report.*
import slreportgen.finder.*
import mlreportgen.dom.*
```

Load the model for this example.

```
model = 'slreportgen_demo_docblock';
load_system(model);
```

Create a container to hold the report content. To avoid a compilation error due to the model containing only virtual blocks, set the `CompileModelBeforeReporting` property of the report object to `false`.

```
rpt = slreportgen.report.Report('ModelDoc', 'docx');
rpt.CompileModelBeforeReporting = false;
```

Add a title page and table of contents.

```
add(rpt,TitlePage('Title',sprintf('%s Model Documentation',model)));
add(rpt,TableOfContents);
```

Find and loop through all the systems in the model.

```
finder = SystemDiagramFinder(model);
for system = find(finder)
```

Create a chapter for each system. Include the system name in the chapter title. Use the chapter to report on the DocBlock content of the system.

```
ch = Chapter('Title',sprintf('System %s', system.Name));
```

Find all the DocBlock blocks in the current system. Each result returns the DocBlock reporter for the found DocBlock. The add method invokes the DocBlock reporter.

```
docBlockFinder = BlockFinder(system);
docBlockFinder.Properties = {'MaskType', 'DocBlock'};
results = find(docBlockFinder);
if ~isempty(results)
    add(ch, results);
else
    add(ch, "This system does not have documentation.");
end
add(rpt, ch)
end
```

Close and view the report.

```
close(rpt);
close_system(model);
rptview(rpt);
```

### See Also

[DocBlock](#) | [slreportgen.finder.BlockFinder](#) | [slreportgen.finder.BlockResult](#) | [slreportgen.finder.SystemDiagramFinder](#) | [slreportgen.report.DocBlock](#)

### More About

- “Report Generation for Simulink and Stateflow Elements” on page 1-9
- “What Is a Reporter?”



## Report Model Notes

This example shows how to create a report that embeds model notes. The report includes a chapter for each system in the model. Each chapter includes a system snapshot and any notes for that system.

### Open Model

Open a model. This example uses a modified version of the `sldemo_autotrans` model where the documentation from the "?" help block has been copied into model notes.

```
model = "slreportgendemo_autotrans";
open_system(model);
```

### Report Setup

Import the Report Generator API packages so that you do not have to use long, fully qualified class names.

```
import mlreportgen.report.*
import slreportgen.report.*
import slreportgen.finder.*
```

Create and open a Simulink report object. To create a Microsoft® Word, HTML, or single-file HTML report, change "pdf" to "docx", "html", or "html-file", respectively.

```
rpt = slreportgen.report.Report(model + "_Notes_Report", "pdf");
open(rpt);
```

Add a title page and table of contents.

```
titlepage = TitlePage("Title", model);
add(rpt, titlepage);
toc = TableOfContents();
add(rpt, toc);
```

### Report on systems

Find and loop through all of the systems in the model.

```
finder = DiagramFinder(model);
while hasNext(finder)
    system = next(finder);
```

Create a new chapter and add the system result, which adds a system snapshot to the report.

```
ch = Chapter("Title", system.Name);
add(ch, system);
```

Add model notes to the current system. If the current system does not have any notes associated with it, nothing is added.

```
notes = Notes(system);
add(ch, notes);
```

Add the chapter to the report

```
add(rpt, ch);
end
```

### **Close Report**

Close and view the report.

```
close(rpt);  
rptview(rpt);
```

### **See Also**

`slreportgen.finder.DiagramResult`

### **More About**

- “Report Generation for Simulink and Stateflow Elements” on page 1-9
- “What Is a Reporter?”

## Report Execution Order of Tasks and Blocks in a Simulink System

This example shows how to create a report that displays information about all tasks executed by a model and the order in which blocks execute during each task.

Block execution can be separated into different tasks based on sample time if the “Treat each discrete rate as a separate task” configuration parameter is selected. Export-function models and systems containing certain blocks, such as asynchronous interrupts or event-triggered subsystems, also group block execution into different tasks. See “Control and Display Execution Order” for more information on viewing task information and block execution order in Simulink®.

This image shows a diagram of the sample model `slreportgen_demo_ExecutionOrder` and the task summary and block execution order for the model.

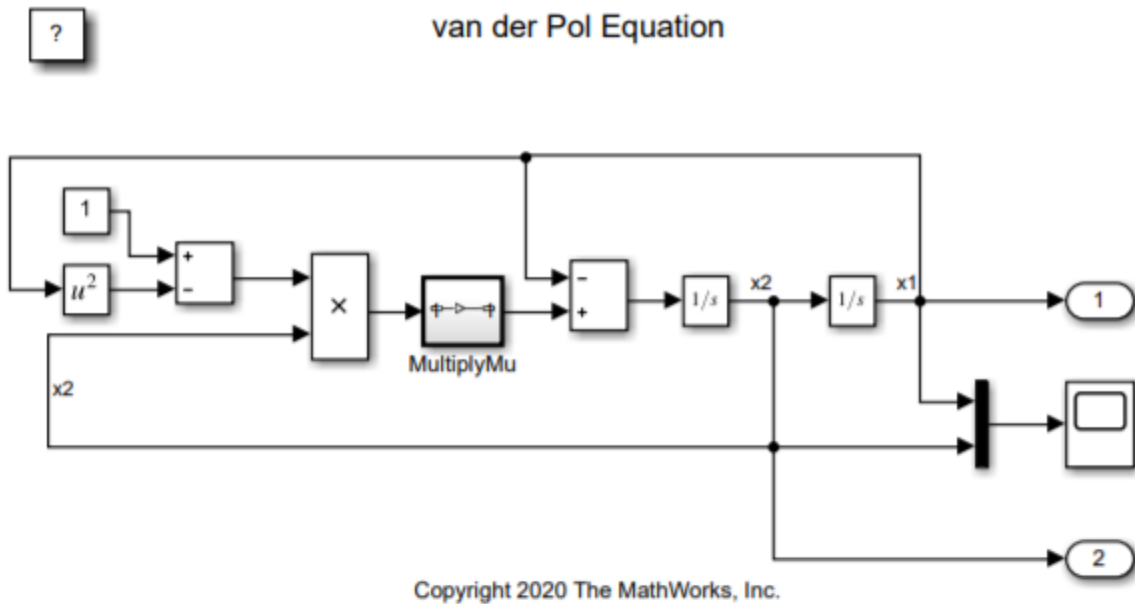


Figure 1.1. slreportgen\_ExecutionOrder\_example

### 1.1. Execution Order

Table 1.1. Tasks

Order	Name	Type	Trigger	TaskID
1	<a href="#">Cont</a>	Periodic	0	TID0
N/A	<a href="#">Constant</a>	Constant		TID1

### Block Execution Order

#### [Cont](#)

1. [x1](#) (Integrator)
2. [Out1](#) (Output)
3. [x2](#) (Integrator)
4. [Out2](#) (Output)
5. [Scope](#) (Scope)
6. [Square](#) (Math)
7. [Sum1](#) (Sum)
8. [Product](#) (Product)
9. [MultiplyMu](#) (SubSystem) [[Block Execution Order](#)]
10. [Sum](#) (Sum)

#### [Constant](#)

1. [Constant](#) (Constant)

Because the model is a continuous system, the main task, Cont, has a sample time value of 0. In all models, constant blocks are separated into Constant tasks.

MultiplyMu is a nonvirtual subsystem. By default, nonvirtual subsystem entries in a block execution order list contain a link to the block execution order list for that subsystem. Alternatively, you can configure the ExecutionOrder reporter options to display subsystem blocks as a nested list.

## Open Model

Open a model. This example uses a single-tasking model, that is, all blocks except constant blocks execute during the same task.

```
model = "slreportgen_ExecutionOrder_example";
open_system(model);
```

## Report Setup

Import the Report Generator API packages so you do not have to use long, fully qualified class names.

```
import mlreportgen.report.*
import slreportgen.report.*
import slreportgen.finder.*
```

Create and open a Simulink report object. To create a Microsoft® Word, HTML, or single-file HTML report, change "pdf" to "docx", "html", or "html-file", respectively.

```
rpt = slreportgen.report.Report(model + "_Report", "pdf");
open(rpt);
```

Add a title page and table of contents.

```
titlepage = TitlePage("Title", model + ": Execution Order Report", "Author", "Jane Doe");
add(rpt, titlepage);
toc = TableOfContents();
add(rpt, toc);
```

## Report on Task and Block Execution Order

Find and loop through all systems in the model.

```
finder = SystemDiagramFinder(model);
while hasNext(finder)
    system = next(finder);
```

Create a new chapter and add the diagram result.

```
ch = Chapter("Title", sprintf("System %s", system.Name));
add(ch, system);
```

Report the execution order of the system only if it is a block diagram or a nonvirtual subsystem. Blocks within virtual subsystems are reported in the parent's block execution order.

```
isNonvirtualSubsystem = strcmp(system.Type, "Simulink.SubSystem") ...
    && strcmp(get_param(system.Object, "IsSubsystemVirtual"), "off");
if strcmp(system.Type, "Simulink.BlockDiagram") || isNonvirtualSubsystem
```

Create an Execution Order section and an ExecutionOrder reporter.

```
eoSect = Section("Execution Order");  
eoRptr = ExecutionOrder(system);
```

For subsystems, set the ExecutionOrder options so that task details are not reported, because this information is already reported by the parent block diagram execution order.

```
if isNonvirtualSubsystem  
    eoRptr.ShowTaskDetails = false;  
end
```

Add the ExecutionOrder reporter to the Execution Order chapter, and add the chapter to the report.

```
add(eoSect, eoRptr);  
add(ch, eoSect);  
end
```

Create a section to include details about each block in the system. Blocks included in ExecutionOrder block execution order lists link to the corresponding block details in this section.

```
blkSect = Section("Blocks");  
blkFinder = BlockFinder(system);  
results = find(blkFinder);  
add(blkSect, results);  
add(ch, blkSect);
```

Add the chapter to the report.

```
add(rpt, ch);  
end
```

### Close and View the Report

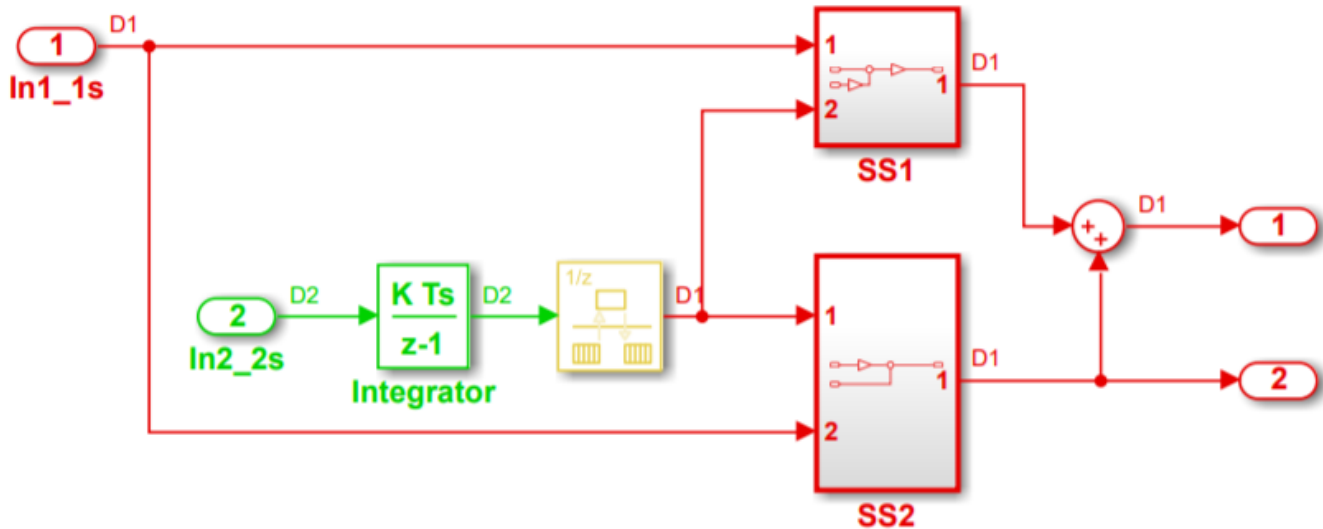
```
close(rpt);  
rptview(rpt);
```

### View Sample Reports

To see how execution order is reported for other types of models, view the sample reports available with this example.

### Multitasking Models

The sample model `slreportgen_demo_Multitasking` is configured to treat each discrete sample time as a separate task. The sample time for blocks `In1_1s`, `SS1`, and `SS2` is 1 second, and the sample time for block `In2_2s` is 2 seconds.



Copyright 2020 The MathWorks, Inc.

The model is also configured to display blocks color-coded by sample time. Blocks that execute at a 1 second sample time are red, and blocks that execute at a 2 second sample time are green. Multirate blocks, such as the rate-transition block between the Integrator block and the two subsystems, are yellow. To programmatically configure a model in this way, execute this command:

```
set_param(model, "SampleTimeColors", "on");
```

The execution order for this model reports two tasks. The Trigger column of the task details table reports the sample time, in seconds, for each task.

**Table 1.1. Tasks**

Order	Name	Type	Trigger	TaskID
1	<a href="#">D1</a>	Periodic	1	TID0
2	<a href="#">D2</a>	Periodic	2	TID1

The model blocks are separated by task. The rate-transition block executes during both tasks, so it is included in both lists. However, only its output port executes during task D1, and only its input port executes during task D2.

## Block Execution Order

### D1

1. [RateTransition](#) (RateTransition) Output Ports: 1
2. [SS1](#) (SubSystem) [[Block Execution Order](#)]
3. [SS2](#) (SubSystem) [[Block Execution Order](#)]
4. [Sum](#) (Sum)
5. [Out1](#) (Outputport)
6. [Out2](#) (Outputport)

### D2

1. [Integrator](#) (DiscreteIntegrator)
2. [RateTransition](#) (RateTransition) Input Ports: 1

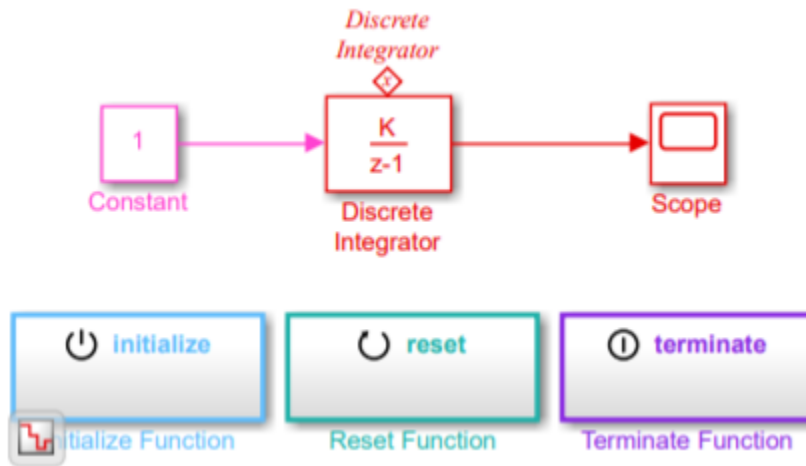
To view the full sample report, execute this command:

```
rptview("slreportgen_demo_Multitasking_Report.pdf")
```

### **Nonperiodic Tasks**

Some tasks, such as those created by asynchronous interrupts or event listeners, do not execute based on sample time. For example, the sample model `slreportgen_demo_InitResetTerm` uses three subsystems with the execution controlled by event listeners. Each event listener is configured to execute a subsystem when it receives an initialize, reset, or terminate function-call event.





The initialize, reset, and terminate events are reported as separate tasks in the execution order. Their execution does not directly depend on the sample time of the model, so they are not given an order number in the task table. The SourceBlock column denotes which block defines the task.

**Table 1.1. Tasks**

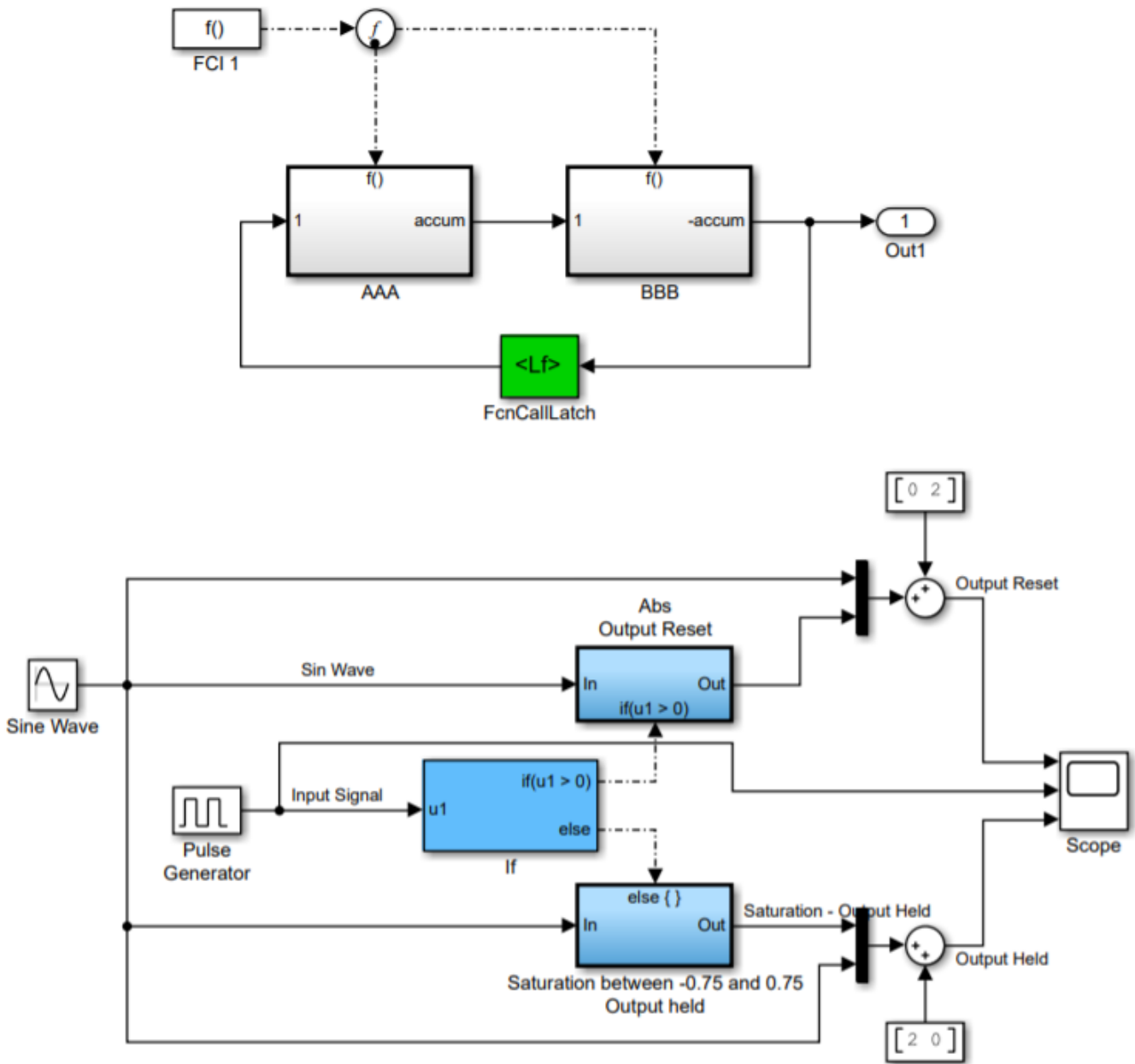
Order	Name	Type	Trigger	TaskID	SourceBlock
1	<a href="#">D1</a>	Periodic	1	TID0	
N/A	<a href="#">Init</a>	Initialize		TID1	<a href="#">slreportgen_demo_InitResetTerm/Initialize Function</a>
N/A	<a href="#">Term</a>	Terminate		TID2	<a href="#">slreportgen_demo_InitResetTerm/Terminate Function</a>
N/A	<a href="#">reset</a>	Reset		TID3	<a href="#">slreportgen_demo_InitResetTerm/Reset Function</a>
N/A	<a href="#">Constant</a>	Constant		TID4	

To view the full sample report, execute this command:

```
rptview("slreportgen_demo_InitResetTerm_Report.pdf")
```

### Conditional Execution

The sample model `slreportgen_demo_ConditionalExecution` contains an If block and a Function-Call Generator block that control when certain subsystems within the model execute.



The conditionally executed subsystems are not reported in the block execution order list because they do not necessarily execute at every time step. Instead, they are included in a Conditional Execution table that is reported after the block execution order list.

**Cont**

1. [FCI 1](#) (S-Function)
2. [Out1](#) (Output)
3. [Sine Wave](#) (Sin)
4. [Pulse Generator](#) (DiscretePulseGenerator)
5. [If](#) (If)
6. [Sum](#) (Sum)
7. [Sum1](#) (Sum)
8. [Scope](#) (Scope)

**Table 1.2. Conditional Execution**

Trigger	Blocks Executed
<a href="#">FCI 1</a> (S-Function)	<ol style="list-style-type: none"> <li>1. <a href="#">AAA</a> (SubSystem) [<a href="#">Block Execution Order</a>]</li> <li>2. <a href="#">BBB</a> (SubSystem) [<a href="#">Block Execution Order</a>]</li> </ol>
<a href="#">If</a> (If)	if( $u1 > 0$ ): <a href="#">Abs Output Reset</a> (SubSystem) [ <a href="#">Block Execution Order</a> ] else: <a href="#">Saturation between -0.75 and 0.75 Output held</a> (SubSystem) [ <a href="#">Block Execution Order</a> ]

To view the full sample report, execute this command:

```
rptview("slreportgen_demo_ConditionalExecution_Report.pdf")
```

**See Also**

`slreportgen.report.ExecutionOrder`

**More About**

- “Report Generation for Simulink and Stateflow Elements” on page 1-9
- “Generate a System Design Report with the Report API” on page 2-8
- “What Is a Reporter?”
- “Treat each discrete rate as a separate task”
- “Control and Display Execution Order”

## Create a Simulink Report Generator Report Interactively

This example shows how to use the Report Explorer to design a report setup file and generate a report that does the following:

- Opens a Simulink model for the van der Pol equation, called the vdp model.
- Sets the **Gain** parameter for the Mu block to five different values.
- Simulates the model each time the **Gain** parameter is set.
- Collects the results. Results that fall within a specified range appear in a table in the generated report.

You do not need to know MATLAB or Simulink software to create and run this example report. However, knowledge of these products will help you understand the MATLAB code and model simulation that executes.

To create this report, you perform these main tasks:

- “Specify Report Options in the Setup File” on page 4-50
- “Add Report Content with Components” on page 4-51

This example includes separate sections for different kinds of report creation and generation task. Each section builds on the previous sections. However, if you want to see the report setup components for a later section without doing the previous sections, in MATLAB you can view the completed report setup file by opening Simulink Dynamic Report. The report is for the vdp model.

### Specify Report Options in the Setup File

To create and configure the report setup file:

- 1 Start Simulink.
- 2 Open the Report Explorer from the Simulink Toolstrip. On the **Apps** tab, in the **Simulation Graphics and Reporting** section, click **Report Generator**.
- 3 Select **File > New** to create a report setup file.
- 4 Save the report setup file.

In the Properties pane:

- a Specify where to save the report setup file. To save it in the current working folder, select **Present Working Directory** from the **Directory** selection list.
- b Specify the report format. In the **File format** selection list, select **Acrobat (PDF)**.

---

**Tip** In your reports, if you want to include hyperlinks in system snapshots, use **Direct PDF (from template)** file format.

---

- c Enter a description for the report. In the **Report description** text box, replace the existing contents with the following text.

---

**Tip** Copy and paste this code from the HTML documentation into the Report Explorer.

---

### Simulink Dynamic Report

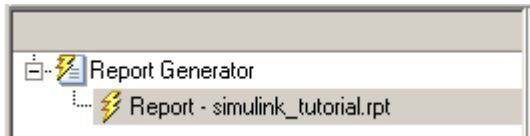
This report opens up a model, sets a block parameter several times, simulates the model, and collects the results. Results that fall within a specified range are displayed in a table after the test is complete.

The report is configured to test the vdp model only. By selecting the Eval String component immediately below the Report component, you can modify

- \* model
- \* block
- \* parameter
- \* tested values

- 5 Click **File > Save As** to save the report setup file as `simulink_tutorial.rpt`.

The Outline pane on the left displays the new file name.



To create the content for the report, see “Add Report Content with Components” on page 4-51.

## Add Report Content with Components

- “Report Components” on page 4-52
- “Add MATLAB Code” on page 4-53
- “Add a Title Page” on page 4-56
- “Open the Simulink Model” on page 4-58
- “Add Logical Then and Logical Else Components” on page 4-59
- “Error If Model Cannot Be Opened” on page 4-61
- “Create the Body of the Report” on page 4-62
- “Process with a Model Loop Component” on page 4-63
- “Add a Paragraph for Each Model” on page 4-64
- “Insert a Snapshot of the Model” on page 4-65
- “Add a Loop for Processing the Model” on page 4-66
- “Block Parameter Value from a MATLAB Expression” on page 4-67
- “Create a Section for Each Iteration” on page 4-68
- “Insert the Block Value” on page 4-69
- “Set a Parameter Value” on page 4-70
- “Check Value Using a Logical If Component” on page 4-71
- “Simulate the Model Using a Model Simulation Component” on page 4-72
- “Create a Post-Test Analysis Section” on page 4-77

## Report Components

Report components specify what information to include in the report. Components are self-contained, modular MATLAB objects that control the report-generation process and insert elements, such as tables, lists, and figures, into a report setup file. Use components to customize the appearance and output of reports.

For more information, see “Report Components” on page 1-24.

The following figure shows a sample page from the report you create in this example, and which components you use to produce this output.

---

**Note** Do not deactivate report components that you add to the report setup file.

---

Chapter/Subsection  
component

# Chapter 1. Model - vdp

## Table of Contents

<i>Iteration_Value -1</i> .....	2
<i>Iteration_Value 0</i> .....	3
<i>Iteration_Value 0.5</i> .....	4
<i>Iteration_Value 1</i> .....	5
<i>Iteration_Value 2</i> .....	6
Post-Test Analysis .....	6

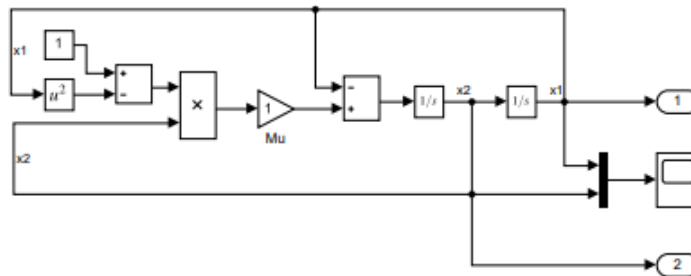
Paragraph  
component

This is a demonstration of the Report Generator's ability to experiment with Simulink systems and auto-document the results. In this report, we will load the model vdp and simulate it 5 times. The report will modify the vdp/Mu block's "Gain" value, setting it to the values [-1 0 0.5 1 2] . Each iteration of the test will include a set of scope snapshots in the report.



van der Pol Equation

System Snapshot  
component



Copyright 2004-2020 The MathWorks, Inc.

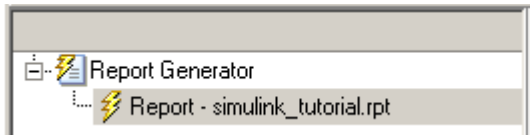
## Add MATLAB Code

**Note** This section builds on the previous tasks described in the step-by-step example summarized in “Create a Simulink Report Generator Report Interactively” on page 4-50.

To see the completed report setup file, open Simulink Dynamic Report. The report is for the vdp model.

The first component to add is the **Evaluate MATLAB Expression** component, which evaluates MATLAB commands in the workspace. The code in this component assigns initial values to variables used in this example.

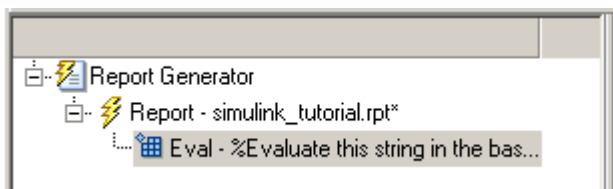
- 1 In the Outline pane on the left, select `simulink_tutorial.rpt`.



- 2 In the Library pane in the middle, under the **MATLAB** category, select **Evaluate MATLAB Expression**.
- 3 In the Properties pane on the right, click the icon next to **Add component to current report** to insert the component into the report.

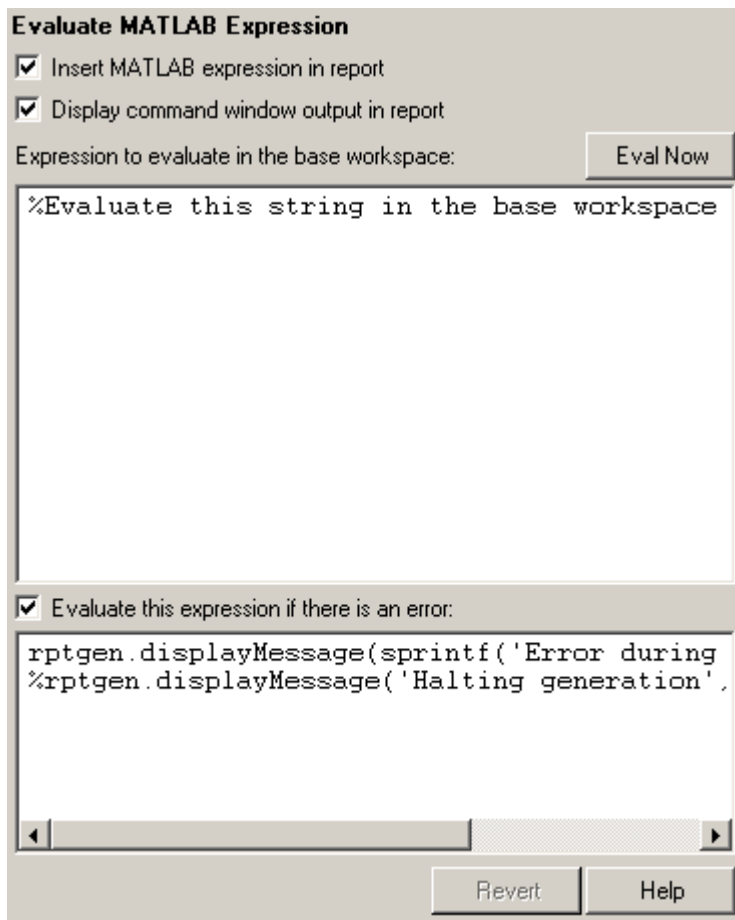
**Note** You cannot edit the component information in the Properties pane on the right until you add the component to the report.

In the Outline pane on the left, the **Evaluate MATLAB Expression** component appears under the `simulink_tutorial` report setup file. The Simulink Report Generator software abbreviates the component name to **Eval**.



The icon in the upper left corner of the **Eval** component icon indicates that this component cannot have child components. By default, any components you add while the **Eval** component is selected are siblings of this component.

The options for the **Evaluate MATLAB Expression** component appear in the Properties pane on the right.



- 4 Clear the **Insert MATLAB expression in report** and the **Display command window output in report** check boxes so you do not include MATLAB code or output in this report.
- 5 Add MATLAB code to the **Expression to evaluate in the base workspace** text box to specify the following values:
  - The model name
  - The block name
  - The block parameter
  - Parameter values
  - Other initial values required for processing the vdp model

Replace the existing text with the following MATLAB code.

```
%The name of the model
%that will be changed
expModel='vdp';
```

```
%The name of the block in the model
%that will be changed
expBlock='vdp/Mu';
```

```
%The name of the block parameter
%that will be changed
```



```
expParam='Gain';

%The values that will be set
%during experimentation
expValue=[-1 0 .5 1 2];

%expValue can be either a vector
%or a cell array

testMin=2.1;
testMax=3;

%---- do not change code below line ---

try
    open_system(expModel);
end

exp0kValues=cell(0,2);
```

---

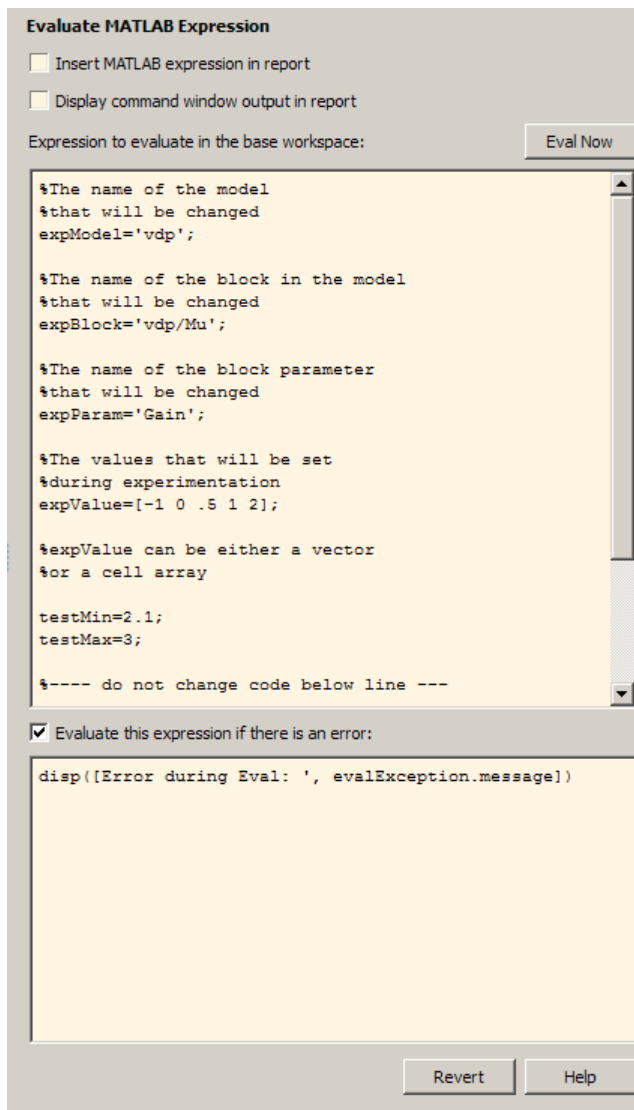
**Note** When you change a field in the Properties pane on the right, the field background changes color (the default is a cream color), indicating that there are unapplied changes to that field. As soon as you perform operations on another component, the Simulink Report Generator software applies the changes, and the background color becomes white again.

---

- 6 Select the **Evaluate this expression if there is an error** check box.
- 7 In the field under the check box, replace the existing text with the following text:

```
disp(['Error during eval: ', evalException.message])
```

The Report Explorer window now looks as follows.



---

**Tip** To run the commands that you specified in your MATLAB expression, click the **Eval Now** button. This button is located at the upper-right corner of the Report Explorer. This is an easy way to ensure that your commands are correct and will not cause report generation problems.

---

- 8 Click **File** > **Save** to save the report setup file.

For information about handling error conditions, see “Error Handling for MATLAB Code”.

### Add a Title Page

---

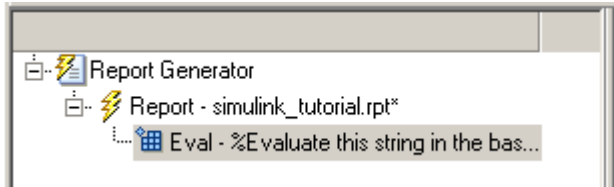
**Note** This section builds on the previous tasks described in the step-by-step example summarized in “Create a Simulink Report Generator Report Interactively” on page 4-50.

To see the completed report setup file, open Simulink Dynamic Report. The report is for the vdp model.

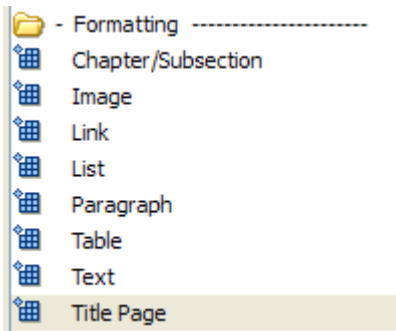
---

Create a custom title page for your report using the **Title Page** component.

- 1 In the Outline pane on the left, select the **Eval** component.

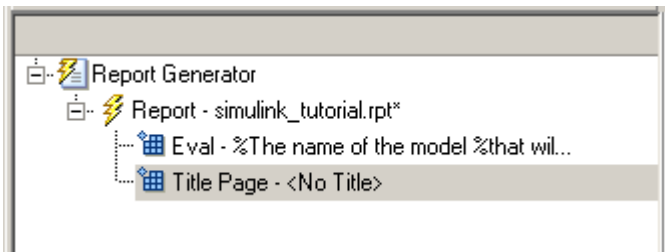


- 2 In the Library pane in the middle, under the **Formatting** category, click **Title Page**.



- 3 Click the icon next to **Add component to current report**.

The **Title Page** component appears in the Outline pane.



**Note** To use the **Title Page** component, you need to have a **Chapter** component in your report . You have not yet added a **Chapter** component, so the Properties pane displays a message indicating that chapters are required for the **Title Page** component to appear correctly. Because later in this example you add **Chapter** components to this report, you can ignore that message.

- 4 In the Properties pane on the right:
  - a In the **Title** text box, enter:  
Dynamic Simulink Report
  - b In the **Subtitle** text box, enter:  
Using Simulink Report Generator to Document Changes
  - c In the **Options** section, choose Custom Author from the selection list.
  - d Enter your name in the text box.
  - e Select the **Include report creation date** check box.

- f Select the default date and time format from the selection list. The Properties pane on the right looks as follows.

**Title Page**

Error

Chapters are required for component "Title Page" (section) to appear correctly. Add chapters to template.

Main Image Abstract Legal notice

Title

Title: Dynamic Simulink Report

Subtitle: Using Report Generator to Document Changes

Options

Custom author: John Q. Engineer

Include report creation date: dd-mmm-yyyy HH:MM:SS (26-Jul-2011 09:36:58)

Include copyright holder and year:

Display legal notice on title page

- 5 Save the report setup file.

### Open the Simulink Model

---

**Note** This section builds on the previous tasks described in the step-by-step example summarized in “Create a Simulink Report Generator Report Interactively” on page 4-50.

To see the completed report setup file, open Simulink Dynamic Report. The report is for the vdp model.

---

The following statement in the **Evaluate MATLAB Expression** component that you created in “Add MATLAB Code” on page 4-53 tries to open the vdp model:

```
try
    open_system(expModel);
end
```

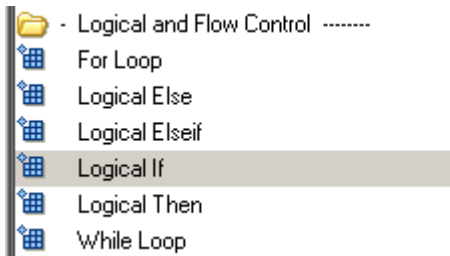
---

**Tip** Select the **Eval** component in the Outline pane on the left to look at this code again.

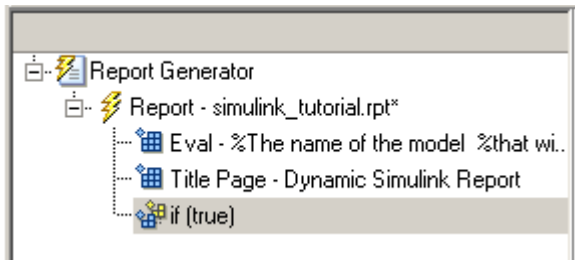
---

To see if the vdp model was successfully opened, test the result of the `open_system` command using a **Logical If** component.

- 1 In the Outline pane on the left, select the **Title Page** component.
- 2 In the Library pane in the middle, under the **Logical and Flow Control** category, select **Logical If**. This component checks to see if a given condition is true or false; in this case, if the model opened successfully.



- 3 In the Properties pane on the right, click the icon next to **Add component to current report**. The **Logical If** component appears as **if** in the Outline pane.



These components are child components of the report and siblings of one another. Components can have parent, child, and sibling relationships.

This component can have child components. “Add Logical Then and Logical Else Components” on page 4-59 explains how to add two child components to the **if** component.

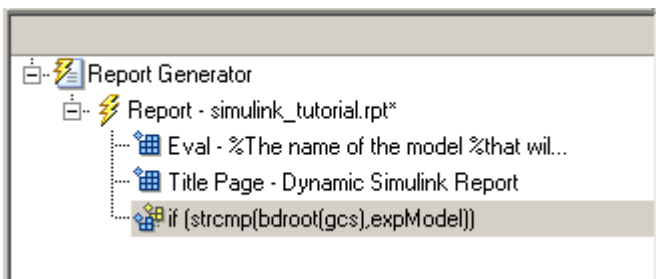
- 4 In the Properties pane on the right, in the **Test expression** text box, replace the default text, `true`, with the following text:

```
strcmp(bdroot(gcs),expModel)
```

The `strcmp` function compares the name of the open Simulink model and the value of `expModel`, which was set to `'vdp'`. It tests to see if the `vdp` model opened successfully. `strcmp` returns `1` (`true`) if the two strings match, and `0` (`false`) if not.

- 5 Save the report setup file.

The **if** component name in the Outline pane changes to include the expression that you added.



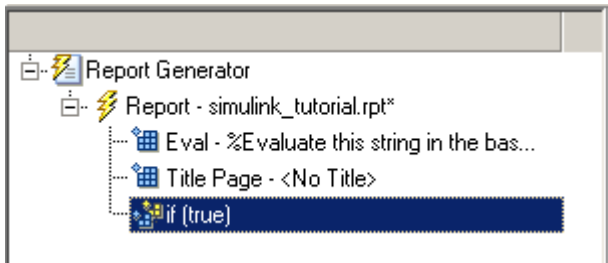
### Add Logical Then and Logical Else Components

**Note** This section builds on the previous tasks described in the step-by-step example summarized in “Create a Simulink Report Generator Report Interactively” on page 4-50.

To see the completed report setup file, open Simulink Dynamic Report. The report is for the vdp model.

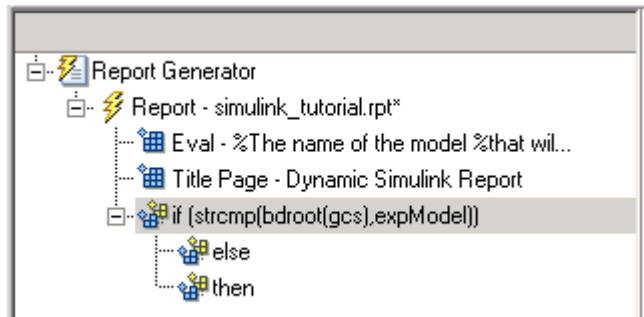
The `if strcmp(bdroot(gcs), expModel)` component has two possible results. Add two child components to the report setup file to process these cases.

- 1 In the Outline pane on the left, select the **if** component.

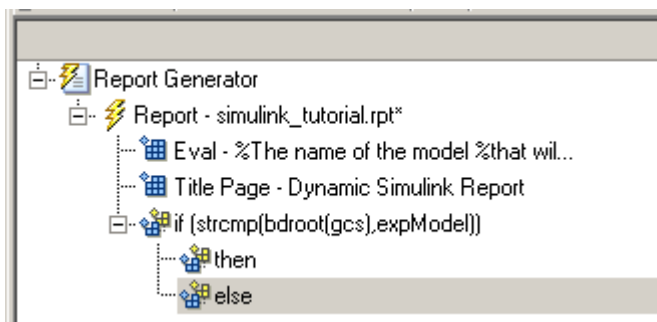


- 2 In the Library pane in the middle, under the **Logical and Flow Control** category, double-click **Logical Then**.
- 3 In the Outline pane on the left, select the **if** component again.
- 4 In the Library pane in the middle, under the **Logical and Flow Control** category, double-click **Logical Else**.

Both elements are added as child components to the if component, as shown in the Outline pane.



- 5 To move the else component under the **then** component, select the else component and click the **down** arrow on the toolbar once. The Outline pane on the left looks as follows.



- 6 Save the report setup file.

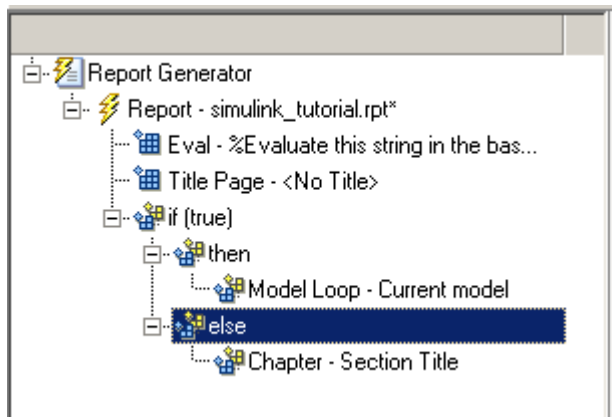
## Error If Model Cannot Be Opened

**Note** This section builds on the previous tasks described in the step-by-step example summarized in “Create a Simulink Report Generator Report Interactively” on page 4-50.

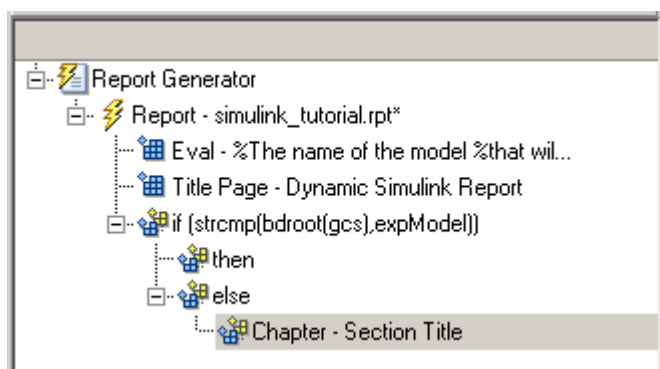
To see the completed report setup file, open **Simulink Dynamic Report**. The report is for the **vdp** model.

If the `if strcmp(bdroot(gcs), expModel)` component fails (the **vdp** model cannot open), the **else** component executes. Display an error message in the report using the **Chapter/Subsection** component.

- 1 In the Outline pane on the left, select the **else** component.



- 2 In the Library pane in the middle, under the **Formatting** category, double-click **Chapter/Subsection** to add it as a child of the **else** component. This component displays an error message if an error occurs when opening the **vdp** model.



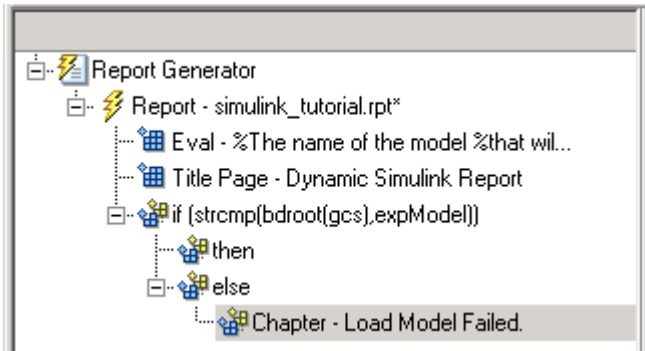
**Note** When you add a component to a report, it is added by default as a child component unless the selected component cannot have child components.

- 3 In the Properties pane on the right, choose **Custom** from the **Title** selection list, and then enter the following text in the text box:

Load Model Failed.

Save the report file.

The Outline pane looks as follows.



- 4 In the Outline pane on the left, select the **Chapter** component.
- 5 In the Library pane in the middle, under **Formatting**, double-click **Paragraph**.
- 6 In the Properties pane on the right, enter the following text in the **Paragraph Text** text box to display the following error message:

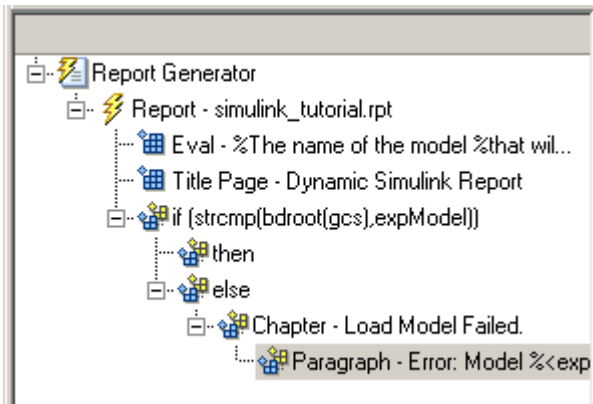
Error: Model %<expModel> could not be opened.

The expression %<expModel> indicates that the value of the workspace variable expModel is inserted into the text, as in the following example.

Error: Model vdp could not be opened.

- 7 In the Outline pane on the left, select the Chapter.
- 8 Save the report setup file.

The Outline pane looks as follows.



### Create the Body of the Report

**Note** This section builds on the previous tasks described in the step-by-step example summarized in “Create a Simulink Report Generator Report Interactively” on page 4-50.

To see the completed report setup file, open Simulink Dynamic Report. The report is for the vdp model.



Creating the body of the report involves setting up components and code for dynamic execution of report components. In this example, you perform the following tasks:

- “Process with a Model Loop Component” on page 4-63
- “Add a Paragraph for Each Model” on page 4-64
- “Insert a Snapshot of the Model” on page 4-65
- “Add a Loop for Processing the Model” on page 4-66
- “Block Parameter Value from a MATLAB Expression” on page 4-67
- “Create a Section for Each Iteration” on page 4-68
- “Insert the Block Value” on page 4-69
- “Set a Parameter Value” on page 4-70
- “Check Value Using a Logical If Component” on page 4-71
- “Simulate the Model Using a Model Simulation Component” on page 4-72
- “Create a Post-Test Analysis Section” on page 4-77

Each action requires a separate component under the **then** component. For information about the then component in this report, see “Add Logical Then and Logical Else Components” on page 4-59.

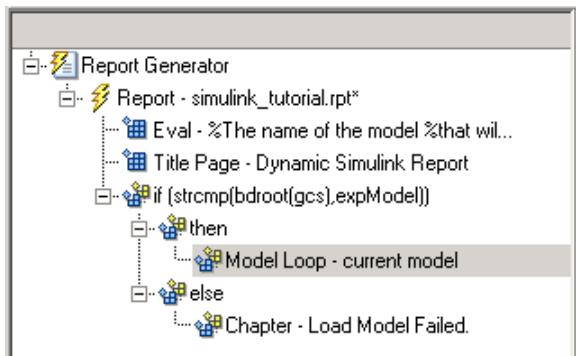
### Process with a Model Loop Component

**Note** This section builds on the previous tasks described in the step-by-step example summarized in “Create a Simulink Report Generator Report Interactively” on page 4-50.

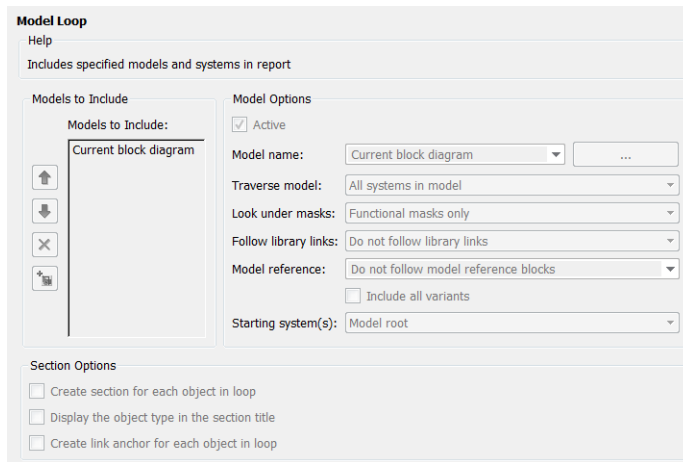
To see the completed report setup file, open **Simulink Dynamic Report**.

The report changes the **Gain** parameter for the Mu block in the vdp model several times. This task requires a **Model Loop** component.

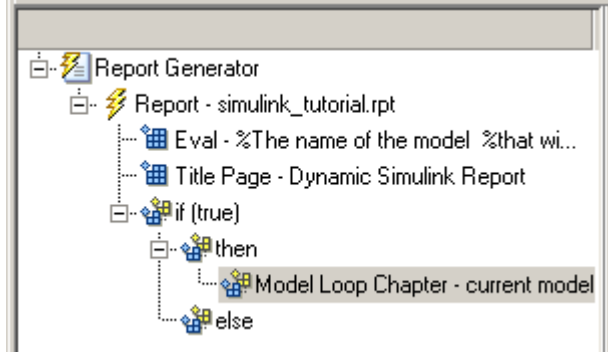
- 1 In the Outline pane on the left, select the **then** component.
- 2 In the Library pane in the middle, scroll down to the **Simulink** category, and then double-click **Model Loop**. It is added as a child of the then component.



The Properties pane on the right looks as follows.



- 3 In the Properties pane on the right:
  - a Select the **Active** check box to process the vdp model.
  - b In the **Traverse model** selection list, select **Selected system(s) only** to traverse only the vdp model.
  - c Select **Model root** from the **Starting system(s)** selection list.
  - d At the bottom of the Properties pane on the left, select the **Create section for each object in loop** check box to create a chapter or section for each model. When you select this check box, the component name in the Outline pane on the left changes to **Model Loop Chapter**.



- e Select the **Display the object type in the section title** check box to include the object type (in this example, model) in the title name.
  - f Clear the **Create link anchor for each object in loop** check box.
- 4 Save the report setup file.

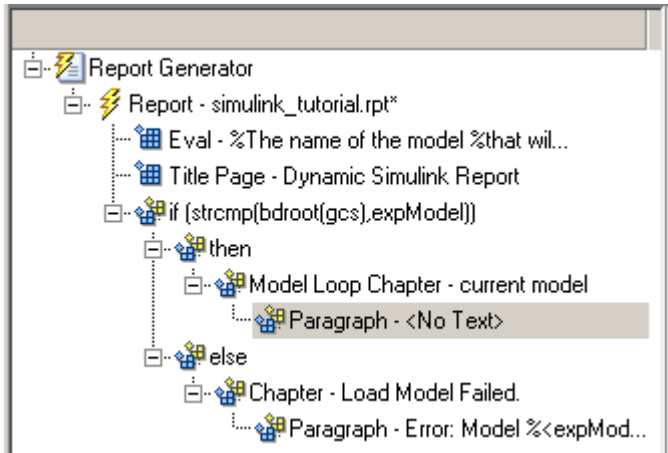
### Add a Paragraph for Each Model

**Note** This section builds on the previous tasks described in the step-by-step example summarized in “Create a Simulink Report Generator Report Interactively” on page 4-50.

To see the completed report setup file, open Simulink Dynamic Report. The report is for the vdp model.

In each **Model Loop Chapter**, add an explanation using the **Paragraph** component.

- 1 In the Outline pane on the left, select the **Model Loop Chapter** component.
- 2 In the Library pane in the middle, scroll up to the **Formatting** category, and then double-click **Paragraph**. The **Paragraph** component is added as a child of the **Model Loop Chapter** component.



- 3 In the Properties pane on the right, in the **Paragraph Text** text box, enter the following text:

This report demonstrates Simulink Report Generator's ability to experiment with Simulink systems and auto-document the results. In this report, you load the model %<expModel> and simulate it %<length> times. This report modifies the %<expBlock> block's "%<expParam>" value, setting it to the values %<expValue>. Each iteration of the test includes a set of scope snapshots in the report.

When this report is generated, the variable names, preceded by percent signs (%) and enclosed in angle brackets (<>), are replaced with the values of those variables in the MATLAB workspace.

- 4 Save the report setup file.

### Insert a Snapshot of the Model

---

**Note** This section builds on the previous tasks described in the step-by-step example summarized in "Create a Simulink Report Generator Report Interactively" on page 4-50.

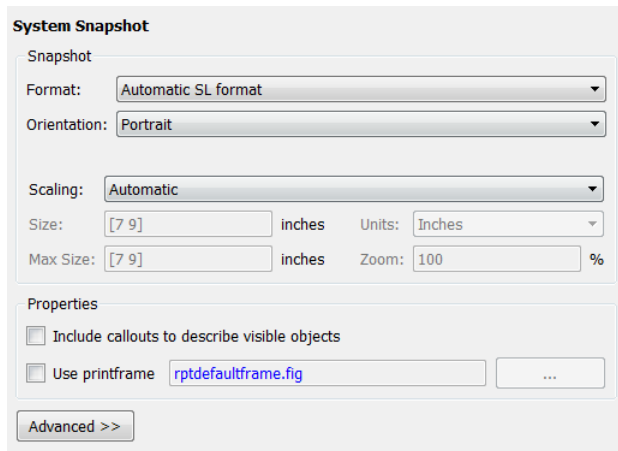
To see the completed report setup file, open Simulink Dynamic Report. The report is for the vdp model.

---

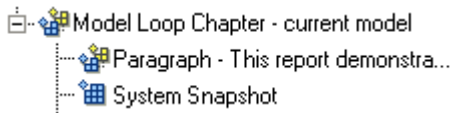
Inside each **Model Loop Chapter** component, include a snapshot of the current model using the **System Snapshot** component.

- 1 In the Outline pane on the left, select the **Model Loop Chapter** component.
- 2 In the Library pane in the middle, scroll down to the **Simulink** category, and then double-click the **System Snapshot** component.

This component inserts an image of the current model into your report. The Properties pane on the right looks as follows.



- 3 In the Properties pane on the right:
  - a Select Zoom from the **Scaling** selection list.
  - b Enter 70 as the % value.
- 4 In the Outline pane on the left, select the **System Snapshot** component.
- 5 Click the **down** arrow on the toolbar once to move it under the **Paragraph** component.



- 6 Save the report setup file.

### Add a Loop for Processing the Model

---

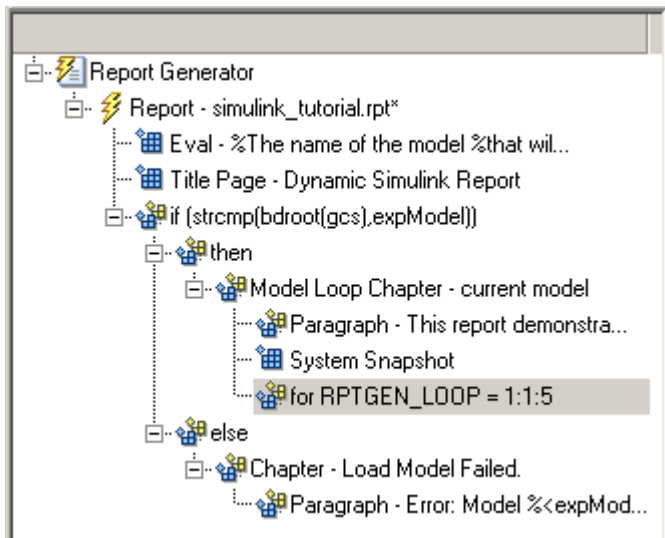
**Note** This section builds on the previous tasks described in the step-by-step example summarized in “Create a Simulink Report Generator Report Interactively” on page 4-50.

To see the completed report setup file, open Simulink Dynamic Report. The report is for the vdp model.

---

Create a loop to process the model %length times using the **For Loop** component.

- 1 In the Outline pane on the left, select the **System Snapshot** component.
- 2 In the Library pane in the middle, under the **Logical and Flow Control** category, double-click **For Loop**. The **For Loop** component is added as a sibling of the **System Snapshot** component.



3 In the Properties pane on the right:

a In the **End** text box, replace the existing text with the following text:

```
length(expValue)
```

expValue is the array of **Gain** parameter values assigned in the **Eval** component with the command `expValue=[ -1 0 0.5 1 2];`. The expression `length(expValue)` evaluates to 5 in this example.

b In the **Variable name** text box, replace the existing text with the name of the for loop variable. Enter the following text:

```
expIteration
```

The name of the **For** component in the Outline pane on the left changes to reflect the loop variable and the termination value.

4 Save the report setup file.

### Block Parameter Value from a MATLAB Expression

---

**Note** This section builds on the previous tasks described in the step-by-step example summarized in “Create a Simulink Report Generator Report Interactively” on page 4-50.

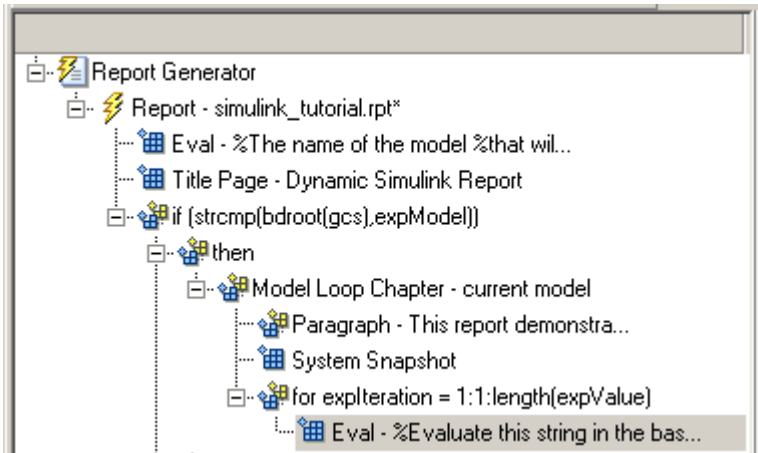
To see the completed report setup file, open Simulink Dynamic Report. The report is for the vdp model.

---

For each iteration, get a value from the expValue array to use as the **Gain** parameter value. This task requires an **Evaluate MATLAB Expression** component.

1 In the Outline pane on the left, select the **for** component.

2 In the Library pane in the middle, under the **MATLAB** category, double-click **Evaluate MATLAB Expression**. In the Outline pane, the component name is shortened to **Eval**.



- 3 On the Properties pane on the right:
  - a Clear the **Insert MATLAB expression in report** and **Display command window output in report** check boxes.
  - b Enter the following text in the **Expression to evaluate in the base workspace** text box:

```
%Evaluate this string in the base workspace

if iscell(expValue)
    Iteration_Value=expValue{expIteration};
else
    Iteration_Value=...
        num2str(expValue(expIteration));
end
```

The Iteration\_Value variable represents the designated array element.

- c Clear the **Evaluate this expression if there is an error** check box.
- 4 Save the report setup file.

### Create a Section for Each Iteration

---

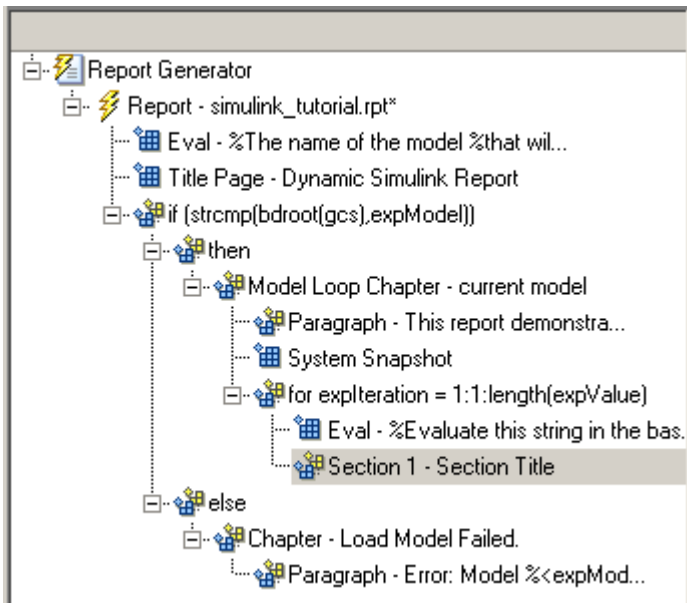
**Note** This section builds on the previous tasks described in the step-by-step example summarized in “Create a Simulink Report Generator Report Interactively” on page 4-50.

To see the completed report setup file, open Simulink Dynamic Report. The report is for the vdp model.

---

Create a separate section for each iteration of the loop that includes the data using the **Chapter/ Subsection** component.

- 1 In the Outline pane on the left, under the **for** component, select the **Eval** component.
- 2 In the Library pane in the middle, under the **Formatting** category, double-click **Chapter/ Subsection** to add it as a sibling. This component is automatically added as **Section 1** because it is inside a **Chapter** component (the **Model Loop Chapter** component).



- 3 In the Properties pane on the right:
  - a In the **Title** selection list, select Custom.
  - b In the text box, enter the following title:

Processing the vdp model

This indicates that the section title comes from the first child component. Do not change any other properties.

- 4 Save the report setup file.

### Insert the Block Value

---

**Note** This section builds on the previous tasks described in the step-by-step example summarized in “Create a Simulink Report Generator Report Interactively” on page 4-50.

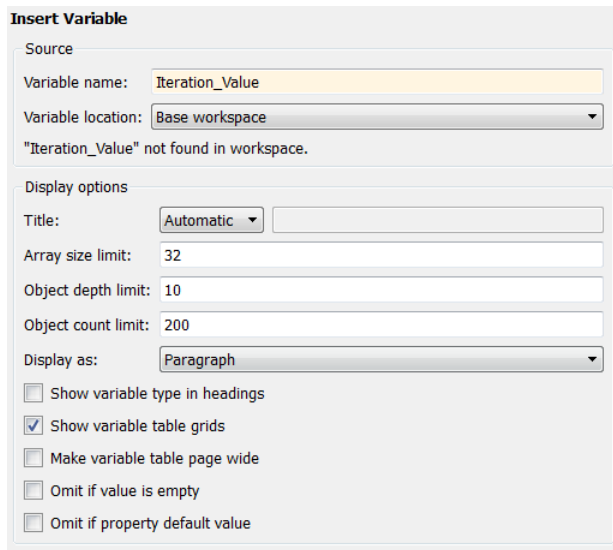
To see the completed report setup file, open Simulink Dynamic Report. The report is for the vdp model.

---

Insert the Gain value that is used for each simulation.

- 1 In the Outline pane on the left, select the **Section 1** component.
- 2 In the Library pane in the middle, under the **MATLAB** category, double-click **Insert Variable**.
- 3 In the Properties pane on the right:
  - a In the **Variable name** text box, enter `Iteration_Value`.
  - b In the **Display as** selection list, select Paragraph.

The Properties pane on the right looks as follows.



- 4 Save the report setup file.

### Set a Parameter Value

---

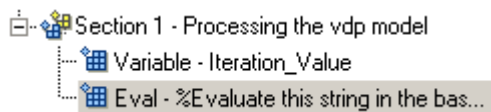
**Note** This section builds on the previous tasks described in the step-by-step example summarized in “Create a Simulink Report Generator Report Interactively” on page 4-50.

To see the completed report setup file, open Simulink Dynamic Report. The report is for the vdp model.

---

For each iteration, set the Gain parameter to the value that you extracted from the expValue array.

- 1 In the Outline pane on the left, select the **Variable** component.
- 2 In the Library pane in the middle, under the **MATLAB** category, double-click **Evaluate MATLAB Expression**. This component is added as a sibling of the **Variable** component.



- 3 In the Properties pane on the right, clear the **Insert MATLAB expression in report** and **Display command window output in report** check boxes.
- 4 In the **Expression to evaluate in the base workspace** text box, replace the existing text with the following text.

```
set_param(expBlock,expParam,Iteration_Value);
okSetValue=(1);
```

The `set_param` command sets the value of the **Gain** parameter for the Mu block in the vdp model to the value of `Iteration_Value`.

- 5 Make sure you select **Evaluate this expression if there is an error**. Enter the following text into the text box:

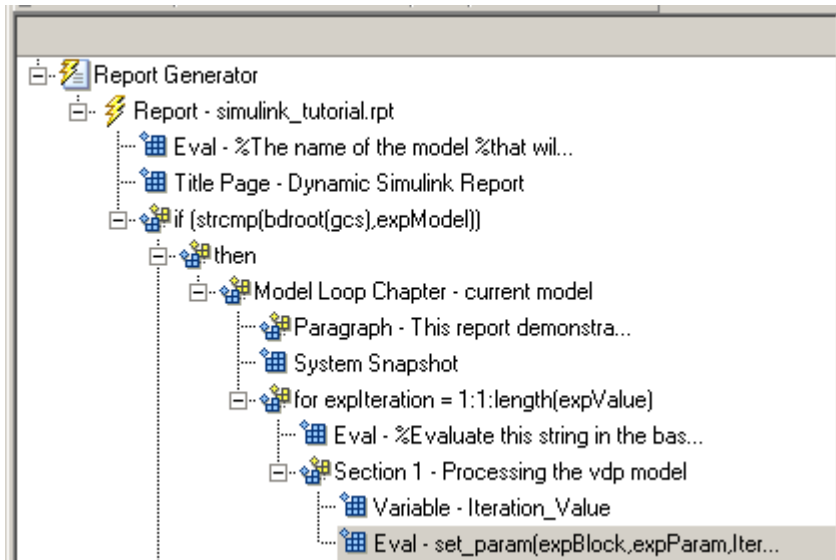
```
okSetValue=logical(0);
```



If the `set_param` command works, `okSetValue` is set to 1. If an error occurs, `okSetValue` is set to 0. The next component then reports the error and terminates processing.

- 6 Save the report setup file.

The Outline pane on the left looks as follows.



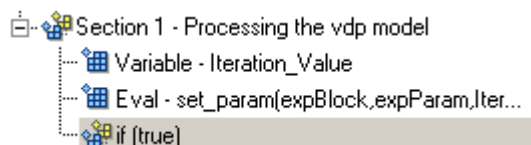
### Check Value Using a Logical If Component

**Note** This section builds on the previous tasks described in the step-by-step example summarized in “Create a Simulink Report Generator Report Interactively” on page 4-50.

To see the completed report setup file, open Simulink Dynamic Report. The report is for the vdp model.

Check the value of `okSetValue` using a **Logical If** component. If the value is 0, the simulation cannot proceed because the **Gain** parameter could not be set.

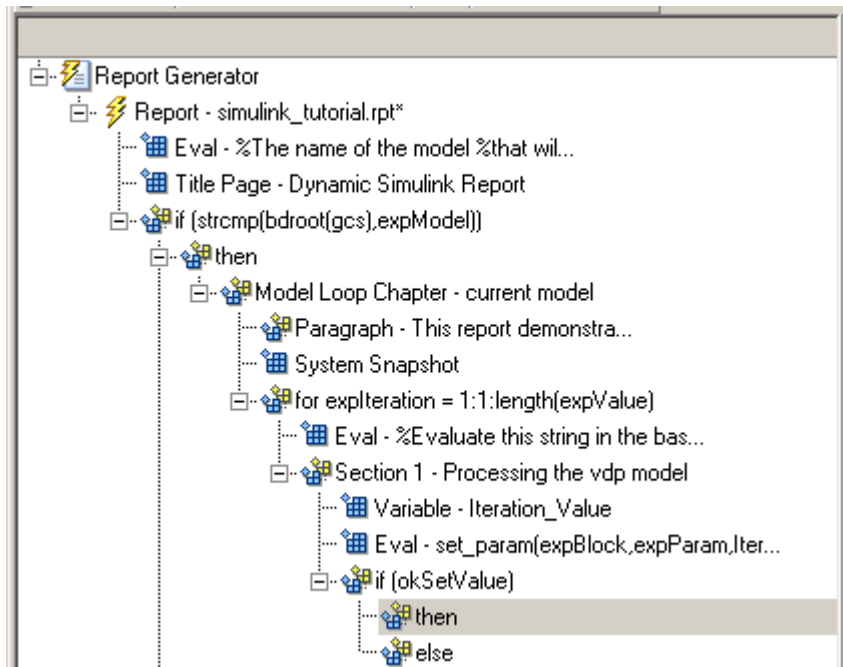
- 1 In the Outline pane on the left, select the **Eval** component for the `set_param` command.
- 2 In the Library pane in the middle, under the **Logical and Flow Control** category, double-click **Logical If**. The component is added as a sibling of **Eval**.



- 3 In the Properties pane on the right, in the **Test expression** text box, replace `true` with `okSetValue`.

`okSetValue` can be 1 (true) or 0 (false), so insert two components — **Logical Then** and **Logical Else** — to process those conditions:

- 1 In the Outline pane on the left, select the **if (okSetValue)** component.
- 2 To insert **Logical Then** and **Logical Else** in the correct order:
  - a In the Library pane in the middle, double-click the **Logical Else** component.
  - b Select the **if (okSetValue)** component again.
  - c Double-click the **Logical Then** component. The Outline pane on the left looks as follows.



- 3 In the Outline pane on the right, select the **else** component.
- 4 In the Library pane in the middle, double-click **Paragraph**.

If `okSetValue = 0`, the **Gain** parameter value is not set and the report displays an error.

- 5 In the Properties pane on the right:
  - a Choose `Custom title` from the **Title Options** selection list.
  - b Enter `Error` in the text box next to the selection list.
  - c Enter the following text into the **Paragraph Text** text box:

```
Could not set %<expBlock> "%<expParam>" to value
%<Iteration_Value>.
```

- 6 Save the report.

### Simulate the Model Using a Model Simulation Component

---

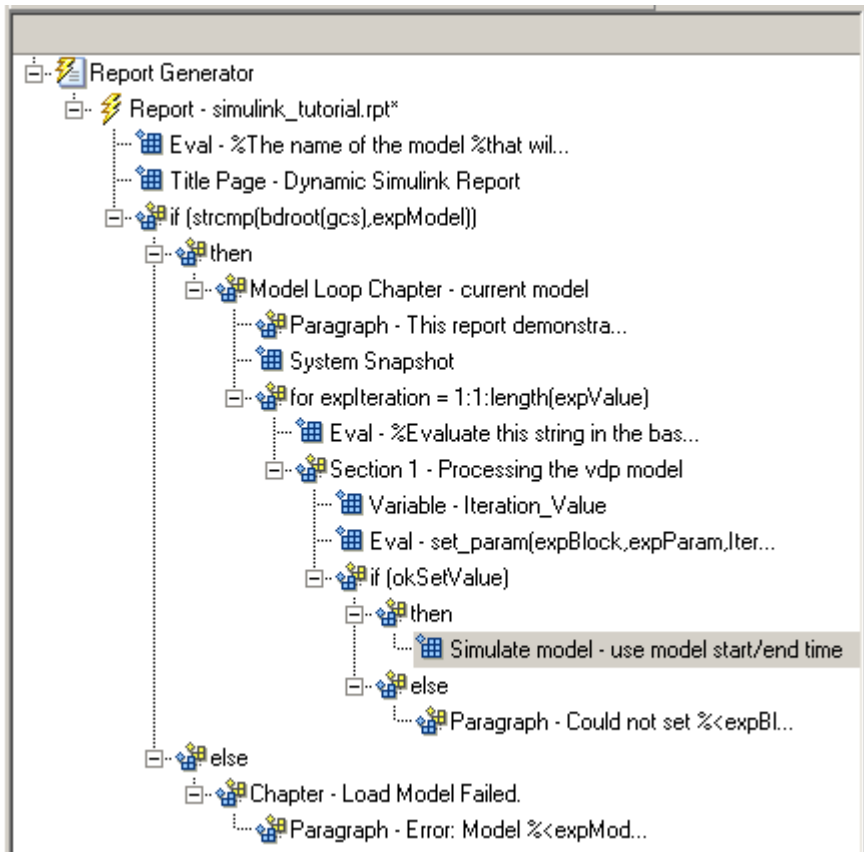
**Note** This section builds on the previous tasks described in the step-by-step example summarized in “Create a Simulink Report Generator Report Interactively” on page 4-50.

To see the completed report setup file, open `Simulink Dynamic Report`. The report is for the `vdp` model.

---

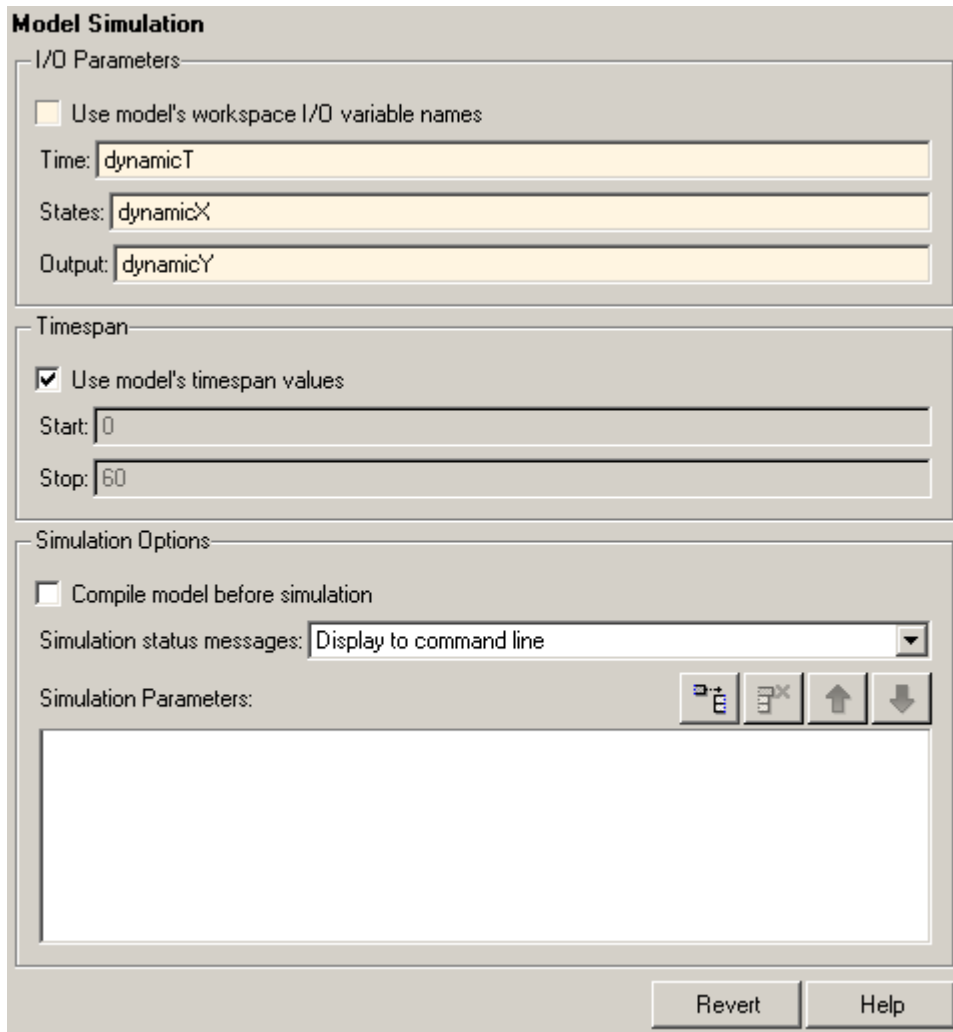
Now that the model is open and the **Gain** parameter is set, use the **Model Simulation** component to simulate the vdp model.

- 1 In the Outline pane on the left, select the **then** component under the **if (okSetValue)** component.
- 2 In the Library pane, under the **Simulink** category, double-click **Model Simulation**. In the Outline pane on the left, this component is renamed **Simulate model**.



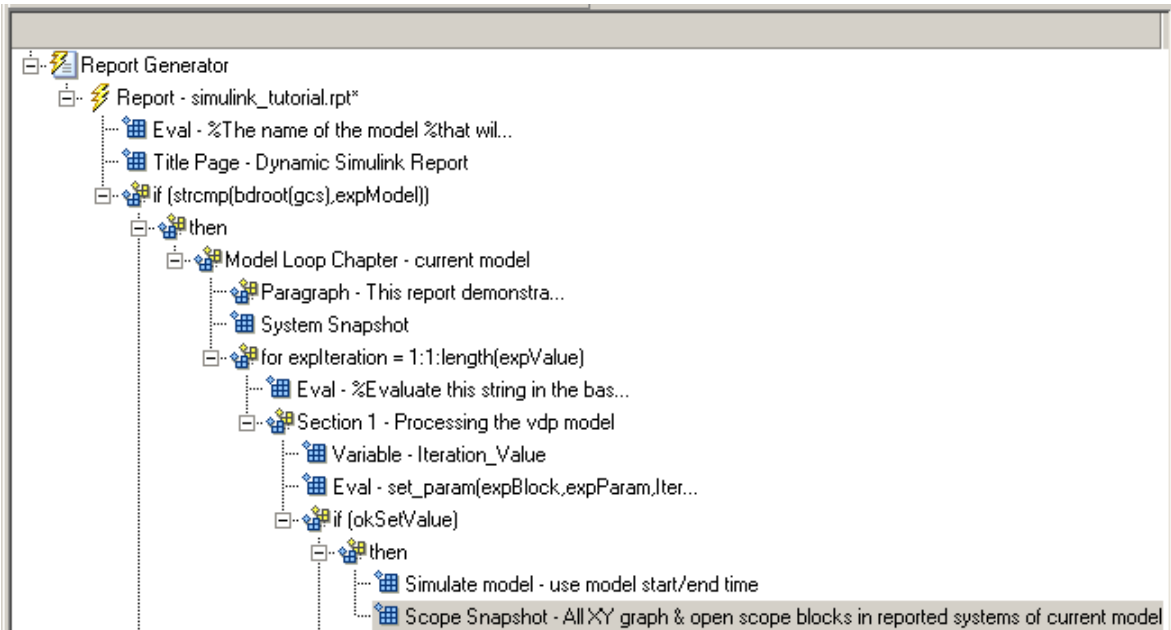
- 3 In the Properties pane on the right:
  - a Clear the **Use model's workspace I/O variable names** check box.
  - b In the **Time** text box, enter `dynamicT`.
  - c In the **States** text box, enter `dynamicX`.
  - d In the **Output** text box, enter `dynamicY`.

The Properties pane on the right looks as follows.



- 4 In the Outline pane on the left, select the **Simulate model** component.
- 5 In the Library pane in the middle:
  - a Scroll down to the **Simulink Blocks** category.
  - b Double-click **Scope Snapshot** to add it as a sibling of the **Simulink Model** component.

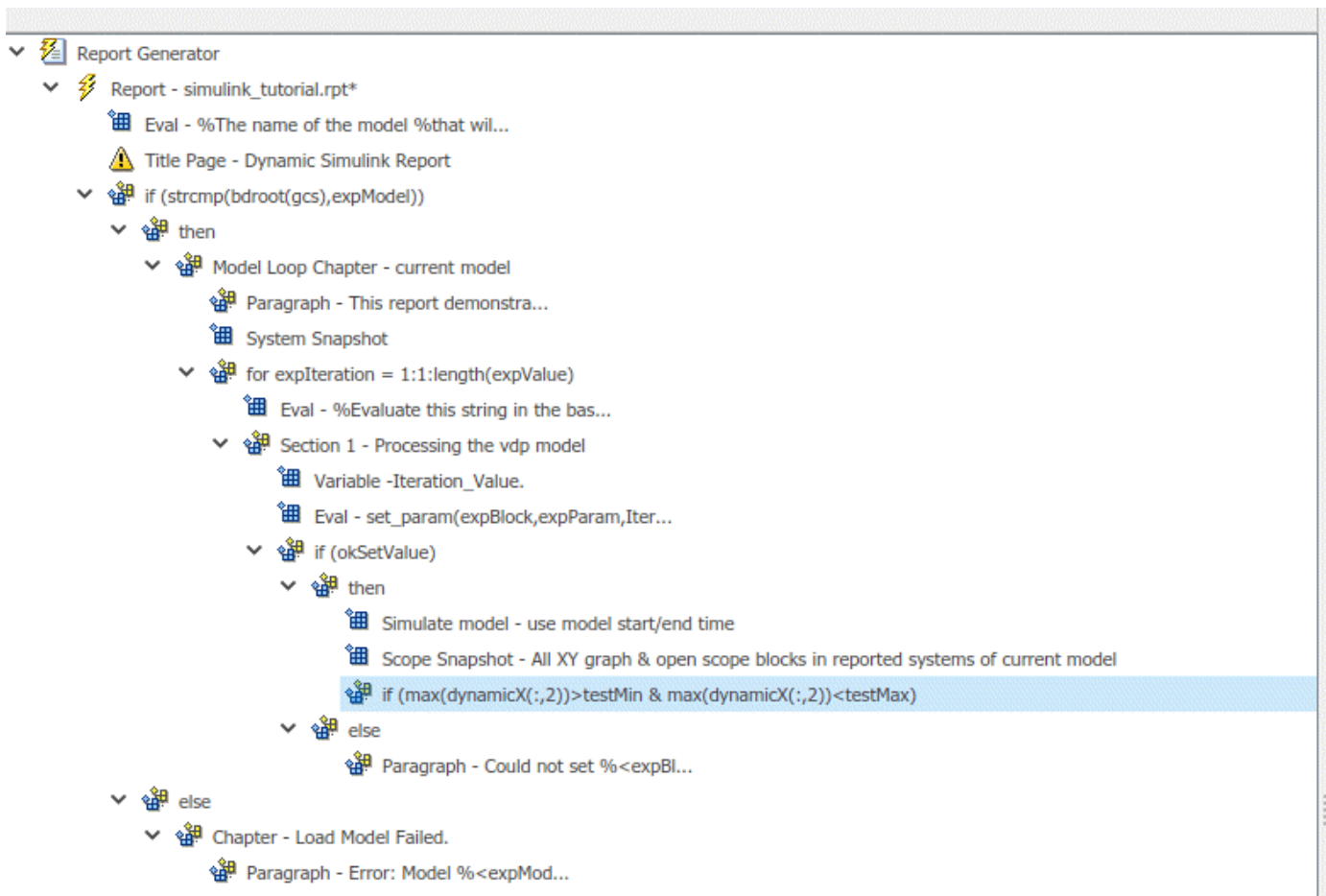
This component captures the scope for each iteration.



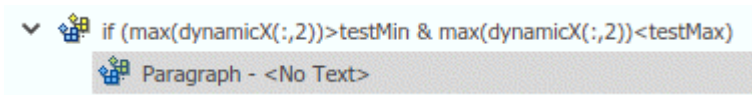
- 6 In the Properties pane on the right:
  - a In the **Paper orientation** selection list, select Portrait.
  - b For the **Image size**, enter [5 4].
  - c In the **Scaling** selection list, select Zoom.
  - d Enter 75 for the % value.
- 7 Save the report setup file.
- 8 To test to see if the signal data falls within a specified range, add another **Logical If** component:
  - a In the Outline pane on the left, select the **Scope Snapshot** component.
  - b In the Library pane in the middle, scroll up to the **Logical and Flow Control** category.
  - c Double-click the **Logical If** component.
- 9 To test the signal data, replace true in the **Test expression** text box with the following in the Properties pane on the right:
 

```
max(dynamicX(:,2))>testMin & max(dynamicX(:,2))<testMax
```
- 10 Save the report.

The Outline pane looks as follows:



- 11 If this condition is true, the signal data falls within the desired range. Add a **Paragraph** component to print information about the signal data in the report.
  - a In the Outline pane on the left, select the **if** component you just added.
  - b In the Library pane in the middle, under the **Formatting** category, double-click **Paragraph** so that it becomes a child of the **if** component.



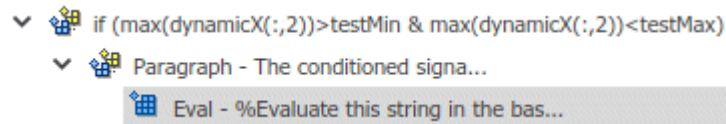
- c In the Properties pane on the right:
  - i From the **Title Options** selection list, select **Custom title**.
  - ii Type **Success** in the text box.
  - iii Enter the following text in the **Paragraph text** text box.

The conditioned signal has a maximum value of  $\%<\max(\text{dynamicX}(:,2))>$ , which lies in the desired range of greater than  $\%<\text{testMin}>$  and less than  $\%<\text{testMax}>$ .

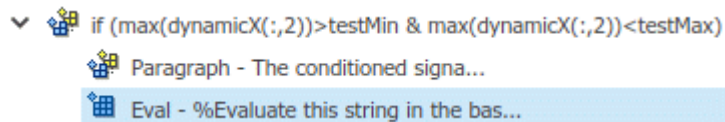
- 12 To save the success values to insert into a table at the end of the iterations, use an **Evaluate MATLAB Expression** component.

- a In the Outline pane on the left, select the **Paragraph** component.
- b In the Library pane in the middle, under the **MATLAB** category, double-click **Evaluate MATLAB Expression**.

An unintended result occurs: the new component is a child of the **Paragraph** component.



- c To make the new component a *sibling* of the **Paragraph** component, in the Outline pane on the left, select the **Eval** component, and then Click the left arrow on the toolbar. The **Eval** component becomes a sibling of the **Paragraph** component.



- 13 In the Properties pane on the right, for the **Eval** component:
  - a Clear the **Insert MATLAB expression in report** and **Display command window output in report** check boxes.
  - b In the **Expression to evaluate in the base workspace** text box, enter the following to save the desired signal values in the exp0kValues array:

```
exp0kValues=[exp0kValues;...
             {Iteration_Value,max(dynamicX(:,2))}];
```

- c Make sure you select **Evaluate this expression if there is an error**. Insert the following text in the text box:

```
disp(['Error during eval: ', evalException.message])
```

- 14 Save the report setup file.

### Create a Post-Test Analysis Section

---

**Note** This section builds on the previous tasks described in the step-by-step example summarized in “Create a Simulink Report Generator Report Interactively” on page 4-50.

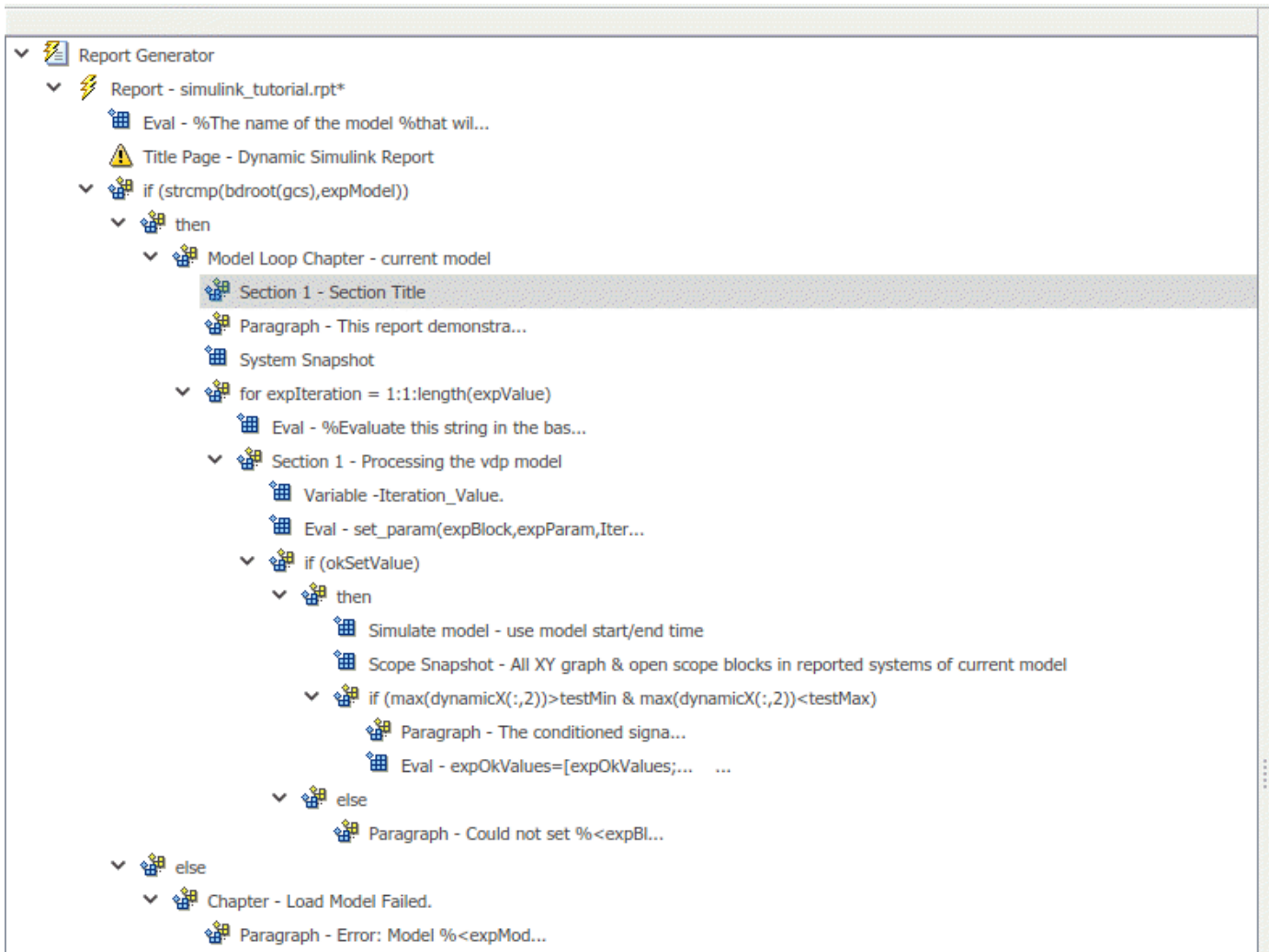
To see the completed report setup file, open Simulink Dynamic Report. The report is for the vdp model.

---

Now that you have collected all the desired values, create the post-test analysis section by creating a table and inserting it into your report at the end of this chapter.

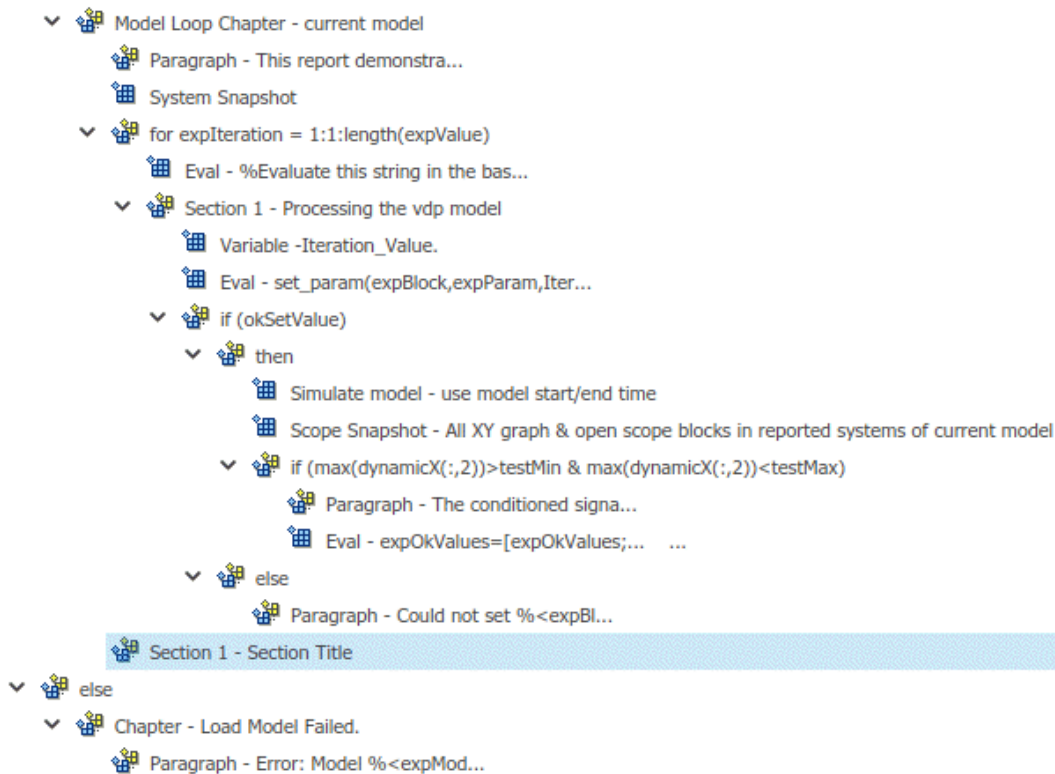
- 1 In the Outline pane on the left, select the **Model Loop Chapter** component.
- 2 In the Library pane in the middle, under the **Formatting** category, double-click **Chapter/Subsection**.

The new section appears at the beginning of the chapter.



Click the **down** arrow three times so **Section 1** moves to the end of the **Model Loop Chapter** component.



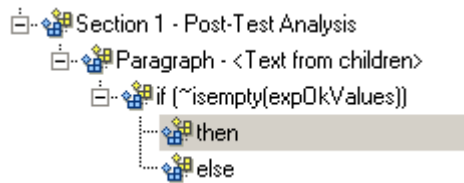


- 3 In the Properties pane on the right:
  - a Select Custom in the **Title** selection list.
  - b Enter Post-Test Analysis in the text box.
- 4 In the Outline pane on the left, select the new **Section 1** component.
- 5 In the Library pane in the middle, under the **Formatting** category, double-click **Paragraph**. Do not change its properties.
- 6 To check whether there are any signal values within the desired range, check the array `expOkValues` with a **Logical If** component. If `expOkValues` is empty, there are no signal values in the desired range. Report the result of this check.
  - a In the Outline pane on the left, select the Paragraph component and add a **Logical If** child component.
  - b In the Properties pane on the right, enter the expression to evaluate in the **Test expression** text box:
 

```
~isempty(expOkValues)
```

This expression evaluates to 0 (false) if `expOkValues` is empty; otherwise, it evaluates to 1 (true).
  - c In the Outline pane on the left, select the **if (~isempty(expOkValue))** component and add the **Logical Else** component as a child.
  - d Select the **if (~isempty(expOkValue))** component again and add the **Logical Then** component as a child.

The two components are siblings in the Outline pane on the left.



- 7 Save the report setup file.
- 8 Now, insert report components to handle the case where `expOkValues` is empty; that is, where no signal values fall within the designated range.
  - a In the Outline pane on the left, select the **else** component.
  - b In the Library pane in the middle, double-click the **Text** component to add it as a child of the **else** component.
  - c In the Properties pane on the right, in the **Text to include in report** text box, enter the following:
 

```
None of the selected iteration values had
a maximum signal value between %<testMin> and %<testMax>.
```
- 9 Now handle the case where `expOkValues` is not empty and you want to insert a table of the acceptable signal values.
  - a In the Outline pane on the left, select the **then** component.
  - b Add a **Text** component as a child to the **then** component.
  - c In the Properties pane on the right, in the **Text to include in report** text box, enter the following text.
 

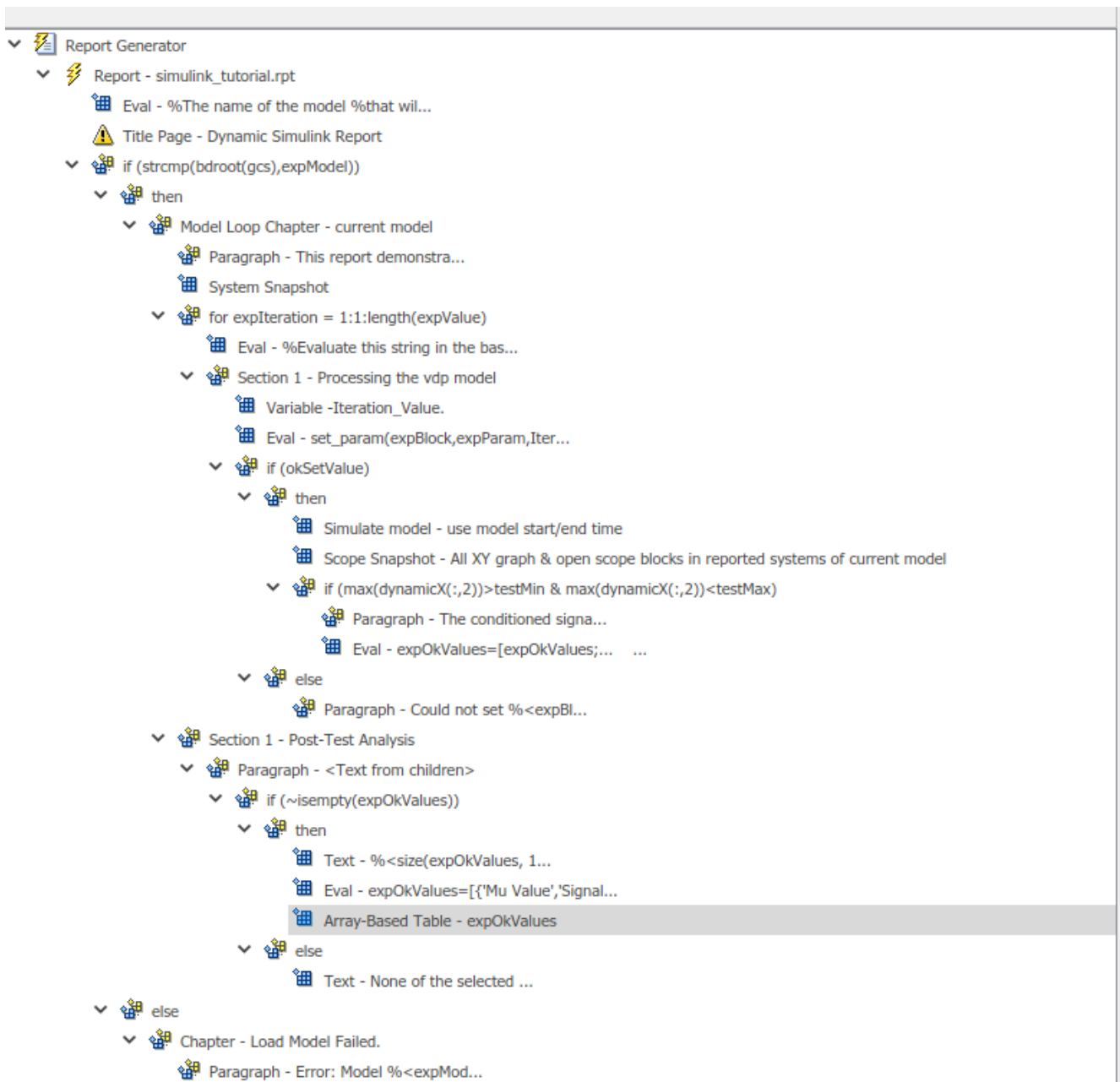
```
%<size(expOkValues, 1)> values for %<expBlock> were
found that resulted in a maximum signal value greater
than %<testMin> but less than %<testMax>. The following
table shows those values and their resulting signal maximum.
```
  - d In the Outline pane on the left, select the **Text** component under the **then** component of the **if (~isempty(expOkValues))** component.
- 10 To create an array for use when formatting the table, use the **Evaluate MATLAB Expression** component.
  - a In the Library pane in the middle, double-click **Evaluate MATLAB Expression**.
  - b In the Properties pane on the right:
    - i Clear the **Insert MATLAB expression in report** and **Display command window output in report** check boxes.
    - ii The next component of the report uses the strings `Mu Value` and `Signal Maximum` as table header values. Add the strings to the front of the `expOkValues` cell array by entering the following text into the **Expression to evaluate in the base workspace** text box:
 

```
expOkValues=[{'Mu Value','Signal Maximum'} expOkValues];
```
    - iii Make sure you select the **Evaluate this expression if there is an error** check box. Enter the following text into the text box:
 

```
disp(['Error during eval: ', evalExpression.message])
```

- 11** In the Outline pane on the left, select the **Eval** component.
- 12** In the Library pane in the middle, under the **Formatting** category, double-click the **Array-Based Table** component so it becomes a sibling of the **Text** and **Eval** components.
- 13** In the Properties pane on the right:
  - a** In the **Workspace variable name** text box, enter `exp0kValues`. The Simulink Report Generator software uses the contents of `exp0kValues` to construct the table.
  - b** In the **Table title** text box, enter `Valid Iteration Values`.
- 14** Save the report setup file.

The Outline pane on the left looks as follows.

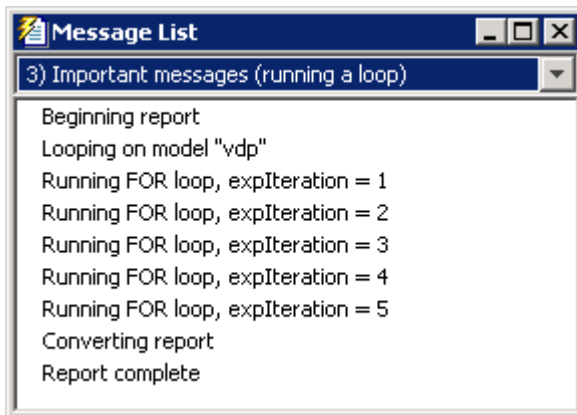


### Generate the Report

To generate the report, click the **Report** icon on the toolbar. The following occurs:

- 1 A Message List window appears, displaying informational and error messages as the report is processed. Specify the level of detail you would like the Message List window to display while the report is being generated. Options range from 0 (least detail) to 6 (most detail). Click the selection list located under the title bar of the Message List window to choose an option.

Message level 3 (Important messages) is used for the remainder of this example.



- 2 The vdp model appears. You can see each time it is simulated.
- 3 The scope window appears. The scope graph changes each time the parameter value changes.
- 4 Each component of the report is highlighted as it executes, in the Outline pane on the left in the Report Explorer window.

When the report generation is complete, the report opens.

# **Dynamic Simulink Report**

## **Using Simulink Report Generator to Document Changes**

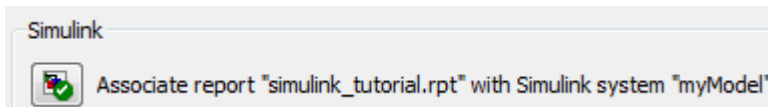
**John Q. Engineer**

## Generate a Report Associated with a Model

You can associate a report with your model. By associating a report with a model, you can use a script to generate the report without specifying the name of the report. You can change the association without modifying the script.

Associating a report with a model sets the model parameter `ReportName` to the name of the report. Each model can have only one report associated with it. You can associate the same report with more than one model.

- 1 Open the model you want to associate with a report.
- 2 In the Report Explorer, from the hierarchy view, select **Report Generator**. The library pane lists your reports.
- 3 Select the report you want to associate with your model.
- 4 In the properties pane, under **Simulink**, click **Associate report with Simulink system**. The report and model names are part of the button label.



- 5 Save the model.
- 6 Create a script to generate the report using `report`. Make sure the model and report template are on the MATLAB path when you run the script.

```
load_system('myModel')
report(get_param('myModel','ReportName'))
bdclose('myModel')
```

You can clear the association clicking **Un-associate Simulink system**.

### See Also

report

### Related Examples

- “Select Report Generation Options”

## Logical and Looping Components

Logical and looping components execute conditionally, determining when a child component executes or how many times a child component executes.

A looping component runs its child components a specified number of times. There are several looping components, such as logical loops, Handle Graphics loops, and model and chart loops. For model and chart loops, you can control aspects such as the order in which the report sorts blocks.

For an example that uses loop components, see “Edit Figure Loop Components”.

You can use loop context functions with loop components. For details, see:

- “Filter with Loop Context Functions” on page 4-87
- “Loop Context Functions” on page 4-89



## Filter with Loop Context Functions

### In this section...

“Create and Save the Setup File” on page 4-87

“Add Components” on page 4-87

“Run the Report” on page 4-88

Use loop context functions to filter the modeling elements to report on and to perform special reporting on specific elements.

In the following example, in a Block Loop component, you use `RptgenSL.getReportedBlock` in a Logical If component to report on targeted blocks within a Block Loop component.

For a summary of loop context functions, see “Loop Context Functions” on page 4-89.

### Create and Save the Setup File

- 1 Open the f14 model.
- 2 At the MATLAB command prompt, enter:  

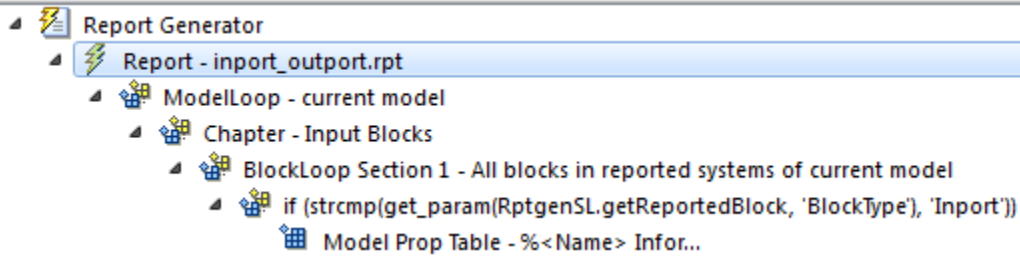
```
report
```
- 3 In the Report Explorer, select **File > New**.
- 4 In the Properties pane, set **Directory** to Present working directory.
- 5 Save the setup file as `inport_output.rpt`.

### Add Components

Add these components to the report, in order.

From this Library Folder	Add this Component	Set this Property
Simulink	Model Loop	N/A
Formatting	Chapter	Title to Inport Blocks
Simulink	Block Loop	N/A
Logical and Flow Control	Logical If	Test Expression to <pre>strcmp(get_param... (RptgenSL.getReportedBlock,'BlockType'),... 'Inport')</pre>
Simulink	Simulink Property Table	N/A

The report setup file looks like this:

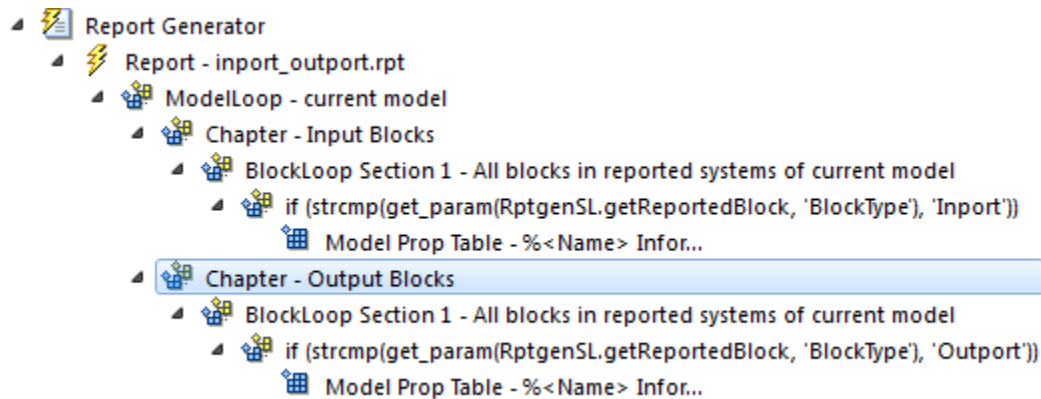


### Run the Report

- 1 Select `inport_outport.rpt`.
- 2 From the context menu, select **Report**.

The report includes a chapter with properties for the Inport blocks only.

If you wish, create a second chapter that reports on Output blocks only, as shown below.



## Loop Context Functions

### In this section...

“For Simulink Modeling Elements” on page 4-89

“For Stateflow Modeling Elements” on page 4-89

You can use these loop context functions in similar ways as shown in “Filter with Loop Context Functions” on page 4-87.

### For Simulink Modeling Elements

Modeling Element	Looping Component	Function
Simulink modeling elements		
Block	<b>Block Loop</b>	RptgenSL.getReportedBlock
Signal	<b>Signal Loop</b>	RptgenSL.getReportedSignal
System	<b>System Loop</b>	RptgenSL.getReportedSystem
Model	<b>Model Loop</b>	RptgenSL.getReportedModel

### For Stateflow Modeling Elements

Modeling Element	Looping Component	Function
Object	<b>Object Loop</b>	RptgenSF.getReportedObject
State	<b>State Loop</b>	RptgenSF.getReportedState
Chart	<b>Chart Loop</b>	RptgenSF.getReportedChart



# **Export Simulink Models to Web Views**

---

## Web Views

### What Is a Web View?

A Web view is an interactive rendition of a model that you can view in a Web browser. You can use Web views to navigate hierarchically to specific subsystems and see properties of blocks and signals. Web views provide a simple way to interactively explore a model. For example, you can view block parameter values without opening a block parameter dialog box.

Use Web views to share models with people who do not have Simulink installed.

You can save Web views of a model over time, creating snapshots of the model as it changes during the development process.

### System Requirements

Although you use Simulink Report Generator software to create Web views, you can display a Web view in a browser, even if you do not have Simulink Report Generator installed.

By default, when you export a Web view, that Web view automatically displays in your default Web browser. Web views require a Web browser that supports SVG natively.

### Web View Files

By default, exporting a Web view creates a zip file that includes the Web view HTML file, as well as files that support Web view display. Supporting files include files include `.svg` and `.png` files. Zip file packaging compresses the files and consolidates the Web view and supporting files into one zip file.

You can choose to export the Web view files as the Web view HTML file and the supporting files, in a folder, without being zipped. You can open the Web view HTML file directly, without having to open a non-zipped file. You can also choose to export the Web view files as both a zip file and as non-zipped files.

The default name of the zip file or folder that contains the non-zipped Web view files is the name of the model that contains the systems to export. You can specify a different file or folder name.

The default location for storing Web view files is the MATLAB current folder. You can choose a different folder.

If you send Web view files to someone else, consider whether you need to explain how to access the Web view file.

### See Also


#### Related Examples

- “Export Models to Web View Files” on page 5-4
- “Display and Navigate a Web View” on page 5-5
- “Create and Use a Web View” on page 5-11

- “Include Model Requirements and Coverage Data in a Web View” on page 5-17

## Export Models to Web View Files

To export a model to a web view file:

- 1 Open the model to export.
- 2 In the Simulink Toolstrip, on the **Simulation** tab, in the **File** section, click **Save** . Under **Export Model To**, select **Web View**.
- 3 Under **Systems to Export**, select the levels of the model to export, in relationship to the system currently displayed or chart currently selected in the Simulink Editor.
- 4 For the systems in the levels that you are exporting, under **Include Options**, select any kinds of systems you want the web view user to be able to navigate below the Subsystem or Model block, to the underlying blocks or models.

If you select more than one kind of system, the criteria for exporting information for interacting with the contents of the systems are applied downward through the model hierarchy. For example, if you did not select **Referenced Models** when you exported the model, regardless of how you set the **Library Links** option, in the web view you cannot interact with a library link block that is inside of a referenced model.

- 5 Under **Systems to Exclude**, select any systems that you do not want to export. To select multiple systems, press the **Ctrl** key and select systems.
- 6 To avoid overwriting existing exported web view packages, select **If package exists, increment name to prevent overwriting**.
- 7 In **Package Type**, specify whether you want to package the web view as a zipped file (the default packaging). In **Package name**, you can specify a name for the zip file or for the folder for the web view files.
- 8 Select optional views.
  - If you have Simulink Coverage™ installed, on the **Optional Views** tab, you can select **Include Coverage view**.
  - If you have Simulink Requirements on the **Optional Views** tab, you can select **Include Requirements view**.
- 9 Click **Export**.

### See Also

#### Functions

slwebview

### Related Examples

- “Display and Navigate a Web View” on page 5-5
- “Create and Use a Web View” on page 5-11
- “Include Model Requirements and Coverage Data in a Web View” on page 5-17
- “Web Views” on page 5-2
- “Web View Files” on page 5-2



## Display and Navigate a Web View

### In this section...

“Display a Web View When You Export It” on page 5-5

“Open a Web View File in a Web Browser” on page 5-5

“View Contents of a System” on page 5-6

“View Block Parameters and Signal Properties” on page 5-7

“Access Optional Web View Information” on page 5-7

### Display a Web View When You Export It

When you export a Web view using the Web View dialog box or from the Report Explorer **Web View** pane, the Web view appears in your system web browser.

### Open a Web View File in a Web Browser

To open a Web view file to display in a web browser, navigate to the folder that contains the Web view files, then open the `webview.html` file. For details about file packaging and location, see “Web View Files” on page 5-2.

Before you can open a Web view file in a Google Chrome™ browser, you must set up the browser to allow the Web view file to access files and subfolders in the Web views folder. The setup depends on the platform that you use.

#### Open a Web View in a Google Chrome Browser on a Windows Platform

Create a shortcut that opens Chrome™ with the `--allow-file-access-from-files` flag. For example, on Windows 10:

- 1 Click the **Start** key and type Chrome.
- 2 In the search results, right-click Chrome and select **Open file location**.

- 3 From the list of applications, drag Chrome to your desktop.
- 4 Right-click the shortcut and select **Properties**.
- 5 Append `--allow-file-access-from-files` to the contents of the **Target** box. Be sure to use two hyphens at the beginning and to put a space between the existing content and the content that you append. Click **OK**.

To open a Web View file:

- 1 Close all open Chrome browsers.
- 2 Open a Chrome browser by using the shortcut that includes the `--allow-file-access-from-files` flag.
- 3 Open the Web view file in the open Chrome browser. For example, drag the file to the browser or right-click the file and select **Open with > Google Chrome**.

### Open a Web View in a Google Chrome Browser on a Macintosh Platform

- 1 Run **Terminal**. You can find it using **Spotlight**, in **Applications/Utilities**.
- 2 Enter the following text:

```
open/Applications Google\Chrome.app --allow-file-access-from-files
```

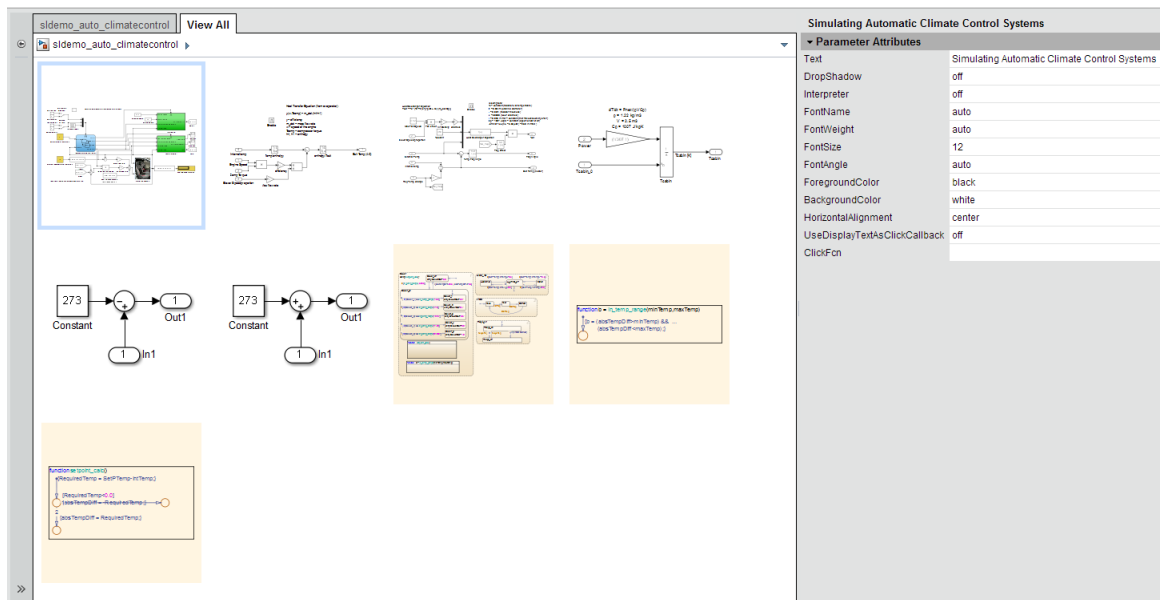
### Open a Web View in a Google Chrome Browser on a Linux Platform

- 1 Run **terminal**.
- 2 Enter the following text:


```
./chromium-browser --allow-file-access-from-files
```

## View Contents of a System

To see a thumbnail of the contents of all of systems in the Web view, click the **View All** tab.



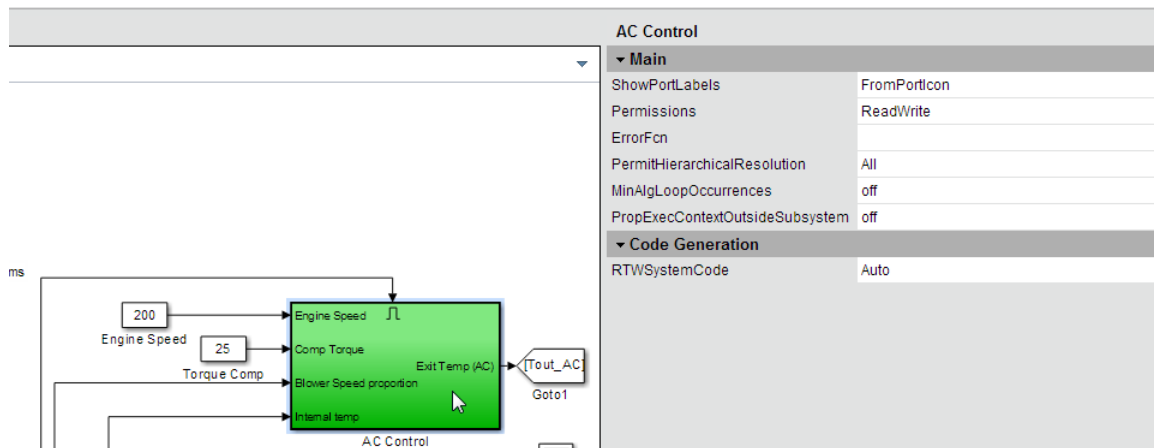
To view the contents of a specific system, use one of these approaches:

- In the model viewer, double-click the system.
- In the model browser, select a system. To expose this pane, click **Hide/Show Model Browser** .
- Click the **View All** tab and click the thumbnail of a system.

To open a system in a separate tab, press **CTRL** and click the system.

## View Block Parameters and Signal Properties

Click a block or signal in the model to see its parameters or properties in the **Object Inspector** pane.



## Access Optional Web View Information

To view the model coverage optional Web view information in a Web view, you must have Simulink Coverage installed. To view the requirements optional Web view information in a Web view, you must have Simulink Requirements installed. To access the information, click a highlighted block (for example, blocks with an orange border have requirements information). The information for that block appears in the **Informers** pane below the model.

## See Also

**Functions**  
slwebview

## Related Examples

- “Create and Use a Web View” on page 5-11
- “Web Views” on page 5-2
- “Web View Files” on page 5-2
- “Include Model Requirements and Coverage Data in a Web View” on page 5-17

## Search a Web View

### In this section...

“Perform a Search” on page 5-8

“Sort Search Results” on page 5-9

“Navigate Between Search Results and Model Elements” on page 5-10

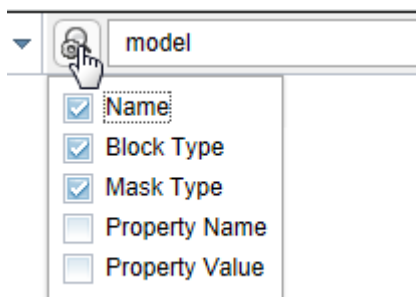
### Perform a Search

1 In a Web View, at the top of the displayed tab, click the search button .

2 In the search box, enter the search term.

Search strings are case-insensitive. The search treats the string as a partial string.

3 To specify search criteria, click the search criteria button and select the types of model element you want to search in.



4 Press **Enter**.

The elements of the model that the search returns appear highlighted. The search results include the name and parent for each returned element.

enablesub
View All

enablesub
output

## Enabled Subsystem Example

**Search Results: enablesub**

Name	Parent
Saturation between -0.75 and 0.75 Output held	enablesub
Abs Output Reset	enablesub
Saturation - Output Held	enablesub
Output Reset	enablesub
Output Held	enablesub

### Sort Search Results

You can sort the search results in alphabetical order. In the search results table, click the **Name** or **Parent** column.

## **Navigate Between Search Results and Model Elements**

To see the corresponding search result for a highlighted model element, click the element.

To highlight the model element for a search result, click the search result.

The **Object Inspector** pane to the right of the model updates to reflect the selected model element or search result.

## Create and Use a Web View

### In this section...

“About This Tutorial” on page 5-11

“Export Specific Systems” on page 5-11

“Navigate the Web View” on page 5-13

“Display Parameters and Properties of Blocks and Signals” on page 5-14

“Open the Web View” on page 5-16


### About This Tutorial

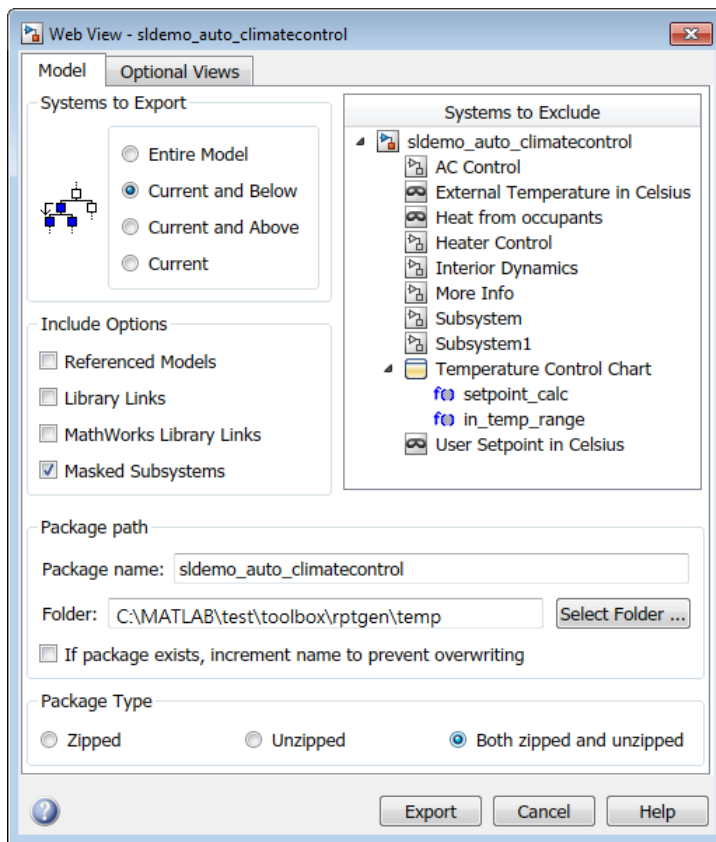
This tutorial takes you through the steps to export a Simulink model to a Web view.

You create a Web view from the Simulink model window using the `sldemo_auto_climatecontrol` model, which is provided with the Simulink software. This model simulates the working of an automatic climate control system in a car.

### Export Specific Systems

When you create the Web view, you can specify export options.

- 1 At the MATLAB command prompt, enter `sldemo_auto_climatecontrol` to open the Simulink model.
- 2 In the Simulink Toolstrip, on the **Simulation** tab, in the **File** section, click **Save** . Under **Export Model To**, select **Web View**.



- 3 In the **Include Options**, select **Masked Subsystems**. This enables users of the Web view to interact with masked blocks.
- 4 In **Systems to Exclude**, select **Temperature Control Chart**.
- 5 Use the **Folder** edit box to specify `climate_control_webview` as the name for the zip file for the exported Web view files.
- 6 Select the **If package exists, increment name to prevent overwriting** check box. Selecting this option prevents overwriting the Web view files if you export multiple Web views from the same model.
- 7 Click **Export**.

Exporting the selected systems to a Web view creates several support files, as well as an HTML file for displaying the systems. In this example, you change the defaults for the naming of the files.

The Web view files are exported, and the Web view appears in a Web browser.



**slidemo\_auto\_climatecontrol** View All

slidemo\_auto\_climatecontrol

**Parameter Attributes**

ModelVersion	1.48
LastModifiedDate	Fri Dec 16 15:19:22 2016
LibraryLinkDisplay	none
ModelBrowserVisibility	off
Dirty	off
Description	Simulating Automatic Climate Control Systems

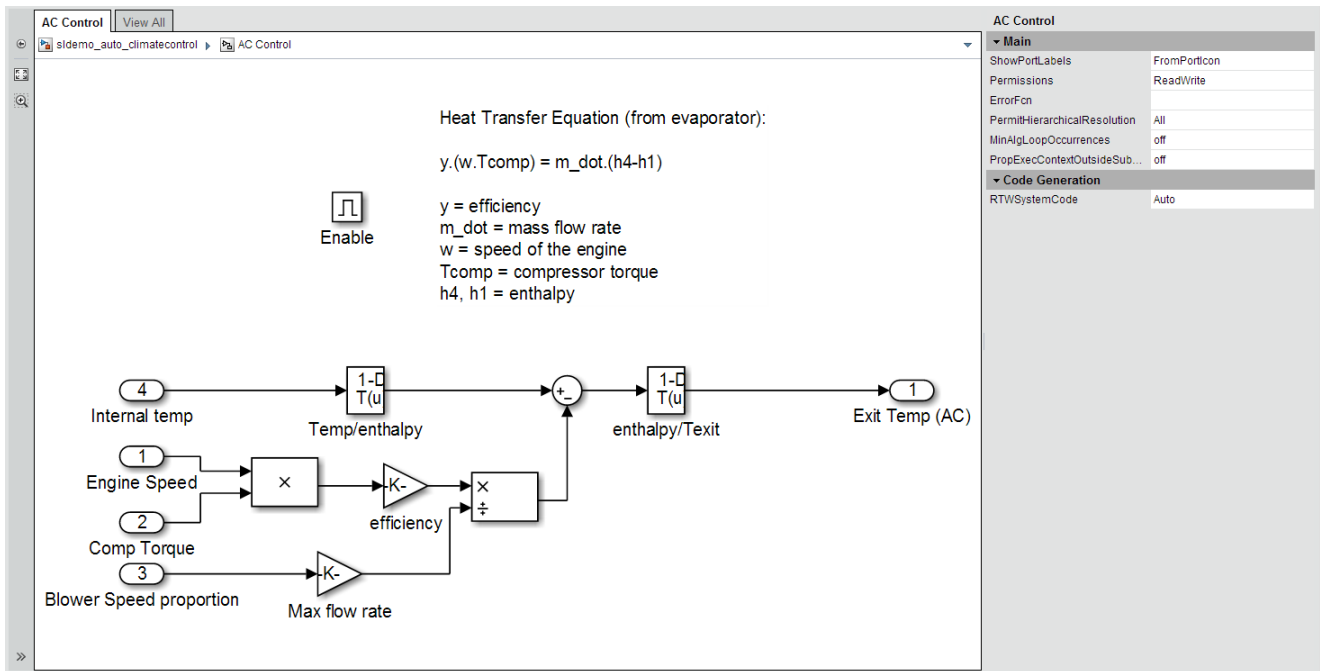
This example shows how to use Simulink(R) and Stateflow(R) to simulate the working of an automatic climate control system in a car. You can enter a temperature value you would like the air in the car to reach by double clicking the User Setpoint in Celsius Block and entering the temperature value. You can also set the External Temperature in Celsius in a similar way. The numerical display on the right-hand side of the model shows the reading of a temperature sensor placed behind the driver's head. This is the temperature that the driver should be feeling. When the model is run and the climate control is active, it is this display box whose value changes to show the change of temperature in the car.

The Temperature Control Chart appears in the top level of the model, but you cannot open that chart in the web view to see its contents.


## Navigate the Web View

By default, if you export a whole model to a Web view, the **Model Viewer** pane shows the whole model. You can display specific systems in the Web view. For example:

- 1 In the model viewer, double-click the AC Control subsystem.



The AC Control subsystem appears in the model viewer. The tab label reflects the name of the currently displayed subsystem.

- 2 Open the model browser (hidden by default). Click **Hide/Show Model Browser** .
- 3 Open another system, in a separate tab. In the model browser, **CTRL+click** select the Heater Control system.
- 4 Drag the AC Control system to the top of model viewer. Place the cursor in the display area, hold down the mouse scroll wheel, and drag.
- 5 Zoom the display with the mouse scroll wheel.

### Display Parameters and Properties of Blocks and Signals

- 1 In the model browser, select `sldemo_auto_climatecontrol`.
- 2 Double-click the AC Control subsystem.
- 3 Click the Temp/enthalpy block to view the block parameter values. The **Object Inspector** pane groups the block parameters by the block parameter dialog box tabs.

AC Control

Heat Transfer Equation (from evaporator):

$$y.(w.Tcomp) = m\_dot.(h4-h1)$$

$y$  = efficiency  
 $m\_dot$  = mass flow rate  
 $w$  = speed of the engine  
 $Tcomp$  = compressor torque  
 $h4, h1$  = enthalpy

Enable

**Table and Breakpoints**

NumberOfTableDimensions: 1

Table: [219.97 230.02 240.02 250.05 260.09 270.11 280.13 285.14 290.16 295.17 300.19 305.22 310.24 315.27 320.29]

BreakpointsForDimension1: [220 230 240 250 260 270 280 285 290 295 300 305 310 315 320]

SampleTime: -1

**Algorithm**

InterpMethod: Linear

ExtrapMethod: Linear

DiagnosticForOutOfRangeInput: None

RemoveProtectionInput: off

IndexSearchMethod: Binary search

BeginIndexSearchUsingPrevi...: off

UseOneInputPortForAllInputD...: off

SupportTunableTableSize: off

**Data Types**

TableDataTypeStr: Inherit: Same as output

TableMin: []

TableMax: []

BreakpointsForDimension1D...: Inherit: Same as corresponding input

BreakpointsForDimension1Min: []

BreakpointsForDimension1Max: []

FractionDataTypeStr: Inherit: Inherit via internal rule

IntermediateResultsDataTyp...: Inherit: Same as output

OutDataTypeStr: Inherit: Same as first input

OutMin: []

OutMax: []

InternalRulePriority: Precision

InputSameDT: on

LockScale: off

RndMeth: Floor

SaturateOnIntegerOverflow: on

- Click the input signal for the Exit Temp (AC) block to display the signal properties.

Heat Transfer Equation (from evaporator):

$$y.(w.T_{comp}) = \dot{m}.(h_4 - h_1)$$

$y$  = efficiency  
 $\dot{m}$  = mass flow rate  
 $w$  = speed of the engine  
 $T_{comp}$  = compressor torque  
 $h_4, h_1$  = enthalpy

The diagram shows a Simulink model with an input 'icy' entering a multiplier block. The output of the multiplier block goes to an adder block (+). The output of the adder block goes to a transfer function block labeled '1-D / T(u)'. The output of the transfer function block goes to an output block labeled '1' with the text 'Exit Temp (AC)' below it.

Try navigating to other parts of the web view.

- 5 Close the web view.

### Open the Web View

In the MATLAB current folder (or wherever you saved the zip file when you performed the steps in “Export Specific Systems” on page 5-11), extract the `climate_control_webview` zip file contents and open the `webview.html` file.

### See Also

#### Related Examples

- “Export Models to Web View Files” on page 5-4
- “Display and Navigate a Web View” on page 5-5

#### More About

- “Web Views” on page 5-2

## Include Model Requirements and Coverage Data in a Web View

You can include these optional views in a model web view:

- Model requirements (requires Simulink Requirements)
- Model coverage (requires Simulink Coverage)


### Prepare the Model for an Optional Web View

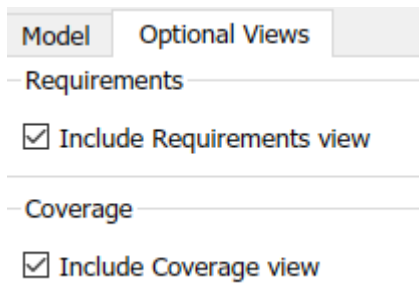
Before you can include an optional view in a web view, the model must contain requirements or coverage data:

- To add requirements to a model, see “Requirements Definition” (Simulink Requirements).
- To add coverage data to a model, simulate the model with coverage enabled. For a web view to include the coverage data, you must also select **Save last run in workspace variable** in the **Coverage > Results** pane of the Configuration Parameters dialog box. See “Specify Code Coverage Options” (Simulink Coverage).

### Add Optional Views to a Web View

To include requirements or coverage data in a model web view:



- 1 Open the Web View dialog box. In the Simulink Toolstrip, on the **Simulation** tab, in the **File** section, click **Save** . Under **Export Model To**, select **Web View**.
- 2 Open the **Optional Views** tab and select the views. By default, both the requirements and coverage views are selected.



- 3 Click **Export**.

### Open an Optional Web View

Open the web view in a web browser. See “Display and Navigate a Web View” on page 5-5. In the lower-left palette of the web view, there is a button for each optional view that you selected.

- To access the requirements view, click the  button. This view highlights model elements that have requirements and outlines model elements that contain elements that have requirements.
- To access the coverage data view, click the  button. Click model elements for coverage information.

## **See Also**

### **Related Examples**

- “Export Models to Web View Files” on page 5-4
- “Display and Navigate a Web View” on page 5-5
- “Introduction to Simulink Requirements” (Simulink Requirements)
- “Model Coverage” (Simulink Coverage)

## Embedded Web View Reports

### In this section...

- “What Is Embedded Web View?” on page 5-19
- “Navigating an Embedded Web View Report” on page 5-20
- “Embedded Web View Packaging” on page 5-22
- “View Embedded Web View Reports” on page 5-22

### What Is Embedded Web View?

Embedded Web View is a MATLAB application programming interface (API) that allows you to create HTML reports containing “Web Views” on page 5-2. Embedded Web View lets you generate compact and navigable reports from Simulink models. For example, the following image shows a control system calibration guide generated by a MATLAB program based on Embedded Web View.

AirChargeDetermination	
- Parameter Attributes	
ModelVer...	1.683
LastModifi...	Mon Mar 18 22:52:07 2013
LibraryLin...	all
ModelBro...	off
Dirty	off
Description	

This calibration guide comprises three hyperlinked panes:

- Left pane — Calibration guide table of contents
- Center pane — Calibration guide content
- Right pane — web view of the Simulink model used to create the control system

Use the MATLAB Report Generator DOM and Report APIs and the Simulink Report Generator Report APIs to generate the calibration guide content. Use the Embedded Web View API to create the report. The report is a user interface with guide content, an embedded Web View, and hyperlinks between the Web View and the guide report text. The resulting guide runs in any standard browser without requiring either MATLAB or Simulink.

## Navigating an Embedded Web View Report

The Embedded Web View API facilitates report navigation by letting you create two-way hyperlinks between report content and the embedded Web View. The hyperlinks are illustrated in the following calibration guide example.

### Navigate Via a Table of Contents

The Embedded Web View API creates a table of contents (TOC) based on your report section headings. Clicking a TOC heading entry displays the corresponding report section in the contents pane. It also displays and flashes the corresponding model element. The linked model block in this calibration guide example is shown highlighted in yellow in the Web View pane.

The screenshot displays a web view report interface with four main panes:

- Table of Contents (TOC):** A list of sections with hyperlinks. An arrow points to "4. Cylinder 2 Fresh Mass".
- Section Header:** "Cylinder 2 Fresh Mass" is highlighted in yellow.
- Best Guess Calibration 3D Plot:** A 3D surface plot showing the relationship between variables.
- Best Guess Calibration 3D Table:** A table with the following data:
 

	Name	Units
Table	ACDFreshMass2	mg
Row Index	ACDECPbp	deg
Column Index	ACDIVCMassbp	mg
- Simulink Model Diagram:** A block diagram titled "Fresh Air Mass Calculation" with a yellow highlight on a specific block.
- Signal Attributes Panel:** A table of parameters for "Cylinder 2 Fresh Mass":
 

- Main	
RowIndex	ACDECPbp
ColumnIn...	ACDIVCMas
Table	ACDFreshMa
LookUpMeth	Interpolation- Use End Values
- Signal Attributes	
OutMin	[]
OutMax	[]
InputSam...	off
OutDataT...	Inherit: Same as first input
LockScale	off
RndMeth	Floor
SaturateO...	off
Other	

### Navigate Via Contents

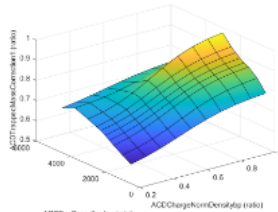
This calibration guide has hyperlinks from any text element in the content to any model element in the Web View pane. Clicking the text element in the contents displays and flashes the corresponding model element. The linked model block in this calibration report example is shown highlighted in yellow in the Web View pane.



3. [Cylinder 1 Fresh Mass Deac Mode](#)
4. [Cylinder 2 Fresh Mass Calculation](#)
8. [Trapped Mass Calculation](#)
  1. [Calibration Requirements in Trapped Mass Calculation](#)
  2. [Cylinder 1 IVC Volume Calibration \(L\)](#)
  3. [Cylinder 2 IVC Volume Calibration \(L\)](#)
  4. [Cylinder 1 Deac Mode Trapped Mass Correction](#)
  5. [Cylinder 1 Trapped Mass Correction](#)
  6. [Cylinder 2 Trapped Mass Correction](#)
  7. [Calibration](#)

### Cylinder 1 Trapped Mass Correction

Best Guess Calibration 3D Plot:



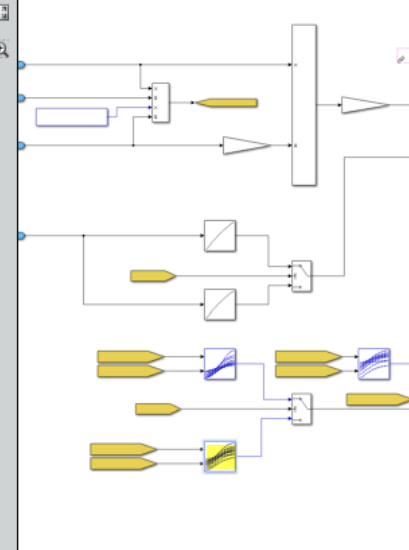
Best Guess Calibration 3D Table:

Name	Unit
Table	ACDTrappedMassCorrection1
Row Index	ACDCChargeNormDensitybp
Column Index	ACDEngSpeedbp

Best Guess Calibration 3D Table:

### Trapped Mass Calculation

View All



### Cylinder 1 Trapped Mass Correction

**- Main**

RowIndex	ACDCChargeNormDensitybp
ColumnIndex	ACDEngSpeedbp
Table	ACDTrappedMassCorrection1
LookUpMethod	Interpolation-Use End Values

**- Signal Attributes**

OutMin	[]
OutMax	[]
InputSampling	off
OutDataTransfer	Inherit: Same as first input
LockScale	off
RndMeth	Floor
SaturateOutput	off

**- Other**

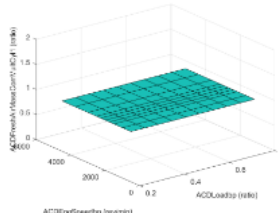
### Navigate Via Web View

This calibration guide has hyperlinks from any element in the Web View pane to any location in the report content pane. Clicking the model element displays the corresponding location in the content pane as shown highlighted in this calibration guide example.

1. [AirChargeDetermination](#)
  1. [Air Charge Determination](#)
  2. [Air Charge Corrections](#)
    1. [Calibration Requirements in Air Charge Corrections](#)
    2. [Fresh Air Charge Correction Cylinder 1](#)
    3. [Fresh Air Charge Correction Cylinder 2](#)
    4. [Fresh Air Charge Correction Deac Mode](#)
  3. [Air Lead Multiplier Calculation](#)
    1. [Calibration Requirements in Air Lead Multiplier Calculation](#)

### Fresh Air Charge Correction Cylinder 1

Best Guess Calibration 3D Plot:



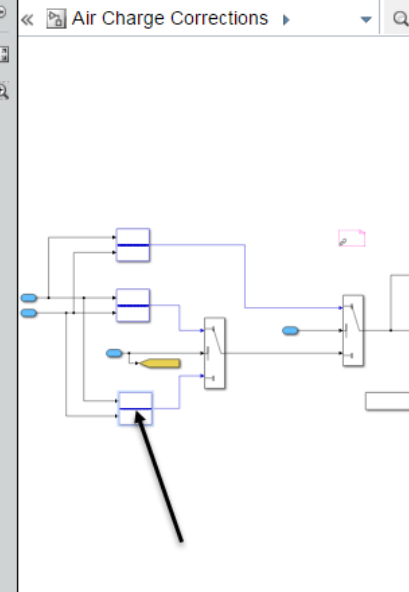
Best Guess Calibration 3D Table:

Name	Unit
Table	ACDFreshAirMassCorrMultCylinder1
Row Index	ACDLoadbp
Column Index	ACDEngSpeedbp

Best Guess Calibration 3D Table:

### Air Charge Corrections

View All



### Fresh Air Charge Correction Cylinder 1

**- Main**

RowIndex	ACDLoadbp
ColumnIndex	ACDEngSpeedbp
Table	ACDFreshAirMassCorrMultCylinder1
LookUpMethod	Interpolation-Use End Values

**- Signal Attributes**

OutMin	[]
OutMax	[]
InputSampling	off
OutDataTransfer	Inherit: Same as first input
LockScale	off
RndMeth	Floor
SaturateOutput	off

**- Other**

## Embedded Web View Packaging

The Embedded Web View API generates a report as a package of HTML, image, style sheet, JavaScript®, and JavaScript Object Notation (JSON) files organized into folders. By default, the API produces both zipped and unzipped versions of a report package in the current MATLAB folder.

## View Embedded Web View Reports

If MATLAB Report Generator is installed on your system, you can use the report generator `rptview` function to view a zipped or unzipped Embedded Web View.

To view an Embedded Web View report on systems that do not have MATLAB installed,

- 1 Unzip the report in an empty, writable folder on your system. This step creates a file named `webview.html` in the folder and a subfolder containing supporting files.
- 2 Open `webview.html` in a browser to view the report.

## See Also

### Related Examples

- “Create an Embedded Web View Report Generator” on page 5-23

## Create an Embedded Web View Report Generator

To create a MATLAB program that generates an Embedded Web View report:

- 1 Create a MATLAB class that defines a report object that generates a Web View report. See “Create an Embedded Web View Report Generator Class” on page 5-23 for a summary of the steps to create and define the class.
- 2 To generate the report, use this class in a MATLAB program. See “Generate an Embedded Web View Report” on page 5-34.

For general information and how to view and navigate a generated report, see “Embedded Web View Reports” on page 5-19.

### Create an Embedded Web View Report Generator Class

To create an Embedded Web View report class, use this workflow:

- 1 Open the MATLAB editor.
- 2 To create a default class definition file, select **New > Class**.
- 3 Replace the default content of the new class definition file with this Embedded Web View class definition template:

```
classdef REPORT_GENERATOR < slreportgen.webview.EmbeddedWebViewDocument
    methods

        % Report generator constructor
        function rptev = REPORT_GENERATOR(reportName,modelName)
            rptev@slreportgen.webview.EmbeddedWebViewDocument(...
                reportName,modelName);
        end

        % Report content generator
        function fillContent(rptev)

        end
    end
end
```

- 4 Replace the placeholder, `REPORT_GENERATOR`, with the name of your report generator.
- 5 To save your new class definition file using the name you used to replace the `REPORT_GENERATOR` placeholder, select **Editor > Save As**.
- 6 To specify the export options and warning suppression option for your report generator, edit the template constructor. See “Specify Export Options for Embedded Web View Report” on page 5-24 and “Suppress Link Warning Messages for Embedded Web View Report” on page 5-33, respectively.
- 7 To include the report content, table of contents, model, and hyperlinks to embed in your report, edit the content generator (`fillContent`) function. These topics describe how to add content to the Embedded Web View report generator. You can perform these tasks in any order.
  - “Specify Document Content for Embedded Web View Report” on page 5-25
  - “Generate Table of Contents for Embedded Web View Report” on page 5-26
  - “Get Model Objects for Embedded Web View Report” on page 5-27
  - “Create Hyperlinks for Embedded Web View Report” on page 5-28
- 8 Save your changes to the new class definition file.

## Specify Export Options for Embedded Web View Report

The `slreportgen.webview.EmbeddedWebViewDocument` base class of your report generator exports and embeds a Web View of the model specified by the constructor in the generated report. By default the base class does not export block diagrams referenced by the top-level model or block diagrams of masked subsystems. To export these types of block diagrams, you must use an `ExportOptions` property inherited from the base class of your generator. You use this property in the constructor of the generator.

For example, the following constructor includes the `ExportOptions` property. This property exports the block diagrams of masked subsystems and models and library blocks referenced by the exported model.

```
function rptvar = SystemDesignVariables...  
    (reportName, modelName)  
    rptvar@slreportgen.webview.EmbeddedWebViewDocument...  
    (reportName,modelName);  
    rptvar.ExportOptions.IncludeMaskedSubsystems = true;  
    rptvar.ExportOptions.IncludeSimulinkLibraryLinks = true;  
    rptvar.ExportOptions.IncludeReferencedModels = true;  
end
```

If you export referenced block diagrams, to view them, click the block icon that references the diagram in the Web View. You can also create hyperlinks from the generated report content to these block diagrams.

For other tasks to create your Embedded Web View generator, see

- “Specify Document Content for Embedded Web View Report” on page 5-25
- “Generate Table of Contents for Embedded Web View Report” on page 5-26
- “Get Model Objects for Embedded Web View Report” on page 5-27
- “Create Hyperlinks for Embedded Web View Report” on page 5-28
- “Suppress Link Warning Messages for Embedded Web View Report” on page 5-33

To generate the Embedded Web View report, see “Generate an Embedded Web View Report” on page 5-34.

## Specify Document Content for Embedded Web View Report

The `fillContent` method fills the document pane of the report that it generates with content specified by DOM and Report API objects. See “Report Generator Creation”. The report generator inherits an `add` method from its `slreportgen.webview.EmbeddedWebViewDocument` base class. Use the `add` method in the `fillContent` method of your report generator. Each call to the `add` method adds its content after the previously added content. The `fillContent` method can create a document of any length and content by repeated calls to the `add` method.

For example, this `fillContent` method begins by creating a report title.

```
function fillContent(rpt)
    import mlreportgen.dom.*
    import mlreportgen.report.*
    add(rpt, TitlePage("Title", "System Design Variable Report"));
    ...
end
```

For other tasks to create your Embedded Web View generator, see:

- “Specify Export Options for Embedded Web View Report” on page 5-24
- “Generate Table of Contents for Embedded Web View Report” on page 5-26
- “Get Model Objects for Embedded Web View Report” on page 5-27
- “Create Hyperlinks for Embedded Web View Report” on page 5-28
- “Suppress Link Warning Messages for Embedded Web View Report” on page 5-33

To generate the Embedded Web View report, see “Generate an Embedded Web View Report” on page 5-34.

## Generate Table of Contents for Embedded Web View Report

The `slreportgen.webview.EmbeddedWebViewDocument` base class of an Embedded Web View report generator embeds JavaScript in the generated Embedded Web View reports. In addition to generating other portions of the report, the JavaScript generates a table of contents from the document section headings. When you open the report in a web browser, the hyperlinked table of contents appears.

To use this feature, your report generator `fillContent` method must use Report API `Chapter` or `Section` objects, or DOM API `Heading` objects to begin the sections and subsections of the report. For example:

```
function fillContent(rpt)

import mlreportgen.dom.*
import mlreportgen.report.*

model = getExportModels(rpt);
model= model{1};
add(rpt, TitlePage("Title", [model " Report"], "Author",""));
finder = slreportgen.finder.ModelVariableFinder(model);

% Create a Variables Chapter
ch = Chapter("Variables");

while hasNext(finder)
    result = next(finder);
    % Create a section for the variable
    s = Section(result.Name);

    reporter = getReporter(result);
    add(s, reporter);

    % Add this section to the chapter
    add(ch, s);
end

% Add the chapter to the report
add(rpt, ch);
end
```

For other tasks to create your Embedded Web View generator, see:

- “Specify Export Options for Embedded Web View Report” on page 5-24
- “Specify Document Content for Embedded Web View Report” on page 5-25
- “Get Model Objects for Embedded Web View Report” on page 5-27
- “Create Hyperlinks for Embedded Web View Report” on page 5-28
- “Suppress Link Warning Messages for Embedded Web View Report” on page 5-33

To generate the Embedded Web View report, see “Generate an Embedded Web View Report” on page 5-34.

## Get Model Objects for Embedded Web View Report

The Embedded Web View report generator `slreportgen.webview.EmbeddedWebViewDocument` base class provides a set of methods for finding model objects to export to the generated report. You use these methods in the report generator `fillContent` method to obtain information about the exported objects for the report document.

To find objects to export from the model, use these methods, which are inherited from the `slreportgen.webview.EmbeddedWebViewDocument` base class of the report generator.

Method	Purpose
<code>getExportModels</code>	Get models to be exported to the report as Web Views.
<code>getExportDiagrams</code>	Get block diagrams to be exported to the report
<code>getExportSimulinkSubSystems</code>	Get subsystem blocks to be exported to the report
<code>getExportStateflowDiagrams</code>	Get Stateflow charts to be exported to the report
<code>getExportStateflowCharts</code>	Get Stateflow chart elements to be exported in this report

This sample code shows how to get a model to use as the Web View.

```
function fillContent(rpt)
    import mlreportgen.dom.*
    model = getExportModels(rpt);
    model = model{1};
    ...
end
```

For other tasks to create your Embedded Web View generator, see:

- “Specify Export Options for Embedded Web View Report” on page 5-24
- “Specify Document Content for Embedded Web View Report” on page 5-25
- “Generate Table of Contents for Embedded Web View Report” on page 5-26
- “Create Hyperlinks for Embedded Web View Report” on page 5-28
- “Suppress Link Warning Messages for Embedded Web View Report” on page 5-33

To generate the Embedded Web View report, see “Generate an Embedded Web View Report” on page 5-34.

## Create Hyperlinks for Embedded Web View Report

To create one-way and two-way hyperlinks between the document pane and the web view embedded in the report, use these methods. These linking methods are inherited from the `slreportgen.webview.EmbeddedWebViewDocument` base class of the report generator.

- `createDiagramTwoWayLink` — Create a two-way link between a document location and a diagram in the embedded web view. Clicking a link created by this method in the document opens the target diagram in the web view. Clicking in the diagram scrolls the document pane to the target document location.
- `createElementTwoWayLink` — Create a two-way link between a document location and a diagram element in the embedded web view. Clicking a link created by this method in a document opens the diagram containing the model element and flashes the element. Clicking the element in the diagram scrolls the document pane to the target document location.
- `createDiagramLink` — Creates a link from the document to a diagram in the embedded web view.
- `createElementLink` — Creates a link from the document to an element of a block diagram in the embedded web view.

In the following example class, `ExampleWebView`, the `fillContent` method uses `createDiagramTwoWayLink` and `createElementTwoWayLink` to create two-way links between the document panel and the embedded web view in an embedded web view report. To create one-way links from the document panel to the embedded web view, replace `createDiagramTwoWayLink` with `createDiagramLink` and `createElementTwoWayLink` with `createElementLink`.

```
classdef ExampleWebView < slreportgen.webview.EmbeddedWebViewDocument

    methods
        function wvdoc = ExampleWebView(reportPath,modelName)
            % Invoke the EmbeddedWebViewDocument constructor, which
            % saves the report path and model name for use by the
            % report's fill methods.
            wvdoc@slreportgen.webview.EmbeddedWebViewDocument(reportPath,modelName);
        end

        function fillContent(wvdoc)
            % Fill the Content hole in the report template with design
            % variable information. You can use DOM or Report API methods
            % to create, format, add, and append content to this report.

            [~, handles] = getExportDiagrams(wvdoc);

            n = numel(handles);
            for i = 1:n
                diagHandle = handles{i};
                diagHeading = createDiagramTwoWayLink(wvdoc,diagHandle, ...
                    mlreportgen.dom.Heading(2,get_param(diagHandle,'Name')));
                append(wvdoc,diagHeading);

                blockFinder = slreportgen.finder.BlockFinder(diagHandle);

                while hasNext(blockFinder)
                    r = next(blockFinder);
                    elemHandle = r.Object;
                end
            end
        end
    end
end
```



```

elemHeading = createElementTwoWayLink(wvdoc,elemHandle, ...
    mlreportgen.dom.Heading(3,get_param(elemHandle,'Name')));

append(wvdoc,elemHeading);
end
end
end
end
end
end
end

```

This code creates an embedded web view report for the vdp model using the ExampleWebView class.

```

model = 'vdp';
open_system(model);
wvdoc = ExampleWebView('myReport',model);
open(wvdoc);
fill(wvdoc);
close(wvdoc);
rptview(wvdoc);

```

Here is the report:

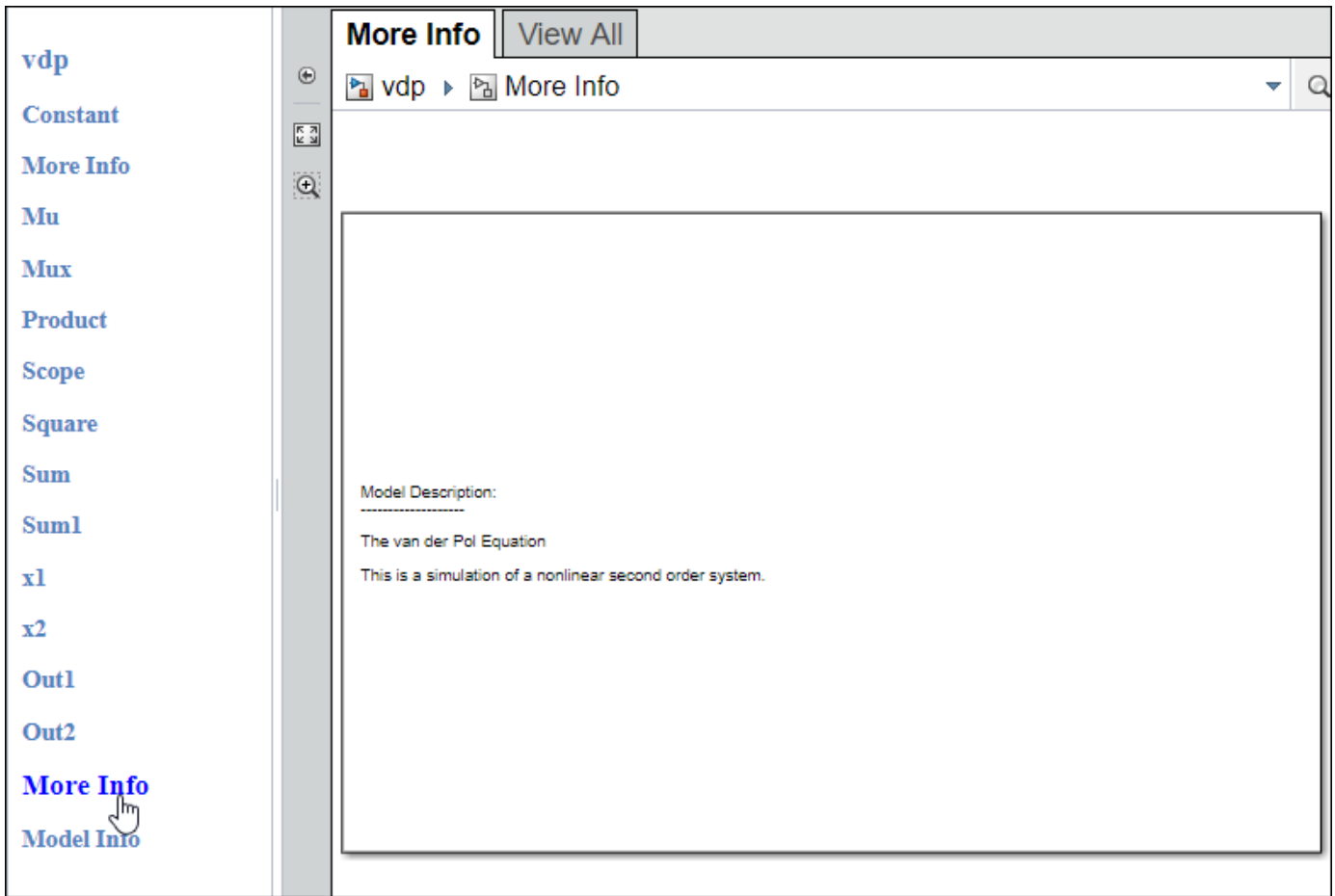
The screenshot displays the MATLAB/Simulink report viewer for the 'vdp' model. The interface is divided into three main sections:

- Left Pane (Document Pane):** A list of model components with hyperlinks for more information. The components listed are: Constant, Mu, Mux, Product, Scope, Square, Sum, Sum1, x1, x2, Out1, Out2, More Info, and Model Info.
- Center Pane (Diagram Pane):** A Simulink block diagram titled 'van der Pol Equation'. The diagram shows a feedback loop with a gain block 'Mu', a summing junction, and two integrators. The outputs are labeled 'Out1' and 'Out2'.
- Right Pane (Parameter Attributes):** A table of model parameters and their values.
 

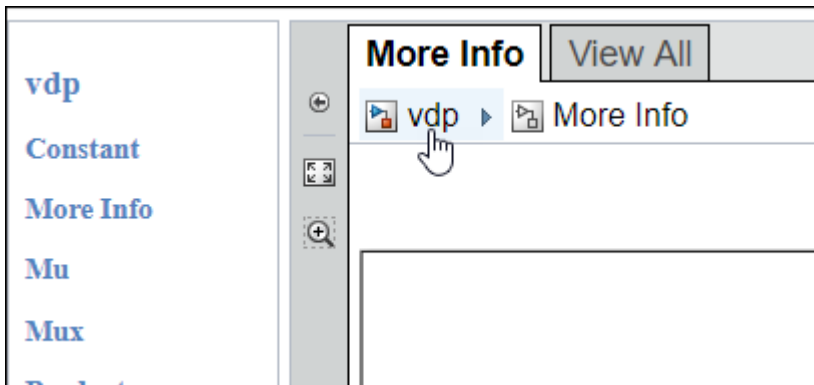
Parameter Attributes	
ModelVersion	2.0
LastModifiedD...	Wed May 20 00:26:14 2020
LibraryLinkDis...	none
ModelBrowser...	off
Dirty	off
Description	The van der Pol Equation  This is a simulation of a nonlinear second order system.

To use the links in the report:

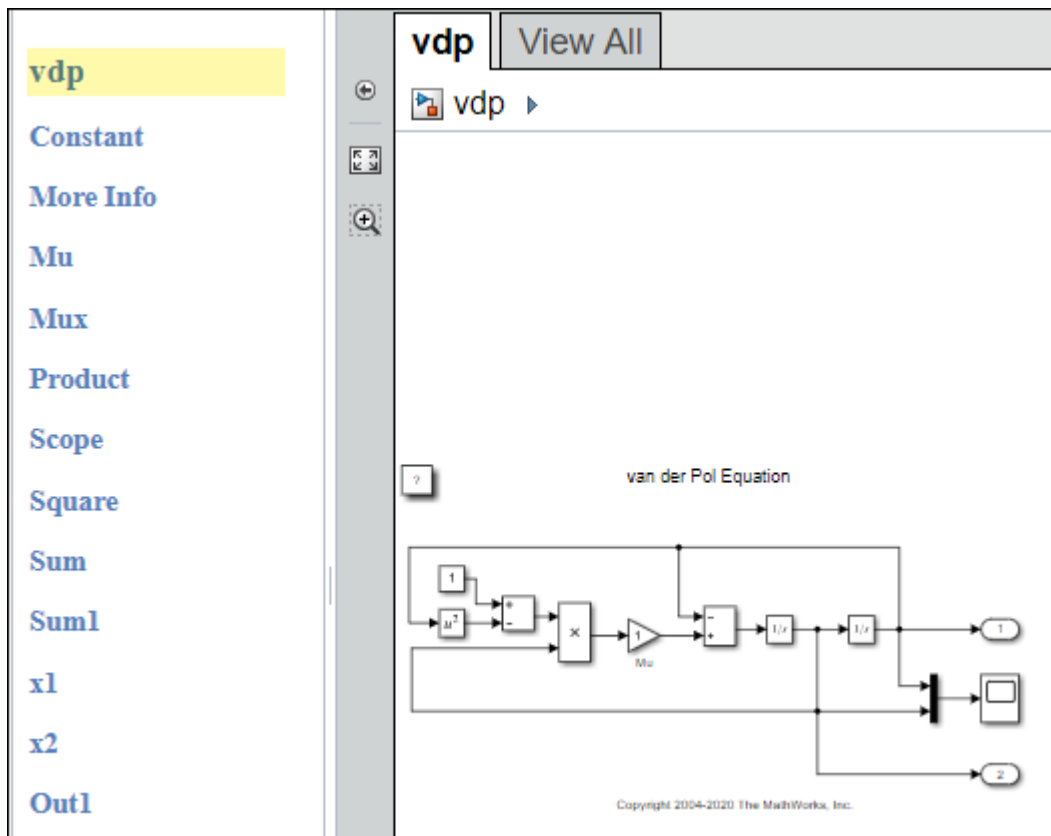
- 1 Click a diagram name in the document pane, for example, **More Info**. The associated diagram opens.



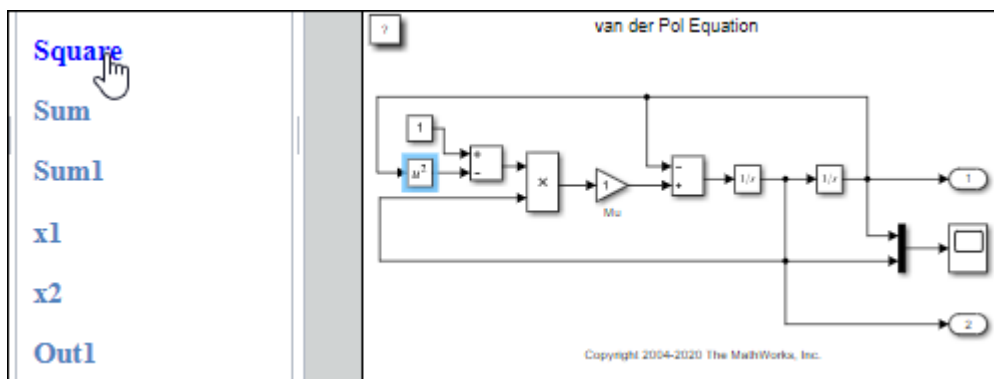
- 2 In the embedded web view, on the More Info tab, click vdp.



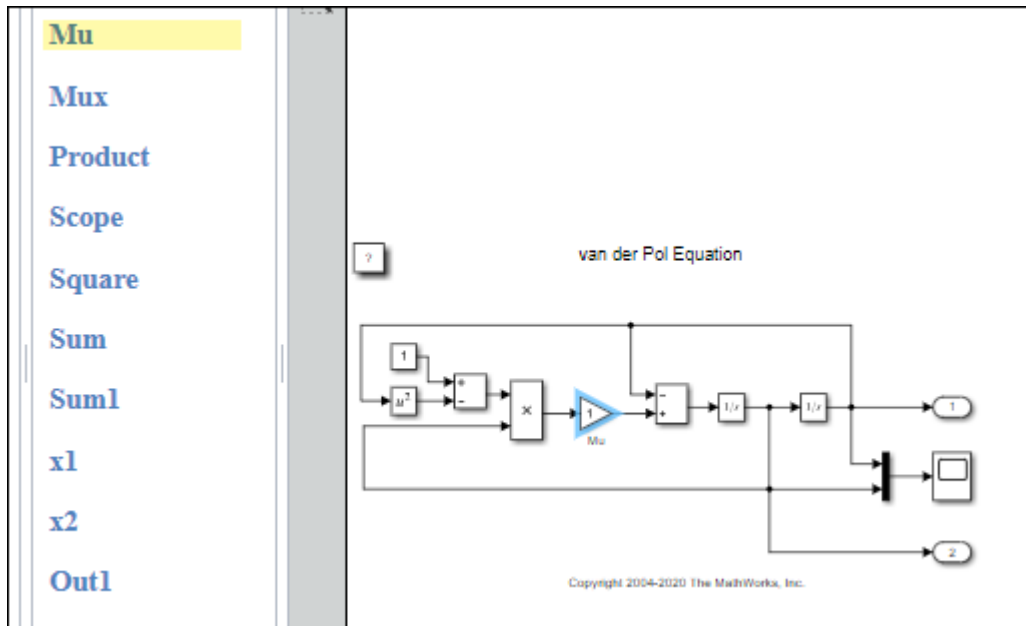
The vdp diagram opens and vdp is highlighted briefly in the document pane.



- 3 Click Square in the document pane, the Square block is highlighted in the embedded web view.



- 4 In the embedded web view, double-click the Mu block. The Mu link in the document pane is highlighted briefly.



For other tasks to create your embedded web view generator, see:

- “Specify Export Options for Embedded Web View Report” on page 5-24
- “Specify Document Content for Embedded Web View Report” on page 5-25
- “Generate Table of Contents for Embedded Web View Report” on page 5-26
- “Get Model Objects for Embedded Web View Report” on page 5-27
- “Suppress Link Warning Messages for Embedded Web View Report” on page 5-33

To generate an embedded web view report, see “Generate an Embedded Web View Report” on page 5-34.

## See Also

`slreportgen.webview.EmbeddedWebViewDocument`

## More About

- “Embedded Web View Reports” on page 5-19

## Suppress Link Warning Messages for Embedded Web View Report

By default, during report generation the `slreportgen.webview.EmbeddedWebViewDocument` base class of the Embedded Web View report generator displays warning messages for invalid links. These warnings appear if you include multiple links to an element in the Web View, although these multiple links are permitted. If you do not want to display these warning messages, you can suppress them. Set the `ValidateLinksAndAnchors` property, which is inherited from the base class, to `false` in the constructor of the report generator. For example,

```
function rptvar = SystemDesignVariables(reportPath, modelName)
    rptvar@slreportgen.webview.EmbeddedWebViewDocument(reportPath, ...
        modelName);
    rptvar.ValidateLinksAndAnchors = false;
end
```

For other tasks to create your Embedded Web View generator, see:

- “Specify Export Options for Embedded Web View Report” on page 5-24
- “Specify Document Content for Embedded Web View Report” on page 5-25
- “Generate Table of Contents for Embedded Web View Report” on page 5-26
- “Get Model Objects for Embedded Web View Report” on page 5-27
- “Create Hyperlinks for Embedded Web View Report” on page 5-28

To generate the Embedded Web View report, see “Generate an Embedded Web View Report” on page 5-34.

## Generate an Embedded Web View Report

To generate an embedded Web View report, create an instance of the class that defines the report generator (see “Create an Embedded Web View Report Generator” on page 5-23). Then, use the `fill` and `close` report generator methods.

For example, suppose that you want to create a report using the `SystemDesignVariables` class example (see “Class Definition File for an Embedded Web View” on page 5-34). These commands generate and display an instance of that report:

```
model = 'f14';
rptName = sprintf('%sVariables', model);
load_system(model);
rpt = SystemDesignVariables(rptName, model);
fill(rpt);
close(rpt);
close_system(model);
rptview(rptName);
```

The `fill(rpt)` command uses the `fill` method, which the report generator inherits from its base class. This method embeds a Web View of the `f14` model in the report. It also calls the `fillContent` method of the report generator, which fills the report document pane with a report about the variables used by the `f14` model.

Here is part of the resulting Embedded Web View report:

The screenshot shows a web browser window displaying an Embedded Web View report for model `f14`. The report is organized into sections:

- Table of Contents (Left):** Lists variables from 1.1 to 1.20, including `Cov`, `Ka`, `Kf`, `Ki`, `Kq`, `Md`, `Mq`, `Mw`, `Swg`, `Ta`, `Tal`, `Ts`, `Uo`, `Vto`, `W1`, `W2`, `Zd`, `Zw`, and `a`.
- Main Content Area (Center):**
  - 1.1. Cov**: `Cov. 1.00`. **Used By:** [f14/Dryden Wind Gust Models/Band-Limited White Noise/Output](#). **Resolved in:** mask workspace (f14/Dryden Wind Gust Models/Band-Limited White Noise).
  - 1.2. Ka**: **Ka. 0.68**. **Used By:** [f14/Controller/Gain3](#). **Resolved in:** base workspace.
- Parameter Table (Right):**

f14 Parameters	
Mo...	1.14
La...	Fri Jun 07 05:21
	2019
Lib...	none
Mo...	off
Dirty	off
De...	

For information on navigating to different parts of the report, see “Navigating an Embedded Web View Report” on page 5-20.

## Class Definition File for an Embedded Web View

This class generates a report for the workspace and data dictionary variables used by a specified Simulink model.

```

classdef SystemDesignVariables < slreportgen.webview.EmbeddedWebViewDocument
    %SystemDesignVariables Report on variables used by a Simulink model
    % Defines a class of report generators to produce HTML reports on
    % the workspace and data dictionary variables used by a Simulink
    % model. The generated report includes this information for
    % each variable:
    %
    % Value (if the value is a scalar, numeric value)
    % Data Type
    % Source (e.g, path of dictionary containing the variable)
    % Source Type (e.g., data dictionary or base workspace)
    % Users (path of blocks that use the variable)

    methods

        function rpt = SystemDesignVariables(reportPath, modelName)

            % Invoke the EmbeddedWebViewDocument constructor, which
            % saves the report path and model name for use by the
            % report's fill methods.
            rpt@slreportgen.webview.EmbeddedWebViewDocument(reportPath,...
                modelName);

            % Turn off duplicate link warnings to avoid warnings for
            % blocks that use multiple design variables.
            rpt.ValidateLinksAndAnchors = false;

            rpt.ExportOptions.IncludeMaskedSubsystems = true;
            rpt.ExportOptions.IncludeSimulinkLibraryLinks = true;
            rpt.ExportOptions.IncludeReferencedModels = true;
        end

        function fillContent(rpt)
            % Fill the Content hole in the report template with design
            % variable information. You can use DOM or Report API methods
            % to create, format, add, and append content to this report.

            %% Set up report
            % Allow use of unqualified names for DOM and Report objects,
            % such as Paragraph instead of mlreportgen.dom.Paragraph and
            % TitlePage instead of mlreportgen.report.TitlePage.
            import mlreportgen.dom.*
            import mlreportgen.report.*

            % Obtain model name, which was saved by the report
            % constructor. getExportedModels returns model names as a
            % cell array, in case a report uses multiple models.
            model = getExportModels(rpt);

            % Extract the model from the cell array. (This report uses
            % only one model.)
            model= model{1};

            % Add a title page to the report
            add(rpt, TitlePage("Title",[model " Report"],"Author",""));

            % Find variables used by the reported model
            finder = slreportgen.finder.ModelVariableFinder(model);

            % Create a Variables Chapter
            ch = Chapter("Variables");

            while hasNext(finder)
                result = next(finder);
                % Create a section for the variable
                s = Section(result.Name);

                % Add variable information to the section using
                % default reporter settings
                reporter = getReporter(result);
                add(s,reporter);
            end
        end
    end
end

```

```
        % Add this section to the chapter
        add(ch,s);
    end

    % Add the chapter to the report
    add(rpt,ch);
end
end
end
```

To generate two-way links between design variable user paths and blocks in the Web View that use the design variables, replace these lines of code:

```
% Add variable information to the section using
% default reporter settings
reporter = getReporter(result);
add(s,reporter);
```

with these lines of code:

```
% Create a Users list with links to the embedded model
usedByPara = Paragraph("Used By:");
usedByPara.Bold = true;
add(s, usedByPara);
users = result.Users;
nUsers = numel(users);
for u = 1:nUsers
    userLink = createElementTwoWayLink(rpt, ...
        users{u}, ...
        Paragraph(users{u}));
    add(s,userLink);
end
```



## Web View

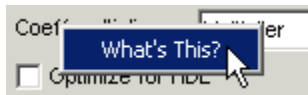
### Web View Export Dialog Box Overview

Use Web View dialog box to set export options, including:

- Systems to export (for example, the current system or current system and systems below)
- Whether to export information to support interacting with contents of referenced models, library links, or masked subsystems
- Where to store the web view files
- How to package the web view files
- Capture and optional view information

#### To get help on an option

- 1 Right-click the option's text label.
- 2 Select **What's This** from the popup menu.



#### See Also

“Export Models to Web View Files” on page 5-4

### Systems to Export

Select which Simulink systems or Stateflow charts to export.

**Note** The systems to export are relative to the system that is currently visible in the Simulink Editor or Stateflow Editor.

Include Systems Option Value	Meaning
<b>Entire Model</b> (Default)	Export all Simulink systems or Stateflow charts in the model
<b>Current and below</b>	Export the system or Stateflow chart that is displayed in the Simulink Editor or in the Stateflow Editor and all subsystems or subcharts that it contains
<b>Current and above</b>	Export the displayed Simulink system or Stateflow chart and all systems or charts that contain it
<b>Current</b>	Export only the Simulink system that is displayed in the Simulink Editor or in the Stateflow Editor

## Referenced Models

Export models referenced by the systems that you select to export. For example, if you select the **Entire Model** option, any models referenced by a Model block the exported model.

## Library Links

Export library blocks linked to from the systems that you select to export. For example, if you select the **Entire Model** option, all library blocks linked to in the exported model.

If you select this option, do not also select the **MathWorks Library Links** option.

## MathWorks Library Links

Export MathWorks built-in library blocks linked to systems that you select to export.

For example, if you select the **Entire Model** option, all MathWorks library blocks linked to in the exported model.

If you select this option, along with the **Masked Subsystems** option, right-clicking a built-in library block in a web view displays the parameter values for that block.

If you select this option, do not also select the **Library Links** option.

## Masked Subsystems

Allow web view interaction with masked subsystem contents in the systems that you select to export. Select this option to view the contents of the masked subsystem and open masked subsystem dialog boxes.

## Package name

The default name is the name of the model that contains the systems to export.

If you use the default **Package Type** setting of Zipped, the **Package name** field specifies the name of the zip file.

If you set **Package Type** to Unzipped, the **Package name** field specifies the name of the folder.

If the zip file or folder with the specified name in the specified folder already exists, Simulink Report Generator overwrites the existing zip file or folder contents. To avoid overwriting an existing Web View files, consider selecting the **If package exists, increment name to prevent overwriting** option.

## Folder

The folder in which to store the exported web view files. The default folder is the MATLAB current folder.

Specify a full path to the folder or click **Select Folder** to navigate to the folder.

## If package exists, increment name to prevent overwriting

Increment the package name by appending a system-generated number to the end of the zip file or folder name.

For example, suppose you use the following settings:

- For **Package Type**, you use the default setting of Zipped.
- For **Package name**, you use `vdp_web_view`.

When you export the Web view the first time, the resulting zip file is called `vdp_web_view`. If you export the Web view again, Simulink Report Generator creates a `vdp_web_view1.zip` file, and preserves the original `vdp_web_view` file.

## Package Type

Specify the file packaging to use when exporting the Web view.

- **Zipped** — (Default) Export as a zipped file that includes the Web view file, along with files that support the display of the Web view. Zip file packaging compresses the files and consolidates the web view and supporting files into one file.
- **Unzipped** — Export as the Web view file and supporting files, without being zipped. You can open an unzipped Web view file directly.
- **Both zipped and unzipped**— Export as both a zipped file and an unzipped file

## Include Model Coverage view

This option appears only if you have Simulink Coverage software installed.

Provides additional information about the model, based on the Model Coverage report.

## Include Embedded Coder view

This option only appears if both these conditions exist:

- You have the Embedded Coder® software installed.
- There is generated code for the systems that you export.

This view provides additional information about the generated code for a block in the web view.

Click a block in the Web view. The generated code for that block appears in the Web Informer pane below the displayed model.

## Include Requirements view

Captures model requirements information. For details, see “Include Model Requirements and Coverage Data in a Web View” on page 5-17.

This option appears only if you have Simulink Requirements software installed.

## **Include Coverage view**

Captures model coverage information based on the Model Coverage report. For details, see “Include Model Requirements and Coverage Data in a Web View” on page 5-17.

This option appears only if you have Simulink Coverage software installed. You must set up a coverage report for the model and simulate the model before you can use this option.

# Components

---

For a list of MATLAB Report Generator components, see the MATLAB Report Generator documentation.

## Annotation Loop

Run child components multiple times for each Simulink annotation in current context

### Description

This component runs its child components multiple times for each Simulink annotation in the current context. The parent component determines the context.

- **Model Loop:** Reports on all annotations inside the reported portion of the reported model.
- **System Loop:** Reports on all annotations inside the current system.
- **Block Loop** or **Signal Loop:** Does nothing.

### Loop Options

The Loop Options pane displays information about the current context. You can sort Alphabetically by text or In traversal order.

Child components of the Annotation Loop consider their context to be annotations when the report is running.

For example, the following components report on the looped annotation:

- **Simulink Automatic Table**
- **Simulink Linking Anchor**
- **Simulink Name**
- **Simulink Property**
- **Simulink Property Table**

Use a Summary Table component to show annotation objects in reports. Each Summary Table component creates a single table with each reported annotation on a single row of the table.

### Section Options

- **Create section for each object in loop:** Inserts a section in the generated report for each object found in the loop.
- **Display the object type in the section title:** Inserts the object type automatically into the section title in the generated report.
- **Create link anchor for each object in loop:** Create a link target for each annotation in the loop so that other parts of the report can link to it.

### See Also

Block Loop, Model Loop, Signal Loop, System Loop, Simulink Linking Anchor, Simulink Name, Simulink Property, Simulink Property Table, Simulink Summary Table

# Block Execution Order List

Create a list or table of all nonvirtual blocks in the model, showing order in which they execute

## Description

This component creates a list or table of all nonvirtual blocks in the model, showing the order in which they execute.

For more information about virtual and nonvirtual blocks, see “Nonvirtual and Virtual Blocks”.

## Properties

- **List Title:**
  - **Automatic:** Generates a list or table title automatically.
  - **Custom:** Enables you to enter a title.
- **Include block type information:** Include each block's BlockType property in the list or table.
- **Look under nonvirtual subsystems:** The default is Automatic (On for models, Off for systems). Set it to On or Off.

## Insert Anything into Report?

Yes. List.

## Class

rptgen\_sl.csl\_blk\_sort\_list

## See Also

Block Loop

## Block Loop

Run child components for each block in the current system, model, or signal

### Description

This component runs its child components for each block contained in the current system, model, or signal.

For conditional processing based of blocks, you can use the `RptgenSL.getReportedBlock` function. For more information, see “Loop Context Functions” on page 4-89.

### Report On

This pane describes the type of object on which this component operates.

- `Automatic list from context`: Report on all blocks in the current context. The parent component of the Block Loop determines its context. If this component does not have the **Model Loop**, **System Loop**, **Signal Loop**, or **Block Loop** as its parent, selecting this option causes this component to report on all blocks in all models.
  - **Model Loop**: Reports on all blocks in the current model.
  - **System Loop**: Reports on all blocks in the current system.
  - **Signal Loop**: Reports on all blocks connected to the current signal.
- `Custom - use block list`: Enables you to specify a list of blocks on which to report. Enter the full path of each block.

### Loop Options

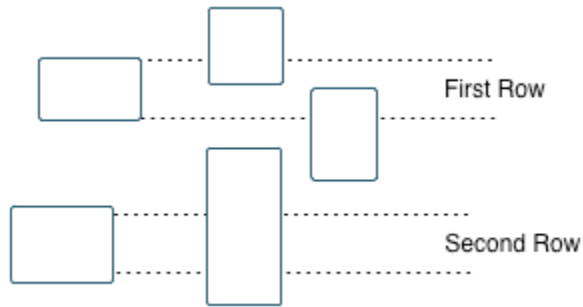
Choose block sorting options and reporting options in this pane.

- **Sort blocks:**

Use this option to select how to sort blocks (applied to each level in a model):

- `Alphabetically by block name`. Sorts blocks alphabetically by their names.
- `Alphabetically by system name`. Sorts systems alphabetically. The report lists blocks in each system, but in no particular order.
- `Alphabetically by full Simulink path`. Sorts blocks alphabetically by Simulink path.
- `By block type`. Sorts blocks alphabetically by block type.
- `By block depth`. Sorts blocks by their depth in the model.
- `By layout (left to right)`: Sorts blocks by their location in the model layout, by rows. The block appearing the furthest toward the left top corner of the model is the anchor for the row. The row contains all other blocks that overlap the horizontal area defined by the top and bottom edges of the anchor block. The other rows use the same algorithm, using as the anchor the next unreported block nearest the left top of the model.





- **By layout (top to bottom):** Sorts blocks by their location in the model layout, by *columns*. The block appearing the furthest toward the left top corner of the model is the anchor for the column. The column contains all other blocks that overlap the vertical area defined by the left and right edges of the anchor block. The other columns use the same algorithm, using as the anchor the next unreported block nearest the left top of the model.
- **By traversal order.** Sorts blocks by traversal order.
- **By simulation order.** Sorts blocks by execution order.
- **%<VariableName>:** Inserts the value of a variable from the MATLAB workspace. The %<> notation can denote a string or cell array. The following example reports on the theta dot integrator block and the theta integrator block in the model `simppend`, using the variable `Z={ 'simppend/theta' }`:

```
simppend/theta dot
%<Z>
```

The generated report includes information about the following blocks:

- `simppend/theta dot`
- `simppend/theta`

For more information, see %<VariableName> Notation on the Text component reference page in the MATLAB Report Generator documentation.

- **Search for Simulink property name/property value pairs:** Reports only on Simulink blocks with specified property name/property value pairs.

## Section Options

- **Create section for each object in loop:** Inserts a section in the generated report for each block found in the loop.
- **Display the object type in the section title:** Automatically inserts the object type into the section title in the generated report.
- **Create link anchor for each object in loop:** Create a link target for each block in the loop so that other parts of the report can link to it. For example, the image created by a **System Snapshot** component can link to the block information only if you select this check box.

## Insert Anything into Report?

Yes, inserts a section if you select the **Create section for each object in loop** option and a link target if you select **Create link anchor for each object in loop**.

## **Class**

rptgen\_sl.csl\_blk\_loop

## **See Also**

Model Loop, Signal Loop, System Loop, Simulink Linking Anchor, Simulink Name, Simulink Property, Simulink Property Table, Simulink Summary Table

# Block Type Count

Count number of each block type in the current model or system

## Description

This component counts the number of each block type in the current model or system. Within a model, this component counts blocks underneath masks and inside library links.

For more information about block types, see “Nonvirtual and Virtual Blocks”.

## Count Types

The parent of this component determines where to count block types:

- **Model Loop:** Reports all block types in the current model:
  - All blocks in model: Counts block types in the entire model.
  - All blocks in reported systems: Counts block types only in systems that appear in the report.
- **System Loop:** Reports all block types in the current system.

## Table Content

- **Table title:** Allows you to enter the table title.
- **Show block names in table:** Includes a column that displays all block names in the table.
- **Sort table:**
  - Alphabetically by block type: Sorts blocks alphabetically by block type.
  - By number of blocks: Sorts by decreasing number of occurrences.
- **Show total count:** Displays total number of block types.

## Insert Anything into Report?

Yes. Table.

## Class

rptgen\_sl.csl\_blk\_count

## See Also

Block Loop, Model Loop, System Loop

## Bus

Create list of signals exiting from Bus Selector block

### Description

This component creates a list of signals exiting a Bus Selector block. The list contains signals leaving from the reported block or downstream buses and signals.

The parent of this component determines which buses appear in the report:

- **Model Loop:** Includes all buses in the current model.
- **System Loop:** Includes all buses in the current system.
- **Block Loop:** If the current block is a bus block, then the report includes that block.
- **Signal Loop:** Includes all buses connected to the current signal.

If the Bus component does not have a looping component as its parent, it reports on all buses in all open models.

### Properties

- **Show Bus Hierarchy:** Specifies whether the list displays downstream buses hierarchically.
- **Insert linking anchor for bus blocks:** Inserts a linking anchor for each bus block. This property designates the list item as the location to which other links for that block point. (For more information, see the [Simulink Linking Anchor](#) or [Link](#) component reference pages.) Do not use this option if you have already specified an anchor location for the bus block with an [Object Linking Anchor](#) component.
- **Insert linking anchor for signals:** Inserts a linking anchor for each signal. This property designates the list item as the location to which other links for that signal point. For more information, see the [Simulink Linking Anchor](#) or [Link](#) component reference pages.) Do not use this option if you have already specified an anchor location for the signal with an [Object Linking Anchor](#) component.
- **Title:** Inserts a title before each list. This attribute supports the %<varname> notation. For more information, see %<VariableName> [Notation](#) on the [Text](#) component reference page in the MATLAB Report Generator documentation.

### Insert Anything into Report?

Yes. List.

### Class

rptgen\_sl.csl\_blk\_bus

### See Also

Block Loop, Model Loop, Signal Loop, Simulink Linking Anchor, System Loop,

# Chart Loop

Run child components for specified Stateflow charts

## Description

This component runs its children for specified Stateflow charts.

For conditional processing for a chart, you can use the `RptgenSF.getReportedChart` function. For more information, see “Loop Context Functions” on page 4-89.

## Report On

- `Automatic list from context`: Report on all chart blocks in the context set by the parent of this component.
  - **Model Loop**: Reports on all Stateflow chart blocks in the current model.
  - **System Loop**: Reports on all Stateflow chart blocks in the current system.
  - **Signal Loop**: Reports on all Stateflow chart blocks connected to the current signal.
  - **Machine Loop**: Reports on the current block if it is in a Stateflow chart.

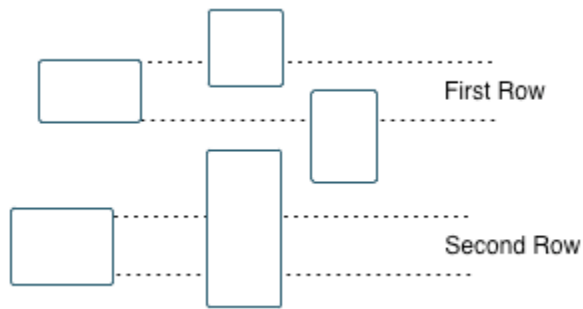
If the **Chart Loop** component has any other type of component as its parent, selecting this option causes it to report on all Stateflow chart blocks.

- `Custom - use block list`: Reports on a specified list of Stateflow chart blocks.

## Loop Options

Choose chart block sorting options and reporting options in this pane.

- **Sort blocks**: Specifies how to sort blocks (applied to each level in a model). This option is available if you select `Automatic list from context` in the **Report On** section, or if you select `Custom - use block list` and the **Sort blocks** option.
  - `Alphabetically by block name`. Sorts blocks alphabetically by name.
  - `Alphabetically by system name`. Sorts systems alphabetically by name. Lists blocks in each system, but in no particular order.
  - `Alphabetically by full Simulink path`. Sorts models alphabetically by their full paths.
  - `By block type`. Sorts blocks alphabetically by block type.
  - `By depth`. Sorts blocks by their depth in the model.
  - `By layout (left to right)`: Sorts blocks by their location in the model layout, by rows. The block appearing the furthest toward the left top corner of the model is the anchor for the row. The row contains all other blocks that overlap the horizontal area defined by the top and bottom edges of the anchor block. The other rows use the same algorithm, using as the anchor the next unreported block nearest the left top of the model.



- **By layout (top to bottom):** Sorts blocks by their location in the model layout, by *columns*. The block appearing the furthest toward the left top corner of the model is the anchor for the column. The column contains all other blocks that overlap the vertical area defined by the left and right edges of the anchor block. The other columns use the same algorithm, using as the anchor the next unreported block nearest the left top of the model.
- **By simulation order.** Sorts blocks by execution order.
- **%<VariableName>:** Inserts the value of a variable from the MATLAB workspace. The %<> notation can denote a string or cell array. For more information, see %<VariableName> **Notation** on the Text component reference page in the MATLAB Report Generator documentation.
- **Search for Simulink property name/property value pairs:** Reports on Simulink blocks with specified property name/property value pairs.
- **Search Stateflow:** Reports on Stateflow charts with specified property name/property value pairs.

## Section Options

- **Create section for each object in loop:** Inserts a section in the generated report for each object found in the loop.
- **Display the object type in the section title:** Inserts the object type automatically into the section title in the generated report.
- **Create link anchor for each object in loop:** Create a link target for each chart in the loop so that other parts of the report can link to it. For example, the image created by a **Stateflow Snapshot** component can link to this information only if you select this check box.

## Insert Anything into Report?

Yes, inserts a section if you select the **Create section for each object in loop** option.

## Class

rptgen\_sf.csf\_chart\_loop

## See Also

Block Loop, Machine Loop, Model Loop, Signal Loop, System Loop, Simulink Function System Loop

# Code Generation Summary

Insert version number information, list of generated files, tables summarizing code generation configuration information, and subsystem maps into report

## Description

This component reports the following information:

- Version number information
- List of generated files
- Code generation configuration information
- Subsystem map

## Summary

- **General information:** Includes the following information in the report:
  - Model name and version
  - Simulink Coder version number
  - List of full paths of generated files
- **Configuration settings:** Includes tables that list optimization and Simulink Coder target selection and build process Configuration Parameter settings.
- **Subsystem map:** Includes in the report a unique mapping between subsystem numbers and subsystem labels in the model.

## Traceability Report

**Use settings from model:** When you select this option, the report uses all of the following configuration settings, as specified in your model. Deselecting this option allows you to turn off one or more of these settings as needed:

- **Eliminated/virtual blocks**
- **Traceable blocks**
- **Traceable Stateflow Objects**
- **Traceable MATLAB Function Blocks**

For more information on these configuration settings, see “Model Configuration Parameters: Code Generation Report” (Simulink Coder).

## Insert Anything into Report?

Yes. Tables and list.

## Class

RptgenRTW.CCodeGenSummary

**See Also**

Import Generated Code



# Data Dictionary Traceability Table

Insert a table that links data dictionary information to requirements

## Description

This component inserts a table into the report. The table links data dictionary information to corresponding requirements. This component reports on the currently open data dictionary. Place this component inside a section, paragraph, or table component.

To use this component, your report setup must include Eval statements that open a data dictionary or determine the data dictionary that is open. To open a template report that shows an example of these Eval statements, at the MATLAB command prompt, enter:

```
setedit([matlabroot ' /toolbox/slrequirements/+rmide/rmide.rpt' ])
```

Find the Eval statements in the if condition at the beginning of the report setup.

## Table Options

Specify information about the table.

- **Table title:** Specify the table title.
  - `No title` — Do not include a table title.
  - `Object name` — Use the name of the data dictionary in the title.
  - `Custom` — Specify your own table title.

## Table Columns

Specify the table columns that you want to include in the report. The **Document name**, **Locations within document**, and **Requirement keyword** check boxes correspond to properties on the Requirements Management Interface Link Editor dialog box.

- **Description** — Include the description of the requirement. The description helps you to identify the requirement the table is linking to. Leave this box selected to improve the readability of your table.
- **Document name** — Include the name of the document where the requirement is located.
- **Locations within document** — Include the identifier of a location in the document.
- **Requirement keyword** — Include the requirement keyword.

## Insert Anything into Report?

Yes. Table.

## Class

RptgenRMI.DDRReqTable

**See Also**

MATLAB Code Traceability Table, Simulink Test Suite Traceability Table, rmi

# Documentation

Insert text extracted from DocBlock blocks in Simulink models

## Description

This component inserts text extracted from DocBlock blocks in Simulink models. It can have the following components as its parent:

- **Model Loop**
- **System Loop**
- **Block Loop**

The specified report format determines the format of the DocBlock block data inserted into the report:

- **HTML:** Imports HTML data into the report.

---

**Note** For non-English HTML DocBlock text that you want to include in a Documentation component, use UTF-8 file encoding. Use a simple text editor to create the HTML code.

---

- **RTF:** Imports RTF data into the report.

## Properties

- **Import file as:** Specifies how to format the imported information. The following example shows how each option works, using the following text as input:

```
First row.
  Second row.
```

```
Third row follows blank line.
```

- **Plain text (ignore line breaks).** Imports plain text without any line breaks (no paragraphs), as in this example:

```
First row. Second row. Third row follows blank line.
```

- **Paragraphs defined by line breaks.** Imports the text contained in paragraphs defined by line breaks (hard returns or carriage returns), as in this example:

```
First row.
Second row.
```

```
Third row follows blank line.
```

- **Paragraphs defined by empty rows.** Imports text contained in paragraphs defined by empty rows (rows that do not contain text), as in this example:

```
First row. Second row.
```

```
Third row follows blank line.
```

- **Text (retain line breaks).** Imports plain text, including line breaks, as in this example:

First row.  
Second row.

Third row follows blank line.

- **Fixed-width text (retain line breaks).** Imports fixed-width text (all letters have the same width or size) including line breaks, as in this example:

First row.  
Second row.

Third row follows blank line.

---

**Tip** This option is useful for importing MATLAB files.

---

- **Insert linking anchor for blocks:** Inserts a linking anchor for each DocBlock block that designates the location where other links for that block point. (See the [Simulink Linking Anchor](#) or [Link](#) component reference pages for more help.) Do not use this option if you have already specified an anchor location for a DocBlock block with an [Object Linking Anchor](#) component.

## Insert Anything into Report?

Yes. Text, paragraph, or external RTF/HTML data.

### Class

rptgen\_sl.csl\_blk\_doc

### See Also

Block Loop, Model Loop, Simulink Linking Anchor, System Loop

# Fixed Point Block Loop

Run child components for the Simulink model, system, or signal defined by parent component

## Description

This component runs its children for the Simulink model, system, or signal that its parent defines. Options for the parent component are:

- **Model Loop**
- **System Loop**
- **Signal Loop**

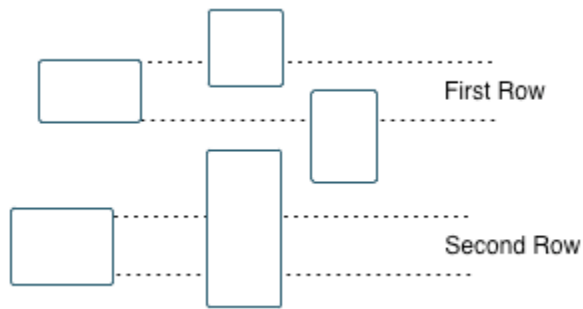
## Report On

- `Automatic list from context`: Reports on all fixed-point blocks in the context of the parent of this component. For example, if the parent component is the **System Loop**, then this component reports on all fixed-point blocks in the current system. If this component does not have a looping component as its parent, then selecting this option causes the component to report on all fixed-point blocks in all models.
- `Custom - use block list::` Reports on a specified list of blocks.

## Loop Options

Choose block sorting options and reporting options in this pane.

- **Sort blocks**: Specifies how to sort blocks (applied to each level in a model). This option is available if you select the `Automatic list from context` option in the **Report On** section, or if you select `Custom - use block list` and the **Sort blocks** options.
  - `Alphabetically by block name`. Sorts blocks alphabetically by name.
  - `Alphabetically by system name`. Sorts systems alphabetically. Lists blocks in each system, but in no particular order.
  - `Alphabetically by full Simulink path`. Sorts blocks alphabetically by Simulink path.
  - `By block type`. Sorts blocks alphabetically by block type.
  - `By block depth`. Sorts blocks by their depth in the model.
  - `By layout (left to right)`: Sorts blocks by their location in the model layout, by rows. The block appearing the furthest toward the left top corner of the model is the anchor for the row. The row contains all other blocks that overlap the horizontal area defined by the top and bottom edges of the anchor block. The other rows use the same algorithm, using as the anchor the next unreported block nearest the left top of the model.



- **By layout (top to bottom):** Sorts blocks by their location in the model layout, by *columns*. The block appearing the furthest toward the left top corner of the model is the anchor for the column. The column contains all other blocks that overlap the vertical area defined by the left and right edges of the anchor block. The other columns use the same algorithm, using as the anchor the next unreported block nearest the left top of the model.
- **By traversal order.** Sorts blocks by traversal order.
- **By simulation order.** Sorts blocks by execution order.
- **%<VariableName>:** Inserts the value of a variable from the MATLAB workspace. For more information, see **%<VariableName> Notation** on the Text component reference page in the MATLAB Report Generator documentation.
- **Search for Simulink name/property value pairs:** Reports only on blocks with the specified property name/property value pairs. To enable searching, click the check box. In the first row of the property name and property value table, click inside the edit box, delete the existing text, and type the property name and value. To add a row, click the **Add row** button.

For information about subsystem property names and values, in “Block-Specific Parameters”, see the “Ports & Subsystems Library Block Parameters” section.

## Section Options

- **Create section for each object in loop:** Inserts a section in the generated report for each object found in the loop.
- **Display the object type in the section title:** Inserts the object type automatically into the section title in the generated report.
- **Create link anchor for each object in loop:** Create a link target for each fixed-point block in the loop so that other parts of the report can link to it. For example, the image created by a **System Snapshot** component can link to this information only if you select this check box.

## Insert Anything into Report?

Yes, inserts a section if you select **Create section for each object in loop** and a link target if you select **Create link anchor for each object in loop**.

## Class

rptgen\_fp.cfp\_blk\_loop

## **See Also**

Block Loop, Model Loop, Signal Loop, Simulink Linking Anchor, System Loop

## Fixed Point Logging Options

Set fixed-point options like in Fixed Point Tool

### Description

This component sets fixed-point options like those set in the Fixed Point Tool (invoked by running the `fxptdlg` function).

This component must be a child of the `Model Loop` component. Use this component to set the following options on the current model:

- Data type override
- Fixed-point instrument mode
- Logging type

This component can have child components. It is a good practice to use this component with a `Model Simulation` component as its child. This approach sets fixed-point properties for the model for the purpose of the simulation, and then restores them to their original values after the simulation is complete.

### Data Type Override

- **Use local settings:** Overrides data types according to the value of this parameter set for each subsystem. Otherwise, settings for parent systems override those of child systems.
- **Scaled double:** Overrides the output data type of all blocks in the current system or subsystem with doubles. However, this option maintains the scaling and bias specified in the mask of each block.
- **Doubles:** Overrides the output data type of all blocks in the current system or subsystem with doubles. The overridden values have no scaling or bias.
- **Singles:** Overrides the output data type of all blocks in the current system or subsystem with singles. The overridden values have no scaling or bias.
- **Off:** Does not perform any data type override on any block in the current system or subsystem.

### Fixed-Point Instrumentation Mode

Specify logging options in this section. For logged blocks, minimum and maximum simulation values are written to the workspace.

- **Use local settings:** Logs data according to the value of this parameter set for each subsystem. Otherwise, settings for parent systems always override those of child systems.
- **Min, max, and overflow:** Logs minimum value, maximum value, and overflow data for all blocks in the current system or subsystem.
- **Overflow only:** Logs only overflow data for all blocks in the current system or subsystem.
- **Force off:** Logs no data for any block in the current system or subsystem. Use this selection to work with models containing fixed point-enabled blocks, if you do not have a Fixed-Point Designer license.



For more information on logging simulation results, see “Propose Fraction Lengths Using Simulation Range Data” (Fixed-Point Designer).

## Logging Type

Specify how to record logs in this section:

- **Overwrite log:** Clears information in the logs before new logging data is entered.
- **Merge log:** Merges new logging data with previously logged information.

## Insert Anything into Report?

No.

## Class

rptgen\_fp.cfp\_options

## See Also

Model Simulation

## Fixed Point Property Table

Insert table that reports on Fixed-Point Designer block property name/property value pairs

### Description

This component inserts a table that reports on Fixed-Point Designer block property name/property value pairs.

### Table

Select a preset table, which is already formatted and configured, in the **Preset table** list in the upper-left corner of the attributes page.

- **Preset table**

Specifies the type of object property table.

- Default
- Mask properties
- Block limits
- Out-of-range errors
- All fixed-point properties
- Blank 4x4

To apply the specified table, select the table and click **Apply**.

- **Split property/value cells:** Split property name/property value pairs into separate cells. For the property name and property value to appear in adjacent horizontal cells in the table, select the **Split property/value cells** check box. In this case, the table is in split mode, so there only one property name/property value pair can exist in a cell. If there is more than one name/property pair in a cell, only the first pair appears in the report. The report ignores all subsequent pairs.

For the property name and property value to appear together in one cell, clear the **Split property/value cells** check box. That option specifies nonsplit mode. Nonsplit mode supports more than one property name/property value pair and text.

Before switching from nonsplit mode to split mode, make sure that there is only one property name/property value pair per table cell. If you have more than one property name/property value pair or text in one cell, only the first value pair appears in the report. Subsequent pairs and text are omitted.

- **Display outer border:** Display the outer border of the table in the generated report.
- **Table spans page width:** Display the table across the entire page in the generated report.

### Table Cells

Select table properties to modify. The selection in this pane affects the available fields in the **Cell Properties** pane.

## Cell Properties

- **Contents**

Modify the contents of the table cell selected in the **Table Cells** pane.

- **Show as:** Specifies the format for the contents of the table cell.
  - PROPERTY Value
  - Value
  - Property Value
  - Property: Value
  - PROPERTY: Value
  - Property - Value
  - PROPERTY - Value
- **Alignment:** Aligns the contents of the table cell.
  - Center
  - Left
  - Right
  - Double justified
- **Lower border:** Displays the lower border of the table in the generated report.
- **Right border:** Displays the right border of the table in the generated report.

### Creating Custom Tables

To create a custom table, edit a preset table, such as the Blank 4x4 table. Add and delete rows and add properties. To open the Edit Table dialog box, click **Edit**.

For details about creating custom property tables, see “Property Table Components”.

## Insert Anything into Report?

Yes. Table.

### Class

rptgen\_fp.cfp\_prop\_table

### See Also

Fixed Point Summary Table

## Fixed Point Summary Table

Table of specified fixed-point block properties or parameters

### Description

This component displays properties or parameters of specified fixed-point blocks in a table.

### Properties

#### Table title

Choose a table title in the generated report:

- **Automatic**: Generates a title automatically from the parameter.
- **Custom**: Specifies a custom title.

### Property Columns

#### Property name

This field displays the object properties to include in the Summary Table in the generated report.

- To add a property:
  - 1 Select the appropriate property level in the menu
  - 2 Select the property to add from the selection list and click **Add**.
- To delete a property, select the property name and click the **Delete** button.
- To move properties up and down in the list, click the **Up** and **Down** buttons.

---

**Note** Some entries in the list of available properties (such as **Depth**) are “virtual” properties that you cannot access using the `get_param` command. The properties used for property/value filtering in the block and system loop components must be retrievable by the `get_param`. Therefore, you cannot configure your Summary Table to report on all blocks of `Depth == 2`.

---

#### Transpose table

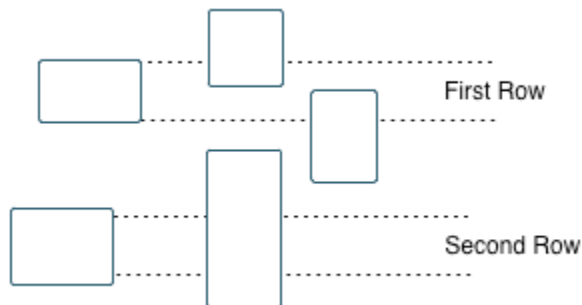
Enabling this check box changes the summary table rows into columns in the generated report, putting the property names in the first column and the values in the other columns.

### Object Rows

- **Insert anchor for each row**: Inserts an anchor for each row in the summary table.
- **Report On**: Specifies blocks on which to report:
  - **Automatic list from context**: Reports on all blocks in the current context.
  - **Custom - use block list**: Reports on a specified list of blocks. To include a given block in the report, specify its full path.

## Loop Options

- **Sort blocks:** Specifies how to sort blocks (applied to each level in a model):
  - **Alphabetically by block name.** Sorts blocks alphabetically by name.
  - **Alphabetically by system name.** Sorts systems alphabetically. Lists blocks in each system, but in no particular order.
  - **Alphabetically by full Simulink path.** Sorts blocks alphabetically by Simulink path.
  - **By block type.** Sorts blocks alphabetically by block type.
  - **By block depth.** Sorts blocks by their depth in the model.
  - **By layout (left to right):** Sorts blocks by their location in the model layout, by *rows*. The block appearing the furthest toward the left top corner of the model is the anchor for the row. The row contains all other blocks that overlap the horizontal area defined by the top and bottom edges of the anchor block. The other rows use the same algorithm, using as the anchor the next unreported block nearest the left top of the model.



- **By layout (top to bottom):** Sorts blocks by their location in the model layout, by *columns*. The block appearing the furthest toward the left top corner of the model is the anchor for the column. The column contains all other blocks that overlap the vertical area defined by the left and right edges of the anchor block. The other columns use the same algorithm, using as the anchor the next unreported block nearest the left top of the model.
- **By traversal order.** Sorts blocks by traversal order.
- **By simulation order.** Sorts blocks by execution order.
- **Search for Simulink property name/property value pairs:** Reports only on Simulink blocks with specified property name/property value pairs.

## Insert Anything into Report?

Yes. Table.

## Class

rptgen\_fp.cfp\_summ\_table

## See Also

Fixed Point Property Table

## Import Generated Code

Import source and header files generated by Simulink Coder software, and custom files specified as part of model

### Description

This component imports source and header files generated by Simulink Coder software. It also imports custom files that you specify as part of your model.

### Properties

- **Source files (auto-generated):** Includes the following files in the report:
  - .c and .cpp source files generated by Simulink Coder software.
  - Simulink Coder source files, such as the setup file and supporting files in the build folder.

This check box is selected by default. Clear it to omit source files.

- **Header files (auto-generated):** Includes the following files in the report:

- .h and .hpp header files generated by Simulink Coder software.
- Simulink Coder header files in the build folder.

This check box is selected by default. Clear it to omit source files.

- **Custom files:** Includes custom source files that you specify in the **Code Generation > Custom Code** pane of the Configuration Parameters dialog box. This check box is deselected by default.

### Insert Anything into Report?

Yes. Generated code listings.

### Class

RptgenRTW.CImportCode

### See Also

Code Generation Summary

# Look-Up Table

Report on lookup table blocks

## Description

The Look-Up Table component reports on the following blocks in the Simulink Lookup Tables library. Some examples of the lookup table blocks include:

- 1-D Lookup Table
- n-D Lookup Table
- Sine, Cosine
- Interpolation Using Prelookup
- Direct Lookup Table (n-D)

The Look-Up Table component inserts a figure and/or table into the report. The table contains input and output numeric values. A figure plots these values.

---

**Note** The Look-Up Table component does not display a table or plot for the Direct Lookup Table (n-D) block if the block is configured to generate the table during simulation as a block input. Instead, the Look-Up Table displays a note in the report to the effect that the table is generated dynamically during simulation.

---

## Look-Up Table Options

This pane allows you to specify the types of lookup table blocks to include in the report and how they appear. If you select none of the check boxes in this pane, the component does not insert anything into the report.

- The Look-Up Table displays results according to the type of its parent component:
  - **Model Loop:** Includes all lookup tables in the current model.
  - **System Loop:** Includes all lookup tables in the current system.
  - **Block Loop:** If the current block is a lookup table, the reports that block.
  - **Signal Loop:** Includes all lookup tables connected to the current signal.
  - If the Look-Up Table does not have any of the looping components as its parent, it includes all lookup tables in all open models.
- **Plot 1-D data:** Plots data from a 1-D Lookup Table block. Choose the plot type, `Line plot` or `Bar plot`, from the corresponding list. The input data appears on the horizontal or x-axis, and the output data appears on the vertical or y-axis.

For more information on line and bar plots, see “2-D and 3-D Plots”.

- **Create table for 1-D data:** Creates a table that contains numeric data values from the 1-D Lookup Table block.
- **Plot 2-D data:** Creates a plot of 2-D Lookup Table blocks. You can specify whether the data appears as a surface plot or a line plot. The line plot is best for small data sets, and the surface plot for larger tables. For more information on surface and line plots, see “2-D and 3-D Plots”.

---

**Note** This option creates a 2-D slice through n-D data.

---

- **Create table for 2-D data:** Creates a table that contains numeric data values from the 2-D Lookup Table block.
- **Create table for N-D data:** Creates a table that contains numeric data values from the n-D Lookup Table block.

## Print Options

- **Image file format:** Specifies the image file format. Select Automatic HG Format (the default) to choose automatically the format best suited for the output format that you chose in the Report component. Otherwise, choose an image format that your output viewer can read.
  - Automatic SL Format (Uses the Simulink file format selected in the Preferences dialog box)
  - Bitmap (16m-color)
  - Bitmap (256-color)
  - Black and white encapsulated PostScript
  - Black and white encapsulated PostScript (TIFF)
  - Black and white encapsulated PostScript2
  - Black and white encapsulated PostScript2 (TIFF)
  - Black and white PostScript
  - Black and white PostScript2
  - Color encapsulated PostScript
  - Color encapsulated PostScript (TIFF)
  - Color encapsulated PostScript2
  - Color encapsulated PostScript2 (TIFF)
  - Color PostScript
  - Color PostScript2
  - JPEG high quality image
  - JPEG medium quality image
  - JPEG low quality image
  - PNG 24-bit image
  - TIFF - compressed
  - TIFF - uncompressed
  - Windows metafile
- **Paper orientation:**
  - Landscape
  - Portrait
  - Rotated
  - Use figure orientation: Uses the orientation for the figure, which you set with the orient command.



- **Full page image (PDF only):** In PDF reports, scales images to fit the full page, minimizes page margins, and maximizes the size of the image by using either a portrait or landscape orientation.

For more information about paper orientation, see the `orient` command in the MATLAB documentation.

- **Image size:** Allows you to specify the image size in the report by selecting **Use figure PaperPositionMode setting** and setting the `PaperPositionMode` property of the Handle Graphics figure.
  - **Automatic (same size as on screen):**
  - **Custom:** Specifies a custom image size. Set the image size using the **Size** field and **Units** list.

For more information on paper position mode, see `orient` in the MATLAB documentation.

- **Size:** Allows you to enter the size of the Handle Graphics figure snapshot in the format `wxh` (width times height). This field is active only if you choose **Custom** in the **Image size** list box.
- **Units:** Allows you to enter for the size of the Handle Graphics figure snapshot. This field is active only if you choose **Custom** in the **Image size** list box.
- **Invert hardcopy:** Causes the Handle Graphics `InvertHardcopy` property to invert colors for printing. In other words, this option changes dark colors to light colors and light colors to dark colors. To change colors in your image, choose one of the following options:
  - **Automatic:** Automatically changes a dark axes colors to light axes colors. If the axes color is a light color, this option does not invert the color.
  - **Invert:** Changes dark axes colors to light axes colors, and light axes colors to dark axes colors.
  - **Don't invert:** Does not change the colors in the image that appears on the screen for printing.
  - **Use figure's InvertHardcopy setting:** Uses the `InvertHardcopy` property set in the Handle Graphics image.
  - **Make figure background transparent:** Makes the image background transparent.

## Display Options

- **Scaling:** Controls size of the image, as displayed in a browser. Making an image larger using this option does not affect the storage size of the image, but the quality of the displayed image may decrease as you increase or decrease the size of the displayed image.

Generally, to achieve the best and most predictable display results, use the default setting of `Use image size`.

- **Use image size:** Causes the image to appear the same size in the report as on screen (default).
- **Fixed size:** Specifies the number and type of units.
- **Zoom:** Specifies the percentage, maximum size, and units of measure.
- **Size:** Specifies the size of the snapshot in the form `[width, height]` format. This field is active only if you choose **Fixed size** in the **Scaling** selection list.
- **Max size:** Specifies the maximum size of the snapshot in the form `[width, height]`. This field is active only if you choose **Zoom** from the **Scaling** selection list.

- **Units:** Allows you to enter units for the size of the snapshot. This field is active only if you choose Zoom or Fixed size in the **Image size** list box.
- **Alignment:** Only reports in PDF or RTF format support this property.
  - Auto
  - Right
  - Left
  - Center
- **Title:** Enter text to appear above the snapshot.
- **Caption:** Enter text to appear under the snapshot.

### Insert Anything into Report?

Yes. Figure and/or table.

### Class

rptgen\_sl.csl\_blk\_lookup

### See Also

Block Loop, Model Loop, Signal Loop, System Loop

# Machine Loop

Run child components for specified Stateflow machines

## Description

This component runs its child components for selected Stateflow machines. The behavior of this component depends on its parent component. If it has no parent, the **Machine Loop** runs its child components for all machines. If it has the **Model Loop** is its parent, it runs its child components for all machines in the model.

## Loop Options

### Search Stateflow

If selected, searches states that you specify in the field that appears under the check box.

## Section Options

- **Create section for each object in loop:** Inserts a section in the generated report for each object in the loop.
- **Display the object type in the section title:** Inserts the object type automatically into the section title in the generated report.
- **Create link anchor for each object in loop:** Create a link target for each machine in the loop so that other parts of the report can link to it.

## Insert Anything into Report?

Yes, inserts a section if you select the **Create section for each object in loop** option.

## Class

rptgen\_sf.csf\_machine\_loop

## See Also

Model Loop

## Missing Requirements Block Loop

Apply all child components to blocks that do not have requirements

### Description

This component runs its child components for each block in the current system, model, or signal that do not have associated requirements.

For more information on working with looping components, see “Logical and Looping Components”.

### Report On

This pane describes the type of object on which this component operates.

- **Automatic list from context:** Report on all blocks in the current context that do not have associated requirements. The parent component of the Block Loop component determines its context. If this component does not have the **Model Loop**, **System Loop**, **Signal Loop**, or **Block Loop** as its parent, selecting this option causes this component to report on all blocks in all models that do not have associated requirements.
  - **Model Loop:** Reports on all blocks in the current model with no associated requirements.
  - **System Loop:** Reports on all blocks in the current system with no associated requirements.
  - **Signal Loop:** Reports on all blocks connected to the current signal with no associated requirements.
- **Custom - use block list:** Enables you to specify a list of blocks on which to report. Enter the full path of each block.

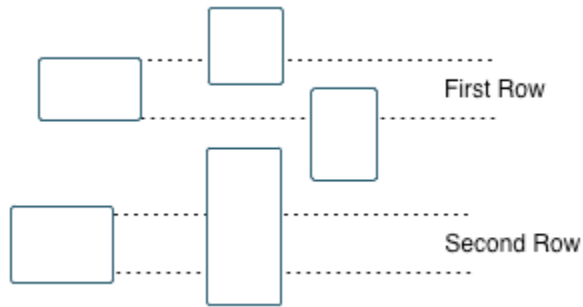
### Loop Options

Choose block sorting options and reporting options in this pane.

- **Sort blocks:**

Use this option to select how to sort blocks (applied to each level in a model):

- **Alphabetically by block name:** Sorts blocks alphabetically by their names.
- **Alphabetically by system name:** Sorts systems alphabetically. Lists the blocks in each system, but in no particular order.
- **Alphabetically by full Simulink path:** Sorts blocks alphabetically by Simulink path.
- **By block type:** Sorts blocks alphabetically by block type.
- **By block depth:** Sorts blocks by their depth in the model.
- **By layout (left to right):** Sorts blocks by their location in the model layout, by rows. The block appearing the furthest toward the left top corner of the model is the anchor for the row. The row contains all other blocks that overlap the horizontal area defined by the top and bottom edges of the anchor block. The other rows use the same algorithm, using as the anchor the next unreported block nearest the left top of the model.



- **By layout (top to bottom):** Sorts blocks by their location in the model layout, by *columns*. The block appearing the furthest toward the left top corner of the model is the anchor for the column. The column contains all other blocks that overlap the vertical area defined by the left and right edges of the anchor block. The other columns use the same algorithm, using as the anchor the next unreported block nearest the left top of the model.
- **By traversal order.** Sorts blocks by traversal order.
- **By simulation order.** Sorts blocks by execution order.
- **%<VariableName>:** Inserts the value of a variable from the MATLAB workspace. The %<> notation can denote a string or cell array. The following example reports on the theta dot integrator block and the theta integrator block in the model `simppend`, using the variable `Z={ 'simppend/theta' }`:

```
simppend/theta dot
%<Z>
```

The generated report includes information about the following blocks:

- `simppend/theta dot`
- `simppend/theta`

For more information, see %<VariableName> Notation on the Text component reference page in the MATLAB Report Generator documentation.

- **Search for Simulink property name/property value pairs:** Reports only on Simulink blocks with specified property name/property value pairs that do not have associated requirements.

## Section Options

- **Create section for each object in loop:** Inserts a section in the generated report for each block found in the loop.
- **Display the object type in the section title:** Automatically inserts the object type into the section title in the generated report.
- **Create link anchor for each object in loop:** Create a link target for each missing requirement in the loop so that other parts of the report can link to it.

## Insert Anything into Report?

Yes, inserts a section if you select **Create section for each object in loop** and a link target if you select **Create link anchor for each object in loop**.

**Class Name**

RptgenRMI.NoReqBlockLoop

**See Also**

Block Loop, Missing Requirements System Loop, Requirements Block Loop, Requirements Documents Table, Requirements Signal Loop, Requirements Summary Table, Requirements System Loop, Requirements Table

# MATLAB Code Traceability Table

Insert a table that links MATLAB code to requirements

## Description

This component inserts a table into the report. The table links MATLAB code to corresponding requirements. This component reports on the currently open .m file. Place this component inside a section, paragraph, or table component.

To use this component, your report setup must include Eval statements that open a .m file or determine the .m file that is open. To open a template report that shows an example of these Eval statements, at the MATLAB command prompt, enter:

```
setedit([matlabroot ' /toolbox/slrequirements/+rmiml/rmiml.rpt' ])
```

Find the Eval statements in the if condition at the beginning of the report setup.

## Table Options

Specify information about the table.

- **Table title:** Specify the table title.
  - `No title` — Do not include a table title.
  - `Object name` — Use the name of the .m file in the title.
  - `Custom` — Specify your own table title.

## Table Columns

Specify the table columns that you want to include in the report. The **Document name**, **Locations within document**, and **Requirement keyword** check boxes correspond to properties on the Requirements Management Interface Link Editor dialog box.

- **Description** — Include the description of the requirement. The description helps you to identify the requirement the table is linking to. Leave this box selected to improve the readability of your table.
- **Document name** — Include the name of the document where the requirement is located.
- **Locations within document** — Include the identifier of a location in the document.
- **Requirement keyword** — Include the requirement keyword.

## Insert Anything into Report?

Yes. Table.

## Class

RptgenRMI.MlReqTable

**See Also**

Data Dictionary Traceability Table, Simulink Test Suite Traceability Table, rmi



# MATLAB Function

Insert information about MATLAB Function block contents

## Description

This component displays tables with information about MATLAB code included in MATLAB Function blocks. You specify which of the following kinds of information to include in the report:

- Function properties — Parameter settings for the MATLAB Function block
- Argument properties — Properties of the function arguments (for example, complexity)
- The function script — MATLAB code of the function
- Function symbol data — Information about the user-defined and (optionally) built-in MATLAB variables and functions invoked by the MATLAB function that computes the block outputs.
- Supporting functions — User-defined functions and, optionally, MATLAB functions that are included in the MATLAB Function block function.

For details about MATLAB Function blocks, see the MATLAB Function block reference page.

Use the MATLAB Function component within a section, paragraph, or table.

---

**Note** To view the contents of a MATLAB Function block in a Web viewer, use the Web view feature of the Simulink Report Generator. In the Web view, hover your cursor over the MATLAB Function block. For details, see “Create Model Web Views”.

---

## Function Properties Table

- **Include function properties:** Generates a table with function property information.
- **Table title:** Insert a title for the function properties table.
  - **Automatic:** Use the default title for the table.
  - **Custom:** Use the title that you specify for the table.
- You can change the header text for property and value columns of the function properties table. In the **Header** column, double-click to change the header text. The **Width** column indicates the relative width, in relative terms, based on the smallest width you specify. For example, for a three-column table, if the first column width is 1, and the column width of the other two columns is 3, then the second and third columns is three times wider than the first column.
- **Grid lines:** Show grid lines for the table.
- **Spans page width:** Make the table as wide as the page.

## Argument Summary Table

- **Include argument summary table:** Generate a table with summary information about the MATLAB Function block function arguments.
- **Table title:** Insert a title for the argument summary table.

- **Automatic:** Use the default title for the table.
- **Custom:** Use the title that you specify for the table.
- **Argument Summary Table Options:** Specify the property columns to include in the table.
  - To add a property column:
    - 1 In the table on the right, select a property near where you want to insert the new property column.
    - 2 From the list of properties to the left of the table, select a property that you want to add to the table.
    - 3 Click the left-arrow button.
    - 4 If necessary, use the up or down arrow button to position the new column.
  - To delete a property column, select the property in the table and click the right-arrow button
  - You can change the header text for property and value columns of the table. In the **Header** column, double-click to change the header text. The **Width** column indicates the relative width, in relative terms, based on the smallest width you specify. For example, for a three-column table, if the first column width is 1, and the column width of the other two columns is 3, then the second and third columns is three times wider than the first column.
- **Grid lines:** Show grid lines for the table.
- **Spans page width:** Make the table as wide as the page.
- **Column alignment:** Align the text in each column:
  - Left
  - Center
  - Right
  - Double justified

## Detailed Argument Report

- **Include detailed argument report:** Generate a table with detailed information about the MATLAB Function block function arguments.
- **Argument Property Table Format Options:** Specify the argument property columns to include in the table.
  - **Table title:** Insert a title for the argument properties table.
    - **Automatic:** Use the default title for the table.
    - **Custom:** Use the title that you specify for the table.
  - You can change the header text for property and value columns of the table. In the Header column, double-click to change the header text.
  - **Grid lines:** Show grid lines for the table.
  - **Spans page width:** Make the variable table as wide as the page on which the table appears.
- **Include function script:** Include the script for the function.
- **Include function symbol data:** Generate a table that includes information about the user-defined and (optionally) built-in MATLAB variables and functions invoked by the MATLAB function that computes the block outputs.

- **Highlight script syntax:** Use colors to highlight syntax keywords.
- **Include supporting functions:** Include a list of functions invoked directly or indirectly by the function script. If you specify to include supporting functions in the report, also specify whether to include both MATLAB and user-defined functions or just user-defined functions.
- **Supporting Function Table Format Options:**
  - **Table title:** Insert a title for the supporting functions table.
    - **Automatic:** Use the default title for the table.
    - **Custom:** Use the title that you specify for the table.
  - You can change the header text for property and value columns of the table. In the **Header** column, double-click to change the header text. The **Width** column indicates the relative width, in relative terms, based on the smallest width you specify. For example, for a three-column table, if the first column width is 1, and the column width of the other two columns is 3, then the second and third columns is three times wider than the first column.
  - **Grid lines:** Show grid lines for the table.
  - **Spans page width:** Make the table as wide as the page.

## Insert Anything into Report?

Yes. Tables and, optionally, code.

### Class

rptgen\_sl.csl\_emlfcn

### See Also

Stateflow Property

## Missing Requirements System Loop

Loop only on systems and subsystems that do not have associated requirements

### Description

This component runs its child components for each system or subsystem defined by the parent component that does not have associated requirements. Insert this component as the child of a **Model Loop** component to include systems and subsystems that do not have any associated requirements in the report.

### Report On

- **Loop on Systems:**
  - **Select systems automatically:** Reports on all systems in the current context that do not have associated requirements.
    - **Model Loop:** Reports on systems in the current model.
    - **System Loop:** Reports on the current system.
    - **Signal Loop:** Reports on the parent system of the current signal.
    - **Block Loop:** Reports on the parent system of the current block.

If this component does not have any of these components as its parent, selecting this option reports on all systems in all models that do not have associated requirements.
- **Custom - use system list:** Reports on a list of specified systems. Specify the full path of each system.
- **%<VariableName>:** Inserts the value of a variable from the MATLAB workspace. The %<> notation can denote a string or cell array. For more information, see %<VariableName> Notation on the Text component reference page.

### Loop Options

- **Sort Systems:** Specifies how to sort systems.
  - **Alphabetically by system name** (default): Sorts systems alphabetically by name.
  - **By number of blocks in system:** Sorts systems by number of blocks. The list shows systems by decreasing number of blocks. In other words, it shows the system with the largest number of blocks that do not have requirements appears first in the list.
  - **By system depth:** Sorts systems by their depth in the model.
  - **By traversal order:** Sorts systems in the traversal order.
- **Search for:** Reports only on blocks with the specified property name/property value pairs. To enable searching, click the check box. In the first row of the property name and property value table, click inside the edit box, delete the existing text, and type the property name and value. To add a row, use the **Add row** button.

For information about subsystem property names and values, in “Block-Specific Parameters”, see the “Ports & Subsystems Library Block Parameters” section.

## Section Options

- **Create section for each object in loop:** Inserts a section in the generated report for each object found in the loop.
- **Display the object type in the section title:** Inserts the object type automatically into the section title in the generated report.
- **Number sections by system hierarchy:** Hierarchically numbers sections in the generated report. Requires that **Sort Systems** be set to **By traversal order**.
- **Create link anchor for each object in loop:** Create a link target for each missing requirement in the loop so that other parts of the report can link to it.

## Insert Anything into Report?

Yes, inserts a section if you select **Create section for each object in loop** and a link target if you select **Create link anchor for each object in loop**.

## Class

RptgenRMI.NoReqSystemLoop

## See Also

Block Loop, Missing Requirements Block Loop, Requirements Block Loop, Requirements Documents Table, Requirements Signal Loop, Requirements Summary Table, Requirements System Loop, Requirements Table, System Loop

## Model Advisor Report

Insert Model Advisor report or link to Model Advisor report for current model

### Description

This component inserts a Model Advisor report for the current model into the report if the report is in HTML format. For other report formats, it inserts a link to a Model Advisor report for the current model. For more information about Model Advisor reports, see “Save and View Model Advisor Check Reports”.

### Properties

**Use existing report:** Includes an existing Model Advisor report in the report. This check box is selected by default. Clearing this option generates a new Model Advisor report.

### Insert Anything into Report?

Yes, a Model Advisor report.

### Class

rptgen\_sl.CModelAdvisor

### See Also

Model Change Log

# Model Change Log

Construct model history table that displays model revision information

## Description

Run this component before you run the **Model Simulation** component. It constructs a model history table that displays information about each logged revision to the model. This model history table includes:

- The author of each change
- The model version of the change
- The time and date of the change
- A description of the change

See “Model Information” for more information..

---

**Tip** If your model has a long revision history, consider limiting the number of revisions reported.

---

## Table Columns

Choose the information displayed in the model revision table in this section:

- **Author name:** Includes the name of the person who last revised the model.
- **Version:** Includes the version number of the model.
- **Date changed:** Includes the revision date of the model.
- **Description of change:** Includes a description of the revision to the model.

## Table Rows

- **Limit displayed revisions to:** Limits the number of revisions that appears in the report.
- **Show revisions since date:** Limits the number of revisions that appears in the report by date. Enter the date in the corresponding text field. This field supports %<varname> notation. For example, the default value, %<datestr(now-14)>, returns revision history for the last two weeks.

## Table Display

Choose how the model revision history table appears in this section.

- **Table title:** Specifies the title of the table.
- **Sort order:** Sorts the table entries from most recent to oldest, or from oldest to most recent.
- **Date format:** Specifies a preferred date format for the date/time stamps in the table.

## **Insert Anything into Report?**

Yes. Table.

### **Class**

rptgen\_sl.csl\_md1\_changelog

### **See Also**

Model Advisor Report



# Model Configuration Set

Insert active configuration set of a model into a report

## Description

This component displays a table with the active configuration set for the model.

For information about configurations sets, see “Manage Configuration Sets for a Model”.

## Display Options

- **Title:** Specifies a title for the table in the generated report.
  - **Automatic:** Generates a title automatically from the parameter.
  - **Custom:** Specifies a custom title.
  - **None:** Uses no title.
- **Show configuration set table grids:** Show grid lines for the table.
- **Make configuration set tables page wide:** Make the table as wide as the page.

## Insert Anything into Report?

Yes. Table.

## Class

rptgen\_sl.csl\_md1\_cfgset

## See Also

Model Loop, System Loop

## Model Loop

Loop on Simulink models and systems, as specified by child components

### Description





This component loops on Simulink models and systems, as specified by child components. For example, you can use a **Model Loop** with a child **System Loop** to report on the subsystems of the specified system.

Consider making these components children of the **Model Loop** (although the **Model Loop** is not necessarily required to be the immediate parent of a given component).

For conditional processing based of blocks, you can use the `RptgenSL.getReportedBlock` function. For more information, see “Loop Context Functions” on page 4-89.

### Models to Include

You can add a model to the list by clicking **Add New Model to List**. The following table shows the buttons you can use to move a model up or down in the list, or to add or delete a model.

Button	Action
	Move a model up in the list.
	Move a model down in the list.
	Remove a model from the list.
	Add a new model to the list.

### Model Options

- **Active:** Includes a given model in the loop. This option is selected by default. Clearing this option omits the model from the loop.

This option allows you to temporarily omit one or more models from a report.

- **Model name:** Specifies the model name.
  - Current block diagram
  - All open models
  - All open libraries
  - Block diagrams in current directory
  - Custom block diagram: Selecting this option automatically sets the **Starting system(s)** field \$top to start in the model root system.
  - `%<VariableName>`: For more information, see `%<VariableName>` Notation on the Text component reference page in the MATLAB Report Generator documentation.

- **Traverse model:** Specifies the systems to traverse.
  - All systems in model
  - Selected system(s) only
  - Selected system(s) and ancestors
  - Selected system(s) and children
- **Look under masks:** Specifies how to handle masks.
  - Functional masks only
  - No masks
  - All masks
  - Graphical masks only

For more information, see “Create Block Masks”.

- **Follow library links:** Specifies library links to include.
  - Do not follow library links: Library links are treated as blocks.
  - Include library links: Library links are treated as subsystems.
  - Include unique library links: With multiple copies of the same library link in a system, one is treated as a subsystem and the others as blocks.

For more information, see “Linked Blocks”.

- **Model reference:** Specifies whether to report on models referenced by a Model block. If you want to report on referenced models, then you can control the depth of the model hierarchy and whether to report on variant models.
  - Do not follow Model blocks: Do not report on blocks contained in referenced models.
  - Follow all Model blocks: Report on blocks contained in all models that any part of the model hierarchy references.
  - Follow Model blocks defined in current model: Report on blocks in models that the currently selected model references.
  - <Custom model reference depth>: Report on blocks in models that your specified level in the model hierarchy references.
- **Include all variants:** Report on all variant models. To enable this option, set the **Model reference** option to report on blocks in referenced models.
- **Starting system(s):** Specifies the system in which to start the loop. Available options depend on the value that you select in the **Traverse model** option. Selecting any option other than All systems in model for **Traverse model** activates the **Starting system(s)** option.

If you do not enter a model name in the **Model name** option, then select either Root model or Current to specify where to start the loop.

If you specify a model name in the **Model name** option, then the **Starting system(s)** option provides an edit box in which you can enter:

- The full path of a subsystem or subsystems
- \$top to start the loop in the model root system
- \$current to start the loop in the currently selected system

## Section Options

- **Create section for each object in loop:** Inserts a section in the generated report for each object found in the loop.
- **Display the object type in the section title:** Inserts the object type automatically into the section title in the generated report.
- **Create link anchor for each object in loop:** Create a link target for each model in the loop so that other parts of the report can link to it.

## Examples

### Example 6.1. Generating Reports on Specified Systems and Their Subsystems

This example shows how to loop over a specified system and its subsystems in the sample model `sldemo_auto_climate_elec`, which the Simulink software includes.

- 1 (Optional) To open the `sldemo_auto_climate_elec` model, at the MATLAB command prompt, enter the following command:
 

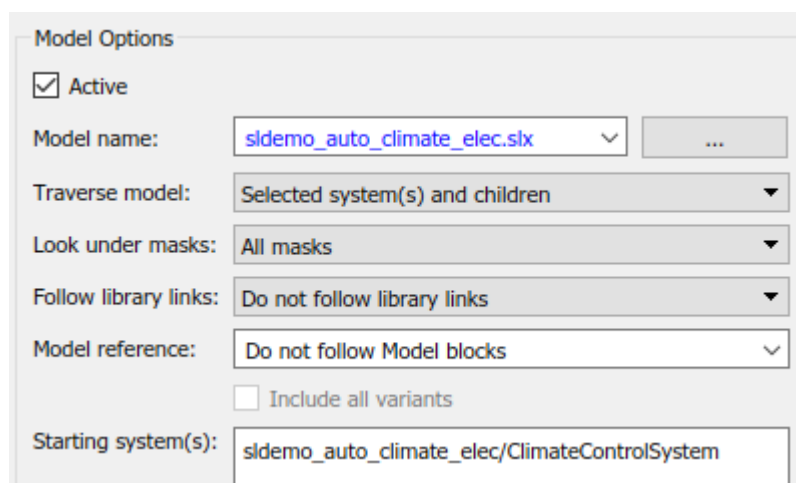
```
sldemo_auto_climate_elec
```

Explore the model to familiarize yourself with its subsystems.
- 2 Open the Report Explorer.
- 3 Create a report setup file by clicking **File > New**.
- 4 Save the report setup file by clicking **File > Save As**. Give it the name `sldemo_auto_report`.
- 5 Add a Chapter/Subsection component to the report setup file to include information about model subsystems:
  - a In the Library pane in the middle, double-click **Chapter/Subsection** to add it to the report setup file.
  - b For **Title**, choose Custom. In the title field, enter `Description of subsystems`.
  - c Add a Model Loop as a child of the Chapter/Subsection component. This loops over the ClimateControlSystem system and its subsystems in the `sldemo_auto_climate_elec` model:
    - i In the Library pane in the middle, double-click **Model Loop** to add it to the report setup file. By default, the Report Explorer adds that component as a child of the Chapter/Subsection component.
    - ii In the Model Loop properties pane, from the **Model name** selection list, select `<Custom block diagram>`.
    - iii In the **Model name** field, delete the text `<Custom block diagram>`, and then enter `sldemo_auto_climate_elec.slx`. Click any component in the report setup file to add this model to the **Models to include** list.
    - iv In the **Traverse model** selection list, select `Selected system(s) and children`.
    - v In the **Look under masks** selection list, select `All masks`.
    - vi In the **Model reference** selection list, select `Do not follow Model blocks`.
    - vii In the **Starting system(s)** field, enter `sldemo_auto_climate_elec/ClimateControlSystem`. Because you selected `Selected system(s) and`

children for **Traverse model**, the Model Loop loops over `sldemo_auto_climate_elec/ClimateControlSystem` and its subsystems.

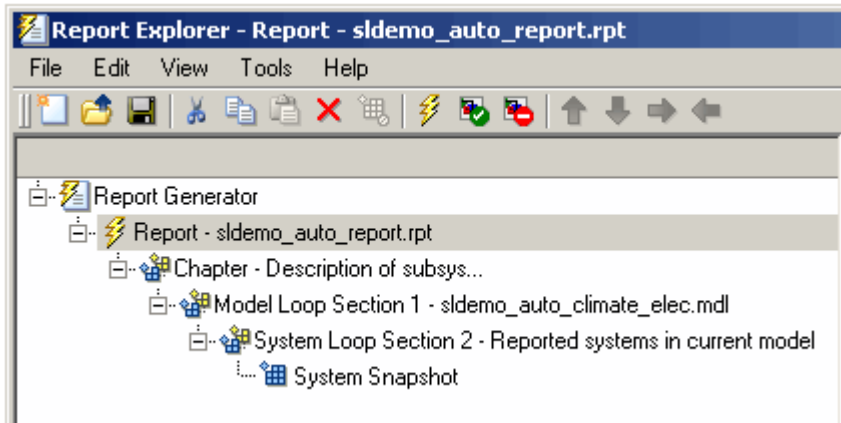
- viii Under **Section Options**, select the **Create section for each object in loop** check box. Selecting this option creates separate sections in the generated report for each model over which the component loops.

The **Model Loop** properties pane looks as follows.



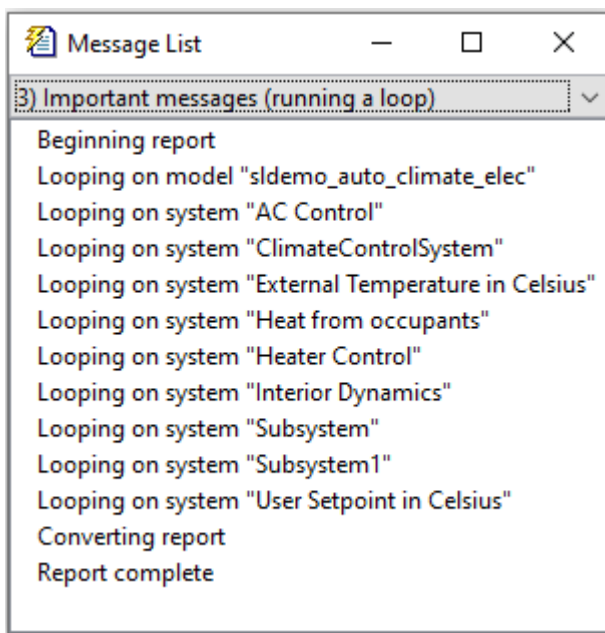
- 6 Save the report by clicking **File > Save**.
- 7 Add a **System Loop** as a child of the **Model Loop** component.
  - a In the Library pane in the middle, double-click **System Loop** to add it to the report setup file. By default, Model Explorer adds this component as a child of the **Model Loop** component.
  - b In the **System Loop** properties pane, under **Loop Options**, select the **Create section for each object in loop** check box. Selecting this option creates a section in the generated report for each subsystem on which the component loops. Accept the default values for all other fields.
- 8 Add a **System Snapshot** component as a child of the **System Loop** component. This step creates snapshots of all the subsystems of `ClimateControlSystem` in the generated report. In the Library pane in the middle, double-click **System Snapshot**. By default, Model Explorer adds this component as a child of the **System Loop** component.
- 9 Save the report.

The report setup file hierarchy now looks as follows.



- 10** Run the report by clicking **File > Report**.

The report loops on the system `ClimateControlSystem` of the `sldemo_auto_climate_elec` model and all of its subsystems, as shown in the following Message List.



Below is an excerpt from the generated report.

## Chapter 1. Description of subsystems

### Table of Contents

[sldemo\\_auto\\_climate\\_elec](#)

[AC Control](#)

[ClimateControlSystem](#)

[External Temperature in Celsius](#)

[Heat from occupants](#)

[Heater Control](#)

[Interior Dynamics](#)

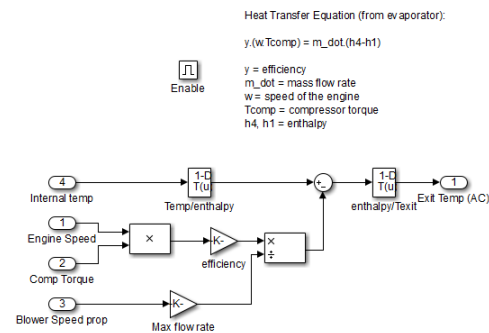
[Subsystem](#)

[Subsystem1](#)

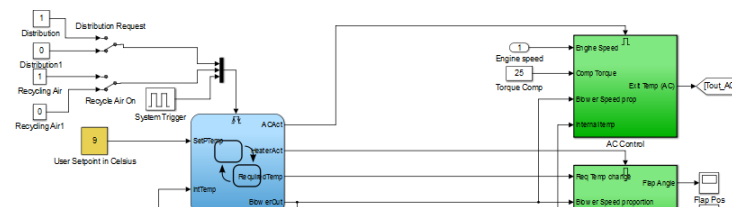
[User Setpoint in Celsius](#)

### sldemo\_auto\_climate\_elec

#### AC Control




#### ClimateControlSystem

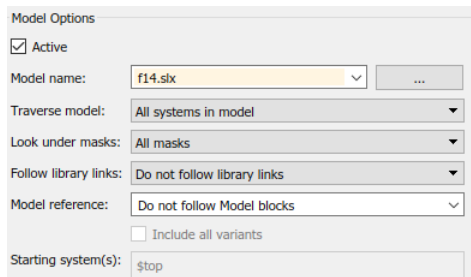


### Example 6.2. Temporarily Omitting a Model from a Loop

This example shows how to use the Model Loop **Active** check box to temporarily omit a model from the loop. This example uses the report setup file that you created in the Generating Reports on Specified Systems and their Subsystems example above, `sldemo_auto_report.rpt`, and the `f14` model, which is included with Simulink

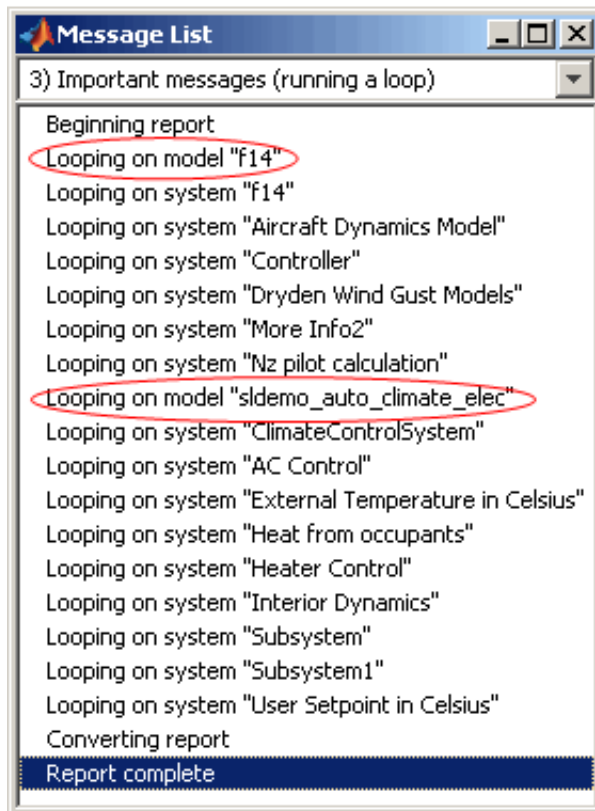
- 1 In the Report Explorer, click **File > Open**, and then open `sldemo_auto_report.rpt` by double-clicking it.
- 2 In the Outline pane on the left, click Model Loop Section 1 - `sldemo_auto_climate_elec`.
- 3 In the **Model Loop** properties pane, click the  button to add a model to the **Models to include** list.
- 4 In the **Model Loop** properties pane, from the **Model name** selection list, select <Custom block diagram>.
- 5 In the **Model name** field, delete the text <Custom block diagram> and enter `f14.slx`.
- 6 In the **Look under masks** selection list, select All masks.

The **Model Loop** properties pane now looks as follows.



- 7 Save the report setup file.
- 8 Generate the report.

The report generation process loops over the specified systems in the f14 and sldemo\_auto\_climate\_elec models, as shown in the following message box.



Below is an excerpt from the generated report.



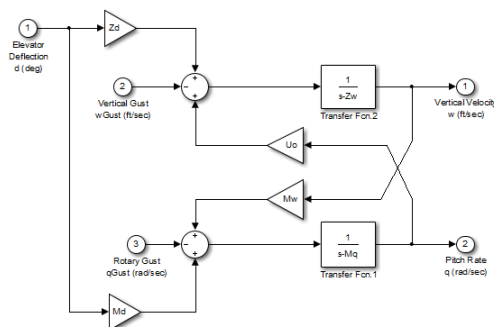
## Chapter 1. Description of subsystems

### Table of Contents

[f14](#)  
[Aircraft Dynamics Model](#)  
[Controller](#)  
[Dryden Wind Gust Models](#)  
[f14](#)  
[Nz pilot calculation](#)  
[sldemo\\_auto\\_climate\\_elec](#)  
[AC Control](#)  
[ClimateControlSystem](#)  
[ElectricalSystem](#)  
[Heater Control](#)  
[Interior Dynamics](#)  
[sldemo\\_auto\\_climate\\_elec](#)  
[Subsystem](#)  
[Subsystem1](#)  
[Variable resistor \(with parasitic L\)](#)

### f14

#### Aircraft Dynamics Model



#### Controller



- 9 In the **Models to include** list, click f14 to select it.
- 10 Clear the **Active** check box to omit f14 model information from the generated report.
- 11 Rerun the report.

The report now includes information only on the `sldemo_auto_climate_elec` model, as shown at the end of the previous example, Generating Reports on Specified Systems and their Subsystems.

- 12 To reactivate the f14 model, in the Model Loop **Models to include** list, select the f14 model and then select the **Active** check box.

## Insert Anything into Report?

Yes, inserts a section if you select **Create section for each object in loop** and a link target if you select **Create link anchor for each object in loop**.

## Class

rptgen\_sl.csl\_md1\_loop

**See Also**

Block Loop, System Loop

# Model Simulation

Run current model with specified simulation parameters

## Description

This component runs the current model using specified simulation parameters. Ensure that this component has the **Model Loop** component as its parent.

For more information on simulation parameters, see “Configure Simulation Conditions”.

## I/O Parameters

### Use model's workspace I/O variable names

Use the names of the parameters specified in the Simulation Parameters dialog box.

The following options are available if you do not select the **Use model's workspace I/O variable names** option:

- **Time** : Specifies a new variable name for the **Time** parameter.
- **States**: Specifies a new variable name for the **States** parameter.
- **Output**: Specifies a new variable name for the **Output** parameter.

## Timespan

**Use model's timespan values**: Use the model's **Start time** and **Stop time** values, as specified in the **Solver** tab in the Simulation Parameters dialog box.

The following options are available if you do not select the **Use model's timespan values** option:

- **Start**: Specifies a simulation starting time.
- **Stop**: Specifies a simulation ending time.

---

**Note** If you set the stop time of your model to `inf` (infinity) in Simulink or on this component attribute page, Simulink Report Generator terminates the model simulation after 60 seconds. Terminating the report prevents the report generation process from entering an infinite loop.

---

## Simulation Options

- **Compile model before simulation**: Compiles the model before simulating, preserving scope content. Select this option if:
  - You use Simulink Coder Summary properties.
  - You sort systems or blocks by simulation order.
  - You use scope snapshots.
- **Simulation status messages**: Displays simulation status messages, or inserts them into the report.

- `Display to command line`: Sends messages to a command-line window.
- `Display to Report Generator Message List`: Sends messages to the Simulink Report Generator message window.
- `Insert into report`: Includes messages in the report.
- **Simulation parameters**: Specifies simulation parameters.

### **Insert Anything into Report?**

No.

### **Class**

`rptgen_sl.csl_mdl_sim`

### **See Also**

Model Loop

# Object Loop

Run child components for Stateflow objects, and then insert table into report

## Description

This component runs its child components for each Stateflow object and inserts a table into the generated report.

For conditional processing of Stateflow objects, you can use the `RptgenSF.getReportedObject` function. For more information, see “Loop Context Functions” on page 4-89.

## Object Types

- **Report on “Data” objects:** Includes Stateflow data objects in the loop.
- **Report on “Event” objects:** Includes Stateflow event objects in the loop.
- **Report on “Transition” objects:** Includes Stateflow transition objects in the loop.
- **Report on “Junction” objects:** Includes Stateflow junction objects in the loop.
- **Report on “Target” objects:** Includes Stateflow target objects in the loop.
- **Report on “Annotation” objects:** Includes Stateflow note objects in the loop.

## Loop Options

- **Report depth:** Specifies the level at which to loop:
  - `Local children only` (Default). Reports only on children one level down.
  - `All objects`. Reports on all Stateflow objects.
- **Skip autogenerated charts under truth tables:** Excludes autogenerated charts under truth tables from the report.
- **Remove objects which do not contain more information than a snapshot:** Excludes objects that contain only a snapshot.
- **Search Stateflow:** Reports on Stateflow charts with specified property name/property value pairs.

## Section Options

- **Create section for each object in loop:** Inserts a section in the generated report for each object found in the loop.
- **Display the object type in the section title:** Automatically inserts the object type into the section title in the generated report.
- **Create link anchor for each object in loop:** Create a link target for each Stateflow object in the loop so that other parts of the report can link to it.

## **Insert Anything into Report?**

Yes, inserts a section if you select **Create section for each object in loop** and a link target if you select **Create link anchor for each object in loop**.

### **Class**

rptgen\_sf.csf\_obj\_loop

### **See Also**

Stateflow Filter, Stateflow Hierarchy Loop, Stateflow Hierarchy Loop, Simulink Function System Loop

# Requirements Block Loop

Apply child components to blocks with requirements

## Description

This component applies its child components to blocks with associated requirements.

## Report On

- **Automatic list from context:** If selected, this option reports on all blocks in the current context. The parent of the Requirements Block Loop component determines its context.
  - **Model Loop:** Reports on all blocks with requirements in the current model.
  - **System Loop:** Reports on all blocks with requirements in the current system.
  - **Signal Loop:** Reports on all blocks with requirements connected to the current signal.

If the Requirements Block Loop does not have the Model Loop, System Loop, Signal Loop, or Block Loop component as its parent, it reports on all blocks in all models.

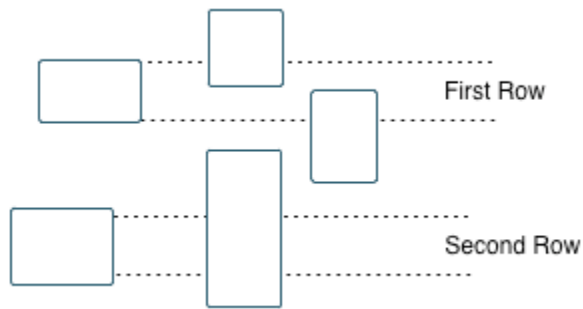
- **Custom - use block list:** Reports on a list of blocks with specified requirements. Enter the full paths of each block into this field.

## Loop Options

- **Sort blocks**

Specify how to sort blocks (applied to each level in a model):

- **Alphabetically by block name:** Sorts blocks alphabetically by name.
- **Alphabetically by system name:** Sorts systems and subsystems alphabetically by name. (Blocks in each system do not appear in alphabetical order).
- **Alphabetically by full Simulink path:** Sorts blocks alphabetically by Simulink path.
- **By block type:** Sorts blocks alphabetically by block type.
- **By block depth:** Sorts blocks by their depth in the model.
- **By layout (left to right):** Sorts blocks by their location in the model layout, by rows. The block appearing the furthest toward the left top corner of the model is the anchor for the row. The row contains all other blocks that overlap the horizontal area defined by the top and bottom edges of the anchor block. The other rows use the same algorithm, using as the anchor the next unreported block nearest the left top of the model.



- **By layout (top to bottom):** Sorts blocks by their location in the model layout, by *columns*. The block appearing the furthest toward the left top corner of the model is the anchor for the column. The column contains all other blocks that overlap the vertical area defined by the left and right edges of the anchor block. The other columns use the same algorithm, using as the anchor the next unreported block nearest the left top of the model.
- **By traversal order:** Sorts blocks by traversal order.
- **By simulation order:** Sorts blocks by execution order.
- **Search for Simulink property name/property value pairs:** Reports on Simulink blocks with specified property name/property value pairs that have associated requirements.

## Section Options

- **Create section for each object in loop:** Inserts a section in the generated report for each block found in the loop that has associated requirements.
- **Display the object type in the section title:** Inserts the object type automatically into the section title in the generated report.
- **Create link anchor for each object in loop:** Create a link target for each requirement in the loop so that other parts of the report can link to it.

## Insert Anything into Report?

Yes, inserts a section if you select **Create section for each object in loop** and a link target if you select **Create link anchor for each object in loop**.

## Class

RptgenRMI.CBlockLoop

## See Also

Missing Requirements Block Loop, Missing Requirements System Loop, Model Loop, Requirements Documents Table, Requirements Signal Loop, Requirements Summary Table, Requirements System Loop, Requirements Table



# Requirements Documents Table

Insert table of linked requirements documents

## Description

This component creates a table that lists all requirements documents linked to model objects.

## Table Options

- **Show documents linked to**
  - **Simulink and Stateflow objects:** Inserts requirements documents linked to both Simulink and Stateflow objects in the model.
  - **Simulink objects:** Inserts requirements documents linked only to Simulink objects in the model.
  - **Stateflow objects:** Inserts requirements documents linked only to Stateflow objects in the model.
- **Table title:** Specifies a title for the table.
  - No title
  - Model name (Default)
  - Custom

## Table Columns

- **Replace document paths with links:** Inserts links to requirements documents when possible.
- **When replacing with links, note absolute vs. relative file path:** Indicates absolute or relative file paths when including links to requirements documents.
- **Include document modification time:** Includes the document modification information.
- **Count # references to each document:** Includes a count of the number of references to the requirements document in the model.

## Document References

- **Replace file names with document IDs in the main body of the report:** Includes shortened IDs to identify requirements documents to simplify the requirements documents table.
- **Retrieve full module path for DOORS links (requires login):** This option applies only to DOORS® requirements. Append the DOORS module ID to the module path in the DOORS database if the module information is not stored with the model.

## Insert Anything into Report?

Yes. Table.

**Class**

RptgenRMI.ReqDocTable

**See Also**

Requirements Summary Table, Requirements Table

# Requirements Signal Loop

Apply all child components to signal groups with requirements

## Description

The Requirements Signal Loop component applies all child components to signal groups that have requirements in Signal Builder blocks.

## Properties

- **Create link anchor for each object in loop:** Create a link target for each requirement in the loop so that other parts of the report can link to it.
- **Display the object type in the section title:** Inserts the object name with requirements into the section title.
- **Create section for each object in loop:** Creates a hyperlink to each object with requirements in the loop.
- **Section Type:** Specifies the section type to insert. If you choose **Automatic**, the Simulink Report Generator software determines the appropriate section type:
  - Book
  - Chapter
  - Section 1
  - Section 2
  - Section 3
  - Section 4
  - Section 5
  - Simple Section
  - Automatic

## Report On

### Loops on signal groups in systems:

- **Collect all Signal Builders:** Processes all Signal Builder blocks, looking for signal groups with requirements.
- **Custom - use list:** Processes all subsystems in the user-defined list. If a subsystem on the list does not have requirements, the Simulink Report Generator software does not include it in the report.

## Insert Anything into Report?

Yes, inserts a section if you select **Create section for each object in loop** and a link target if you select **Create link anchor for each object in loop**.

## **Class**

RptgenRMI.CSystemLoop

## **See Also**

Missing Requirements Block Loop, Missing Requirements System Loop, Requirements Block Loop, Requirements Documents Table, Requirements Summary Table, Requirements System Loop, Requirements Table, Signal Loop

# Requirements Summary Table

Properties of blocks, systems, or Stateflow objects with associated requirements

## Description

This component displays properties of blocks, systems, or Stateflow objects with associated requirements.

## Object Type

Choose the object type to display in the generated report.

- Block (Default)
- System
- Stateflow

The selected object type affects the options available in the **Property Columns** pane.

## Table Title

Specify a table title in the generated report.

- **Automatic:** Generates a title automatically from the parameter.
- **Custom:** Specifies a custom title.

## Property Columns

- Object properties to include in the Requirements Summary Table appear in a list.
  - To add a property:
    - 1 Select the appropriate property level in the text box on the left.
    - 2 In the text box on the right, select the property that you want to add and click **Add**.
  - To delete a property, select the property name and click **Delete**.

`%<SplitDialogParameters>` is a unique property that you can specify for Requirements Summary Tables where the object type is **Block**. This property generates multiple summary tables, grouped by block type. Each Summary Table group contains the dialog box parameters for that block.

Some entries in the list of available properties (such as **Depth**) are “virtual” properties that you cannot access using the `get_param` command. The properties used for property/value filtering in the block and System Loop components must be retrievable by the `get_param`. Therefore, you cannot configure your Requirements Summary Table to report on all blocks of `Depth == 2`.

- **Remove empty columns:** Removes empty columns from the table.
- **Transpose table:** Changes the summary table rows into columns in the generated report, putting the property names in the first column and the values in the other columns.

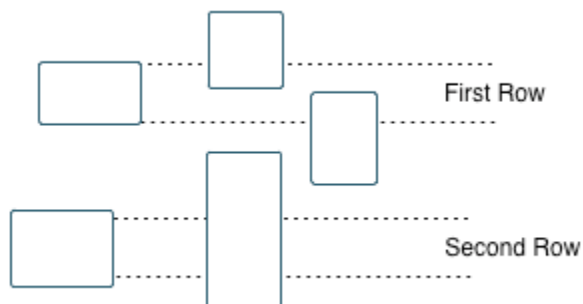
## Object Rows

- **Insert anchor for each row:** Inserts an anchor for each row in the Requirements Summary Table.
- **Report On**
  - **Automatic list from context:** Reports on all blocks in the current context. The parent of this component determines its context.
  - **Custom - use block list:** Reports on a list of blocks that you specify, and enters the block names in the corresponding field. Specify the full path of each block.

## Loop Options

Choose block sorting options and reporting options in this pane.

- **Sort blocks:** Use this option to select how to sort blocks (applied to each level in a model):
  - **Alphabetically by block name:** Sorts blocks alphabetically by name.
  - **Alphabetically by system name.** Sorts systems alphabetically. Lists blocks in each system, but in no particular order.
  - **Alphabetically by full Simulink path:** Sorts blocks alphabetically by Simulink path.
  - **By block type:** Sorts blocks alphabetically by block type.
  - **By block depth:** Sorts blocks by their depth in the model.
  - **By layout (left to right):** Sorts blocks by their location in the model layout, by *rows*. The block appearing the furthest toward the left top corner of the model is the anchor for the row. The row contains all other blocks that overlap the horizontal area defined by the top and bottom edges of the anchor block. The other rows use the same algorithm, using as the anchor the next unreported block nearest the left top of the model.



- **By layout (top to bottom):** Sorts blocks by their location in the model layout, by *columns*. The block appearing the furthest toward the left top corner of the model is the anchor for the column. The column contains all other blocks that overlap the vertical area defined by the left and right edges of the anchor block. The other columns use the same algorithm, using as the anchor the next unreported block nearest the left top of the model.
- **By traversal order:** Sorts blocks by traversal order.
- **By simulation order:** Sorts blocks by execution order.
- **Search for Simulink property name/property value pairs:** Reports on blocks with specified property name/property value pairs.

## **Insert Anything into Report?**

Yes. Table.

### **Class**

RptgenRMI.CSummaryTable

### **See Also**

Block Loop, Missing Requirements Block Loop, Missing Requirements System Loop, Requirements Block Loop, Requirements Documents Table, Requirements Signal Loop, Requirements System Loop, Requirements Table

## Requirements System Loop

Apply child components to systems with requirements

### Description

This component applies its child components to systems with associated requirements.

### Report On

- **Loop on systems**

- **Select systems automatically:** If selected, this option reports on all systems in the current context. The parent of the component determines the context of this setting:
  - **Model Loop:** Reports on systems in the current model.
  - **System Loop:** Reports on the current system.
  - **Signal Loop:** Reports on the parent system of the current signal.
  - **Block Loop:** Reports on the parent system of the current block.

If the Requirement System Loop does not have any of these components as its parent, selecting this option reports on all systems with requirements in all models.

- **Custom - use system list:** Reports on a list of specified systems. Enter the full path of each system.

### Loop Options

- **Sort Systems:**
  - **Alphabetically by system name (default):** Sorts systems alphabetically by name.
  - **By number of blocks in system:** Sorts systems by the number of blocks in the system. The list displays systems by decreasing number of blocks; the system with the largest number of blocks appears first in the list.
  - **By system depth:** Sorts systems by their depth in the model.
  - **By traversal order:** Sorts systems in the traversal order .
- **Search for:** Reports on Simulink blocks with specified property name/property value pairs.

### Section Options

- **Create section for each object in loop:** Inserts a section in the generated report for each object found in the loop.
- **Display the object type in the section title:** Inserts the object type automatically into the section title in the generated report.
- **Number sections by system hierarchy:** Numbers sections in the generated report hierarchically. Requires that **Sort Systems** be set to **By traversal order**.
- **Create link anchor for each object in loop:** Create a link target for each requirement in the loop so that other parts of the report can link to it.



## Insert Anything into Report?

Yes, inserts a section if you select **Create section for each object in loop** and a link target if you select **Create link anchor for each object in loop**.

### Class

RptgenRMI.CSystemLoop

### See Also

Missing Requirements Block Loop, Missing Requirements System Loop, Requirements Block Loop, Requirements Documents Table, Requirements Signal Loop, Requirements Summary Table, Requirements Table, System Loop

## Requirements Table

Requirements links for current context

### Description

This component creates a table that contains information from the Simulink Requirements software. Objects can have multiple requirements. Each requirement is a row in the table.

---

**Note** If you want to generate a Microsoft Word report, to enable Simulink Requirements hyperlinks in your report, at the MATLAB command prompt, enter:

```
rmipref('ReportNavUseMatlab',true)
rmipref('UnsecureHttpRequests',true)
```

---

### Table Options

- **Show requirements for current:** Specifies the object type to display.
  - Simulink object
  - Stateflow object
- **Table title:** Specifies a title for the table.
  - No title
  - Object name (Default)
  - Custom

### Table Columns

- **Description:** Includes the object description in the table.
- **Document name:** Includes the report name in the table.
- **Locations within document:** Includes the locations of the object within the document in the table.
- **Requirement keyword:** Includes the requirement keyword for the object in the table.

### Insert Anything into Report?

Yes. Table.

### Class

RptgenRMI.CReqTable

**See Also**

Missing Requirements Block Loop, Missing Requirements System Loop, Requirements Block Loop, Requirements Documents Table, Requirements Signal Loop, Requirements Summary Table, Requirements System Loop, Stateflow Automatic Table, Stateflow Name

## Scope Snapshot

Insert images of scopes and XY graphs

### Description

This component inserts images of scopes and XY graphs. Examples of blocks for which this component inserts snapshots include:

- Scope (and Floating Scope) blocks and the XY Graph block (Simulink)
- Spectrum Analyzer and Time Scope blocks (DSP System Toolbox™)
- Video Viewer (Computer Vision Toolbox™)
- Blocks in the Simulink Control Design™ Linear Analysis Plots library (for example, the Bode Plot block)

If the model has not been simulated, scopes are empty. For more information, see the [Model Simulation](#) component reference page.

The parent component of the `Scope Snapshot` determines its behavior.

- `Model Loop` or no Simulink looping component: Includes all XY graphs and scopes in the current model.
- `System Loop`: Includes all XY graphs and scopes in the current system.
- `Block Loop`: Includes the current block when it is an XY graph or scope.
- `Signal Loop`: Includes all XY graphs and scopes connected to the current signal.

If the `Scope Snapshot` does not have any of the Simulink looping components as its parent, it includes all XY graphs and scopes in all open models.

### Scope Options

- **Report on closed scopes:** Takes a snapshot of all scopes in context. This option forces closed scopes to open when the report is generating.
- **Autoscale time axis:** Scales the Simulink scope time axis to include the entire log.

### Print Options

- **Image file format:** Specifies the image file format (for example, JPEG, TIFF, etc.). Select `Automatic HG Format` (the default) to choose the format best suited for the specified output format automatically. Otherwise, choose an image format that your output viewer can read.
  - `Automatic HG Format` (uses the Simulink file format selected in the Preferences dialog box)
  - `Bitmap (16m-color)`
  - `Bitmap (256-color)`
  - `Black and white encapsulated PostScript`
  - `Black and white encapsulated PostScript (TIFF)`

- Black and white encapsulated PostScript2
- Black and white encapsulated PostScript2 (TIFF)
- Black and white PostScript
- Black and white PostScript2
- Color encapsulated PostScript
- Color encapsulated PostScript (TIFF)
- Color encapsulated PostScript2
- Color encapsulated PostScript2 (TIFF)
- Color PostScript
- Color PostScript2
- JPEG high quality image
- JPEG medium quality image
- JPEG low quality image
- PNG 24-bit image
- TIFF - compressed
- TIFF - uncompressed
- Windows metafile
- **Paper orientation:**
  - Landscape
  - Portrait
  - Rotated
  - Use figure orientation: Uses the orientation for the figure, which you set with the `orient` command.
  - Full page image (PDF only): In PDF reports, scales images to fit the full page, minimizes page margins, and maximizes the size of the image by using either a portrait or landscape orientation.

For more information about paper orientation, see the `orient` reference page in the MATLAB documentation.

- **Image size:** Specifies the size of the Handle Graphics figure snapshot in the form [w h] (width, height). In the units text box, select one of the following options:
  - Inches
  - Centimeters
  - Points
  - Normalized
- **Invert hardcopy:** Inverts colors for printing; changes dark colors to light colors and light colors to dark colors.
  - Automatic: Automatically changes dark axes colors to light axes colors. If the axes color is a light color, this option does not invert the color.
  - Invert: Changes dark axes colors to light axes colors and light axes colors to dark axes colors.

- **Don't invert:** Does not change the colors in the image on the screen for printing.
- **Use figure's InvertHardcopy setting:** Uses the InvertHardcopy property set in the Handle Graphics image.
- **Make figure background transparent:** Makes the image background transparent.

## Display Options

- **Scaling:** Controls size of the image, as displayed in a browser. Making an image larger using this option does not affect the storage size of the image, but the quality of the displayed image may decrease as you increase or decrease the size of the displayed image.

Generally, to achieve the best and most predictable display results, use the default setting of `Use image size`.

- **Use image size:** Causes the image to appear the same size in the report as on screen (default).
- **Fixed size:** Specifies the number and type of units.
- **Zoom:** Specifies the percentage, maximum size, and units of measure.
- **Size:** Specifies the size of the snapshot in the form `w h` (width, height). This field is active only if you choose `Fixed size` from the **Scaling** selection list.
- **Max size:** Specifies the maximum size of the snapshot in the form `w h` (width, height). This field is active only if you choose `Zoom` from the **Scaling** selection list.
- **Units:** Specifies the units for the size of the snapshot. This field is active only if you choose `Zoom` or `Fixed size` in the **Image size** list box.
- **Alignment:** Only reports in PDF or RTF format support this property.
  - Auto
  - Right
  - Left
  - Center
- **Title:** Specifies a title for the snapshot figure.
  - `Block name:` Uses the block name as the title.
  - `Full Simulink path name:` Uses the Simulink path as the title.
  - `Custom:` Specifies a custom title.
- **Caption:** Select or enter a short text description for the snapshot figure.
  - `No caption`
  - `Automatic (use block description).` Uses the Simulink block description as the caption.
  - `Custom.` Specifies a short text description for the snapshot figure.

## Insert Anything into Report?

Yes. Image.

**Class**

rptgen\_sl.csl\_blk\_scope

**See Also**

Block Loop, Model Loop, Signal Loop, System Loop

## Signal Loop

Run child components for each signal contained in current system, model, or block

### Description

The `Signal Loop` component runs its child components for each signal contained in the current system, model, or block. The parent component determines the behavior of this component.

- `Model Loop`: Loops on all signals in the current model.
- `System Loop`: Loops on all signals in the current system. Choose not to report on the following types of signals by clearing the corresponding option in the **Section options** area:
  - **Include system input signals**
  - **Include system output signals**
  - **Include system internal signals**
- `Signal Loop`: Loops on the current signal.
- `Block Loop` : Loops on all signals connected to the current block. Choose not to report on the following types of signals by clearing the corresponding option in the **Section options** area:
  - **Include block input signals**
  - **Include block output signals**
- If the `Signal Loop` does not have a looping component as its parent, it loops on all signals in all models. Choose not to report on the following types of signals by clearing the corresponding option in the **Section options** area:
  - **Include block input signals**
  - **Include block output signals**
  - **System input signals**
  - **System output signals**
  - **System internal signals**

For conditional processing of signals, you can use the `RptgenSL.getReportedSignal` function. For more information, see “Loop Context Functions” on page 4-89.

### Select Signals

- **Include block input signals**: Loops on signals that feed into blocks. This option is valid only when the parent component of this component is a `Block Loop`.
- **Include block output signals**: Loops on signals that leave the block. This option is valid only when the parent component of this component is a `Block Loop`.
- **Include system input signals**: Loops on signals coming from inports. This option is valid only when the parent component of this component is a `System Loop`.
- **Include system internal signals**: Loops on system internal signals. This option is valid only when the parent component of this component is a `System Loop`.



- **Include system output signals:** Loops on signals going to outputs. This option is valid only when the parent component of this component is a `System Loop`.
- **Sort signals:** Specifies how to sort signals:
  - `Alphabetically by signal name:` Sorts signals alphabetically by name.
  - `Alphabetically by signal name (exclude empty):` Sorts signals alphabetically by name.
  - `Alphabetically by system name:` Sorts alphabetically by parent system names. Lists signals in each system, but in no particular order.
  - `By signal depth:` Sorts signals by their depth in the model.

## Section Options

- **Create section for each object in loop:** Inserts a section in the generated report for each object found in the loop.
- **Display the object type in the section title:** Automatically inserts the object type into the section title in the generated report.
- **Create link anchor for each object in loop:** Create a link target for each signal in the loop so that other parts of the report can link to it.

## Insert Anything into Report?

Yes, inserts a section if you select **Create section for each object in loop** and a link target if you select **Create link anchor for each object in loop**.

## Class

`rptgen_sl.csl_sig_loop`

## See Also

`Block Loop`, `Model Loop`, `System Loop`

## Simulink Automatic Table

Insert two-column table with information on selected model, system, signal, or block

### Description

This component inserts a two-column table that contains details for the selected model, system, signal, or block into a generated report.

### Options

- **Show current:** Modeling object to specify properties for.
  - Automatic: Uses the context of the parent loop.
  - Model
  - System
  - Block
  - Annotation
- **Properties list:** Specifies whether to have Report Explorer select properties automatically or to list the properties to report on.
  - Determine properties automatically: Let the Report Explorer automatically select the properties to report.

Modeling Component Selected in the Show current Field	Listed Properties
Model	Description
System	Description
Block	Block parameter dialog box prompt properties
Annotation	Text
Signal	Description

- **Show properties:** Specify a list of properties to report. Enter the names of object properties that you want the report to include for the modeling object you specified in the **Show current** field. Use this option to display properties that the Report Explorer does not include automatically.

Property names often differ from the Simulink dialog box prompts. Refer to the Simulink documentation to determine property names for blocks, signals, and other modeling objects. You can also use the MATLAB `get` command to determine the property names of an object. For example, to determine the property names of the block currently selected in a model, enter the following at the MATLAB command line:

```
get(get_param(gcf, 'Handle'))
```

- **Show full path name:** Displays the full path of the selected Simulink model.
- **Display property names as prompts:** Displays property names as prompts in the generated report. The report includes the dialog box string instead of the underlying code property.

## Display Options

- **Table title:** Displays a table title in the generated report.
  - Name: Automatically generates a title from the parameter.
  - Custom: Specifies a custom title.
  - No title: Does not include a title.
- **Header row:** Select a header row for the table in the generated report.
  - No header: Includes no header row.
  - Type and Name: Includes a header row with columns for name and object type.
  - Custom: Includes a custom header.
- **Don't display empty values:** Excludes empty parameters in the generated report.

## Insert Anything into Report?

Yes. Table.

## Class

rptgen\_sl.csl\_auto\_table

## See Also

Block Loop, Model Loop, Signal Loop, System Loop

# Simulink Data Dictionary

Report Simulink data dictionary information

## Description

This component reports on the data dictionary currently active in the data dictionary loop specified by the Data Dictionary Loop component. Include this component as a child of a Simulink Data Dictionary Loop component.

## Presentation Format

The report for a data dictionary includes a table that summarizes the properties of each variable in the dictionary. The report also includes a dictionary details section that fully reports the properties and value of each variable in the dictionary. If you use a conversion template to generate the report, you can specify template-defined styles for the summary table title and the summary table.

To use a conversion template, in the Report Options dialog box, set **File format** to one of the from template options, for example, **Direct PDF (from template)**.

- **Table title style name:** Specifies the style to use for the data dictionary table title. To specify the default style name `rgTableTitle`, which the default conversion template defines, use `Auto`. To specify a custom style defined in a custom template that you use with this report, select `Specify`.
- **Table style name:** Specifies the style to use for the data dictionary table. To specify the default table style name `rgUnruledTable`, which is the default conversion template defines, use `Auto`. To specify a custom style defined in a custom template that you use with this report, select `Specify`.

## Options

You can specify whether to include dictionaries referenced by a dictionary and how to present the referenced information.

- **Include referenced data dictionaries:** Includes information from the data dictionaries that the dictionary currently active in the data dictionary loop specified by the Data Dictionary Loop component references. The referenced information displays at the end of the table for the referencing data dictionary, unless you select **Make separate table for each referenced dictionary**.
- **Make separate table for each referenced dictionary:** If you select **Include referenced data dictionaries**, display a table for each referenced data dictionary.
- **Include referenced dictionaries list:** If you select **Include referenced data dictionaries**, following the referencing data dictionary summary table, include a list of the referenced data dictionaries.

## Sections to Report

You can specify the data dictionary sections to include data for.

- **Design Data** (default): Include information from the Design Data section of the current data dictionary.
- **Configuration**: Include information from the Configuration section of the current data dictionary.
- **Other Data**: Include information from the Other Data section of the current data dictionary.

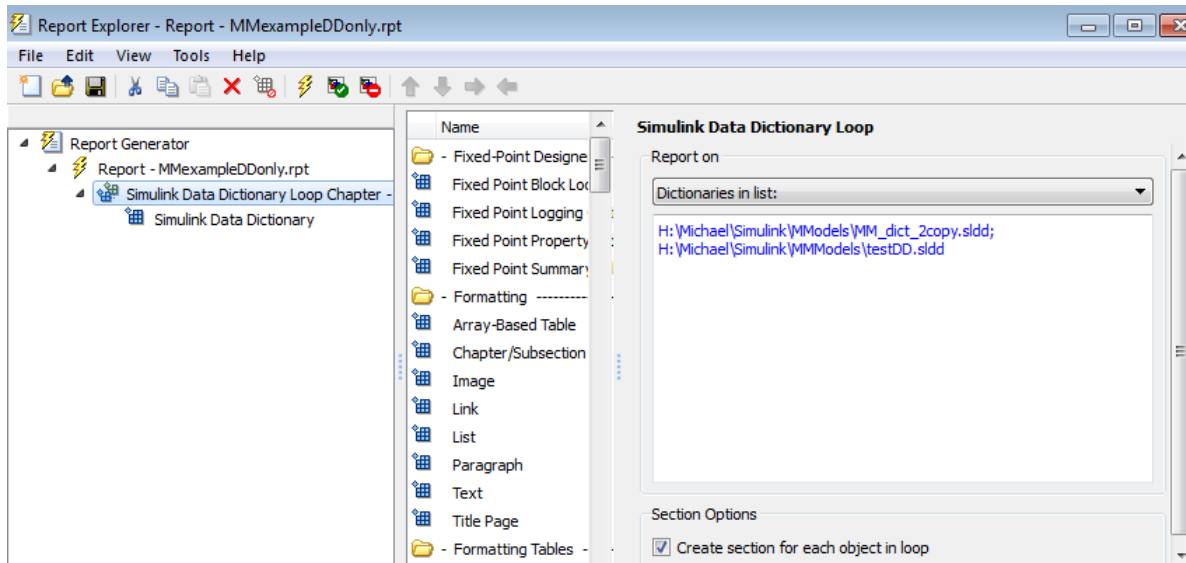
## Fields to Report

The current dictionary summary table lists properties of the variables that it contains. The table always includes the variable name and value. In addition, it optionally includes these properties:

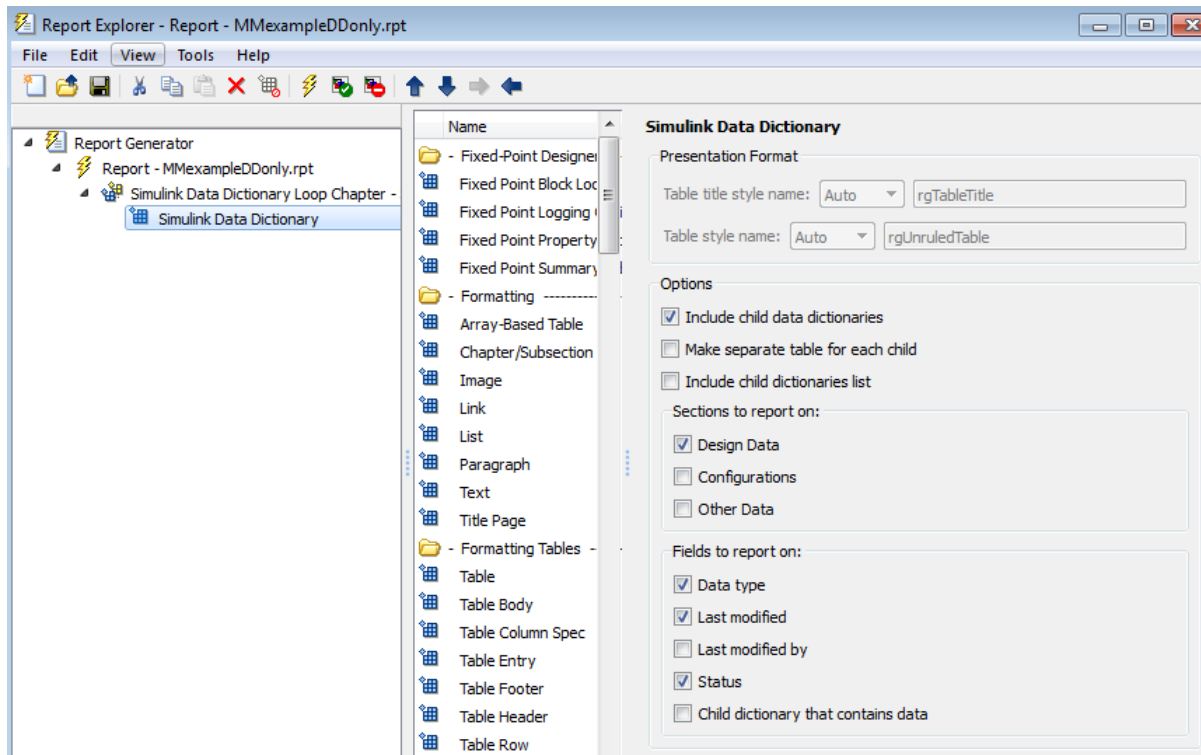
- **Data type**
- **Last modified**
- **Last modified by**
- **Status**
- **Referenced dictionary that contains data**

## Example

Suppose that you configure an HTML report with the Simulink Data Dictionary Loop component.



Then you configure the Simulink Data Dictionary component.



Here is the resulting report.

**Table of Contents**[1. H:\Michael\Simulink\MMmodels\MM\\_dict\\_2copy.sldd](#)[2. H:\Michael\Simulink\MMmodels\testDD.sldd](#)**Chapter 1. H:\Michael\Simulink\MMmodels\MM\_dict\_2copy.sldd****MM\_Entry1**

- Name: MM\_Entry1
- Value: 27
- class: double
- LastModified: 2015-Jan-20 01:15:09.619512
- Status: Unchanged

**MM\_d2\_Entry1**

- Name: MM\_d2\_Entry1
- Value: 27
- class: double
- LastModified: 2015-Jan-30 23:30:23.182407
- Status: Modified

**MM\_d2\_phone\_number**

- Name: MM\_d2\_phone\_number
- Value: 508-647-8103
- class: char
- LastModified: 2015-Jan-30 23:30:05.414130
- Status: Modified

**MM\_phone\_number**

- Name: MM\_phone\_number
- Value: 508-647-8103
- class: char
- LastModified: 2015-Jan-20 01:15:09.782512
- Status: Unchanged

**Chapter 2. H:\Michael\Simulink\MMmodels\testDD.sldd****testVar**

- Name: testVar
- Value: testVarVal
- class: char
- LastModified: 2015-Jan-30 23:37:08.139822
- Status: New

**Class**

rptgen\_sl.csl\_data\_dictionary

**See Also**

Simulink Data Dictionary Loop

## Simulink Data Dictionary Loop

Run Simulink Data Dictionary child component for each Simulink data dictionary in specified context

### Description

This component runs the Simulink Data Dictionary child component for each Simulink data dictionary in the specified context. You can specify whether to have each data dictionary in the loop.

### Report on

Specify the data dictionaries to report on.

- **Dictionaries in MATLAB path:** Report on all data dictionaries on the MATLAB path. If you select **Include child data dictionaries**, then also reports on child data dictionaries whose parent is on the MATLAB path.
- **Dictionaries in list:** Report on all data dictionaries that you specify in the text box. Enter data dictionary names, separated by either a comma or semicolon. You can use multiple lines. If you do not specify the full path to a data dictionary, the loop includes that data dictionary only if the dictionary is on the MATLAB path.

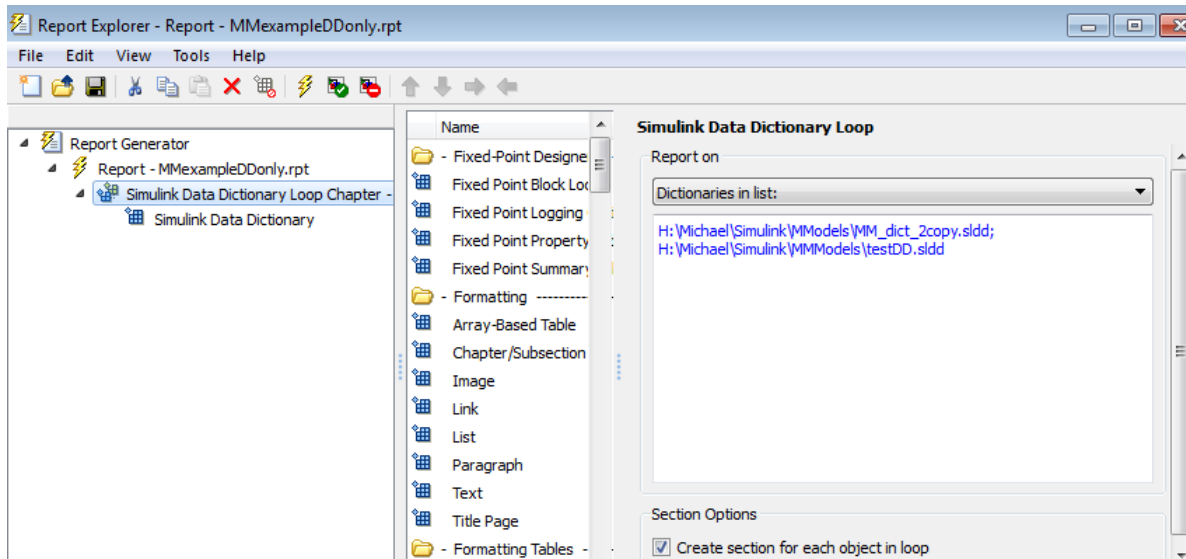
Use a Summary Table component to show annotation objects in reports. Each Summary Table component creates a single table with each reported annotation on a single row of the table.

### Section Options

**Create section for each object in loop:** Create a separate chapter for each data dictionary.

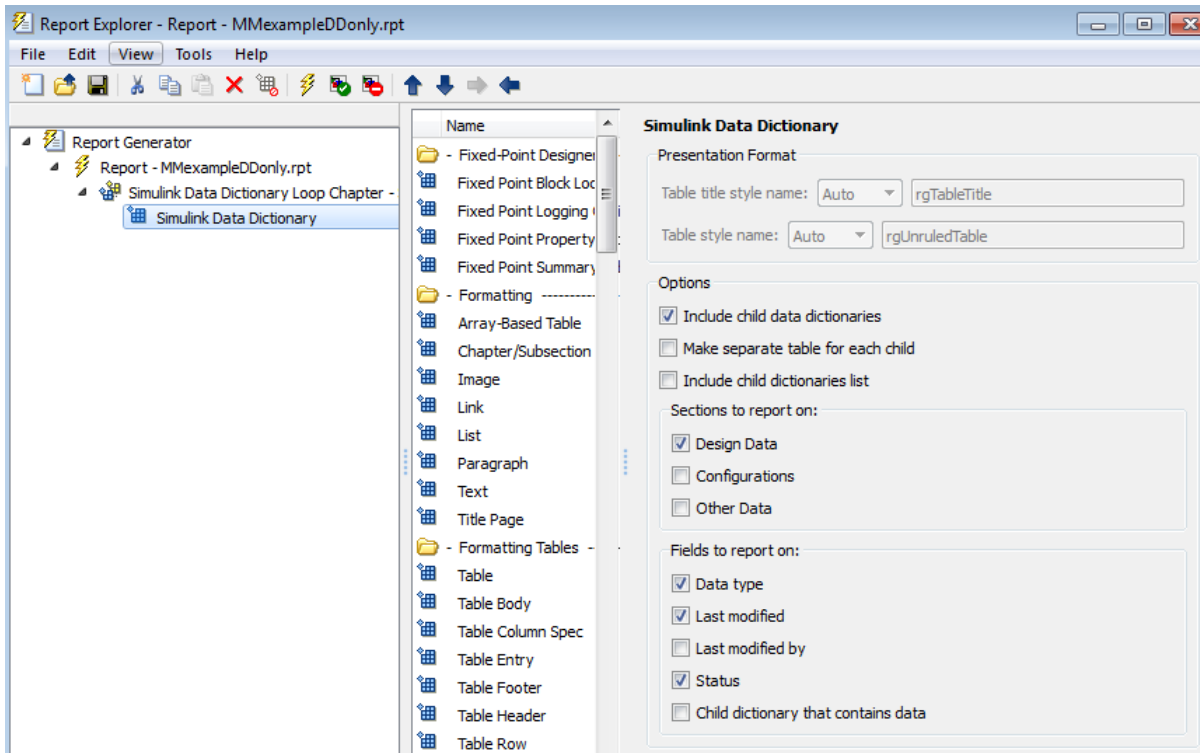
### Example

Suppose you have an HTML report with the Simulink Data Dictionary Loop component configured like this:





Then you configure the Simulink Data Dictionary component like this:



The resulting report looks like this:

**Table of Contents**

[1. H:\Michael\Simulink\MMmodels\MM\\_dict\\_2copy.sldd](#)

[2. H:\Michael\Simulink\MMmodels\testDD.sldd](#)

**Chapter 1. H:\Michael\Simulink\MMmodels\MM\_dict\_2copy.sldd**

**MM\_Entry1**

- Name: MM\_Entry1
- Value: 27
- class: double
- LastModified: 2015-Jan-20 01:15:09.619512
- Status: Unchanged

**MM\_d2\_Entry1**

- Name: MM\_d2\_Entry1
- Value: 27
- class: double
- LastModified: 2015-Jan-30 23:30:23.182407
- Status: Modified

**MM\_d2\_phone\_number**

- Name: MM\_d2\_phone\_number
- Value: 508-647-8103
- class: char
- LastModified: 2015-Jan-30 23:30:05.414130
- Status: Modified

**MM\_phone\_number**

- Name: MM\_phone\_number
- Value: 508-647-8103
- class: char
- LastModified: 2015-Jan-20 01:15:09.782512
- Status: Unchanged

**Chapter 2. H:\Michael\Simulink\MMmodels\testDD.sldd**

**testVar**

- Name: testVar
- Value: testVarVal
- class: char
- LastModified: 2015-Jan-30 23:37:08.139822
- Status: New

**Class**

rptgen\_sl.csl\_data\_dict\_loop

**See Also**

Simulink Data Dictionary

# Simulink Dialog Snapshot

Insert snapshots of Simulink editor dialog boxes

## Description

This component takes snapshots of Simulink editor dialog boxes. You use it to display the current settings associated with an object or document the appearance of your custom mask dialog boxes.

The parent component of this component determines the behavior of this component.

- **Block Loop:** Documents the dialog box of the current reported block.
- **System Loop:** Documents the dialog box of the current reported system.

## Format

- **Image file format:** Specifies the format for the snapshot image file. The **automatic** format chooses BMP format for PDF files, and PNG for other formats.
- **Show all tabs:** Automatically generates images for all the tabs for the dialog box. If you clear this check box, Simulink Report Generator creates an image of only the first tab.

## Display Options

- **Scaling:** Controls size of the image, as displayed in a browser. Making an image larger using this option does not affect the storage size of the image, but the quality of the displayed image may decrease as you increase or decrease the size of the displayed image.

Generally, to achieve the best and most predictable display results, use the default setting of **Use image size**.

- **Use image size:** Causes the image to appear the same size in the report as on screen (default).
- **Fixed size:** Specifies the number and type of units.
- **Zoom:** Specifies the percentage, maximum size, and units of measure.
- **Size:** Specifies the size of the snapshot in the format **w h** (width, height). This field is active only if you choose **Fixed size** from the **Scaling** selection list.
- **Max size:** Specifies the maximum size of the snapshot in the format **w h** (width, height). This field is active only if you choose **Zoom** from the **Scaling** selection list.
- **Units:** Specifies the units for the size of the snapshot. This field is active only if you choose **Zoom** or **Fixed size** in the **Image size** list box.
- **Alignment:** Only reports in PDF or RTF format support this property.
  - Auto
  - Right
  - Left
  - Center

- **Title:** Specifies text to appear above the snapshot.
- **Caption:** Specifies text to appear under the snapshot.

## **Insert Anything into Report?**

Yes. Snapshot.

### **Class**

rptgen\_sl.CDialogSnapshot

### **See Also**

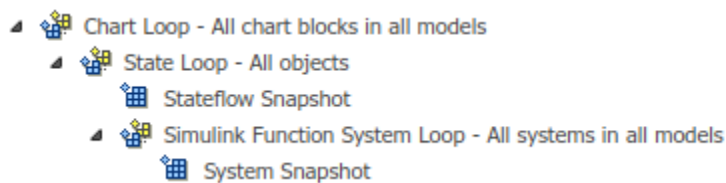
Block Loop, System Loop

# Simulink Function System Loop

Report on Simulink functions specified in a Stateflow loop

## Description

This component loops over the Simulink systems that implement a Stateflow Simulink Function object, including the function's parent system, subsystems, and optionally the systems that implement Simulink Functions nested in this function. This component must be a descendant of a `State Loop` component that is descendant of a `Chart Loop` component. This component executes when the current object in the state loop is a Simulink Function. For example, this structure creates a snapshot of each Simulink Function in a chart followed by snapshots of the systems that implement the function:



## Report On

**Include subsystems in nested Simulink functions:** Specifies whether to include subsystems in nested Simulink functions. By default, this option is enabled.

## Loop Options

- **Sort Systems:** Specifies how to sort systems.
  - `Alphabetically by system name` (default): Sorts systems alphabetically by name.
  - `By number of blocks in system`: Sorts systems by number of blocks. The list shows systems by decreasing number of blocks; that is, the system with the largest number of blocks appears first in the list.
  - `By system depth`: Sorts systems by their depth in the model.
  - `By traversal order`: Sorts systems in traversal order.
- **Search for:** Reports only on Subsystem blocks with the specified property name/property value pairs. To enable searching, click the check box. In the first row of the property name and property value table, click inside the edit box, delete the existing text, and type the property name and value.

For information about subsystem property names and values, in “Block-Specific Parameters”, see the “Ports & Subsystems Library Block Parameters” section.

## Section Options

- **Create section for each object in loop:** Inserts a section in the generated report for each object found in the loop.

- **Display the object type in the section title:** Inserts the object type automatically into the section title in the generated report.
- **Number sections by system hierarchy:** Hierarchically numbers sections in the generated report. Requires that **Sort Systems** be set to `By traversal order`.
- **Create link anchor for each object in loop:** Create a link target for each Simulink Function system in the loop so that other parts of the report can link to it.

## Insert Anything into Report?

Yes, inserts a section if you select **Create section for each object in loop** and a link target if you select **Create link anchor for each object in loop**.

### Class

```
rptgen_sl.csl_sys_loop
```

### See Also

Object Loop, State Loop, Chart Loop, System Loop, Block Loop, Model Loop, Signal Loop

# Simulink Functions and Variables

Create table that displays workspace variables and MATLAB functions used by reported blocks in Simulink models

## Description

This component creates a table that displays workspace variables and MATLAB functions used by blocks in a Simulink model. The `Model Loop` component specifies the current model and systems in which the blocks appear. For example, suppose a Simulink Gain block has a string `cos(x)` instead of a number. The Simulink software looks for a variable `x` in the workspace and uses the `cos` function.

## Functions

- **Include table of functions:** Includes a table of Simulink functions in the generated report.
- **Table Title:** Specifies a title for the table in the generated report:
  - **Automatic:** Generates a title automatically from the parameter.
  - **Custom:** Specifies a custom title.
- **Parent block:** Includes a column in the table that includes the name of the block, which contains the reported variable.
- **Calling string:** Includes the MATLAB code that calls the reported variable.
- **Include fixed-point functions (sfix, ufix, ...):** Includes Fixed-Point Designer functions in the report.

## Variables

- **Include table of variables:** Includes a table of Simulink variables in the generated report.
- **Table title:** Specifies a title for the table in the generated report.
  - **Automatic:** Generates a title automatically from the parameter.
  - **Custom:** Specifies a custom title.
- **Include Workspace I/O parameters:** Reports on variables that contain parameters with time vectors and state matrices. Set these parameters in the **Workspace I/O** pane in the Simulation Parameters dialog box in a Simulink model.

In the following table, if any of the entries in the first column are `on`, the component looks for the variable listed in the second column. If the component finds the variable in the workspace, it includes it in the report.

Parameter name	Variable name
LoadExternalInput	ExternalInput
SaveTime	TimeSaveName
SaveState	StateSaveName
SaveOutput	OutputSaveName

Parameter name	Variable name
LoadInitialState	InitialState
SaveFinalState	FinalStateName

- **Parent block:** Includes the name of the block that contains the reported variable.
- **Calling string:** Includes the MATLAB code that calls the reported variable.
- **Size of variable:** Includes the size of the reported variable.
- **Class of variable:** Includes the variable class to which the reported variable belongs.
- **Memory size:** Includes the amount of memory in bytes that the reported variable needs.
- **Value in workspace:** Includes the value of the reported variable.

Large arrays may appear as [MxN CLASS]. For example, if you have a 300-by-200 double array, it appears in the report as [300x200 DOUBLE].

- **Storage class:** Include the storage class of the reported variable.

The title of this column is **Storage Class**. This option looks at the model's TunableVars property to see if any of the model variables specify their storage class. If you specify the storage class, TunableVarsStorageClass and TunableVarsTypeQualifier appear in a table column in the model variables table.

The column entries are TunableVarsStorageClass (TunableVarsTypeQualifier) when TunableVarsTypeQualifier is not empty. If TunableVarsTypeQualifier is empty, the column entry is TunableVarsStorageClass.

Values for TunableVarsStorageClass include:

- Exported Global
  - Auto
  - ImportedExtern
  - ImportedExtern Pointer
- **Data object properties:** For variables that are Simulink.Parameter data objects, includes the values of the object properties that you list in the edit box.

### Example

This table is an example of a table created by the Model Variables component. This Property Table reports on the variables in the Controller in the f14 model.

Variable Name	Parent Blocks	Calling String	Value
Ka	f14/Controller/Gain3	Ka	0.677
Kf	f14/Controller/Gain	Kf	-1.746
Ki	f14/Controller/Proportional plus integral compensator	[Ki]	-3.864
Kq	f14/Controller/Gain2	Kq	0.8156



## **Insert Anything into Report?**

Yes. Table.

### **Class**

`rptgen_sl.csl_obj_fun_var`

### **See Also**

Block Loop, Model Loop, Signal Loop, System Loop

## Simulink Library Information

Insert table that lists library links in the current model, system, or block

### Description

This component inserts a table that lists library links in the current model, system, or block.

### Table Columns

- **Block:** Includes the Simulink block name in the generated table.
- **Library:** Includes the Simulink library root name in the generated table.
- **Reference block:** Includes the Simulink reference block name in the generated table.
- **Link status:** Includes the link status in the generated table.

### Display Options

- **Title:** Specifies a title for the generated report.
- **Sort table by:**
  - **Block:** Sorts the table by block name.
  - **Library:** Sorts the table by library name.
  - **Reference Block:** Sorts the table by reference block name.
  - **Link Status:** Sorts the table by link status.
- **Merge repeated rows:** Merges sorted rows in the generated table.

### Example

The following table sorts based on Reference Block column. The Report Explorer uses the `aero_guidance` model with **Merge repeated rows** deselected to generate the table.

Block	Library	Reference Block	Status
Equations of Motion (Body Axes)	Aerospace	Equations of Motion (Body Axes)	resolved
Incidence & Airspeed	Aerospace	Incidence & Airspeed	resolved
Fin Actuator	Aerospace	2nd Order Nonlinear Actuator	resolved
3DoF Animation	Aerospace	3DoF Animation	resolved
Atmosphere	Aerospace	Atmosphere model	resolved
Cm	Simulink	Interpolation (n-D) using PreLookup	resolved
Cx	Simulink	Interpolation (n-D) using PreLookup	resolved

Block	Library	Reference Block	Status
Cz	Simulink	Interpolation (n-D) using PreLookup	resolved
Kg	Simulink	Interpolation (n-D) using PreLookup	resolved
Ki	Simulink	Interpolation (n-D) using PreLookup	resolved
Alpha Index	Simulink	PreLookup Index Search	resolved
Mach Index	Simulink	PreLookup Index Search	resolved
Mach Index	Simulink	PreLookup Index Search	resolved
Alpha  Index	Simulink	PreLookup Index Search	resolved

When you select **Merge repeated rows**, the generated table collapses rows in the Block column. Each row in the Reference Block column is unique, as shown in the following table.

Block	Library	Reference Block	Status
Equations of Motion (Body Axes)	Aerospace	Equations of Motion (Body Axes)	resolved
Incidence & Airspeed	Aerospace	Incidence & Airspeed	resolved
Fin Actuator	Aerospace	2nd Order Nonlinear Actuator	resolved
3DoF Animation	Aerospace	3DoF Animation	resolved
Atmosphere	Aerospace	Atmosphere model	resolved
Cm	Simulink	Interpolation (n-D) using PreLookup	resolved
Cx			
Cz			
Kg			
Ki			
Alpha Index	Simulink	PreLookup Index Search	resolved
Mach Index			
Mach Index			
Alpha  Index			

## Insert Anything Into Report?

Yes. Table.

**Class**

rptgen\_sl.CLibinfo

**See Also**

Block Loop, Model Loop, System Loop

# Simulink Linking Anchor

Designate locations to which links point

## Description

This component designates a location to which links point. Use the `Model Loop`, `System Loop`, `Block Loop`, or `Signal Loop` component as the parent component for this component.

## Properties

- **Insert text:** Specifies text to appear after the linking anchor.
- **Link from current:** Sets the current model, system, block, or signal as the linking anchor.
  - **Automatic:** Automatically selects the appropriate model, system, block, or signal as a linking anchor. If the `Model Loop` component is the parent component, the linking anchor is set on the current model. Similarly, if the `Block Loop` or `Signal Loop` is the parent component, the linking anchor is inserted for the current system, block, or signal, respectively.
  - **Model:** Sets the linking anchor to the current model.
  - **System:** Sets the linking anchor to the current system.
  - **Block:** Sets the linking anchor to the current block.
  - **Annotation:** Sets the linking anchor to the current annotation.
  - **Signal:** Sets the linking anchor to the current signal.

---

**Note** Use only one anchor per report each object. For more information, see the [Simulink Summary Table](#) component reference page.

---

## Insert Anything into Report?

Yes. A link, and possibly text, depending on attribute choices.

## Class

`rptgen_sl.csl_obj_anchor`

## See Also

`Block Loop`, `Model Loop`, `Signal Loop`, `System Loop`

## Simulink Name

Insert name of a Simulink model, system, block, or signal into report

### Description

This component inserts the name of a Simulink model, system, block, or signal into the report.

Using this component as the first child component of a **Chapter/Subsection** component allows the current Simulink model, system block, or signal name to be the chapter or section title.

### Properties

- **Object type**
  - **Automatic:** Automatically selects the appropriate model, system, block, or signal name as the Simulink object name to include in the report. If the Model Loop component is the parent component, the object name is the current model name. If the System Loop, Block Loop, or Signal Loop is the parent component, then the object name is the name of the current system, block, or signal, respectively.
  - **Model:** Includes the current model name in the report.
  - **System:** Includes the current system name in the report.
  - **Block:** Includes the current block name in the report.
  - **Signal:** Includes the current signal name in the report. If the signal name is empty, the signal `<handle>`, which is a unique numerical identifier to that signal, appears in the report.
  - **Annotation:** Includes the current annotation name in the report.
- **Display name as:** Display the Simulink object name in the report.
  - **Name:** For example, f14
  - **Type Name:** For example, Model f14
  - **Type - Name:** For example, Model - f14
  - **Type: Name:** For example, Model: f14
- **Show full path name:** Displays the full path of a system or block. Choosing this option for a block causes the Simulink block name to appear as `<Model Name>/<System Name>/<Block Name>`.

---

**Note** This option is not available for models or signals.

---

### Insert Anything into Report?

Yes. Text.

### Class

rptgen\_sl.csl\_obj\_name

## **See Also**

Chapter/Subsection

## Simulink Property

Insert property name/property value pair for current Simulink model, system, block, or signal

### Description

This component inserts a single property name/property value pair for the current Simulink model, system, block, or signal.

### Simulink Object and Parameter

- **Object type:** Specifies the Simulink object type to include in the report.
  - System
  - Model
  - Block
  - Signal
  - **Annotation:** Reports annotation content as plain text for web (HTML), Acrobat (PDF), and Word Document (rtf) output types or as formatted text for other output types.
- **System parameter name:** Specifies a Simulink parameter name to include in the generated report:
  - If you select Model for **Object type**, this option appears as **Model parameter name**.
  - If you select Block for **Object type**, this option appears as **Block parameter name**.
  - If you select Signal for **Object type**, this option appears as **Signal parameter name**.
  - If you select Annotation for **Object type**, the **Signal parameter name** field does not appear.

### Display Options

These display options appear for every **Object type**, except Annotation.

- **Title:** Choose a title to display in the generated report:
  - **Automatic:** Uses the parameter name as the title.
  - **Custom:** Specifies a custom title.
  - **None:** Uses no title.
- **Array size limit:** Limits the width of the display in the generated report. Units are in pixels. The size limit for a given table is the hypotenuse of the table width and height [ $\sqrt{w^2+h^2}$ ]. The size limit for text is the number of characters squared. If you exceed the size limit, the variable appears in condensed form. Setting a size limit of 0 displays the variable in full, regardless of its size.
- **Object depth limit:** Specifies the maximum number of nesting levels to report on for a variable value
- **Object count limit:** Specifies the maximum number of nested objects to report on for a variable value



- **Display as:** Specifies a display style.
  - **Table or paragraph depending on data type:** Displays as a table or paragraph.
  - **Table:** Displays as a table.
  - **Paragraph:** Displays as a text paragraph.
  - **Inline text:** Displays inline with the surrounding text.
- **Show variable type in headings:** Show data type of this variable in the title of its report.
- **Show variable table grids:** Show grid lines for the table used to report the value of this variable.
- **Make variable table page wide:** Make the variable table as wide as the page on which the table appears.
- **Omit if value is empty:** Exclude empty parameters from the generated report.
- **Omit if property default value:** Exclude object property from the report if that property uses the default value.

## Insert Anything into Report?

Yes. Text.

## Class

`rptgen_sl.csl_property`

## See Also

Block Loop, Model Loop, System Loop

## Simulink Property Table

Insert table that reports on model-level property name/property value pairs

### Description

This component inserts a table that reports on model-level property name/property value pairs.

### Properties

**Select Object:** Choose the object for the Property Table in the generated report.

- Model
- System
- Block
- Signal
- Annotation

For more information about selecting object types in Property Table components, see “Select Object Types”.

### Table

Select a preset table, which is already formatted and set up, in the preset table list in the upper-left corner of the attributes page.

- **Preset table:** Specifies the type of the object property table.
  - Default
  - Simulation Parameters
  - Version Information
  - Simulink Coder Information
  - Summary (req. Simulink Coder)
  - Blank 4x4

To apply a preset table, select the table and click **Apply**.

- **Split property/value cells:** Split property name/property value pairs into separate cells. For the property name and property value to appear in adjacent horizontal cells, select the **Split property/value cells** check box. In this case, the table is in split mode, so there is only one property name/property value pair per cell. If there is more than one name/property pair in a cell, only the first pair appears in the report. All subsequent pairs are ignored.

For the property name and property value to appear together in one cell, clear the **Split property/value cells** check box. That setting specifies nonsplit mode. Nonsplit mode supports more than one property name/property value pair and text per cell.

Before switching from nonsplit mode to split mode, make sure that there is only one property name/property value pair per table cell. If there is more than one property name/property value

pair or text per cell, only the first property name/property value pair appears in the report. The report omits subsequent pairs and text.

- **Display outer border:** Displays the outer border of the table in the generated report.
- **Table Cells:** Specifies table properties to modify. The selection in this pane affects available fields in the **Cell Properties** pane.

## Cell Properties

The options in this pane depend on the object selected in the **Table Cells** pane. If you select %<Name> Information, only **Contents** and **Show** appear. If you select any other object in the **Table Cells** pane, **Lower border** and **Right border** display.

- **Contents:** Enables you to change the contents of the table cell selected in the **Table Cells** pane.
- **Show as:** Specifies the format for the contents of the table cell.
  - Value
  - Property Value
  - PROPERTY Value
  - Property: Value
  - PROPERTY: Value
  - Property - Value
  - PROPERTY - Value
- **Alignment:** Specifies the alignment of the contents of the selected table cell in the **Table Cells** pane.
  - Left
  - Center
  - Right
  - Double justified
- **Lower border:** Displays the lower border of the table in the generated report.
- **Right border:** Displays the right border of the table in the generated report.

## Creating Custom Tables

To create a custom table, edit a preset table, such as the Blank 4x4 table. Add and delete rows and add properties. To open the Edit Table dialog box, click **Edit**.

For more information on creating custom property tables, see “Property Table Components”.

If the Simulink Coder software is not installed, Summary (req Simulink Coder) does not appear in this list. If you are using a report setup file that contains a summary property, the property name appears in the report, but the property value does not.

## Example

The following report displays information on the f14 model using the **Simulation Parameters** preset table.

<i>Solver</i> ode45	<i>ZeroCross</i> on	<i>StartTime</i> 0.0 <i>StopTime</i> 60
<i>RelTol</i> 1e-4	<i>AbsTol</i> 1e-6	<i>Refine</i> 1
<i>InitialStep</i> auto	<i>FixedStep</i> auto	<i>MaxStep</i> auto
<i>LimitMaxRows</i> off	<i>MaxRows</i> 1000	<i>Decimation</i> 1

## **Insert Anything into Report?**

Yes. Table.

### **Class**

rptgen\_sl.csl\_prop\_table

### **See Also**

Model Loop, Signal Loop, System Loop

# Simulink Sample Time

Insert title of Simulink sample time into report

## Description

This component inserts a title for a Simulink sample time into the report.

## Properties

- **Table Options**
  - **Title:** Specifies a title for the table in the generated report.
  - **Grid lines:** Show grid lines for the table.

## Insert Anything into Report?

Yes. Table.

## Class

rptgen\_sl.CSampleTime

## See Also

Chapter/Subsection

## Simulink Summary Table

Properties or parameters of specified Simulink models, systems, blocks, or signals in table

### Description

This component displays properties or parameters of selected Simulink models, systems, blocks, or signals in a table.

### Object type

Choose the object type to display in the generated report.

- Block (Default)
- Model
- System
- Signal
- Annotation

The selected object type affects the options available in the **Property Columns** pane.

### Table title

Choose a title to appear in the generated report:

- Automatic: Automatically generates a title from the parameter.
- Custom: Specifies a custom title.

### Property Columns

This pane displays object properties to include in the Summary Table in the generated report.

- To add a property:
  - 1 Select the appropriate property level in the text box on the left.
  - 2 In the text box on the right, select the property that you want to add and click **Add**.
- To delete a property, select the property name and click **Delete**.

The **Parent** property column displays only the name of the parent system. To display the full path of the parent system, specify the property as **parent** with a lowercase **p**.

`%<SplitDialogParameters>` is a unique property for Simulink Summary Tables, where the object type is **Block**. This property generates multiple summary tables, organized by block type. Each Summary Table group contains the dialog box parameters for that block.

Some entries in the list of available properties (such as **Depth**) are “virtual” properties that you cannot access using the `get_param` command. The properties used for property/value filtering in the block and System Loop components must be retrievable by the `get_param`. Therefore, you cannot configure your Summary Table to report on all blocks of `Depth == 2`.

You can create multiple values for a property in a Simulink Summary Table. For example, to report on blocks of type Inport, Outport and Constant:

- 1 Check the **Search for Simulink property name/property value pairs** box.
- 2 Make sure that you set **Property Name** to BlockType.
- 3 Type the following text into the **Property Value** field:

```
\<(Inport|Outport|Constant)\>
```

**Remove empty columns:** Removes empty columns from the table.

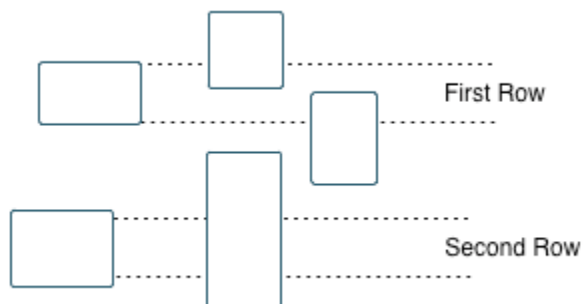
**Transpose table:** Changes the summary table rows into columns in the generated report, putting the property names in the first column and the values in the other columns.

## Object Rows

- **Insert anchor for each row:** Inserts an anchor for each row in the summary table.
- **Report On:**
  - **Automatic list from context:** Reports on all blocks in the current context, as set by the parent component.
  - **Custom - use block list:** Reports on a list of specified blocks. Specify the full path of each block.

## Loop Options

- **Sort blocks**
  - **Alphabetically by block name:** Sorts blocks alphabetically by name.
  - **Alphabetically by system name:** Sorts systems alphabetically by name. Lists blocks in each system, but in no particular order.
  - **Alphabetically by full Simulink path:** Sorts blocks alphabetically by Simulink path.
  - **By block type:** Sorts blocks alphabetically by block type.
  - **By block depth:** Sorts blocks by their depth in the model.
  - **By layout (left to right):** Sorts blocks by their location in the model layout, by rows. The block appearing the furthest toward the left top corner of the model is the anchor for the row. The row contains all other blocks that overlap the horizontal area defined by the top and bottom edges of the anchor block. The other rows use the same algorithm, using as the anchor the next unreported block nearest the left top of the model.



- **By layout (top to bottom):** Sorts blocks by their location in the model layout, by *columns*. The block appearing the furthest toward the left top corner of the model is the anchor for the column. The column contains all other blocks that overlap the vertical area defined by the left and right edges of the anchor block. The other columns use the same algorithm, using as the anchor the next unreported block nearest the left top of the model.
- **By traversal order:** Sorts blocks by traversal order.
- **By simulation order:** Sorts blocks by execution order.
- **%<VariableName>:** Inserts the value of a variable from the MATLAB workspace. The %<> notation can denote a string or cell array. For more information, see %<VariableName> Notation on the Text component reference page.
- **Search for Simulink property name/property value pairs:** Reports on blocks with specified property name/property value pairs.

### Example

Specify the following options to generate a Summary Table in a report for on the model f14:

- Sort on systems by system depth.
- Include the system parameters Name and Block in the table.

The following table appears in the report.

Name	Blocks
f14	u, Actuator Model, Aircraft Dynamics Model, Angle of Attack, Controller, Dryden Wind Gust Models, Gain, Gain1, Gain2, Gain5, More Info, More Info1, Nz pilot calculation, Pilot, Pilot G force Scope, Stick Input, Sum, Sum1, alpha (rad), Nz Pilot (g)
Aircraft Dynamics Model	Elevator Deflection d (deg), Vertical Gust wGust (ft/sec), Rotary Gust qGust (rad/sec), Gain3, Gain4, Gain5, Gain6, Sum1, Sum2, Transfer Fcn.1, Transfer Fcn.2, Vertical Velocity w (ft/s), Pitch Rate q (rad/s)
Controller	Stick Input (in), alpha (rad), q (rad/s), Alpha-sensor Low-pass Filter, Gain, Gain2, Gain3, Pitch Rate Lead Filter, Proportional plus integral compensator, Stick Prefilter, Sum, Sum1, Sum2, Elevator Command (deg)
Dryden Wind Gust Models	Band-Limited White Noise, Q-gust model, W-gust model, Wg, Qg
More Info	None
More Info1	None
Nz pilot calculation	w, q, Constant, Derivative, Derivative1, Gain1, Gain2, Product, Sum1, Pilot g force (g)

### Insert Anything into Report?

Yes. Table.

### Class

rptgen\_sl.csl\_summ\_table



**See Also**

Block Loop, Model Loop, Signal Loop, System Loop, Simulink Function System Loop

## Simulink Test Suite Traceability Table

Insert a table that links a Simulink test suite to requirements

### Description

This component inserts a table into the report. The table links a Simulink test suite to corresponding requirements. This component reports on the currently open Simulink test suite. Place this component inside a section, paragraph, or table component.

To use this component, your report setup must include Eval statements that open a Simulink test suite or determine the test suite that is open.

### Table Options

Specify information about the table this component inserts.

- **Table title:** Specify the table title.
  - `No title` — Do not include a table title.
  - `Object name` — Use the name of the Simulink test suite in the title.
  - `Custom` — Specify your own table title.

### Table Columns

Specify the table columns that you want to include in the report. The **Document name**, **Locations within document**, and **Requirement keyword** check boxes correspond to properties on the Requirements Management Interface Link Editor dialog box.

- **Description** — Include the description of the requirement. The description helps you to identify the requirement the table is linking to. Leave this box selected to improve the readability of your table.
- **Document name** — Include the name of the document where the requirement is located.
- **Locations within document** — Include the identifier of a location in the document.
- **Requirement keyword** — Include the requirement keyword.

### Insert Anything into Report?

Yes. Table.

### Class

`RptgenRMI.TMReqTable`

### See Also

Data Dictionary Traceability Table, MATLAB Code Traceability Table, `rmi`

# Simulink Workspace Variable

Report on workspace variables used in model, in loop generated by Simulink Workspace Variable Loop component

## Description

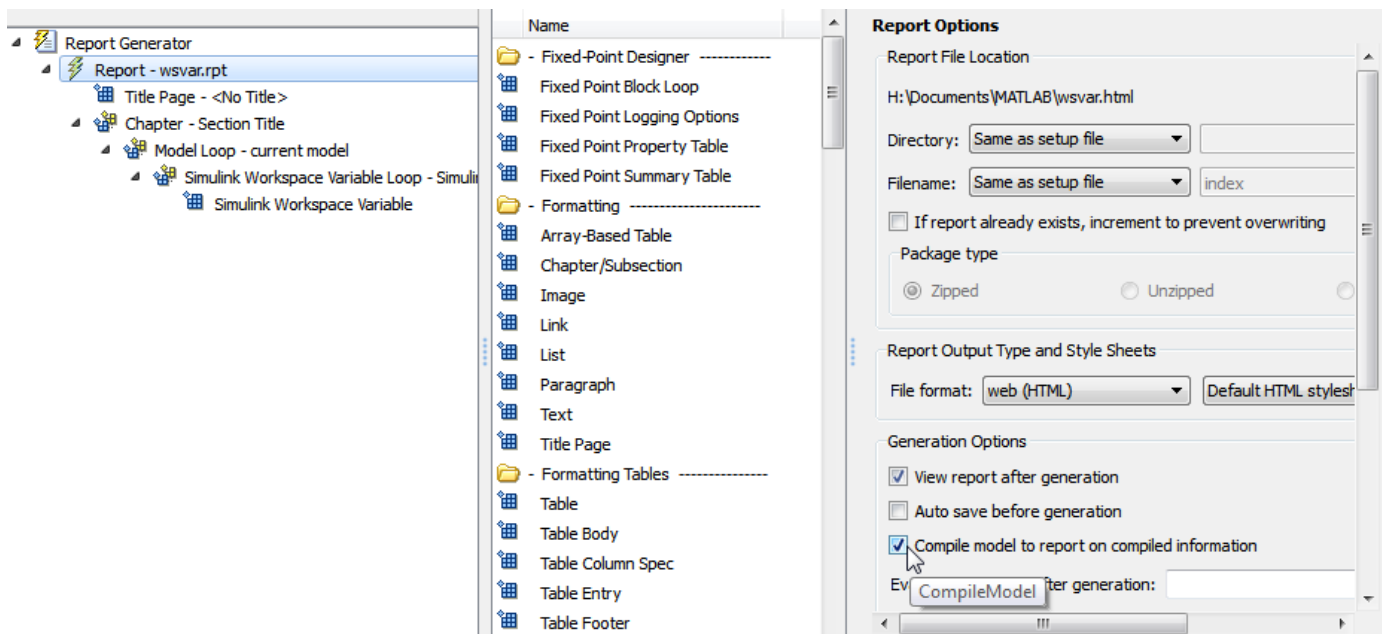
This component provides information about those workspace variables that the Simulink model uses, in a loop generated by a Simulink Workspace Variable Loop component. Your report setup must include Simulink Workspace Variable component as a child of a Simulink Workspace Variable Loop component.

The report includes the name and value each variable. Optionally, you can include the following information for each variable:

- Variable source (MATLAB workspace, model workspace, or data dictionary)
- Blocks that use the variable

For variables that are Simulink data objects (for example, a `Simulink.Parameter` object), the report includes the properties of the object. You can filter out properties to streamline the report.

Use a Simulink Workspace Variable Loop component as a parent for a Simulink Workspace Variable component. In the Report Options dialog box, select **Compile model to report on compiled information**.



## Options

The following options specify additional information that the report can include about each variable:

- **Show workspace:** Report the source of each variable — MATLAB workspace, model workspace, or data dictionary.
- **Show blocks that use variable:** Report the blocks that use each variable.

For variables whose values are Simulink data objects, you can filter the properties to include in the report, using one of the following approaches:

- Use the **Filter Properties** area of the dialog box to specify a standard filter.

The standard filter options apply to all variables whose values are instances of the class or classes that you specify. For example, you can use a standard filter to filter out the **Description** property for all variables used by the model whose values use a `Simulink.Parameter` object.

- Select the **Use custom property filter** option and write MATLAB code for filtering.

Writing custom filtering code allows you to do kinds of filtering that the standard filter does not perform. Some common examples of custom filters that you might want to create are filters that filter out:

- A property for some, but not all, instances of a class
- Properties that match a regular expression

The **Filter Properties** area of the dialog box, where you specify a standard filter, has these fields.

- **Class name (\* for all classes):** Specify the class of the variables for which you want to filter out specific properties. You can specify one class at a time, or enter an asterisk (\*) to specify all classes. After you enter the class name, move the cursor outside of the edit box.
- **Available Properties:** If the class that you entered in **Class name (\* for all classes)** is on the MATLAB path, then this list displays the properties of that class.
- **Filtered Properties:** Displays the properties to filter out. Use the right-arrow button to add to the **Filtered Properties** list the properties that you selected in the **Available Properties** list.
- If the class that you enter is *not* on the MATLAB path, then a **Comma-separated list of properties to be filtered** edit box appears. Enter the names of properties to use for filtering.
- **Convert to Custom:** Generate custom MATLAB code that implements your **Filter Properties** standard filter settings.

---

**Note** Selecting the **Convert to Custom** button overwrites any existing MATLAB custom filtering code for this component.

---

To create and apply custom filtering MATLAB code, select the **Use custom property filter** check box. Selecting this check box opens an edit box where you define a MATLAB function for filtering properties. The edit box includes a sample function (commented out) that you can use as a starting point for your filtering function. Use the `isFiltered` variable for the output of your function. For example:

- To filter out the `Owner` and `testProp` properties, in the edit box enter:

```
isFiltered = strcmpi(propertyName, 'Owner') || ...
            strcmpi(propertyName, 'testProp');
```

- To filter out all properties *except* for the `CoderInfo` property, in the edit box, enter:

```
isFiltered = ~strcmpi(propertyName, 'CoderInfo');
```

If you clear the **Use custom property filter** check box, Simulink Report Generator saves your custom MATLAB filtering code, but does not use that code to filter properties.

## **Insert Anything into Report?**

Yes. List.

### **Class**

rptgen\_sl.csl\_ws\_variable

### **See Also**

Simulink Workspace Variable Loop, Bus, Simulink Functions and Variables

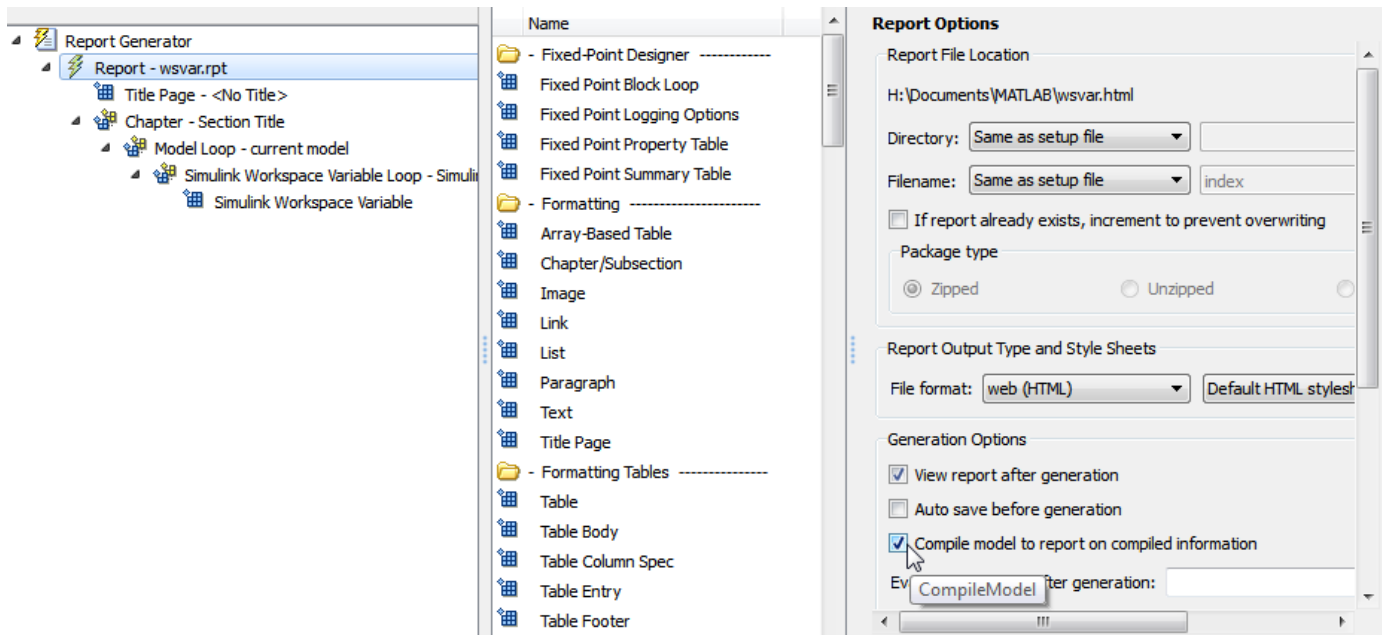
## Simulink Workspace Variable Loop

Generates a model variable loop

### Description

This component generates a model variable loop used by the Simulink Workspace Variable component to report on those workspace variables that the Simulink model uses.

You can limit the variables included in the loop to those that match property name and value pairs that you specify. If you want to report on model variables, your report setup file must include this component as a child of a Model Loop component and must include a Simulink Workspace Variable component as its child. Also, in the Report Options dialog box, select **Compile model to report on compiled information**. For example:



### Loop Options

- **Sort**
  - Alphabetically by text: Sort variables alphabetically by name.
  - By data type: Sort variables alphabetically by data type.
- **Search for Simulink property name/property value pairs:** Reports on variables with specified property name/property value pairs.

### Section Options

- **Create section for each object in loop:** Creates a separate section in the output for each variable.

- If you specify to create a section for each variable, you can select the **Display the object type in the section title** to insert a variable name in each section title.
- **Create link anchor for each object in loop:** Create a link target for each workspace variable in the loop so that other parts of the report can link to it.

## Insert Anything into Report?

Yes, inserts a section if you select **Create section for each object in loop** and a link target if you select **Create link anchor for each object in loop**.

## Class

rptgen\_sl.csl\_ws\_var\_loop

## See Also

Simulink Workspace Variable, Bus, Simulink Functions and Variables

## State Loop

Run child components for all states in current context

### Description

This component runs its children for all states in its context. The parent component of this component determines the context.

- **Model Loop:** Includes all states in the models.
- **System Loop:** Includes all states in the systems.
- **Machine Loop:** Includes all states in the machines.
- **Chart Loop:** Includes all states in the charts.
- **State Loop:** Includes all states in the current state.

For conditional processing based on states, you can use the `RptgenSF.getReportedState` function. For more information, see “Loop Context Functions” on page 4-89.

### State Types

- **Include “and” and “or” states:** Includes AND and OR states in the loop.
- **Include “box” states:** Includes “box” states in the loop.
- **Include functions:** Includes “function” states in the loop.
- **Include truth tables:** Includes truth tables in the loop.
- **Include MATLAB functions:** Includes MATLAB functions in the loop.

### Loop Options

- **Report depth:** Specifies the level on which to loop.
  - Local children only
  - All objects
- **Skip autogenerated charts under truth table:** Keeps autogenerated state objects under truth tables from appearing in the report.
- **Search Stateflow:** Indicates specific states to include in the loop.

### Section Options

- **Create section for each object in loop:** Inserts a section in the generated report for each object found in the loop.
- **Display the object type in the section title:** Inserts the object type automatically into the section title in the generated report.
- **Create link anchor for each object in loop:** Create a link target for each state in the loop so that other parts of the report can link to it. For example, the image created by a `Stateflow Snapshot` component can link to the chart information only if you select this check box.



## Insert Anything into Report?

Yes, inserts a section if you select **Create section for each object in loop** and a link target if you select **Create link anchor for each object in loop**.

### Class

rptgen\_sf.csf\_state\_loop

### See Also

Chart Loop, Machine Loop, Model Loop, System Loop, Simulink Function System Loop

## State Transition Matrix

Inserts state transition matrix contents into report

### Description

This component inserts the contents of state transition matrices into a report. A state transition matrix is an alternative view of a state transition table. In the state transition matrix, you can easily see how the state transition table reacts to each condition and event.

### Options

- **Title**
  - `No title` (default): Report uses no title for the state transition matrix.
  - `Use Stateflow name`: For the title in the report, uses the names of the State Transition Table blocks from which the state transition matrices are generated.
  - `Custom`: In the text field, specify a custom name for the state transition matrix.
- **Display condition actions on matrix**: Include the state transition matrix condition actions. A condition action is an action that executes as soon as a condition evaluates to true. The condition action is part of a transition label.

### Insert Anything into Report?

Yes, inserts state transition matrices and optionally, condition actions.

### Class

`rptgen_sf_csf_statetransitionmatrix`

### See Also

State Transition Table

# State Transition Table

Inserts state transition tables into report

## Description

This component inserts the state transition tables into a report. A state transition table is an alternative way of expressing sequential modal logic. Instead of drawing states and transitions graphically in a Stateflow® chart, you express the modal logic in tabular format.

## Options

- **Title**
  - `No title` (default): Report uses no title for the state transition table.
  - `Use Stateflow name`: Uses the name of the State Transition Table block as the title.
  - `Custom`: In the text field, specify a custom name for the state transition table.

## Insert Anything into Report?

Yes, inserts state transition table.

## Class

`rptgen_sf_csf_statetransitiontable`

## See Also

State Transition Matrix

## Stateflow Automatic Table

Insert table with properties of current Stateflow object

### Description

This component inserts a table that contains the properties of the current Stateflow object. Parents of this component can be:

- Machine Loop
- State Loop
- Chart Loop
- Graphics Object Loop

### Display Options

- **Table title:** Specifies a title for the table in the generated report.
  - No title: Includes no title.
  - Custom: Includes a custom title.
  - Name (default): Uses an object name as the title.
    - Object name
    - Object name with Stateflow path
    - Object name with Simulink and Stateflow path
- **Header row:** Selects a header row for the table in the generated report.
  - No header: Includes no header row.
  - Type and Name: Includes a header row with columns for name and object type. When selected, this option creates a header row for the table with object name and type.
  - Custom: Includes a custom header.
- **Don't display empty values:** Excludes empty values from the generated report.

### Insert Anything into Report?

Yes. Table.

### Class

`rptgen_sf.csf_auto_table`

### See Also

Chart Loop, Graphics Object Loop, Machine Loop, State Loop

# Stateflow Count

Count number of Stateflow objects in current context

## Description

This component counts the number of Stateflow objects in the current context.

## Properties

- **Search depth:** Specifies the search depth for the count.
  - **Immediate children only (default):** Searches only children one level under the Stateflow object.
  - **All descendants:** Searches all children of the Stateflow object.
- **Sort results:** Specifies the sort method for the count results.
  - **Numerically decreasing by object count (Default)**
  - **Alphabetically increasing by object type**
- **Include a list of objects in table:** Inserts a column containing the counted objects.
- **Show total count:** Displays a total of counted objects.

## Insert Anything into Report?

Yes. Table.

## Class

`rptgen_sf.csf_count`

## See Also

State Loop

# Stateflow Dialog Snapshot

Insert snapshots of Stateflow editor dialog boxes

## Description

This component reports on the current reported Stateflow dialog box object, depending on its context. If this component is the child of a `State Loop`, for example, the report includes information about the dialog box of the current State. Display the current settings associated with an object or document the appearance of your custom mask dialog boxes.

## Format

- **Image file format:** Specifies the format for the snapshot image file. The `Automatic` format uses BMP format for PDF files and PNG for other formats.
- **Show all tabs:** Automatically generates images for all the tabs for the dialog box. If you clear this check box, the Simulink Report Generator software creates an image of only the first tab.

## Display Options

- **Scaling:** Controls size of the image, as displayed in a browser. Making an image larger using this option does not affect the storage size of the image, but the quality of the displayed image may decrease as you increase or decrease the size of the displayed image.

Generally, to achieve the best and most predictable display results, use the default setting of `Use image size`.

- `Use image size:` Causes the image to appear the same size in the report as on screen (default).
- `Fixed size:` Specifies the number and type of units.
- `Zoom:` Specifies the percentage, maximum size, and units of measure.
- **Size:** Specifies the size of the snapshot in the form `w h` (width, height). This field is active only if you choose `Fixed size` in the **Scaling** selection list.
- **Max size:** Specifies the maximum size of the snapshot in the form `w h` (width, height). This field is active only if you choose `Zoom` in the **Scaling** selection list.
- **Units:** Specifies the units for the size of the snapshot. This field is active only if you choose `Zoom` or `Fixed size` in the **Image size** list box.
- **Alignment:** Aligns your snapshot. Only reports in PDF or RTF format support this property.
  - Auto
  - Right
  - Center
  - Left
- **Title:** Specifies text to appear above the snapshot.
- **Caption:** Specifies text to appear under the snapshot.

**Insert Anything into Report?**

Yes. Snapshot.

**Class**

rptgen\_sl.Cdialog boxesnapshot

**See Also**

State Loop

## Stateflow Filter

Run child components only if current object type matches specified object type

### Description

This component runs its children only if the current object type, as set by its parent Stateflow Hierarchy Loop, matches the selected object type.

### Properties

- **Object type:** Specifies the Stateflow object type to include in the report.
- **Run only if Stateflow object has at least the following number of Stateflow children:** Specifies a minimum number of children that a Stateflow object must have to include in the report.
- **Automatically insert linking anchor:** Inserts a linking anchor before the reported object. If an anchor for this object exists, this option does not create a second anchor.

### Insert Anything into Report?

No.

### Class

rptgen\_sf.csf\_obj\_filter

### See Also

Stateflow Hierarchy Loop



# Stateflow Hierarchy

Provide visual representation of the hierarchy of a Stateflow object

## Description

This component inserts a tree that shows the hierarchy of a given Stateflow object.

## Tree Options

- **Construct tree from:** Specifies the object to use for the tree representation.
  - Current object
  - Root of current object: Starts reporting from the top of the hierarchy.
- **Emphasize current object in tree:** Highlights the current object in the tree representation.
- **Show number of parents:** Specifies the number of parents to include in the tree representation.
- **Show siblings:** Displays siblings in the tree representation.
- **Show children to depth:** Specifies the depth of children to display for each object in the tree representation.

## Children

- **Show junctions:** Specifies the level of junction detail to display in the generated report.
  - All
  - Non-redundant
  - None
- **Show transitions:** Specifies the level of transition detail to display in the generated report.
  - All
  - Labeled or non-redundant
  - Non-redundant
  - Labeled
  - None
- **Skip autogenerated charts under truth tables:** Excludes autogenerated charts under truth tables.

## List Formatting

- **List style:**
  - Bulleted list
  - Numbered list: Allows you to specify numbering options in the **Numbering style** section.
    - **Numbering style:** Allows you to specify a numbering style. This setting supports only the RTF/DOC report format.

- 1,2,3,4...
- a,b,c,d...
- A,B,C,D...
- i,ii,iii,iv...
- I,II,III,IV...

To show the parent number in each list entry, select `Show parent number in nested list (1.1.a)`. To show only the current number or letter, select `Show only current list value (a)`.

### **Insert Anything into Report?**

Yes. Tree graphic.

### **Class**

`rptgen_sf.csf_hier`

### **See Also**

Stateflow Hierarchy Loop

# Stateflow Hierarchy Loop

Run child components on Stateflow object hierarchy

## Description

This component runs its child components on the Stateflow object hierarchy.

## Loop Options

- **Minimum legible font size:** Specifies the minimum font size to use in the report. The default font size, 8, is the smallest recommended font size.
- **Skip autogenerated charts under truth tables;** Excludes autogenerated charts under truth tables in the report.
- **Search Stateflow:** Reports on Stateflow charts with specified property name/property value pairs.

## Section Options

- **Create section for each object in loop:** Inserts a section in the generated report for each object found in the loop.
- **Display the object type in the section title:** Inserts the object type automatically into the section title in the generated report.
- **Create link anchor for each object in loop:** Create a link target on each object in the loop so that other parts of the report can link to it.

## Insert Anything into Report?

No.

## Class

rptgen\_sf.csf\_hier\_loop

## See Also

Stateflow Hierarchy

## Stateflow Linking Anchor

Designate locations to which links point

### Description

This component designates a location to which other links point. The linking anchor is set to the current object, as defined by the parent component.

This component must have the `Chart Loop`, `State Loop`, `Machine Loop`, or `Stateflow Filter` component as its parent.

### Properties

**Insert text:** Specifies text to appear after the linking anchor.

### Insert Anything into Report?

Yes. A link, and possibly text, depending on attribute choices.

### Class

`rptgen_sf.csf_obj_anchor`

### See Also

`Chart Loop`, `Machine Loop`, `State Loop`, `Stateflow Filter`,

# Stateflow Name

Insert into report name of Stateflow object specified by parent component

## Description

This component inserts the name of the Stateflow object, as defined by its parent component, into the report. This component must have the `State Loop`, `Chart Loop`, or `Stateflow Filter` component as its parent.

Using this component as the first child component of a `Chapter/Subsection` component allows the current Stateflow object name to be the chapter or section title.

## Properties

- **Display name as:** Displays the Stateflow object name in the report.
  - Name: For example, `Object`
  - Type Name: For example, `Object <ObjectName>`
  - Type - Name: For example, `Object - <ObjectName>`
  - Type: Name: For example, `Object: <ObjectName>`
- **Display name as:** Specifies the level of detail with which the Stateflow object name displays the report.
  - Object name
  - Object name with Stateflow path
  - Object name with Simulink and Stateflow path

## Insert Anything into Report?

Yes. Text.

## Class

`rptgen_sf.csf_obj_name`

## See Also

`Chapter/Subsection`, `Chart Loop`, `State Loop`, `Stateflow Filter`

## Stateflow Property

Insert into report table, text, or paragraph with information on selected Stateflow object property

### Description

This component inserts a table, text, or paragraph that contains details of the selected Stateflow object property.

### Property to Display

**Property name:** Specifies the Stateflow property name to display. If the Stateflow object is an annotation, the **Display options** are ignored. Annotation content is reported as plain text for web (HTML), Acrobat (PDF), and Word Document (rtf) output types or as formatted text for other output types.

### Display Options

- **Title:** Specifies a title to display in the generated report.
  - **Automatic:** Uses the parameter name as the title.
  - **Custom:** Specifies a custom title.
  - **None:** Specifies no title.
- **Size limit:** Specifies the width of the display in the generated report. Units are in pixels. The size limit for a given table is the hypotenuse of the width and height of the table,  $\sqrt{w^2+h^2}$ . The size limit for text is the number of characters squared. If you exceed the size limit, the variable appears in condensed form.

Setting a size limit of 0 always displays the variable in long form, regardless of its size.

- **Display as:** Specifies a display style from the menu.
  - **Auto table/paragraph (default):** Displays as a table or paragraph based on the information.
  - **Table:** Displays as a table.
  - **Paragraph:** Displays as a text paragraph.
  - **Inline text:** Displays in line with the surrounding text.
- **Ignore if value is empty:** Excludes empty parameters from the generated report.

### Insert Anything into Report?

Yes. Text, paragraph, or table.

### Class

rptgen\_sf.csf\_property

## **See Also**

Paragraph, Table, Text, Stateflow Name

## Stateflow Property Table

Insert into report property-value table for Stateflow object

### Description

This component inserts a property-value table for a Stateflow object into the report. Use the `Stateflow Filter` component as the parent of this component.

For more information on working with Property Table components, see “Property Table Components”.

### Table

Select a preset table, which is already formatted and set up, in the preset table list in the upper-left corner of the attributes page.

- **Preset table:** Specifies a type of table to display the object property table.
  - Default
  - Machine
  - Chart
  - State
  - Truth table
  - EM function
  - Data
  - Event
  - Junction

To apply a preset table, select the table and click **Apply**.

- **Split property/value cells:** Splits property name/property value pairs into separate cells.
  - For the property name and property value to appear in adjacent horizontal cells, select the **Split property/value cells** check box. In this case, the table is in split mode, there is only one property name/property value pair per cell. If there is more than one name/property pair in a cell, only the first pair appears in the report. The report ignores all subsequent pairs.
  - For the property name and property value to appear together in one cell, clear the **Split property/value cells** check box. This setting is nonsplit mode. Nonsplit mode supports more than one property name/property value pair and text.
  - Before switching from nonsplit mode to split mode, make sure that there is only one property name/property value pair per table cell. When there is more than one property name/property value pair or any text in a given cell, only the first property name/property value pair appears in the report. The report omits subsequent pairs and text.
- **Display outer border:** Displays the outer border of the table in the generated report.
- **Table Cells:** Specifies table properties to modify. The selection in this pane affects the available fields in the **Cell Properties** pane.



## Cell Properties

The options in the **Title Properties** pane depend on the object selected in the **Table Cells** pane. If you select %<Name>, only **Contents** and **Show** appear. If you select any other object in the **Table Cells** pane, **Lower border** and **Right border** appear.

- **Contents:** Modifies the contents of the table cell selected in the **Table Cells** pane.
- **Alignment:** Justifies the contents of the selected table cell in the **Table Cells** pane.
  - Left
  - Center
  - Right
  - Double justified
- **Show As:** Specifies the format for the contents of the table cell.
  - Value
  - Property Value
  - PROPERTY Value
  - Property: Value
  - PROPERTY: Value
  - Property - Value
  - PROPERTY - Value
- **Lower border:** Displays the lower border of the table in the generated report.
- **Right border:** Displays the right border of the table in the generated report.

### Creating Custom Tables

You can edit a preset table, such as the Blank 4x4 table, to create a custom table. Add and delete rows and add properties. To open the Edit Table dialog box, click **Edit**.

For details about creating custom property tables, see “Property Table Components”.

## Insert Anything into Report?

Yes. Table.

### Class

rptgen\_sf.csf\_prop\_table

### See Also

Stateflow Filter

## Stateflow Snapshot

Insert into report snapshot of Stateflow object

### Description

This component inserts a snapshot (screen capture) of a Stateflow object, defined by the Stateflow Filter parent component, into a report.

This component executes only if the selected object in the Stateflow Filter component is a graphical object, such as Chart, State, Transition, or Frame.

For HTML and Direct PDF (from template) output, the state charts in the resulting image can link to associated report information. To enable this linking, on the Chart Loop component that this component is a descendant of, select the **Create link anchor for each object in loop** check box. For Direct PDF (from template) output, you also need to set the output format to Automatic SF format or Scalable Vector Graphics.

### Snapshot

- **Format:** Specifies the image file format. Select Automatic SF Format (default) to choose the format best suited for the specified report output format automatically. Otherwise, choose an image format that your output viewer can read.
  - Automatic SF Format — Select this option, or Scalable Vector Graphics, with Direct PDF (from template) report output format if you are linking the snapshot to related report content.
  - Bitmap
  - JPEG high quality image
  - JPEG low quality image
  - JPEG medium quality image
  - PNG 24-bit image
  - Scalable Vector Graphics
- **Paper orientation:**
  - Portrait
  - Landscape
  - Rotated
  - Largest dimension vertical: Positions the image so that its largest dimension is vertical.
  - Use Chart PaperOrientation setting: Uses the paper orientation setting for the chart. Use the Simulink PaperOrientation parameter to specify the orientation.
  - Full page image (PDF only): In PDF reports, scales images to fit the full page, minimizes page margins, and maximizes the size of the image by using either a portrait or landscape orientation.

For more information about paper orientation, see the `orient` command in the MATLAB documentation.

- **Image sizing:**
  - **Shrink image to minimum font size specified in Stateflow Hierarchy Loop:** Resizes the image so that the text label font size is the minimum font size.
  - **Fixed and Zoom:** Specifies the size of the image.
- **Scaling:** Specifies the percentage of the image size to which to scale it.
- **Maximum size:** Specifies the maximum size for the snapshot in the generated report in the selected units. Use [width, height] format. In the units text box, select Inches, Centimeters, Points, or Normalized.
- **Use printframe:** Inserts a frame around your image. Use the default frame or create a custom one.
- **Use printframe paper settings:** Uses the dimensions and parameters as set by the specified **printframe** to size your image. If you choose this option, all other options (except for **Image file format**) become inactive.

## Properties

- **Include callouts to describe visible objects:** Displays descriptive callouts for visible objects.
- **Insert anchors for transitions and junctions:** Inserts anchors for transitions and junctions into the report.
  - None
  - Redundant children only
  - All
- **Run only if Stateflow object has at least the following number of children:** Specifies the minimum number of children that the current Stateflow object must have to include in the report. This option is inactive unless the selected object in the parent Stateflow Filter component is a graphical object.

---

**Tip** This option allows you to exclude certain images to decrease the size of the report for large models.

---

## Display Options

- **Scaling:**
  - **Use image size:** Uses the image size that you specify in the snapshot option.
  - **Zoom and Fixed size:** Allows you to specify the size of the image.
- **Size:** Specifies a size in inches for your image. The default is 7-by-9.
- **Max size:** Specifies the maximum size of the snapshot in the format w h (width, height). This field is active only if you choose Zoom from the **Scaling** selection list.
- **Units:** Specifies the units for the size of the snapshot. This field is active only if you choose Zoom or Fixed size in the **Image size** list box.
- **Alignment:** Only reports in PDF or RTF format support this property.
  - Auto

- Right
- Center
- Left
- **Image title:**
  - None(Default).
  - Object name: Uses the object name as the title.
  - Full Stateflow name: Specifies the Stateflow path and the name of the object.
  - Full Simulink + Stateflow name: Specifies the Simulink path and name of the object.
  - Custom: Enter a different title.
- **Caption:** Specifies a caption for your image.
  - None(Default).
  - Custom: Specifies a custom caption.
  - Description: Sets the caption to the value of the object Description property.

## Insert Anything into Report?

Yes. Image.

### Class

rptgen\_sf.csf\_obj\_snap

### Class

rptgen\_sf.csf\_prop\_table

### See Also

Stateflow Filter

# Stateflow Summary Table

Table of properties or parameters of specified Stateflow object

## Description

This component displays a table of properties or parameters of specified Stateflow objects. It can have the following parents:

- Any Stateflow looping component
- Any Simulink looping component (Model Loop, System Loop, Block Loop, or Signal Loop)

## Properties

- **Object type:** Specifies the object type to display in the generated report. This value affects the options available in the **Property Columns** pane.
- **Table title:** Specifies a title for the Summary Table in the generated report.
  - **Automatic:** Generates a title automatically from the parameter.
  - **Custom:** Specifies a custom title.

## Property Columns

- **Property columns:** Displays the object properties to include in the Summary Table in the generated report.
  - To add a property:
    - Select the appropriate property level in the text box.
    - In the context list under the text box, select the property that you want to add and click **Add**.
  - To delete a property, select the property name and press the **Delete** key.

Some entries in the list of available properties (such as **Depth**) are “virtual” properties that you cannot access using the `get_param` command. The properties used for property/value filtering in the block and System Loop components must be retrievable by the `get_param`. Therefore, you cannot configure your Summary Table to report on all blocks of `Depth == 2`.

- **Remove empty columns:** Removes empty columns from the Summary Table in the generated report.
- **Transpose table:** Changes the summary table rows into columns in the generated report, putting the property names in the first column and the values in the other columns.

## Object Rows

**Insert anchor for each row:** Inserts an anchor for each row in the summary table.

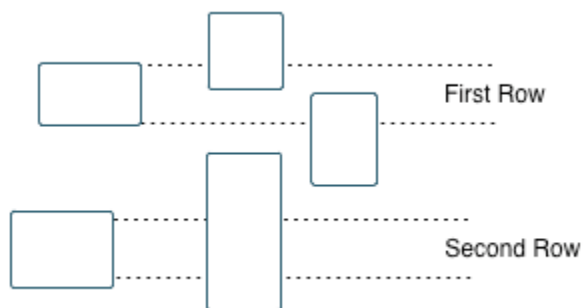
## Report On

- **Automatic list from context:** Reports on all blocks in the current context, as set by the parent component.
- **Custom - use block list:** Reports on a specified list of blocks. Specify the full path of each block.

## Loop Options

Choose block sorting options and reporting options in this pane.

- **Sort blocks:** Specifies how to sort blocks (applied to each level in a model):
  - **Alphabetically by block name:** Sorts blocks alphabetically by name.
  - **Alphabetically by system name:** Sorts systems alphabetically by name. Lists blocks in each system, but in no particular order.
  - **Alphabetically by full Simulink path:** Sorts blocks alphabetically by Simulink path.
  - **By block type:** Sorts blocks alphabetically by block type.
  - **By block depth:** Sorts blocks by their depth in the model.
  - **By layout (left to right):** Sorts blocks by their location in the model layout, by *rows*. The block appearing the furthest toward the left top corner of the model is the anchor for the row. The row contains all other blocks that overlap the horizontal area defined by the top and bottom edges of the anchor block. The other rows use the same algorithm, using as the anchor the next unreported block nearest the left top of the model.



- **By layout (top to bottom):** Sorts blocks by their location in the model layout, by *columns*. The block appearing the furthest toward the left top corner of the model is the anchor for the column. The column contains all other blocks that overlap the vertical area defined by the left and right edges of the anchor block. The other columns use the same algorithm, using as the anchor the next unreported block nearest the left top of the model.
- **By traversal order:** Sorts blocks by traversal order.
- **By simulation order:** Sorts blocks by execution order.
- **Search for Simulink property name/property value pairs:** Reports on blocks with specified property name/property value pairs.
- **Search Stateflow:** Reports on Stateflow charts with specified property name/property value pairs.

## Insert Anything into Report?

Yes. Table.

### Class

rptgen\_sf.csf\_summ\_table

### See Also

Block Loop, Chart Loop, Model Loop, Object Loop, Signal Loop, State Loop, Stateflow Hierarchy Loop, System Loop

## System Filter

Run child components if current system meets specified conditions

### Description

This component runs its child components if the current system meets the conditions that you specify with this component.

### Properties

- **Report only if system has at least N blocks:** Specifies the minimum number of blocks that the system must include for any of the child components to run. If you enter 0, child components run regardless of the number of blocks in the system.
- **Report only if system has at least N subsystems:** Specifies the minimum number of subsystems that the system must include for the child components to run. If you enter 0, child components run regardless of the number of subsystems in the system.
- **Report only if system mask type is:** Specifies which masks to include in the generated report.
  - Either masked or unmasked
  - Masked
  - Unmasked
- **Custom filtering MATLAB code:** Specifies custom MATLAB filtering code that the System Filter applies when determining which systems and subsystems to report on in a System Loop component. The edit box includes a sample function (commented out) that you can use as a starting point for your own filtering function. Use the `isFiltered` variable for the output of your function. For example, to filter out systems and subsystems whose names start with `engine`, enter:

```
isFiltered = strncmpi( currentSystem, 'engine', 6);
```

### Insert Anything into Report?

No.

### Class

`rptgen_sf.csf_obj_filter`

### See Also

System Loop



# System Hierarchy

Create nested list that shows hierarchy of specified system

## Description

This component creates a nested list that shows the hierarchy of a specified system. The list can display all systems in a model, or the parents and children of the current system.

## Starting System

- **Build list from:** Specifies the system or model from which to build the list.
  - Current system
  - Current model
- **Emphasize current system:** Highlights the current system or model in the generated report.

## Display Systems

- **Show number of parents:** Specifies the number of parents to list.
- **Display peers of current system:** Shows the peers of the current system in the generated report.
- **Show children to depth:** Specifies the depth of children to list.

## List Formatting

- **List style:**
  - Bulleted list
  - Numbered list: Allows you to select numbering options in the **Numbering style** section.
- **Numbering style:** Allows you to select a numbering style in the selection list, by setting **List style** to Numbered list. Only the RTF/DOC report format supports this option.
  - 1,2,3,4,...
  - a,b,c,d,...
  - A,B,C,D,...
  - i,ii,iii,iv,...
  - I,II,III,IV,...

## Insert Anything into Report?

Yes. List.

## Class

rptgen\_sl.csl\_sys\_list

**See Also**

Model Loop, System Loop

# System Loop

Specify systems and subsystems on which to loop, as defined by parent component

## Description

This component runs its child components for each system defined by the parent component. For example, to include systems and subsystems within a given model in the report, you can include this component as the child of a `Model Loop` component.

For conditional processing systems, you can use the `RptgenSL.getReportedSystem` function. For more information, see “Loop Context Functions” on page 4-89.

## Report On

- **Loop on Systems:**
  - **Select systems automatically:** Reports on all systems in the current context as set by the parent component.
    - `Model Loop`: Reports on systems in the current model.
    - `System Loop`: Reports on the current system.
    - `Signal Loop`: Reports on the parent system of the current signal.
    - `Block Loop`: Reports on the parent system of the current block.

If this component does not have any of these components as its parent, selecting this option reports on all systems in all models.
- **Custom - use system list:** Reports on a list of specified systems. Specify the full path of each system.
- **%<VariableName>:** Inserts the value of a variable from the MATLAB workspace. The %<> notation can denote a string or cell array. For more information, see %<VariableName> Notation on the Text component reference page.
- **Include subsystems in Simulink functions:** Specifies whether to include subsystems in Simulink functions. By default, this option is enabled.

## Loop Options

- **Sort Systems:** Specifies how to sort systems.
  - **Alphabetically by system name (default):** Sorts systems alphabetically by name.
  - **By number of blocks in system:** Sorts systems by number of blocks. The list shows systems by decreasing number of blocks; that is, the system with the largest number of blocks appears first in the list.
  - **By system depth:** Sorts systems by their depth in the model.
  - **By traversal order:** Sorts systems in traversal order.
- **Search for:** Reports only on blocks with the specified property name/property value pairs. To enable searching, click the check box. In the first row of the property name and property value

table, click inside the edit box, delete the existing text, and type the property name and value. To add a row, use the **Add row** button.

For information about subsystem property names and values, in “Block-Specific Parameters”, see the “Ports & Subsystems Library Block Parameters” section.

## Section Options

- **Create section for each object in loop:** Inserts a section in the generated report for each object found in the loop.
- **Display the object type in the section title:** Inserts the object type automatically into the section title in the generated report.
- **Number sections by system hierarchy:** Hierarchically numbers sections in the generated report. Requires that **Sort Systems** be set to `By traversal order`.
- **Create link anchor for each object in loop:** Create a link target for each system in the loop so that other parts of the report can link to it. For example, the image created by a `System Snapshot` component can link to the subsystem section only if you select this check box.

## Examples

For an example of how to use this component with a `Model Loop` as its parent, see `Model Loop`.

## Insert Anything into Report?

Yes, inserts a section if you select **Create section for each object in loop** and a link target if you select **Create link anchor for each object in loop**.

## Class

`rptgen_sl.csl_sys_loop`

## See Also

`Block Loop`, `Model Loop`, `Signal Loop`

# System Snapshot

Insert snapshot of the current system into report

## Description

This component inserts a snapshot (screen capture) of the current system into a report. A **System Snapshot** must be a child or descendant of a **System Loop** component.

For HTML and **Direct PDF (from template)** output, the blocks and subsystems in the resulting image can link to associated report information. For example, a block in the report can link to a block property table. A subsystem can link to the subsystem block diagram or to the block properties.

---

**Note** For PDF reports in which you want to include hyperlinks in system snapshots, use **Direct PDF (from template)** file format. If you use **Acrobat (PDF)** format, snapshots do not include hyperlinks.

---

To enable this linking, select the **Create link anchor for each object in loop** check box on the appropriate loop component:

- For blocks and masked subsystems in this system, select the check box on the **Block Loop** component that reports on the system's blocks.
- To provide links to the sections that report on the unmasked subsystems of this system, select the check box on the **System Loop** component.

For **Direct PDF (from template)** output, you also need to set the output format to **Automatic SL** format or **Scalable Vector Graphics**.

## Snapshot Options

- **Format:** Specifies the image file format. Select **Automatic SL Format** (the default) to choose the format best suited for the specified report output format automatically. Otherwise, choose an image format that your output viewer can read.
  - **Automatic SL Format** — Select this option. or **Scalable Vector Graphics**, with **Direct PDF (from template)** report output format if you are linking the snapshot to related report content.
  - **Bitmap**
  - **JPEG high quality image**
  - **JPEG low quality image**
  - **JPEG medium quality image**
  - **PNG 24-bit image**
  - **Scalable Vector Graphics**
- **Orientation:**
  - **Largest dimension vertical:** Positions the image so that its largest dimension is vertical.

- Landscape
- Portrait
- **Use system orientation:** Uses the paper orientation setting for the system. Use the Simulink `PaperOrientation` parameter to specify the orientation.
- **Full page image (PDF only):** In PDF reports, scales images to fit the full page, minimizes page margins, and maximizes the size of the image by using either a portrait or landscape orientation.
- **Scaling:** Controls the size of the image in the image file.
  - **Automatic (default):** Automatically scales the image to output dimensions.
  - **Custom:** Specifies image size.
  - **Zoom:** Enlarges or reduces the image size to the percent that you specify. Use **Max Size** to specify the maximum size other than the default for the image.

---

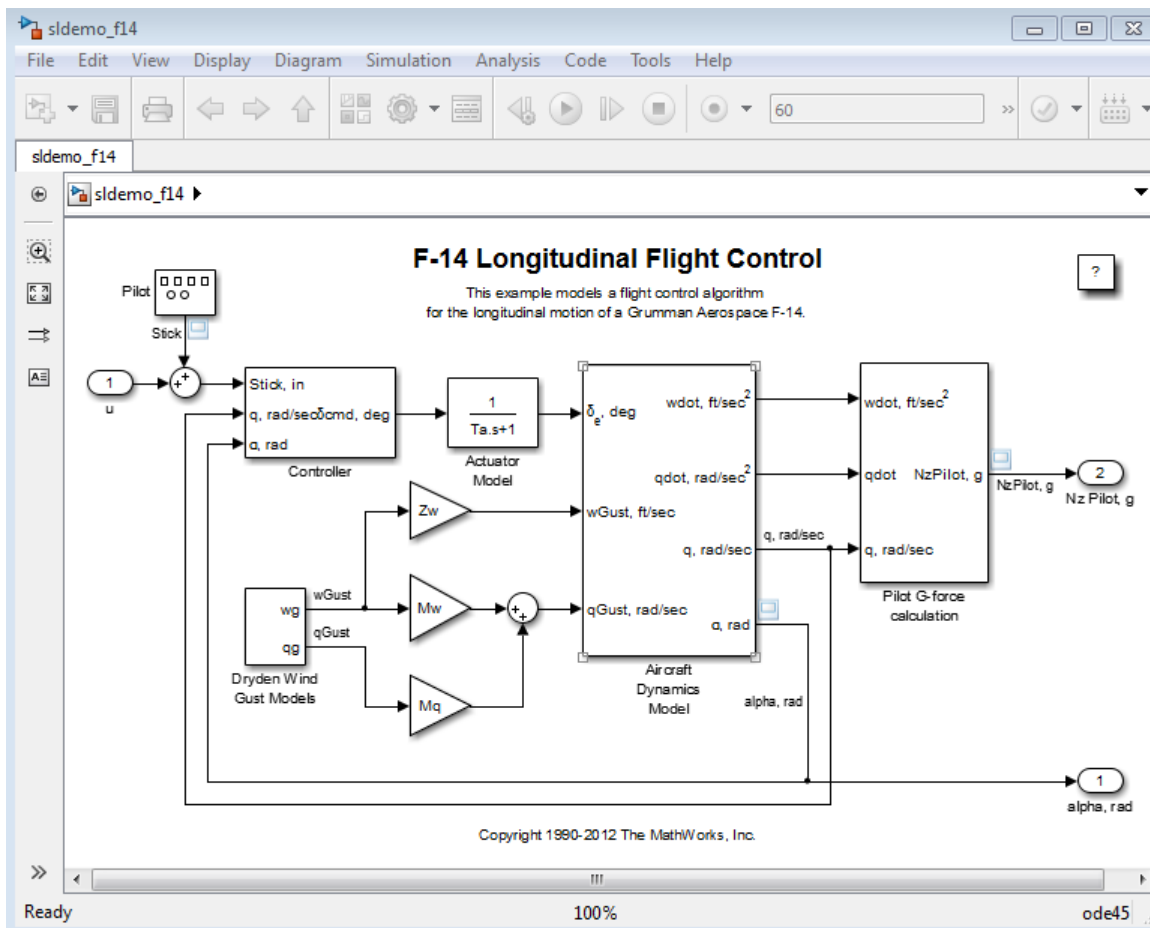
**Note** Selecting **Use printframe paper settings** deactivates the **Custom** and **Zoom** options and automatically scales the image to the print frame size.

---

## Properties Options

- **Include callouts to describe visible objects:** Displays descriptive callouts for visible objects
- **Use printframe:** Prints a frame around the image. You can use the default frame, `rptdefaultframe.fig`, or use the Frame Editor to build a custom frame. For more information, see the `frameedit` function in Simulink documentation.

The default frame is five inches wide and four inches high. It includes the name of the system and the model folder. This frame is optimized for use with a portrait paper orientation. The Flight Control Model in the f14 Simulink model appears here with the default Simulink Report Generator frame option.



## Display Options

To access the display options, click the **Advanced** button.

- **Scaling:** Controls size of the image, as displayed in a browser. Making an image larger using this option does not affect the storage size of the image, but the quality of the displayed image may decrease as you increase or decrease the size of the displayed image.

Generally, to achieve the best and most predictable display results, use the default setting of Use image size.

- **Use image size:** Causes the image to appear the same size in the report as on screen (default).
- **Fixed size:** Specifies the number and type of units.
- **Zoom:** Specifies the percentage, maximum size, and units of measure.
- **Size:** Specifies the size of the snapshot in a browser, using the format w h (width, height). This field is active only if you choose **Fixed size** in the **Scaling** selection list.
- **Max size:** Specifies the maximum size of the snapshot in a browser, using the format w h (width, height). This field is active only if you choose **Zoom** in the **Scaling** selection list.
- **Units:** Specifies the units for the size of the snapshot in a browser. This field is active only if you choose **Fixed size** in the **Image size** list box.

- **Alignment:** Only reports in PDF or RTF format support this property.
  - Auto
  - Right
  - Left
  - Center
- **Image title:**
  - None (Default)
  - System name: Uses the system name as the image name.
  - Full system name: Uses the system name, with path information, as the image name.
  - Custom: Specifies a custom title.
- **Caption:**
  - None (Default)
  - Description (use system description)
  - Custom: Specifies a custom caption.

### Insert Anything into Report?

Yes. Image.

### Class

rptgen\_sl.csl\_sys\_snap

### See Also

System Loop, Block Loop



# Test Sequence

Capture Test Sequence block information

## Description

This component captures information about Simulink Test™ Test Sequence blocks. The report includes the test sequence using the tabular series of steps from the Test Sequence block.

## Test Sequence Block Section Title

Title to use for the Test Sequence block section:

- **Use Test Sequence name (default):** Use the name of the Test Sequence block as the section title.
- **Custom:** Specify a custom section title in the text box.

## Step Content

Select the step content to include in the report.

- **Include description, action, and transition table (default):** Include all the step data in the report, i.e., step descriptions, action statements, transition table, and when condition.
- **Include description only:** Include only the description of each step in the report.
- **Include action and transition table only:** Include the action statements, transition table, and when condition in the report.
- **None:** Do not include the description, action, or transition table in the report.
- **Include requirements:** Include links to requirements that are attached to steps in the Test Sequence block.

## Insert Anything into Report?

Yes. Test Sequence block name or specified title and optional step information.

## Class

rptgen\_stm.cstm\_testseq

## To Workspace Plot

Capture plot figure created in the MATLAB workspace

### Description

This component captures a plot figure created in the MATLAB workspace, and then inserts one or both of the following into the report:

- A table that includes input and output numeric values.
- A figure that plots the values included in the table.

### Print Options

- **Image file format:** Specifies the image file format (for example, JPEG or TIFF) from this list. Select `Automatic HG Format` (the default) to choose the format best suited for the specified report output format automatically. Otherwise, choose an image format that your output viewer can read. Other options are:
  - `Automatic HG Format` (Uses the file format selected in the Preferences dialog box)
  - `Bitmap (16m-color)`
  - `Bitmap (256-color)`
  - `Black and white encapsulated PostScript`
  - `Black and white encapsulated PostScript (TIFF)`
  - `Black and white encapsulated PostScript2`
  - `Black and white encapsulated PostScript2 (TIFF)`
  - `Black and white PostScript`
  - `Black and white PostScript2`
  - `Color encapsulated PostScript`
  - `Color encapsulated PostScript (TIFF)`
  - `Color encapsulated PostScript2`
  - `Color encapsulated PostScript2 (TIFF)`
  - `Color PostScript`
  - `Color PostScript2`
  - `JPEG high quality image`
  - `JPEG medium quality image`
  - `JPEG low quality image`
  - `PNG 24-bit image`
  - `TIFF - compressed`
  - `TIFF - uncompressed`
  - `Windows metafile`
- **Paper orientation:**

- Landscape
- Portrait
- Rotated
- Use `figure orientation`: Uses the orientation for the figure, which you set with the `orient` command.
- Full page image (PDF only): In PDF reports, scales images to fit the full page, minimizes page margins, and maximizes the size of the image by using either a portrait or landscape orientation.

For more information about paper orientation, see the `orient` command in the MATLAB documentation.

- **Image size:**
  - Use figure `PaperPositionMode` setting: Uses the `PaperPositionMode` property of the Handle Graphics figure to set the image size in the report. For more information about paper position mode, see the `orient` command in the MATLAB documentation.
  - Automatic (same size as on screen): Sets the image in the report to the same size as it appears on the screen.
  - Custom: Specifies a custom image size. Specify the image size in the **size** and **units** fields.
- **Size:** Specifies the size of the Handle Graphics figure snapshot in the format `wxh` (width times height). This field is active only if you choose `Custom` in the **Image size** list box.
- **Units:** Specifies units for the size of the Handle Graphics figure snapshot. This field is active only if you choose `Set image size` in the **Custom** list box.
- **Invert hardcopy:** Uses the Handle Graphics figures `InvertHardcopy` property, which inverts colors for printing; it changes dark colors to light colors, and light colors to dark colors.
  - Automatic: Automatically changes dark axes colors to light axes colors. If the axes color is a light color, it is unchanged.
  - Invert: Changes dark axes colors to light axes colors, and light axes colors to dark axes colors.
  - Don't invert: Retains image colors displayed on screen in the printed report.
  - Use figure's `InvertHardcopy` setting: Uses the `InvertHardcopy` property set in the Handle Graphics image.
  - Make figure background transparent: Makes the image background transparent.

## Display Options

- **Scaling:** Controls size of the image, as displayed in a browser. Making an image larger using this option does not affect the storage size of the image, but the quality of the displayed image may decrease as you increase or decrease the size of the displayed image.

Generally, to achieve the best and most predictable display results, use the default setting of `Use image size`.

- `Use image size`: Causes the image to appear the same size in the report as on screen (default).
- `Fixed size`: Specifies the number and type of units.

- **Zoom:** Specifies the percentage, maximum size, and units of measure.
- **Size:** Specifies the size of the snapshot in the format `w h` (width, height). This field is active only if you choose `Fixed size` in the **Scaling** list.
- **Max size:** Specifies the maximum size of the snapshot in the format `w h` (width, height). This field is active only if you choose `Zoom` from the **Scaling** list.
- **Units:** Specifies units for the size of the snapshot. This field is active only if you choose `Zoom` or `Fixed size` in the **Image size** list box.
- **Alignment:** Only reports in PDF or RTF format support this property.
  - Auto
  - Right
  - Left
  - Center
- **Title:** Specifies text to appear above the snapshot.
- **Caption:** Specifies text to appear under the snapshot.

## Insert Anything into Report?

Yes. Figure.

### Class

`rptgen_sl.csl_blk_toworkspace`

### See Also

Figure Loop

# Truth Table

Report on truth tables in Simulink and Stateflow models

## Description

The Truth Table component reports on truth tables in Simulink and Stateflow models. It displays both the condition table and the action table. The parent component of the Truth Table determines its behavior.

- **Model Loop:** Reports on all truth tables in the current model.
- **System Loop:** Reports on all truth tables in the current system.
- **Block Loop :** Reports on all truth tables in the current block.
- **Signal Loop:** Reports on all truth tables in the current signal.

## Title

**Title:** Specifies a title for the truth table.

- No title
- Use Stateflow name
- Custom

## Condition Table

Specify display parameters for the condition table.

- **Show header:** Displays the column headers in the table.
- **Show number:** Displays the condition number column in the table.
- **Show condition:** Displays the condition column in the table.
- **Show description:** Displays the description column in the table.
- **Wrap if column count:** Specifies how many columns to display before creating a table continuation. If the specified number is greater than the number of columns that can appear on the page, some columns do not appear in the report.

## Action Table

- **Show header:** Displays the column headers in the table.
- **Show number:** Displays the condition number column in the table.
- **Show condition:** Displays the condition column in the table.
- **Show description:** Displays the description column in the table. If you do not select this option, no action table appears in the report.

## Insert Anything into Report?

Yes. Table.

**Class**

rptgen\_sf.csf\_truthtable

**See Also**

Block Loop, Model Loop, Signal Loop, System Loop

# Classes

---

## slreportgen.webview.EmbeddedWebViewDocument class

**Package:** slreportgen.webview

Create a report generator that generates an HTML report containing an interlinked document and associated web view

### Description

Creates a report generator that generates an HTML report containing a document and a web view of one or more Simulink models, with two-way hyperlinks between the document and the web view.

This class provides the following facilities for generating embedded web view reports:

- A report generator based on an `slreportgen.report.Report` object. You can use DOM and Report APIs to fill the document content.
- An HTML template with three panels for a table of contents (TOC), document content, and a web view, respectively
- Template holes to be filled with the document content and a web view, respectively. The hole for the web view is named `slwebview` and is located in the right panel of the report. The hole for document content is named `Content` and is located in the center panel of the report.
- Methods for filling the document and web view holes.
- Methods for creating two-way hyperlinks between the document content and embedded webview(s)
- JavaScript that generates a TOC from document headings when the report opens in a browser
- Model export options that allow you to specify the models and subsystems to be embedded as web views in the generated report
- Methods for retrieving elements (diagrams, blocks, charts, etc.) from models to be embedded as web views in the report

### Construction

`rptgen = slreportgen.webview.EmbeddedWebViewDocument(rptname,model)` creates a report generator that generates a report having the specified file name and containing a web view of the specified model. Use the generator's `fill` method to generate the web view and embed the web view in the document. Use the generator's `close` method to output the document as a zip file or folder containing the HTML document.

`rptgen = slreportgen.webview.EmbeddedWebViewDocument(rptname,model1,model2,...,modeln)` creates a report generator that includes two or more models in the web view that it creates. This constructor assigns an array of default `slreportgen.webview.ExportOptions` objects to the generator's `ExportOptions` property, one for each of the models to be included in the generated document's web view. You can use the objects to specify custom export options for each of the models to be included in the web view exported to the generated document.



`rptgen = slreportgen.webview.EmbeddedWebViewDocument(rptname, {model1, model2, ... modeln})` creates a generator that includes the specified models in the web view that it embeds in the output document.

`rptgen = slreportgen.webview.EmbeddedWebViewDocument(rptname)` creates a generator that embeds models specified by the `Diagrams` property of the generator's `ExportOptions` property, for example:

```
import slreportgen.webview.*
rptgen = EmbeddedWebViewDocument('myDoc');
rptgen.ExportOptions.Diagrams = 'myModel';
```

## Input Arguments

### **rptname** — Name of output report file and/or folder

character vector

Name of the zip file and/or folder containing the report generated by this generator. Use this generator's `PackageType` property to specify whether to package the generated report as a file or a folder or both. If you specify an extension, the extension must be `.htm`. If you do not specify an extension, the report generator appends `.htm`.

### **model** — Name of model to export

character vector

Name of model, specified as a character vector, to be embedded in the generated report as a web view.

## Output Arguments

### **rptgen** — Embedded web view report generator

`slreportgen.webview.EmbeddedWebViewDocument`

## Properties

### **CurrentHoldID** — Identifier of current hole in document

character vector

Identifier of current hole in document, stored as a character vector. This is a read-only property.

### **ExportOptions** — Web view export options

`slreportgen.webview.ExportOptions` (default)

An array of `slreportgen.webview.ExportOptions` objects, one for each model or set of models to be included in the web view exported to the generated report. The generator's constructor sets this property with default values for the models you specify. Use the properties of the export options object or objects to customize export of the models to the generated web view. For example, you can specify additional models to include or whether to include the block diagrams of masked subsystems and library blocks.

### **ForceOverwrite** — Overwrite existing report

True (default) | False

Whether to overwrite an existing report with the same name. True overwrites the existing report. False generates the report under a new name.

### **OpenStatus — Status of the report being generated**

unopened (default) | opened

Status of the report being generated, either 'unopened' or 'opened'. This is a read-only property.

### **OutputPath — Path of the document output directory**

current working directory (default)

Path of the report output directory.

### **PackageType — Packaging for files generated**

'both' (default) | 'zipped' | 'unzipped'

Packaging to use for output document, specified as one of these character vectors:

- 'both' — Creates both zipped and unzipped output
- 'zipped' — Creates a zip file with an .htmx extension
- 'unzipped' — Creates a folder of files

### **TemplatePath — Path of the template used to generate this report**

character vector

Path to the HTML template to use to generate this report, specified as a character vector. The template has an .html extension. This property points by default to a default template. To use a custom template, set this property to the path of the custom template.

### **TitleBarText — Text for HTML browser title bar**

character vector

Text to display in the title bar of the HTML browser used to display the generated report. The default text is "Simulink Web View - Created by Simulink Report Generator."

### **ValidateLinksAndAnchors — Whether to check validity of hyperlinks between the generated document and web view**

True (default) | False

Generates a warning at the command line if the link target that you specify does not exist or if you specify a link from a model element that already has a link. Validation checking increases the time required to generate a report. For this reason, consider using link validation checking only when debugging your report.

## **Methods**

<b>Method</b>	<b>Purpose</b>
createDiagramTwoWayLink	Creates a two-way link between a location in the document in the center panel and a diagram in the web view in the right panel

Method	Purpose
createElementTwoWayLink	Creates a two-way link between a document panel location and a diagram element in the web view
createDiagramLink	Creates a link from the document panel to a diagram in the model web view
createElementLink	Creates a link from the document panel to an element in the model web view
fill	Invoke the embedded web view report generator's hole filling methods to fill the holes in its template.
fillslwebview	Fills template's slwebview hole with a web view
getExportModels	Names of models to be included in the web view
getExportDiagrams	Paths and handles of block diagrams to be included in the web view
getExportSimulinkSubSystems	Paths and handles of subsystem blocks to be included in this web view
getExportStateflowCharts	Paths and handles of Stateflow charts to be included in this web view
getExportStateflowDiagrams	Array of Stateflow diagram paths
getReportObject	Returns the report object for the embedded web view report

## Compatibility Considerations

### Items in table of contents are not numbered

*Behavior changed in R2019b*

Starting in R2019b, items in the table of contents in an embedded web view report are not numbered. To generate numbered items in a table of contents:

- 1 Copy the default template for an embedded web view report to the current folder.

```
copyfile(fullfile(matlabroot, '/toolbox/slreportgen/webview/resources/templates/embedded_webview.htmxx'))
```
- 2 Unzip the template into the subfolder myTemplate.

```
unzipTemplate('embedded_webview.htmxx', 'myTemplate');
```
- 3 In myTemplate, navigate to the stylesheets subfolder. Open the root.css file and remove this line from the ol.toc style:

```
list-style-type: none;
```
- 4 Save the root.css file.
- 5 Navigate to the folder that contains the original zipped template file. Package the unzipped template files into a zipped template file.

```
zipTemplate('Newtemplate.htmxx', 'myTemplate');
```
- 6 Set the TemplatePath property of the embedded web view report generator object to the path and name of the new template file. For example, suppose that MyEmbeddedWebView is a subclass

of `slreportgen.webview.EmbeddedWebViewDocument`. Here is the code to set the `TemplatePath` property of a `MyEmbeddedWebView` object:

```
rpt = MyEmbeddedWebView(rptName,model);  
rpt.TemplatePath = 'NewTemplate.htm';
```

## **See Also**

`slreportgen.webview.ExportOptions`

## **Topics**

“Create an Embedded Web View Report Generator” on page 5-23

“Embedded Web View Reports” on page 5-19

## **Introduced in R2017a**

# slreportgen.report.BusObject class

**Package:** slreportgen.report

Simulink bus object reporter

## Description

Creates a reporter that generates information about a Simulink.Bus object in a report.

---

**Note** To use a bus object reporter in a report, you must create the report using the slreportgen.report.Report class or subclass.

---

The slreportgen.report.BusObject class is a handle class.

## Class Attributes

HandleCompatible	true
------------------	------

For information on class attributes, see “Class Attributes”.

## Creation

### Description

`reporter = slreportgen.report.BusObject()` creates an empty slreportgen.report.BusObject reporter object. Customize the content and formatting of the information reported for a bus object by using the reporter object properties. Before you add the reporter to a report, you must set the Object property of the reporter to an slreportgen.report.ModelVariableResult or Simulink.VariableUsage object that specifies a Simulink.Bus object. Adding an empty reporter to a report produces an error.

`reporter = slreportgen.report.BusObject(object)` creates a reporter for the Simulink.Bus object specified by an slreportgen.report.ModelVariableResult or Simulink.VariableUsage object. See the Object property.

`reporter = slreportgen.report.BusObject(Name, Value)` sets the reporter properties using name-value pairs. You can specify multiple name-value pair arguments in any order. Enclose each property name in single or double quotes.

## Properties

### Object — Object that specifies a Simulink.Bus object

slreportgen.finder.ModelVariableResult object | Simulink.VariableUsage object

Object that specifies the Simulink.Bus object to report, specified as an slreportgen.finder.ModelVariableResult object or a Simulink.VariableUsage object.

**Name — Name of bus object**`string scalar`

This read-only property contains the name of bus object to report, specified as a string scalar.

**ReportedBusProperties — Bus object properties to report**`[] (default) | string array | cell array of character vectors`

Bus object properties to report, specified as a string array or a cell array of character vectors. The properties specified by the `ReportedBusProperties` property are further filtered by the `PropertyFilterFcn` property. If the `ReportedBusProperties` property is empty, the reporter includes all properties in the report, except the properties filtered by the `PropertyFilterFcn` property. The reporter excludes any bus object property that is not valid for the bus object.

**ReportedElementProperties — Bus element properties to report**`[] (default) | string array | cell array of character vectors`

Bus element properties to report, specified as a string array or a cell array of character vectors. The properties specified by the `ReportedElementProperties` property are further filtered by the function or code specified in the `PropertyFilterFcn` property. If the `ReportedElementProperties` property is empty, the reporter includes all properties in the report, except the properties filtered by the `PropertyFilterFcn` property. The reporter excludes any bus element property that is not valid for the bus element.

**ShowName — Whether to show name of bus**`false (default) | true`

Whether to show the name of the bus object in the report, specified as `true` or `false`.

**ShowHierarchy — Whether to show hierarchy of the bus object**`true (default) | false`

Whether to include a nested list that represents the bus hierarchy in the report, specified as `true` or `false`.

**ShowProperties — Whether to show bus object properties**`true (default) | false`

Whether to show the bus object properties table in the report, specified as `true` or `false`.

**ShowElements — Whether to show bus object elements properties**`true (default) | false`

Whether to show the bus element properties table in the report, specified as `true` or `false`.

**ShowUsedBy — Whether to show blocks that use bus object**`true (default) | false`

Whether to show a list of the blocks that use the bus object, specified as `true` or `false`. If the `ShowUsedBy` property is set to `true`, the reporter includes a list of the blocks that use the bus object in the report. If the `ShowUsedBySnapshot` property is also set to `true`, the reporter includes a diagram snapshot for each parent subsystem that uses the bus object. Blocks that use the bus object are highlighted in the snapshot.

**ShowUsedBySnapshot — Whether to show diagram snapshots highlighting blocks that use bus object**`true (default) | false`

Whether to show diagram snapshots of parent subsystems and highlight the blocks that use the bus object, specified as `true` or `false`. If the `ShowUsedBySnapshot` property is set to `true`, the report includes a snapshot for each parent subsystem that uses the bus object. Blocks that use the bus object are highlighted in the snapshot. If a parent subsystem has more than one block that uses the bus object, the reporter shows only one diagram snapshot that highlights the blocks that use the bus object.

**CreateSections — Whether to create sections**`true (default) | false`

Whether to create a separate section for each type of information about the bus object in the report. If the `CreateSections` property is set to `true`, the reporter creates an `mlreportgen.report.Section` with a title for each of these types of information:

If the `CreateSections` property is set to `false`, the reporter generates labels for tables and lists. For a table, the reporter generates a table title. For a list, the reporter generates text that precedes the list.

**HierarchyListFormatter — List formatter for hierarchy**`mlreportgen.dom.UnorderedList | mlreportgen.dom.OrderedList`

List formatter that formats the generated bus hierarchy, specified as an `mlreportgen.dom.UnorderedList` object or an `mlreportgen.dom.OrderedList` object. To customize the list formatting, modify the list object properties or replace the list object with a customized list object that does not contain list items.

**UsedByListFormatter — List formatter for blocks that use the bus object**`mlreportgen.dom.UnorderedList | mlreportgen.dom.OrderedList`

List formatter that formats the generated list of blocks that use the bus object, specified as an `mlreportgen.dom.UnorderedList` object or an `mlreportgen.dom.OrderedList` object. To customize the list formatting, modify the list object properties or replace the list object with a customized list object that does not contain list items.

**PropertiesTableReporter — Table reporter for bus object properties**`mlreportgen.report.BaseTable`

Table reporter used to format the table of bus object properties, specified as an `mlreportgen.report.BaseTable` object. To customize the appearance of the table, modify the properties of the default table reporter or replace it with a customized table reporter. If you add content to the `Title` property of the default or customized table reporter, the content appears in front of the table title in the generated report.

**ElementsTableReporter — Table reporter for bus element properties**`mlreportgen.report.BaseTable`

Table reporter used to format the table of bus element properties, specified as an `mlreportgen.report.BaseTable` object. To customize the appearance of the table, modify the properties of the default table reporter or replace it with a customized table reporter. If you add

content to the `Title` property of the default or customized table reporter, the content appears in front of the table title in the generated report.

### HorizontalElementsTable — Whether to display element properties horizontally

`false` (default) | `true`

Whether to display properties horizontally in the table of element properties, specified as `true` or `false`.

If the `HorizontalElementsTable` property is set to `true`, the table has one column for each property. For example:

Name	DataType	Complexity	Min	Max
a	double	real	[]	[]
b	double	real	[]	[]

If the `HorizontalElementsTable` property is set to `false`, the property and value cells in the row for the element are split into multiple rows. For example:

Element	Property	Value
a	DataType	double
	Complexity	real
	Min	[]
	Max	[]
b	DataType	double
	Complexity	real
	Min	[]
	Max	[]

### SectionReporter — Section reporter

`mlreportgen.report.Section`

Reporter for formatting sections when the `CreateSections` property is set to `true`, specified as an `mlreportgen.report.Section` object. To customize the appearance of the section, modify the properties of the default section reporter or replace it with a customized section reporter.

### PropertyFilterFcn — Function or expression to filter properties of a reported bus or bus element

[] (default) | function handle | string scalar | character vector

Function or expression to filter the properties of a reported bus or bus element from a report. Specify a function as a function handle. Specify an expression as a string scalar or character vector.

If you provide a function handle, the associated function must:

For example, this code prevents the display of the `HeaderFile` and `Description` properties of a bus object and the `Complexity` property of a bus element:

```
import slreportgen.finder.*
import slreportgen.report.*

rpt = slreportgen.report.Report('busrpt', 'pdf');
```



```

model = load_system('sldemo_bus_arrays');

modelVariableFinder = ModelVariableFinder(model);
results = find(modelVariableFinder);
for result = results
    if isa(getVariableValue(result),'Simulink.Bus')
        busRptr = slreportgen.report.BusObject(result);
        busRptr.PropertyFilterFcn = @busPropertyFilter;
        % Create a Chapter
        chapter = mlreportgen.report.Chapter(busRptr.Name);
        add(chapter, busRptr);
        add(rpt,chapter)
    end
end
close(rpt);

close_system(model);
rptview(rpt);

function tf = busPropertyFilter(~, variableObject,propertyName)
if isa(variableObject, 'Simulink.Bus')
    tf = (propertyName == "HeaderFile") || ...
        (propertyName == "Description");
else
    % Filter Simulink.BusElement Complexity property
    tf = propertyName == "Complexity";
end
end

```

If you provide a string scalar or a character vector, it must contain an expression. The expression:

For example, this code filters the HeaderFile property of a bus object from the report:

```

import slreportgen.finder.*
import slreportgen.report.*

rpt = slreportgen.report.Report('busrpt','pdf');

model = load_system('sldemo_bus_arrays');

modelVariableFinder = ModelVariableFinder(model);
results = find(modelVariableFinder);
for result = results
    if isa(getVariableValue(result),'Simulink.Bus')
        busRptr = slreportgen.report.BusObject(result);
        busRptr.PropertyFilterFcn = "isFiltered = " + ...
            "isa(variableObject, 'Simulink.Bus') && " + ...
            "propertyName == 'HeaderFile';";
        % Create a Chapter
        chapter = mlreportgen.report.Chapter(busRptr.Name);
        add(chapter, busRptr);
        add(rpt,chapter)
    end
end
close(rpt);

close_system(model);
rptview(rpt);

```

### TemplateSrc — Source of template for this reporter

[ ] (default) | character vector | string scalar | reporter or report | DOM document or document part

Source of the template for this reporter, specified in one of these ways:

The specified template must be the same type as the report to which this reporter is appended. For example, for a Microsoft Word report, `TemplateSrc` must be a Word reporter template. If the `TemplateSrc` property is empty, this reporter uses the default reporter template for the output type of the report.

### TemplateName — Name of template for this reporter

character vector | string scalar

Name of the template for this reporter, specified as a character vector or string scalar. The template for this reporter must be in the template library of the template source (`TemplateSrc`) for this reporter.

### LinkTarget — Hyperlink target for this reporter

[] (default) | character vector | string scalar | `mlreportgen.dom.LinkTarget` object

Hyperlink target for this reporter, specified as a character vector or string scalar that specifies the link target ID, or an `mlreportgen.dom.LinkTarget` object. A character vector or string scalar value is converted to a `LinkTarget` object. The link target immediately precedes the content of this reporter in the output report.

## Methods

### Public Methods

<code>slreportgen.report.BusObject.createTemplate</code>	Create bus object reporter template
<code>slreportgen.report.BusObject.customizeReporter</code>	Create custom bus object reporter
<code>slreportgen.report.BusObject.getClassFolder</code>	Bus object reporter class definition file location
<code>copy</code>	Create copy of reporter object and make deep copies of property values that reference a reporter, <code>ReporterLayout</code> , or DOM object
<code>getImpl</code>	Get implementation of reporter

## Examples

### Report on Bus Objects Using Bus Object Reporters

Report on bus objects in a model by using a model variable finder to find all variables used in the model and then creating a bus reporter for each variable that is a bus object.

```
% Create a Report
rpt = slreportgen.report.Report("MyReport", "pdf");
open(rpt);

% Load a model
model_name = "sldemo_bus_arrays";
load_system(model_name);

% Find all variables used by the model
finder = slreportgen.finder.ModelVariableFinder(model_name);

% Create a Bus object reporter object for all results representing a
```

```

% Simulink.Bus object
while hasNext(finder)
    result = next(finder);
    if isa(getVariableValue(result), "Simulink.Bus")
        % Create a Bus object reporter
        busReporter = slreportgen.report.BusObject(result);
        % Create a Chapter
        chapter = mlreportgen.report.Chapter(busReporter.Name);
        % Add bus to chapter
        add(chapter, busReporter)
        % Add chapter to the report
        add(rpt,chapter);
    end
end

% Close and view the report
close(rpt);
rptview(rpt);

```

### Customize the Reported Content and Formatting for Bus Objects

Customize the reported content and formatting of the content by setting properties of the bus object reporter. This example uses the ReportedElementProperties property to constrain the element properties that are reported. It uses the HorizontalElementsTable property to generate a properties table with one column for each property.

```

% Create a Report
rpt = slreportgen.report.Report("MyReport", "pdf");
open(rpt);

% Load a model
model_name = "sldemo_bus_arrays";
load_system(model_name);

% Find all variables used by the model
finder = slreportgen.finder.ModelVariableFinder(model_name);

% Create a Bus object reporter object for all results representing a
% Simulink.BusObject object
while hasNext(finder)
    result = next(finder);
    if isa(getVariableValue(result), "Simulink.Bus")
        % Create a Bus object reporter
        busReporter = slreportgen.report.BusObject(result);
        % Limit the properties that are reported
        busReporter.ReportedElementProperties = {'Name', 'DataType', 'Min', 'Max'};
        % Display element properties horizontally
        busReporter.HorizontalElementsTable = true;
        % Create a Chapter
        chapter = mlreportgen.report.Chapter(busReporter.Name);
        % Add bus to chapter
        add(chapter, busReporter)
        % Add chapter to the report
        add(rpt,chapter);
    end
end

% Close and view the report

```

```
close(rpt);  
rptview(rpt);
```

**See Also**

[Simulink.VariableUsage](#) | [Simulink.findVars](#) |  
[slreportgen.finder.ModelVariableFinder](#) | [slreportgen.finder.ModelVariableResult](#)  
| [slreportgen.report.ModelVariable](#)

**Topics**

“Report Generation for Simulink and Stateflow Elements” on page 1-9  
“What Is a Reporter?”

**Introduced in R2019b**

# slreportgen.report.DataDictionary class

**Package:** slreportgen.report

Simulink data dictionary reporter

## Description

Use an object of the `slreportgen.report.DataDictionary` class to report on a Simulink data dictionary. Create a `DataDictionary` object to report on a specific data dictionary. Alternatively, use an `slreportgen.finder.DataDictionaryFinder` object to find data dictionaries and use the `getReporter` method of an `slreportgen.finder.DataDictionaryResult` object to return the reporter for the result.

---

**Note** To use an `slreportgen.report.DataDictionary` reporter in a report, you must create the report using the `slreportgen.report.Report` class or subclass.

---

The `slreportgen.report.DataDictionary` class is a handle class.

## Class Attributes

`HandleCompatible` true

For information on class attributes, see “Class Attributes”.

## Creation

### Description

`reporter = slreportgen.report.DataDictionary()` creates a `DataDictionary` reporter object based on the default template. Use the reporter properties to specify a data dictionary and report options. You must specify the data dictionary to report. Adding an empty data dictionary reporter object to a report produces an error.

`reporter = slreportgen.report.DataDictionary(dictionaryName)` creates a `DataDictionary` reporter object and sets the `Dictionary` property to the specified data dictionary. Use the reporter properties to specify report options.

`reporter = slreportgen.report.DataDictionary(Name,Value)` sets the reporter properties using name-value pairs. You can specify multiple name-value pair arguments in any order. Enclose each property name in single or double quotes.

## Properties

### Dictionary — Data dictionary to report

character vector | string scalar

Data dictionary to report, specified as a character vector or string scalar that contains the file name of a dictionary on the MATLAB path or the path and file name of a data dictionary. The path can be relative or absolute.

Example: "sldemo\_fuelsys\_dd.sldd"

Example: "myDictionaries/myDataDictionary.sldd"

### SummaryProperties — Properties to report

["Name" "Value" "Class" "LastModified" "LastModifiedBy" "Status" "DataSource"] (default) | string array | cell array of character vectors

Properties to report for each data dictionary entry in the summaries table, specified as a string array or cell array of character vectors. Valid properties are:

The **Value** entry contains the value if the data type is numeric scalar, logical scalar, string scalar, or character vector. Otherwise, the **Value** entry is `See details`.

Example: ["Name" "Value" "Class" "LastModified" "LastModifiedBy" "Status" "DataSource"]

Example: {'Name' 'Value' 'Class'}

### ShowDesignData — Whether to report Design Data section

true (default) | false

Whether to report the **Design Data** section of the data dictionary, specified as true or false.

Data Types: logical

### ShowConfigurations — Whether to report Configurations section

false (default) | true

Whether to report the **Configurations** section of the data dictionary, specified as true or false.

Data Types: logical

### ShowOtherData — Whether to report Other Data section

false (default) | true

Whether to report the **Other Data** section of the data dictionary, specified as true or false.

Data Types: logical

### IncludeReferencedDictionaries — Whether to include referenced dictionaries

true (default) | false

Whether to include the dictionaries that are referenced by the dictionary that this reporter reports, specified as true or false. The `ReferencedDictionaryPolicy` property determines how referenced dictionaries are reported.

Data Types: logical

### ReferencedDictionaryPolicy — Display policy for referenced dictionary

"SameTable" (default) | character vector | string scalar

Display policy for reporting a referenced dictionary, specified as one of these values:

**EntryFilterFcn — Data dictionary entry filter**

[] (default) | function handle | string scalar | character vector

Data dictionary entry filter, specified as a function handle, string scalar, or character vector. If you do not provide `EntryFilterFcn`, all entries are included in the report.

If you provide a function handle, the associated function must:

For example, this code uses the `EntryFilterFcn` property to prevent reporting of entries that are `Simulink.Parameter` objects:

```
rpt = slreportgen.report.Report("MyReport", "pdf");
ddPath = which("sldemo_fuelsys_dd.sldd");

ch = mlreportgen.report.Chapter("sldemo_fuelsys_dd.sldd");
rptr = slreportgen.report.DataDictionary(ddPath);

filterFcnHandle = @(entryObject, entryValue) ...
    isa(entryValue, "Simulink.Parameter");
rptr.EntryFilterFcn = filterFcnHandle;

append(ch, rptr);
append(rpt, ch);

close(rpt);
rptview(rpt);
```

If you provide a string scalar or a character vector, it must contain an expression. The expression:

For example, this code uses the `EntryFilterFcn` property to prevent reporting of entries that are `Simulink.Bus` objects:

```
rpt = slreportgen.report.Report("MyReport", "pdf");
ddPath = which("sldemo_fuelsys_dd.sldd");

ch = mlreportgen.report.Chapter("sldemo_fuelsys_dd.sldd");
rptr = slreportgen.report.DataDictionary(ddPath);
filterStr = "isFiltered = isa(entryValue, 'Simulink.Bus');";
rptr.EntryFilterFcn = filterStr;

append(ch, rptr);
append(rpt, ch);

close(rpt);
rptview(rpt);
```

**SummaryTableReporter — Formatter for entry summary tables**

mlreportgen.report.BaseTable object

Formatter for the entry summary tables, specified as an `mlreportgen.report.BaseTable` object. The default value of this property is a `BaseTable` object with the `TableStyleName` property set to the `DataDictionaryTable` style which is defined in the default template for a `DataDictionary` reporter. To customize the appearance of the table, modify the properties of the default `BaseTable` object or replace the object with a customized `BaseTable` reporter. If you add content to the `Title` property, the content appears in front of the table title in the generated report.

**DetailsReporter — Formatter for entry details**

mlreportgen.report.MATLABVariable object

Formatter for the entry details, specified as an `mlreportgen.report.MATLABVariable` object. The default value of this property is a `MATLABVariable` object with default property values. To customize the appearance of the entry details, modify the properties of the default `MATLABVariable` object or replace the object with your own `MATLABVariable` reporter. The `Variable`, `Location`, and `LinkTarget` properties of the `MATLABVariable` reporter are ignored.

### ListFormatter — List formatter for referenced dictionary list

`mlreportgen.dom.UnorderedList` object | `mlreportgen.dom.OrderedList` object

List formatter for a referenced dictionary list, specified as an `mlreportgen.dom.UnorderedList` object or `mlreportgen.dom.OrderedList` object. The list formatter is used when the `ReferencedDictionaryPolicy` property is set to "List". The `UnorderedList` or `OrderedList` object must not contain list items.

The default value of this property is an `UnorderedList` object with the `StyleName` property set to the `DataDictionaryList` style, which is defined in the default template of a `DataDictionary` reporter. To customize the appearance of the list, modify the properties of the default `UnorderedList` object or replace the object with your own `UnorderedList` or `OrderedList` object.

### TemplateSrc — Source of template for this reporter

[] (default) | character vector | string scalar | reporter or report | DOM document or document part

Source of the template for this reporter, specified in one of these ways:

The specified template must be the same type as the report to which this reporter is appended. For example, for a Microsoft Word report, `TemplateSrc` must be a Word reporter template. If the `TemplateSrc` property is empty, this reporter uses the default reporter template for the output type of the report.

### TemplateName — Name of template for this reporter

character vector | string scalar

Name of the template for this reporter, specified as a character vector or string scalar. The template for this reporter must be in the template library of the template source (`TemplateSrc`) for this reporter.

### LinkTarget — Hyperlink target for this reporter

[] (default) | character vector | string scalar | `mlreportgen.dom.LinkTarget` object

Hyperlink target for this reporter, specified as a character vector or string scalar that specifies the link target ID, or an `mlreportgen.dom.LinkTarget` object. A character vector or string scalar value is converted to a `LinkTarget` object. The link target immediately precedes the content of this reporter in the output report.

## Methods

### Public Methods

`slreportgen.report.DataDictionary.createTemplate`

Copy the default `slreportgen.report.DataDictionary` reporter template



slreportgen.report.DataDictionary.customizeReporter	Create subclass of slreportgen.report.DataDictionary class
slreportgen.report.DataDictionary.getClassFolder	Get location of folder that contains the slreportgen.report.DataDictionary class definition file
copy	Create copy of reporter object and make deep copies of property values that reference a reporter, ReporterLayout, or DOM object
getImpl	Get implementation of reporter

## Examples

### Report on Data Dictionary

Use an object of the `slreportgen.report.DataDictionary` class to report on a Simulink data dictionary.

Import the MATLAB Report and Simulink Report API packages so that you do not have to use long, fully qualified class names.

```
import slreportgen.report.*
import mlreportgen.report.*
```

Create a Simulink report.

```
rpt = slreportgen.report.Report("MyReport", "pdf");
```

Specify the path to the data dictionary used by the model `sldemo_fuelsys_dd`.

```
ddPath = "sldemo_fuelsys_dd.sldd";
```

Create a chapter for the data dictionary information.

```
ch = Chapter("sldemo_fuelsys_dd.sldd");
```

Create a reporter for the data dictionary.

```
rptr = DataDictionary(ddPath);
```

Append the reporter to the chapter and the chapter to the report.

```
append(ch, rptr);
append(rpt, ch);
```

Close and view the report.

```
close(rpt);
rptview(rpt);
```

### See Also

`slreportgen.finder.DataDictionaryFinder` | `slreportgen.finder.DataDictionaryResult` | `slreportgen.report.Report`

**Topics**

“What Is a Data Dictionary?”

“Report Generation for Simulink and Stateflow Elements” on page 1-9

“What Is a Reporter?”

**Introduced in R2020b**

# slreportgen.report.Diagram class

**Package:** slreportgen.report

Create diagram reporter

## Description

Create a diagram reporter, including a diagram snapshot and caption, for a Simulink or Stateflow diagram.

---

**Note** To use a Diagram reporter in a report, you must create the report using the `slreportgen.report.Report` class.

---

## Construction

`diagram = Diagram()` creates an empty diagram reporter. Set its properties to capture a Simulink or Stateflow diagram.

`diagram = Diagram(source)` creates a diagram reporter for the Simulink or Stateflow diagram specified by `source`. Adding this reporter to a report creates a snapshot of the diagram. Then, the snapshot displays in the report as an image with a caption. The snapshot image is stored in the temporary folder of the report. When the report is closed, the snapshot image is copied into the report and then, the image is deleted from the temporary folder. To prevent the snapshot image file from being deleted, use the `Debug` property of the report. See `slreportgen.report.Report`

`diagram = Diagram(Name, Value)` sets properties using name-value pairs. You can specify multiple name-value pair arguments in any order. Enclose each property name in single or double quotes.

## Input Arguments

**source — Diagram snapshot image source**

string | character vector

See the `Source` property.

## Properties

**Source — Diagram snapshot image source**

string scalar | character vector | handle | `slreportgen.finder.DiagramResult` | ...

Diagram snapshot image source, specified as one of these values.

- Name of an open or loaded Simulink model
- Path of a Simulink subsystem block that contains the Simulink diagram or Stateflow chart
- `slreportgen.finder.DiagramResult` object
- Handle to a subsystem block containing a Simulink diagram or Stateflow chart

- `Stateflow.Chart` or `Stateflow` subchart object. Subcharts are graphical objects that can contain the same objects as a top-level chart, including other subcharts. Subcharts are commonly specified by a `Stateflow.State`, `Stateflow.Function`, or `Stateflow.Box` object.

### Snapshot — Snapshot reporter

`mlreportgen.report.FormalImage` object

Snapshot reporter, set by default to an object of the `mlreportgen.report.FormalImage` class. You do not need to set this property yourself. The `FormalImage` object adds the diagram snapshot to a report. To control the size of the snapshot, set the `mlreportgen.report.FormalImage` properties.

### SnapshotArea — Diagram area to capture in snapshot

`[]` (default) | 1-by-4 array of doubles

Diagram area to capture in the snapshot, specified as a 1-by-4 array of doubles. The first two values of the array are the x and y coordinates, in pixels, of the top left corner of the diagram area in the Simulink Editor coordinate space. The last two values are the width and height, in pixels. An empty array specifies the entire diagram.

You can set up the view that you want to capture in the Simulink Editor and then set the `SnapshotArea` property to the output of the `slreportgen.utils.getCurrentEditorView` function. For an example, see “Take Snapshot of Part of a Diagram” on page 7-28.

### SnapshotFormat — Snapshot image format

`'svg'` (default) | ...

Snapshot image format, specified as a character vector or string scalar. Supported formats are:

- `'bmp'` — Bitmap image.
- `'gif'` — Graphics Interchange format.
- `'jpg'` — JPEG image.
- `'png'` — PNG image.
- `'emf'` — Enhanced metafile, supported only in DOCX output on Windows platforms.
- `'svg'` — Scalable Vector Graphics.
- `'tif'` — Tag Image File format, not supported in HTML output.
- `'pdf'` — PDF image.

See “Compatibility Considerations” on page 7-29.

### HyperlinkDiagram — Hyperlinks of diagram elements

`true` (default) | `false`

Choice to include a hyperlink of each diagram element, specified as a logical. If this property is `true`, each element becomes a hyperlink to an object in the report that describes it. This property applies only to PDF and HTML reports. Hyperlinks allow you to navigate the report using Simulink and Stateflow charts.

The `Diagram`, `SimulinkObjectProperties`, and `StateflowObjectProperties` reporters work together to enable navigation using hyperlinks. Each reporter prefaces the report object it creates with a hyperlink target. The ID of that target is based on the path of the reported element in the model. The `Diagram` reporter also overlays elements of a diagram snapshot with hyperlinks to the

corresponding element-based target ID. The report object to which a diagram element links depends on the element type.

- A diagram-based block (subsystem, chart, model) links to the diagram of the block.
- Other blocks link to textual block descriptions, typically block property tables.
- Masked subsystem blocks that have mask parameters link to the textual description of the block, such as the mask parameter tables. This linking to the textual descriptions is true only if the `MaskedSystemLinkPolicy` property of the Diagram reporter is set to 'block' or 'default'. Otherwise, the masked system block links to its diagram.
- Masked subsystem blocks that do not have mask parameters link to the diagram of the block.

To customize diagram-based navigation, create custom link targets based on target IDs generated by the `slreportgen.utils.getObjectID` utility function.

### **MaskedSystemLinkPolicy — Policy for masked system blocks hyperlinks targets**

character vector | string

Policy to determine the targets for the hyperlinks of masked system blocks, specified as one of these values.

- 'default' — Masked system blocks that have parameters link to textual descriptions, such as mask parameter tables. Masked system blocks that do not have parameters link to the corresponding block diagram in the report.
- 'system' — Masked system blocks link to their block diagram in the report.
- 'block' — Masked system blocks link to their textual description, such as a table of masked parameters or subsystem parameters.

### **Scaling — Scaling options for diagram snapshot image**

auto (default) | custom | zoom

Scaling options for diagram snapshot image, specified as the string, `auto`, `custom`, or `zoom`. `Scaling` controls size of the diagram snapshot image in the image file.

- `auto` — For PDF or Word (docx) output, `auto` scales the diagram snapshot image to fit in the current page layout while maintaining its aspect ratio. First, the diagram snapshot image is scaled to the page width. If the image height exceeds the page height, then the image is again scaled down. This additional scaling assures that the image fits in the current page with an extra 1" spacing. The extra spacing allows for extra text, such as a caption. Scaling does not apply to HTML output.
- `custom` — Scales the diagram snapshot image based on the values of the `Height` and `Width` properties
- `zoom` — Enlarges or reduces the snapshot image size to the percent value specified by the `Zoom` property. To specify the maximum image height and maximum image width, use the `MaxHeight` and `MaxWidth` properties, respectively.

---

**Note** A `java.lang.OutOfMemoryError` can occur when either of these combinations of property settings occur:

- `Scaling` set to `zoom`, and `Zoom`, `MaxHeight`, and `MaxWidth` properties set to large values
- `Scaling` set to `custom`, and `Height` and `Width` properties set to large values

To avoid this error, for zoom `Scaling`, use smaller `Zoom`, `MaxHeight`, and `MaxWidth` property values. For custom `Scaling`, use smaller `Height` and `Width` property values. Using smaller values ensures that the diagram fits on the page.

---

### **Height — Height of snapshot image**

character vector | string scalar

Height of snapshot image, specified as a character vector or string scalar that consists of a number followed by an abbreviation for a unit of measurement. For example, '2in' specifies two inches. Valid abbreviations are:

- `px` — pixels (default)
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points

Example: '2in'

### **Width — Width of snapshot image**

character vector | string scalar

Width of snapshot image, specified as a character vector or string scalar that consists of a number followed by an abbreviation for a unit of measurement. For example, '2in' specifies two inches. Valid abbreviations are:

- `px` — pixels (default)
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points

Example: '3in'

### **Zoom — Amount to zoom diagram snapshot image**

string

Amount to zoom the diagram snapshot image, specified as a string. The `Zoom` format is *value%*, where *value* is the percentage by which the diagram snapshot image is enlarged or reduced.

### **MaxHeight — Maximum height for zoom scaling**

string

Maximum height for zoom scaling, specified as a string. This property applies only if `Scaling` is set to `zoom`.

The `MaxHeight` format is *valueUnits*, where *Units* is an abbreviation for the height units and *value* is the number of units. See the `Height` property for a table of valid *Units* abbreviations.

**MaxWidth — Maximum width for zoom scaling**

string

Maximum width for zoom scaling, specified as a string. This property applies only if `Scaling` is set to `zoom`.

The `MaxWidth` format is *valueUnits*, where *Units* is an abbreviation for the height units and *value* is the number of units. See the `Height` property for a table of valid *Units* abbreviations.

**TemplateSrc — Source of template for this reporter**

[] (default) | character vector | string scalar | reporter or report | DOM document or document part

Source of the template for this reporter, specified in one of these ways:

- Character vector or string scalar that specifies the path of the file that contains the template for this reporter
- Reporter or report whose template is used for this reporter or whose template library contains the template for this reporter
- DOM document or document part whose template is used for this reporter or whose template library contains the template for this reporter

The specified template must be the same type as the report to which this reporter is appended. For example, for a Microsoft Word report, `TemplateSrc` must be a Word reporter template. If the `TemplateSrc` property is empty, this reporter uses the default reporter template for the output type of the report.

**TemplateName — Name of template for this reporter**

character vector | string scalar

Name of the template for this reporter, specified as a character vector or string scalar. The template for this reporter must be in the template library of the template source (`TemplateSrc`) for this reporter.

**LinkTarget — Hyperlink target for this reporter**[] (default) | character vector | string scalar | `mlreportgen.dom.LinkTarget` object

Hyperlink target for this reporter, specified as a character vector or string scalar that specifies the link target ID, or an `mlreportgen.dom.LinkTarget` object. A character vector or string scalar value is converted to a `LinkTarget` object. The link target immediately precedes the content of this reporter in the output report.

**Methods**

<code>createTemplate</code>	Create diagram template
<code>customizeReporter</code>	Create custom diagram reporter class
<code>getClassFolder</code>	Diagram class definition file location
<code>getSnapshotImage</code>	Diagram snapshot image file location

**Inherited Methods**

copy	Create copy of reporter object and make deep copies of property values that reference a reporter, ReporterLayout, or DOM object
getImpl	Get implementation of reporter

**Examples****Add Top Level of Model Diagram**

Add a snapshot of the top level of the vdp model to a report.

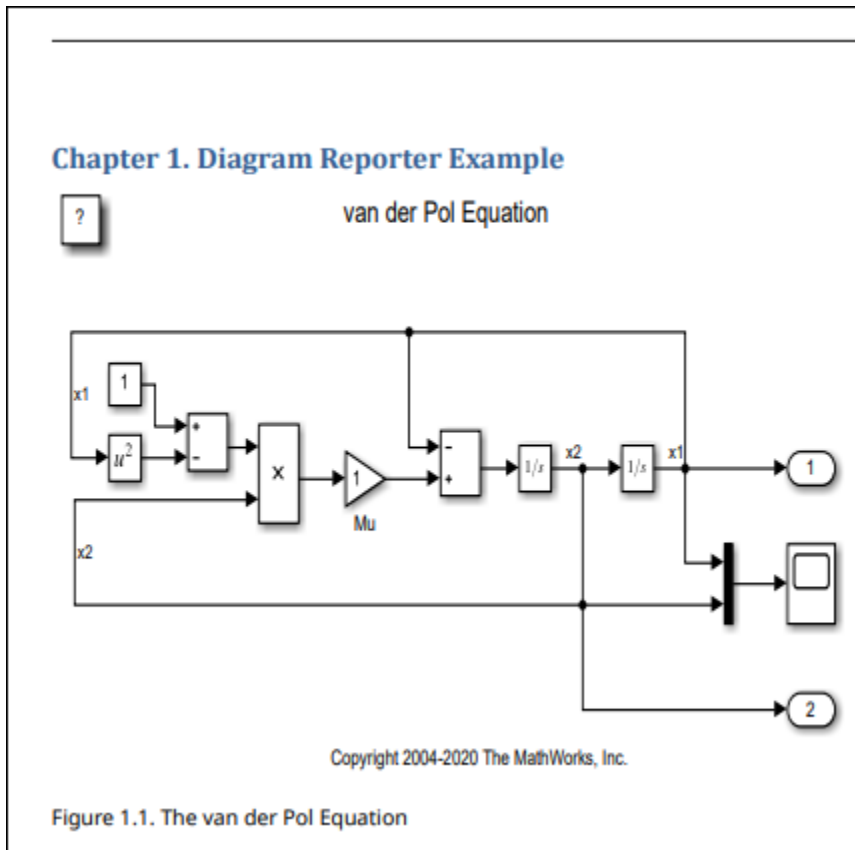
```
load_system('vdp')
import slreportgen.report.*
import mlreportgen.report.*
rpt = slreportgen.report.Report('output','pdf');

chapter = Chapter();
chapter.Title = 'Diagram Reporter Example';

diagram = Diagram("vdp");
diagram.Snapshot.Caption = 'The van der Pol Equation';
diagram.Snapshot.Format = 'svg';
diagram.Snapshot.Height = '4in';

add(chapter,diagram);
add(rpt,chapter);
rptview(rpt);
```





### Add Hyperlinked Diagram to a Report

Create a PDF report and add diagram snapshots of the root system and a subsystem from the `sf_car` model to it. Add a hyperlink to the transmission subsystem and add a paragraph as the target for that link.

```
import mlreportgen.report.*
import slreportgen.report.*
import slreportgen.utils.*
import mlreportgen.dom.*
rpt = slreportgen.report.Report('output','pdf');
chapter = Chapter('sf_car');

load_system('sf_car');
diag1 = Diagram('sf_car');
diag1.Snapshot.Caption = 'Root System: sf_car';
add(chapter,diag1);
add(chapter,PageBreak);

diag2 = Diagram('sf_car/Engine');
diag2.Snapshot.Caption = 'Subsystem: sf_car/Engine';
add(chapter,diag2);
add(chapter, PageBreak);

para = Paragraph('Custom target for sf_car/transmission');
id = getObjectID('sf_car/transmission');
append(para,mlreportgen.dom.LinkTarget(id));
```

```
add(chapter, para);
add(chapter, PageBreak);
```

```
add(rpt, chapter);
close(rpt);
rptview(rpt);
```

## Chapter 1. sf\_car

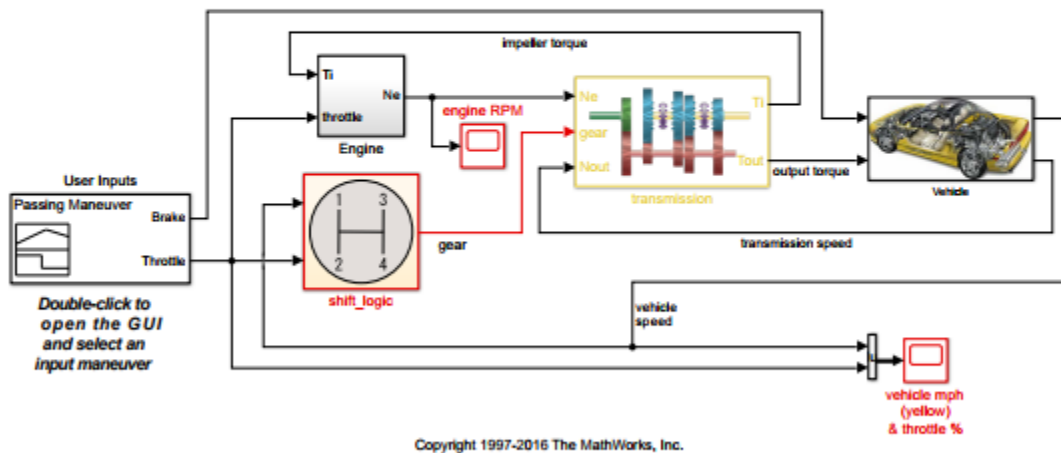


Figure 1.1. Root System: sf\_car

### Take Snapshot of Part of a Diagram

Use the `SnapshotArea` property to specify the area of the diagram to capture in the snapshot. This example sets up the view in the Simulink Editor and then sets the `SnapshotArea` property to that view by calling `slreportgen.utils.getCurrentEditorView`.

Open the model.

```
f14
```

In the Simulink Editor, display the part of the diagram that you want to capture in the snapshot. Get the current Simulink Editor view area by calling `slreportgen.utils.getCurrentEditorView`.

```
editorViewArea = getCurrentEditorView();
```

Create the report and diagram reporter. Set the diagram snapshot area to the current editor viewing area. Add the diagram reporter to the report.

```
import slreportgen.report.*
import slreportgen.utils.*
rpt = Report('output', 'pdf');
```

```
diag = Diagram('f14');  
diag.SnapshotArea = editorViewArea;  
add(rpt, diag);  
  
close(rpt);  
rptview(rpt);
```

## Compatibility Considerations

### Default value of SnapshotFormat is 'svg' for all report types

*Behavior changed in R2019b*

Starting in R2019b, Scalable Vector Graphics (SVG) images are supported for Word reports. For all report types (HTML, PDF, and Word), the default value of the `SnapshotFormat` property is 'svg' and a value of 'auto' indicates 'svg'. In previous releases, the default value of the `SnapshotFormat` property was 'auto', which indicated 'svg' for HTML and PDF reports and 'emf' or 'png' for Word reports, depending on the platform.

Word reports that contain SVG images require Word 2016 or a later version. In MATLAB R2019b or a later release, to generate a report with images that are compatible with earlier versions of Word, set the `SnapshotFormat` property to a value other than 'svg'. To specify the image format used by default in earlier releases of MATLAB, set `SnapshotFormat` to:

- 'emf' for a Windows platform
- 'png' for a UNIX® or Mac platform

## See Also

`slreportgen.finder.AnnotationFinder` | `slreportgen.finder.BlockFinder` |  
`slreportgen.finder.ChartDiagramFinder` | `slreportgen.finder.DiagramElementFinder` |  
`slreportgen.finder.DiagramFinder` | `slreportgen.finder.StateFinder` |  
`slreportgen.finder.StateflowDiagramElementFinder` |  
`slreportgen.finder.SystemDiagramFinder` | `slreportgen.report.Report` |  
`slreportgen.report.SimulinkObjectProperties` |  
`slreportgen.report.StateflowObjectProperties`

## Topics

“Report Generation for Simulink and Stateflow Elements” on page 1-9

“What Is a Reporter?”

## Introduced in R2017b

## slreportgen.report.DocBlock class

**Package:** slreportgen.report

DocBlock reporter

### Description

Creates a DocBlock reporter.

---

**Note** To use a DocBlock reporter in a report, you must create the report using the `slreportgen.report.Report` class or subclass.

---

The reporter adds the DocBlock content or a link to the content to a report, depending on the type of content in the DocBlock and the report type. For Microsoft Word and PDF reports, when a DocBlock contains HTML, you can specify that the reporter links to the content by setting the `ConvertHTML` property to `true`. This table shows when the report includes the content and when it links to the content.

DocBlock Content Type	Report Type	ConvertHTML Property	Report Contains DocBlock Content	Report Contains Link to DocBlock Content
text	HTML	N/A	yes	no
text	Word	N/A	yes	no
text	PDF	N/A	yes	no
HTML	HTML	N/A	yes	no
HTML	Word	true	yes	no
HTML	Word	false	no	yes
HTML	PDF	true	yes	no
HTML	PDF	false	no	yes
RTF	Word	N/A	yes	no
RTF	PDF	N/A	no	yes
RTF	HTML	N/A	no	yes

The `slreportgen.report.DocBlock` class is a `handle` class.

### Class Attributes

`HandleCompatible` `true`

For information on class attributes, see “Class Attributes”.

## Creation

### Description

`rptr = slreportgen.report.DocBlock()` creates an empty DocBlock reporter based on a default template. Customize the content and format of the generated content by using the reporter properties. Before you add the reporter to a report, you must specify the DocBlock in the Object property of the reporter. Adding an empty reporter to a report produces an error.

`rptr = slreportgen.report.DocBlock(docBlockObj)` creates a DocBlock reporter for the DocBlock specified by `docBlockObj`, which can be a DocBlock path or handle. See the Object property.

`rptr = slreportgen.report.DocBlock(Name, Value)` sets the reporter properties using name-value pairs. You can specify multiple name-value pair arguments in any order. Enclose each property name in single or double quotes.

### Properties

#### Object — Simulink DocBlock block

[] (default) | string scalar | character vector | handle

Simulink DocBlock block, specified as a string scalar or character vector that contains the path to a DocBlock or as a handle to a DocBlock block.

---

**Note** If you use a finder to find DocBlock blocks and add the results directly to a report, DocBlock reporters are used to report on the DocBlock blocks rather than Simulink object property reporters.

---

#### ImportTextInline — Whether to import text content in line

false (default) | true

Whether to import plain text content in line, specified as `true` or `false`. If `ImportTextInline` is `false`, before the reporter appends the content to a hole, it wraps the content in one or more paragraphs, depending on the value of the `TexSep` property. Set `ImportTextInline` to `true` to append DocBlock text content to a hole in a paragraph (an inline hole).

#### TextSep — Separator for delimiting paragraphs in text content

"Ignore" (default) | "LineFeed" | "BlankLine"

Separator used to delimit paragraphs in plain text content, specified as one of the values in the table. You can specify the value as a string scalar or a character vector.

Value	Description
"Ignore"	Wrap text in a single paragraph regardless of whether it contains separators. (default)
"LineFeed"	If a text segment ends with a line feed, wrap it in a paragraph.
"BlankLine"	If a text segment ends with a blank line, wrap it in a paragraph.

**ConvertHTML — Whether to include or link to HTML content**`true (default) | false`

`ConvertHTML` applies only to Word and PDF reports. If `ConvertHTML` is `true`, HTML content is converted to DOM objects and appended to a report. If `ConvertHTML` is `false`, HTML content is saved in a file and a link to the file is appended to the report. If the report is an HTML or HTML file report, the HTML content is included in the report, regardless of the value of `ConvertHTML`.

**ParagraphFormatter — Paragraph formatter for plain text**`m1reportgen.dom.Paragraph`

Paragraph formatter object that formats plain text if the `ImportTextInline` property is `false`, specified as an `m1reportgen.dom.Paragraph` object. The initial value of the `ParagraphFormatter` property is an `m1reportgen.dom.Paragraph` object with default property values. To customize the appearance of the paragraph, modify the `m1reportgen.dom.Paragraph` object properties or replace the object with a customized `m1reportgen.dom.Paragraph` object. If you add content to the default or replacement paragraph object, the content appears in front of the `DocBlock` content in the generated report.

**TextFormatter — Text formatter for plain text**`m1reportgen.dom.Text`

Text formatter object that formats plain text if the `ImportTextInline` property is `true`, specified as an `m1reportgen.dom.Text` object. The initial value of the `TextFormatter` property is an `m1reportgen.dom.Text` object with default property values. To customize the appearance of the text, modify the `m1reportgen.dom.Text` object properties or replace the object with a customized `m1reportgen.dom.Text` object. If you add content to the default or replacement paragraph object, the content appears in front of the `DocBlock` content in the generated report.

**TemplateSrc — Source of template for this reporter**`[] (default) | character vector | string scalar | reporter or report | DOM document or document part`

Source of the template for this reporter, specified in one of these ways:

The specified template must be the same type as the report to which this reporter is appended. For example, for a Microsoft Word report, `TemplateSrc` must be a Word reporter template. If the `TemplateSrc` property is empty, this reporter uses the default reporter template for the output type of the report.

**TemplateName — Name of template for this reporter**`character vector | string scalar`

Name of the template for this reporter, specified as a character vector or string scalar. The template for this reporter must be in the template library of the template source (`TemplateSrc`) for this reporter.

**LinkTarget — Hyperlink target for this reporter**`[] (default) | character vector | string scalar | m1reportgen.dom.LinkTarget object`

Hyperlink target for this reporter, specified as a character vector or string scalar that specifies the link target ID, or an `m1reportgen.dom.LinkTarget` object. A character vector or string scalar value is converted to a `LinkTarget` object. The link target immediately precedes the content of this reporter in the output report.

## Methods

### Public Methods

slreportgen.report.DocBlock.createTemplate	Create DocBlock reporter template
slreportgen.report.DocBlock.customizeReporter	Create custom DocBlock reporter class
slreportgen.report.DocBlock.getClassFolder	Get location of DocBlock reporter class definition file
copy	Create copy of reporter object and make deep copies of property values that reference a reporter, ReporterLayout, or DOM object
getImpl	Get implementation of reporter

## Examples

### Include DocBlock Content in a Report

Include the content from the Sensor Info DocBlock of the `sldemo_fuelsys` model in a report by adding a DocBlock reporter to the report. Specify that paragraphs in the DocBlock are delimited by a linefeed.

```
% Import the API package
import slreportgen.report.*
import mlreportgen.report.*

% Load the model
model_name = 'sldemo_fuelsys';
load_system(model_name);
docBlock = 'sldemo_fuelsys/To Controller/Sensor Info';

% Create a report
rpt = slreportgen.report.Report('output','pdf');

% Create a chapter reporter
chapter = Chapter(docBlock);

% Create a DocBlock reporter
% Specify that paragraphs are delimited by a linefeed
rptr = DocBlock(docBlock);
rptr.TextSep = 'LineFeed';

% Add the DocBlock reporter to the chapter
% Add the chapter to the report
add(chapter, rptr);
add(rpt, chapter);

% Close and view the output report
close(rpt);
close_system(model_name);
rptview(rpt);
```

Here is the content from the Sensor Info DocBlock in the generated report:

## Chapter 1. sldemo\_fuelsys/To Controller/Sensor Info

### Measurement Descriptions

=====

Throttle Angle (degrees):

min=3, max=90, precision=0.1

Engine Speed (radians/sec.):

min=0, max=620, precision=.1

Ego Sensor (volts):

min=0, max=12, precision=.05

Manifold Pressure (bar):

min=0.001, max=1, precision=.001

### See Also

`DocBlock | slreportgen.finder.BlockFinder | slreportgen.utils.isDocBlock`

### Topics

“Reporting on DocBlock Blocks” on page 4-37

“Report Generation for Simulink and Stateflow Elements” on page 1-9

“What Is a Reporter?”

### Introduced in R2019b



# slreportgen.report.ElementDiagram class

**Package:** slreportgen.report

Element diagram snapshot and caption reporter

## Description

Create a Simulink or Stateflow element diagram reporter. When added to a report, the `ElementDiagram` reporter creates a snapshot of an element. The reporter adds the snapshot to the report in the form of an image with a caption. Use the “Source” on page 7-0 property to specify the desired element.

---

**Note** To use an `ElementDiagram` reporter in a report, you must create the report using the `slreportgen.report.Report` class.

---

## Construction

`diagram = ElementDiagram()` creates an empty element diagram reporter. Set its properties to capture a Simulink or Stateflow element snapshot.

`diagram = ElementDiagram(source)` creates a diagram reporter for an element of a block diagram or chart specified by `source`. Adding this reporter to a report creates a snapshot of the element's diagram and adds the snapshot, with a caption, to the report. The snapshot image file is stored in the report's temporary folder. When the report is closed, the image file is copied into the report and the temporary folder is deleted. To prevent the deletion, use the `Debug` property of the report. See `mlreportgen.report.Report`.

## Input Arguments

**source** — Diagram element source

character vector | string scalar | handle | object | `slreportgen.finder.DiagramElementResult` object

See the Source on page 7-0 property.

## Properties

**Source** — Diagram element source

character vector | string scalar | handle | object | `slreportgen.finder.DiagramElementResult` object

Diagram element source, specified as one of these values:

- Character vector or string scalar that contains the path to a Simulink block or Stateflow chart block
- Handle to a Simulink block or Stateflow chart block
- Stateflow object

- Simulink Identifier (SID) of a block, annotation, or Stateflow object
- `slreportgen.finder.DiagramElementResult` object

---

**Note** `Simulink.Port` objects are not valid sources for this reporter.

---

### Snapshot — Snapshot reporter

`mlreportgen.report.FormalImage` object

Snapshot reporter, set by default to an object of the `mlreportgen.report.FormalImage` class. You do not need to set this property yourself. The `FormalImage` object adds the element diagram snapshot to a report. To control the size of the snapshot, set its `mlreportgen.report.FormalImage` properties.

### SnapshotFormat — Snapshot image format

'svg' (default) | ...

Snapshot image format, specified as a character vector or string scalar. Supported formats are:

- 'bmp' — Bitmap image.
- 'gif' — Graphics Interchange format.
- 'jpg' — JPEG image.
- 'png' — PNG image.
- 'emf' — Enhanced metafile, supported only in DOCX output on Windows platforms.
- 'svg' — Scalable Vector Graphics.
- 'tif' — Tag Image File format, not supported in HTML output.
- 'pdf' — PDF image.

See “Compatibility Considerations” on page 7-41.

### Scaling — Options for scaling a diagram element image

string | character vector

Options for scaling a diagram element image, specified as a string or character vector. Valid scaling options are:

- `auto` — For PDF or Word (`docx`) output, `auto` scales the element image to fit on a page while maintaining its aspect ratio. First, the element image is scaled to the page width. If the image height exceeds the page height, the image is again scaled down. This additional scaling ensures that the image fits the current page with a 1" margin. The margin allows space for a caption. Scaling does not apply to HTML output.
- `custom` — Sets the element image height and width to the values of this reporter's `Height` and `Width` properties.
- `zoom` — Enlarges or reduces the element image size to the percent value specified by this reporter's `Zoom` property. To specify the maximum image height and maximum image width, use the `MaxHeight` and `MaxWidth` properties, respectively.

---

**Note** A `java.lang.OutOfMemoryError` can occur when either of these combinations of property settings occur:

- `Scaling` set to `zoom`, and `Zoom`, `MaxHeight`, and `MaxWidth` properties set to large values
- `Scaling` set to `custom`, and `Height` and `Width` properties set to large values

To avoid this error, for `zoom Scaling`, use smaller `Zoom`, `MaxHeight`, and `MaxWidth` property values. For `custom Scaling`, use smaller `Height` and `Width` property values. Using smaller values ensures that the diagram fits on the page.

### Height — Height of diagram element

string

Height to set diagram element image, specified as a string. This property applies only if this reporter's `Scaling` property is set to `custom`.

The `Height` format is *valueUnits*, where *Units* is an abbreviation for the height units and *value* is the number of units. The table shows the valid *Units* abbreviations.

Units	Units Abbreviation
pixels	px
centimeters	cm
inches	in
millimeters	mm
picas	pc
points	pt

### Width — Width of diagram element image

string

Width to set diagram element image, specified as a string. This property applies only if this reporter's `Scaling` property is set to `custom`.

The `Width` format is *valueUnits*, where *Units* is an abbreviation for the height units and *value* is the number of units. See the `Height` property for a table of valid *Units* abbreviations.

### Zoom — Amount to zoom diagram element image

string

Amount to zoom the diagram element image, specified as a string. The `Zoom` format is *value%*, where *value* is the percentage by which the diagram element image is enlarged or reduced.

### MaxHeight — Maximum height for zoom scaling

string

Maximum height for zoom scaling, specified as a string. This property applies only if this reporter's `Scaling` property is set to `zoom`. The `MaxHeight` format is *valueUnits*, where *Units* is an abbreviation for the height units and *value* is the number of units. See this reporter's `Height` property for a table of valid *Units* abbreviations.

### MaxWidth — Maximum width for zoom scaling

string

Maximum width for zoom scaling, specified as a string. This property applies only if this reporter's `Scaling` property is set to `zoom`. The `MaxWidth` format is *valueUnits*, where *Units* is an abbreviation

for the height units and *value* is the number of units. See this reporter's `Height` property for a table of valid *Units* abbreviations.

### TemplateSrc — Source of template for this reporter

[] (default) | character vector | string scalar | reporter or report | DOM document or document part

Source of the template for this reporter, specified in one of these ways:

- Character vector or string scalar that specifies the path of the file that contains the template for this reporter
- Reporter or report whose template is used for this reporter or whose template library contains the template for this reporter
- DOM document or document part whose template is used for this reporter or whose template library contains the template for this reporter

The specified template must be the same type as the report to which this reporter is appended. For example, for a Microsoft Word report, `TemplateSrc` must be a Word reporter template. If the `TemplateSrc` property is empty, this reporter uses the default reporter template for the output type of the report.

### TemplateName — Name of template for this reporter

character vector | string scalar

Name of the template for this reporter, specified as a character vector or string scalar. The template for this reporter must be in the template library of the template source (`TemplateSrc`) for this reporter.

### LinkTarget — Hyperlink target for this reporter

[] (default) | character vector | string scalar | `mlreportgen.dom.LinkTarget` object

Hyperlink target for this reporter, specified as a character vector or string scalar that specifies the link target ID, or an `mlreportgen.dom.LinkTarget` object. A character vector or string scalar value is converted to a `LinkTarget` object. The link target immediately precedes the content of this reporter in the output report.

## Methods

<code>createTemplate</code>	Create element diagram template
<code>customizeReporter</code>	Create custom element diagram reporter class
<code>getClassFolder</code>	Element diagram class definition file location
<code>getSnapshotImage</code>	Element diagram snapshot image file location

### Inherited Methods

<code>copy</code>	Create copy of reporter object and make deep copies of property values that reference a reporter, <code>ReporterLayout</code> , or DOM object
<code>getImpl</code>	Get implementation of reporter

## Examples

### Add Element Diagram and Caption

```
import slreportgen.report.*
import mlreportgen.report.*

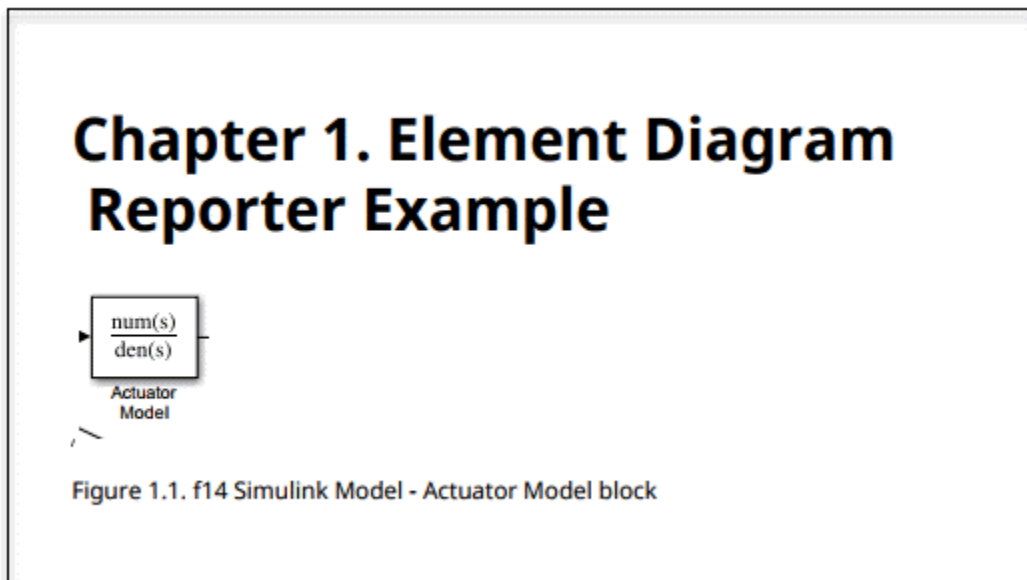
load_system('f14')

rpt = slreportgen.report.Report("output","pdf");
open(rpt)
chap = Chapter();
chap.Title = "Element Diagram Reporter Example";

diag = ElementDiagram("f14/Actuator Model");
diag.Snapshot.Caption = "f14 Simulink Model - Actuator Model block";

add(chap,diag)
add(rpt,chap)

close(rpt)
rptview(rpt)
```



### Add Element Diagrams and Property Tables

```
load_system('f14')
modelsys = "f14/Aircraft Dynamics Model";

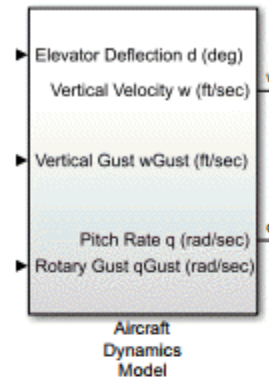
rpt = slreportgen.report.Report("output","pdf");
open(rpt)
chapter = mlreportgen.report.Chapter();
chapter.Title = "Element Snapshots";

diag = slreportgen.report.ElementDiagram(modelsys);
add (chapter,diag)
```

```
blkfinder = slreportgen.finder.BlockFinder(modelsys);
blks = find(blkfinder);
for blk = blks
    blkDiag = slreportgen.report.ElementDiagram...
        (blk.Object);
    blkDiag.Snapshot.Caption = strcat(blk.DiagramPath,...
        "/", blk.Name);
    add(chapter,blkDiag)    % Add diagram element image
    add(chapter,blk)      % Add property table
end

add(rpt,chapter)
close(rpt)
rptview(rpt)
```

# Chapter 1. Element Snapshots



1  
Elevator Deflection d (deg)

Figure 1.1. f14/Aircraft Dynamics Model/Elevator Deflection d (deg)

**Table 1.1. f14/Aircraft Dynamics Model/Elevator Deflection d (deg) Properties**

Property	Value
Type	Block
Block Type	Inport
Port number	1
Port dimensions (-1 for inherited)	-1
Sample time (-1 for inherited)	-1
Data type	Inherit: auto

2  
Vertical Gust wGust (ft/sec)

Figure 1.2. f14/Aircraft Dynamics Model/Vertical Gust wGust (ft/sec)

## Compatibility Considerations

**Default value of SnapshotFormat is 'svg' for all report types**  
*Behavior changed in R2019b*

Starting in R2019b, Scalable Vector Graphics (SVG) images are supported for Word reports. For all report types (HTML, PDF, and Word), the default value of the `SnapshotFormat` property is `'svg'` and a value of `'auto'` indicates `'svg'`. In previous releases, the default value of the `SnapshotFormat` property was `'auto'`, which indicated `'svg'` for HTML and PDF reports and `'emf'` or `'png'` for Word reports, depending on the platform.

Word reports that contain SVG images require Word 2016 or a later version. In MATLAB R2019b or a later release, to generate a report with images that are compatible with earlier versions of Word, set the `SnapshotFormat` property to a value other than `'svg'`. To specify the image format used by default in earlier releases of MATLAB, set `SnapshotFormat` to:

- `'emf'` for a Windows platform
- `'png'` for a UNIX or Mac platform

### **See Also**

`slreportgen.finder.BlockFinder` | `slreportgen.finder.DiagramElementFinder` |  
`slreportgen.finder.StateFinder` |  
`slreportgen.finder.StateflowDiagramElementFinder` | `slreportgen.report.Diagram` |  
`slreportgen.report.Report`

### **Topics**

“Report Generation for Simulink and Stateflow Elements” on page 1-9  
“What Is a Reporter?”

### **Introduced in R2018b**



# slreportgen.report.ExecutionOrder class

**Package:** slreportgen.report

System task and block execution order reporter

## Description

Use an object of the `slreportgen.report.ExecutionOrder` class to report the tasks of a model or nonvirtual subsystem and the blocks in each task, sorted by execution order. By default, an `ExecutionOrder` reporter generates:

- A table of task names and properties
- A list of the blocks in each task

Conditionally executed blocks, such as subsystems triggered by a function call or an If block, are not displayed in the block execution order list. Instead, these blocks are displayed in a `Conditional Execution` table that follows the block execution order list. The table lists the conditionally executed blocks and the blocks that trigger their execution.

Use the `ExecutionOrder` reporter properties to filter the reported content and customize the content formatting.

---

**Note** To use an `slreportgen.report.ExecutionOrder` reporter in a report, you must create the report using the `slreportgen.report.Report` class or subclass. An `ExecutionOrder` reporter does not generate content if it is added to an `slreportgen.report.Report` object that has the `CompileModelBeforeReporting` set to `false`.

---

The `slreportgen.report.ExecutionOrder` class is a `handle` class.

## Class Attributes

`HandleCompatible` true

For information on class attributes, see “Class Attributes”.

## Creation

### Description

`reporter = slreportgen.report.ExecutionOrder()` creates an empty `ExecutionOrder` reporter object based on the default template. You must specify a model or subsystem for which to report the execution order by setting the `Object` property. Use other properties to specify report options.

`reporter = slreportgen.report.ExecutionOrder(system)` creates an `ExecutionOrder` reporter and sets the `Object` property to the specified model or subsystem.

`reporter = slreportgen.report.ExecutionOrder(Name, Value)` sets the reporter properties using name-value pairs. You can specify multiple name-value pair arguments in any order. Enclose each property name in single or double quotes.

## Properties

### Object — Model or nonvirtual subsystem block to report

`[]` (default) | character vector | string scalar | handle | `slreportgen.finder.DiagramResult` object | `slreportgen.finder.BlockResult`

Model or nonvirtual subsystem block to report, specified as one of these types of values:

Specifying a `BlockResult` or `DiagramResult` that represents an unloaded model or a virtual subsystem results in an error.

### ShowTaskDetails — Whether to include a task details table

`true` (default) | `false`

Whether to include a task details table, specified as `true` or `false`. If `ShowTaskDetails` is `true`, the reporter generates a table that displays names and properties of tasks in the specified system.

Data Types: logical

### ShowBlockExecutionOrder — Whether to include block execution order lists

`true` (default) | `false`

Whether to include block execution order lists, specified as `true` or `false`. If `ShowBlockExecutionOrder` is `true`, the reporter includes a list of blocks, sorted in order of execution, for each task in the system. Conditionally executed blocks, such as subsystems triggered by a function call or `If` block, are not displayed in the execution order list. Instead, these blocks are displayed in a `Conditional Execution` table that follows the block execution order list. The table lists the conditionally executed blocks and the blocks that trigger their execution.

Data Types: logical

### TaskProperties — Properties to report for task

`["Order" "Name" "Type" "Trigger" "TaskID" "SourceBlock"]` (default) | string array | cell array of character vectors

Properties to report for each system task, specified as a string array or cell array of character vectors. By default, all properties are included. Valid properties are:

Example: `["Order" "Name" "Type" "Trigger" "TaskID" "SourceBlock"]`

Example: `{'Order' 'Name' 'Type'}`

### ShowEmptyColumns — Whether to show empty columns in task details table

`false` (default) | `true`

Whether to show empty columns in the task details table, specified as `true` or `false`. If `ShowEmptyColumns` is `true`, the task details table includes columns that do not have any data.

Data Types: logical

**ShowBlockType — Whether to show the block type in execution order lists**

true (default) | false

Whether to show the type of each block in the block execution order lists, specified as `true` or `false`. If `ShowBlockType` is `true`, the reporter includes the type of each block next to the block name in the execution order lists.

Data Types: logical

**ShowHiddenBlocks — Whether to show blocks created at compile time**

true (default) | false

Whether to show blocks created at compile time, specified as `true` or `false`. If `ShowHiddenBlocks` is `true`, the reporter includes blocks that Simulink inserts when the model is compiled. If `ShowHiddenBlocks` is `false`, the reporter includes only user-added blocks.

Data Types: logical

**IncludeSubsystemBlocks — Whether to reference block lists of nonvirtual subsystems**

true (default) | false

Whether to reference block lists of nonvirtual subsystems, specified as `true` or `false`. If `IncludeSubsystemBlocks` is `true`, the reporter includes references to nonvirtual subsystem blocks. The `SubsystemBlocksDisplayPolicy` property determines how the nonvirtual subsystem blocks are referenced.

Data Types: logical

**SubsystemBlocksDisplayPolicy — Policy for referencing execution order lists of blocks in nonvirtual subsystems**

"Link" (default) | "NestedList"

Policy for referencing execution order lists of blocks that are in nonvirtual subsystems, specified as one of these string scalars or character vectors:

**TaskFilterFcn — Function or expression to filter system tasks**

[] (default) | function handle | string scalar | character vector

Function or expression to filter system tasks from a report, specified as a function handle, string scalar, or character vector. Specify a function as a function handle. Specify an expression as a string scalar or character vector. If `TaskFilterFcn` is empty, all tasks are included in the report.

If you provide a function handle, the associated function must:

For example, this code uses the `TaskFilterFcn` property to report only periodic tasks:

```
import slreportgen.finder.*
import slreportgen.report.*
import mlreportgen.report.*

model_name = "vdp";
load_system(model_name);

rpt = slreportgen.report.Report("ExecutionOrder_example", "html");

finder = DiagramFinder(model_name);

ch = Chapter("Diagrams");
```

```

while hasNext(finder)
    result = next(finder);
    % Only report block diagrams and nonvirtual subsystems
    if (strcmpi(result.Type,"Simulink.SubSystem")...
        && strcmpi(get_param(result.Object,"IsSubsystemVirtual"),"off")) ...
        || strcmpi(result.Type,"Simulink.BlockDiagram")
        sect = Section(result.Name);
        append(sect,result);
        % Create ExecutionOrder reporter and add to report
        rptr = ExecutionOrder(result);
        % Filter all but periodic tasks
        filterFcnHandle = @(taskName, taskType, trigger, sourceBlock) ...
            ~strcmpi(taskType, "Periodic");
        rptr.TaskFilterFcn = filterFcnHandle;
        append(sect,rptr);

    append(ch,sect);
end
end

append(rpt,ch);
close(rpt);
rptview(rpt);

```

If you provide a string scalar or a character vector, it must contain an expression. The expression:

For example, this code uses the `TaskFilterFcn` property to report only periodic tasks:

```

import slreportgen.finder.*
import slreportgen.report.*
import mlreportgen.report.*

model_name = "vdp";
load_system(model_name);

rpt = slreportgen.report.Report("ExecutionOrder_example","html");

finder = DiagramFinder(model_name);

ch = Chapter("Diagrams");
while hasNext(finder)
    result = next(finder);
    % Only report block diagrams and nonvirtual subsystems
    if (strcmpi(result.Type,"Simulink.SubSystem")...
        && strcmpi(get_param(result.Object,"IsSubsystemVirtual"),"off")) ...
        || strcmpi(result.Type,"Simulink.BlockDiagram")
        sect = Section(result.Name);
        append(sect,result);
        % Create ExecutionOrder reporter and add to report
        rptr = ExecutionOrder(result);
        % Filter all but periodic tasks
        % Code string to include only asynchronous tasks
        filterStr = "isFiltered = ~strcmpi(taskType, ""Periodic"");";
        rptr.TaskFilterFcn = filterStr;
        append(sect,rptr);

    append(ch,sect);
end
end

append(rpt,ch);
close(rpt);
rptview(rpt);

```

### TableReporter — Formatter for task details table

`mlreportgen.report.BaseTable` object

Formatter for the task details table, specified as an `mlreportgen.report.BaseTable` object. The default value of this property is a `BaseTable` object with the `TableStyleName` property set to the

ExecutionOrderTable style, which is defined in the default template for an ExecutionOrder reporter. To customize the appearance of the table, modify the properties of the default BaseTable object or replace the object with your own BaseTable object. If you add content to the Title property of the BaseTable object, the content appears in front of the table title in the generated report.

### ListFormatter — Formatter for block execution order lists

mlreportgen.dom.OrderedList object | mlreportgen.dom.UnorderedList object

Formatter for the block execution order lists, specified as an mlreportgen.dom.OrderedList object or mlreportgen.dom.UnorderedList object. The OrderedList or UnorderedList object must not contain list items.

The default value of this property is an OrderedList object with the StyleName property set to the ExecutionOrderList style, which is defined in the default template for an ExecutionOrder reporter. To customize the appearance of the list, modify the properties of the default OrderedList object or replace the object with a your own OrderedList or UnorderedList object.

### TemplateSrc — Source of template for this reporter

[] (default) | character vector | string scalar | reporter or report | DOM document or document part

Source of the template for this reporter, specified in one of these ways:

The specified template must be the same type as the report to which this reporter is appended. For example, for a Microsoft Word report, TemplateSrc must be a Word reporter template. If the TemplateSrc property is empty, this reporter uses the default reporter template for the output type of the report.

### TemplateName — Name of template for this reporter

character vector | string scalar

Name of the template for this reporter, specified as a character vector or string scalar. The template for this reporter must be in the template library of the template source (TemplateSrc) for this reporter.

### LinkTarget — Hyperlink target for this reporter

[] (default) | character vector | string scalar | mlreportgen.dom.LinkTarget object

Hyperlink target for this reporter, specified as a character vector or string scalar that specifies the link target ID, or an mlreportgen.dom.LinkTarget object. A character vector or string scalar value is converted to a LinkTarget object. The link target immediately precedes the content of this reporter in the output report.

## Methods

### Public Methods

slreportgen.report.ExecutionOrder.createTemplate	Create execution order reporter template
slreportgen.report.ExecutionOrder.customizeReporter	Create custom execution order reporter class
slreportgen.report.ExecutionOrder.getClassFolder	Get location of execution order reporter class definition file

copy	Create copy of reporter object and make deep copies of property values that reference a reporter, ReporterLayout, or DOM object
getImpl	Get implementation of reporter

## Examples

### Report System Tasks and Task Blocks in Execution Order

For each block diagram or virtual subsystem of the vdp model, report the system tasks and the blocks in each task, in execution order.

Import the MATLAB and Simulink Report API packages so that you do not have to use long, fully qualified class names.

```
import mlreportgen.report.*
import slreportgen.finder.*
import slreportgen.report.*
```

Open the model and create a report.

```
model_name = 'vdp';
load_system(model_name);

rpt = slreportgen.report.Report("ExecutionOrder_example", "pdf");
```

Create a finder to find all of the diagrams in the model. Create a Diagrams chapter.

```
finder = DiagramFinder(model_name);
ch = Chapter("Diagrams");
```

For each diagram that is a block diagram or a nonvirtual subsystem, report the system tasks and blocks in execution order, using the default values of the `slreportgen.report.ExecutionOrder` reporter properties.

```
while hasNext(finder)
    result = next(finder);
    if (strcmpi(result.Type, "Simulink.SubSystem") &&...
        strcmpi(get_param(result.Object, "IsSubsystemVirtual"), "off")) ...
        || strcmpi(result.Type, "Simulink.BlockDiagram")
        sect = mlreportgen.report.Section(result.Name);
        append(sect, result);
        rptr = slreportgen.report.ExecutionOrder(result);
        append(sect, rptr);
        append(ch, sect);
    end
end
```

Append the chapter to the report. Close and view the report.

```
append(rpt, ch);
close(rpt);
rptview(rpt);
```

## **See Also**

`slreportgen.finder.BlockResult` | `slreportgen.finder.DiagramResult`

## **Topics**

“Report Generation for Simulink and Stateflow Elements” on page 1-9

“What Is a Reporter?”

**Introduced in R2020b**

## slreportgen.report.LookupTable class

**Package:** slreportgen.report

Lookup table block reporter

### Description

Create a Simulink lookup table block reporter. See the Object property for a list of supported blocks.

---

**Note** To use a LookupTable reporter in a report, you must create the report using the slreportgen.report.Report class.

---

### Construction

`rptr = LookupTable()` creates an empty LookupTable block reporter based on a default template. Use its properties to specify the lookup table block on which to report and specify report options.

`rptr = LookupTable(lutobj)` creates a LookupTable block reporter for the lookup table block specified by lutobj. By default, the reporter generates a table and a plot of output values versus breakpoints, and a data types table.

`rptr = LookupTable(Name, Value)` sets LookupTable reporter properties using name-value pairs. You can specify multiple name-value pair arguments in any order. Enclose each property name in single or double quotes.

### Input Arguments

**lutobj** — Lookup table block object

path | handle

See the Object property.

### Properties

**Object** — Lookup table block

path | handle

Lookup table block to report on, specified as the path or handle of the block.

These lookup table blocks are supported.

- 1-D Lookup Table
- 2-D Lookup Table
- n-D Lookup Table
- Interpolation Using Prelookup
- Direct Lookup Table (n-D)



- Lookup Table Dynamic

---

**Note** If you use a finder to find a Lookup Table block and add it directly to a report, this `LookupTable` reporter is used rather than a Simulink object property reporter.

---

### **IncludeTable — Include table of lookup table data**

`true` (default) | `false`

Whether to include a table of lookup table data, specified as a logical. If `true`, the report includes a table that lists output values and breakpoints of the lookup table.

For a 1-D lookup table, the data table lists the breakpoints in the first column of the table. The second column lists the corresponding output values.

For a 2-D or greater dimension lookup table, the data table lists the first set of breakpoints in the first row of the table. It lists the second set of breakpoints in the first column. The output appears in the corresponding table cells. For lookup tables greater than 2-D, the `LookupTable` report generator shows slices of the table as separate output versus breakpoint tables.

### **IncludePlot — Include lookup table data plot**

`true` (default) | `false`

Whether to include a lookup table data plot, specified as a logical. If `true`, the report includes a plot of the output values versus the breakpoints of the lookup table block.

For a 1-D lookup table, the plot is a line plot of output values versus breakpoints.

For 2-D tables and slices, the plot is a surface or mesh plot of output values versus breakpoints. Use the `PlotType` property to specify whether to use a surface or mesh plot.

### **PlotType — Plot type for 2-D lookup table data plot**

'Surface Plot' (default) | 'Mesh Plot'

Plot type for a 2-D lookup table data plot, specified as either a character array ('Surface Plot' or 'Mesh Plot') or string ("Surface Plot" or "Mesh Plot").

### **DataReporter — Lookup table data reporter**

`BaseTable reporter` (default) | `custom reporter`

Lookup table data reporter, specified as a `BaseTable` reporter or custom reporter. The `LookupTable` reporter uses the specified reporter to create a table of lookup table data.

To customize the appearance of the table of lookup table data, customize the default `BaseTable` reporter or replace it with a custom version of the `BaseTable` reporter.

To customize the title of the table of lookup table data, specify the contents in the `Title` property of the default or replacement reporter. The content you specify is placed at the front of the default title. If the lookup table is too wide to fit legibly on a page, use the `MaxCols` property of the `DataReporter` to generate the table as a set of slices that fit legibly. To determine an optimal value, iterate setting the `MaxCol` value and viewing the report .

Example: `lutable.DataReporter.Title = 'New Title'`

**PlotReporter — Lookup table data plot reporter**

Figure reporter (default) | custom reporter

Lookup table data plot reporter, specified as a `Figure` reporter or custom reporter. The `LookupTable` reporter uses the specified reporter to create a plot of output values versus breakpoints of the lookup table.

To customize the appearance of the plot of the lookup table data, customize the default `Figure` reporter or replace it with a custom version of the `Figure` reporter. To customize the caption of the plot, specify the contents in the `Caption` property of the default or replacement reporter. The content you specify is placed at the front of the default caption.

**MaxTableColumns — Maximum table columns to display**`inf` (default) | integer

Maximum number of table columns to display for the output values versus breakpoints, specified as `inf` or an integer. This property applies only if the `IncludeTable` property is true.

If the number of lookup table columns is greater than the value of this property, the data is shown only as a surface plot. The plot appears only if the `IncludePlot` property is true.

The default value of this property is `inf`, which causes the reporter to use a table regardless of the size of the lookup table data array. Depending on the size of the data being displayed, some tables can be illegible. To avoid creation of illegible tables, change the default setting of this property to a smaller value.

**TemplateSrc — Source of template for this reporter**

[] (default) | character vector | string scalar | reporter or report | DOM document or document part

Source of the template for this reporter, specified in one of these ways:

- Character vector or string scalar that specifies the path of the file that contains the template for this reporter
- Reporter or report whose template is used for this reporter or whose template library contains the template for this reporter
- DOM document or document part whose template is used for this reporter or whose template library contains the template for this reporter

The specified template must be the same type as the report to which this reporter is appended. For example, for a Microsoft Word report, `TemplateSrc` must be a Word reporter template. If the `TemplateSrc` property is empty, this reporter uses the default reporter template for the output type of the report.

**TemplateName — Name of template for this reporter**

character vector | string scalar

Name of the template for this reporter, specified as a character vector or string scalar. The template for this reporter must be in the template library of the template source (`TemplateSrc`) for this reporter.

**LinkTarget — Hyperlink target for this reporter**[] (default) | character vector | string scalar | `mlreportgen.dom.LinkTarget` object

Hyperlink target for this reporter, specified as a character vector or string scalar that specifies the link target ID, or an `mlreportgen.dom.LinkTarget` object. A character vector or string scalar

value is converted to a `LinkTarget` object. The link target immediately precedes the content of this reporter in the output report.

## Methods

<code>createTemplate</code>	Create Simulink lookup table block reporter template
<code>customizeReporter</code>	Create custom <code>LookupTable</code> reporter class
<code>getClassFolder</code>	Lookup Table reporter class definition file location

## Inherited Methods

<code>copy</code>	Create copy of reporter object and make deep copies of property values that reference a reporter, <code>ReporterLayout</code> , or DOM object
<code>getImpl</code>	Get implementation of reporter

## Examples

### Report on Lookup Table Block

Create a PDF report generator that reports on a lookup table block. This example uses the `sf_car` model and reports on its `engine torque` Lookup Table (n-D) block. This block is a 2-D lookup table. The `engine torque` block is in the `Engine` subsystem of the `sf_car` model. The report, by default, includes a table of output values versus breakpoints, a surface plot, a table of block data types, and notes about possible differences between reported values and values obtained from simulation.

```
import slreportgen.report.*
import mlreportgen.report.*

model_name = 'sf_car';
load_system(model_name)
lutable = 'sf_car/Engine/engine torque';

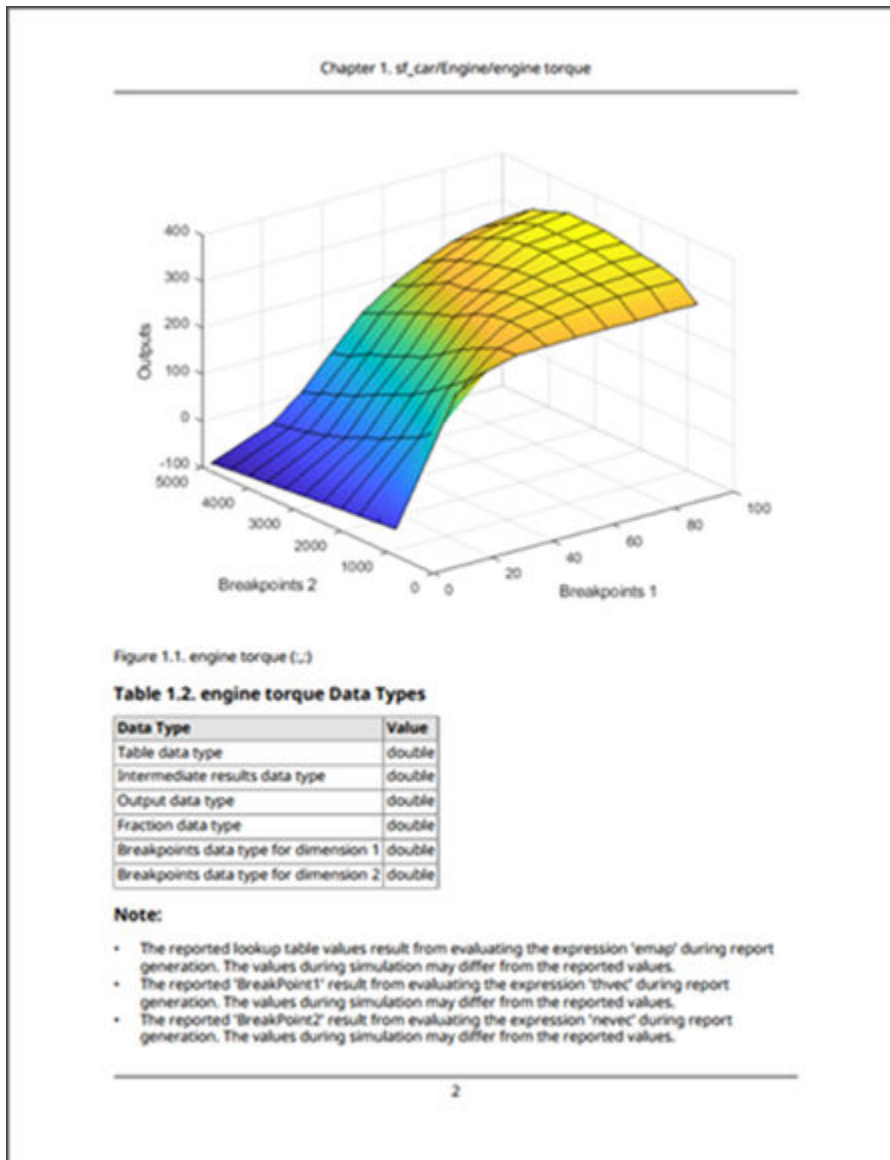
rpt = slreportgen.report.Report('output','pdf');
chapter = Chapter(lutable);
rptr = LookupTable(lutable);
add(chapter,rptr)
add(rpt,chapter)

close(rpt)
close_system(model_name)
rptview(rpt)
```

## Chapter 1. sf\_car/Engine/engine torque

Table 1.1. engine torque (:,:)

	BP2										
	800.0000	1200	1.600 0e+03	2.000 0e+03	2400	2.800 0e+03	3.200 0e+03	3.600 0e+03	4.000 0e+03	4400	4800
<b>0</b>	-40	-44	-49	-53	-57	-61	-65	-70	-74	-78	-82
<b>20</b>	215	117	85	66	44	29	10	-2	-13	-22	-32
<b>30</b>	245	208	178	148	122	104	85	66	48	33	18
<b>40</b>	264	260	241	219	193	167	152	133	119	96	85
<b>50</b>	264	279	282	275	260	238	223	208	189	171	152
<b>60</b>	267	290	293	297	290	275	260	256	234	212	193
<b>70</b>	267	297	305	305	305	301	293	282	267	249	226
<b>80</b>	267	301	308	312	319	323	319	316	297	279	253
<b>90</b>	267	301	312	319	327	327	327	327	312	293	267
<b>100</b>	267	301	312	319	327	334	334	334	319	305	275



### Change Lookup Table Plot Height and Width

Create a PDF report generator that specifies the plot height and width returned by the LookupTable reporter. This example uses the sf\_car model and reports on its Torque ratio Lookup Table (n-D) block. This block is a 1-D lookup table and is in the transmission/Torque Converter subsystem of the sf\_car model. To set the height and width of the plot, use the PlotReporter property.

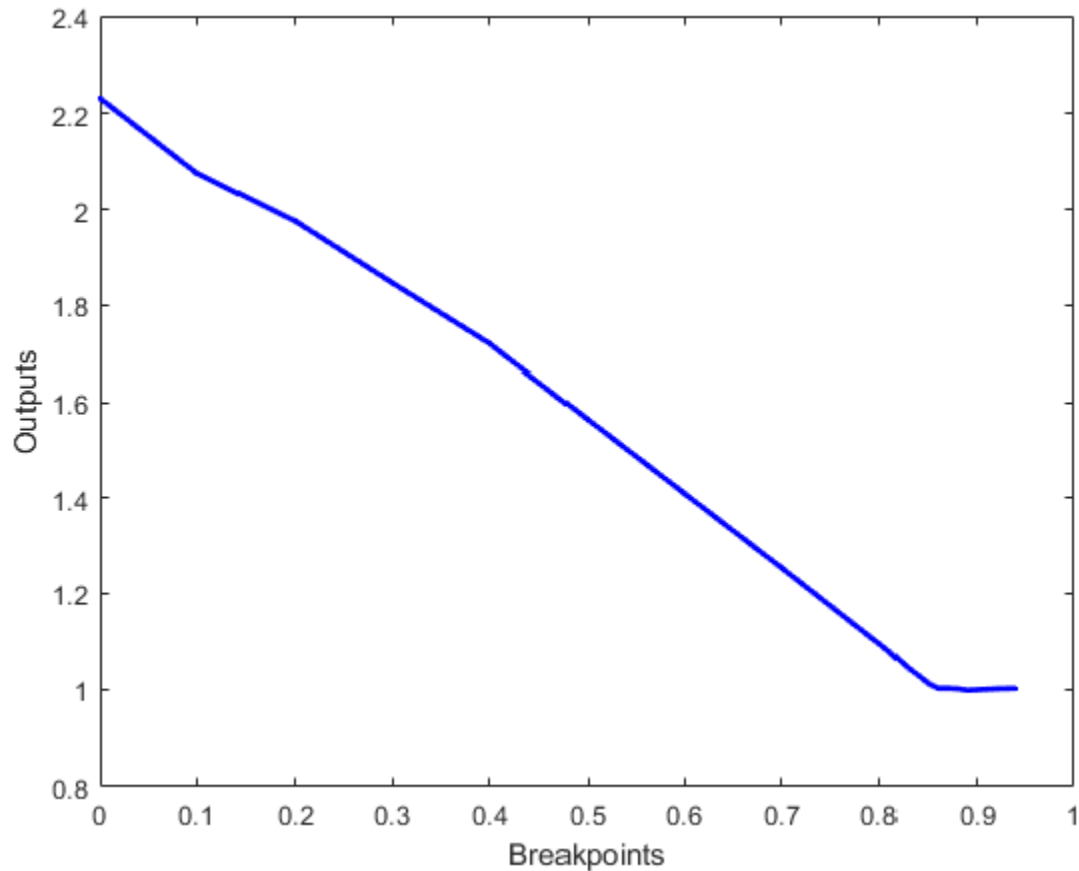
```
import slreportgen.report.*
import mlreportgen.report.*

model_name = 'sf_car';
load_system(model_name);
lutable = 'sf_car/transmission/Torque Converter/Torque ratio';

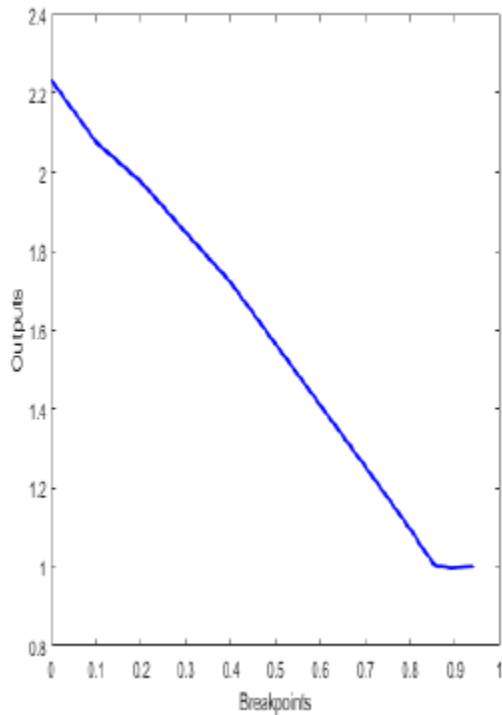
rpt = slreportgen.report.Report('output','pdf');
chapter = Chapter(lutable);
```

```
rptr = LookupTable(lutable);  
rptr.IncludeTable = false;  
add(chapter, rptr)  
  
rptr_resized = LookupTable(lutable);  
rptr_resized.IncludeTable = false;  
rptr_resized.PlotReporter.Snapshot.Width = '3in';  
rptr_resized.PlotReporter.Snapshot.Height = '4in';  
add(chapter, rptr_resized)  
  
add(rpt, chapter)  
  
close(rpt)  
close_system(model_name)  
rptview(rpt)
```

The default plot on the first page of the report uses predefined sizing to fit the plot to the page size.



The resized plot on the second page of the report uses the specified 3" width and 4" height.



### Find and Report on Lookup Tables and Other Blocks

Create a PDF report generator that finds all blocks in the Engine subsystem of the `sf_car` model. The report generator program then loops through the blocks and tests whether the block is a lookup table block. For lookup table blocks, it uses the `LookupTable` reporter to report information about the block. For other blocks, the generated report reports on block properties, which are the results of the `BlockFinder` class.

```
import slreportgen.report.*
import slreportgen.finder.*

model_name = 'sf_car';
load_system(model_name)
subsys_name = 'sf_car/Engine';
rpt = slreportgen.report.Report;

blkfinder = BlockFinder(subsys_name);
blks = find(blkfinder);

for i=1:length(blks)
    if slreportgen.utils.isLookupTable(blks(i).Object)
        rptr = LookupTable(blks(i).Object);
        ch = Chapter(blks(i).Name);
        add(ch,rptr)
        add(rpt,ch)
    else
        ch = Chapter(blks(i).Name);
        add(ch,blks(i))
        add(rpt,ch)
    end
end
```

```

end

close(rpt)
close_system(model_name)
rptview(rpt)

```

The first chapter shows the default properties reported for Inport block.

## Chapter 1. Ti

**Table 1.1. sf\_car/Engine/Ti Properties**

Property	Value
Type	Block
Block Type	Inport
Port number	1
Port dimensions (-1 for inherited)	-1
Sample time (-1 for inherited)	-1
Data type	Inherit: auto

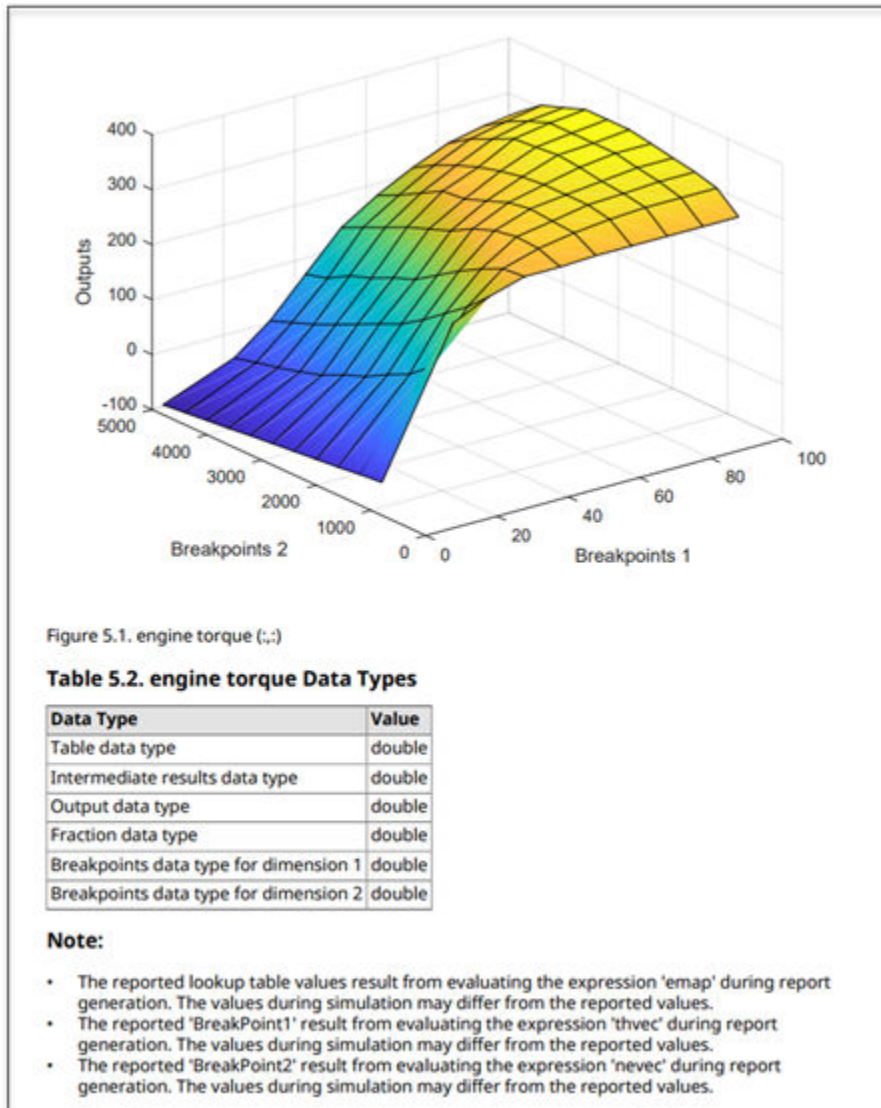
The fifth chapter shows the default output for a Lookup Table block reporter. The default output is a table of output values versus breakpoints table, a plot, and a table of data types.



# Chapter 5. engine torque

Table 5.1. engine torque (:,:)

		BP2										
		800.0000	1200	1.600 0e+03	2.000 0e+03	2400	2.800 0e+03	3.200 0e+03	3.600 0e+03	4.000 0e+03	4400	4800
BP1	0	-40	-44	-49	-53	-57	-61	-65	-70	-74	-78	-82
	20	215	117	85	66	44	29	10	-2	-13	-22	-32
	30	245	208	178	148	122	104	85	66	48	33	18
	40	264	260	241	219	193	167	152	133	119	96	85
	50	264	279	282	275	260	238	223	208	189	171	152
	60	267	290	293	297	290	275	260	256	234	212	193
	70	267	297	305	305	305	301	293	282	267	249	226
	80	267	301	308	312	319	323	319	316	297	279	253
	90	267	301	312	319	327	327	327	327	312	293	267
	100	267	301	312	319	327	334	334	334	319	305	275



## Customize LookupTable Reporter Output

This example shows how to add fixed content to a customized HTML LookupTable reporter template. You can also customize your lookup table report output by editing the report generator program directly. The advantage of customizing a template is that you can reuse it as a basis for customizing another report generator program.

The template and style sheets for the LookupTable reporter are located in the `matlab\toolbox\shared\slreportgen\rpt\rpt\+slreportgen\+report\@LookupTable\resources\templates` folder. You do not need to specify this path when you copy the default template.

- 1 Create a copy of the default html template. In this example, the template package is saved as a zipped file named `CustomTemplate.htm` in the current working folder.

```
import mlreportgen.report.*
import slreportgen.report.*
```

```
LookupTable.createTemplate('CustomTemplate','html');
```

- 2 Unzip the template package.

```
unzipTemplate('CustomTemplate.htmtx');
```

The unzipped template package is a folder of document, style sheet, and image files. In this example, the unzipped folder of files is named "CustomTemplate" and is saved in the current working folder. The `root.css` file, which is in the `stylesheets` subfolder, defines the styles that control the appearance and formatting of the generated report. The `docpart_templates.html` file specifies the holes that hold the report contents when the report is generated.

- 3 From the `CustomTemplate` folder, open the `docpart_templates.html` file in a text editor outside of MATLAB.

```
<html>
  <head>
    <meta charset="utf-8" />
    <title>Document Part Templates</title>
    <link rel="StyleSheet" href="./stylesheets/root.css" type="text/css" />
  </head> <body>
    <dplibrary>

      <!-- NOTE: temporary in the template library until the -->
      <!  DOM supports a source without template name -->
      <dptemplate name="LookupTable">
        <hole id="Content">LUT_CONTENT</hole>
        <hole id="LUTDataTypes">DATA_TYPE</hole>
        <hole id="FootNoteContent">LUT_FOOTNOTE_CONTENT</hole>
      </dptemplate>
      <dptemplate name="LookupTableContent">
        <hole id="TableContent">TABLECONTENT</hole>
        <hole id="FigureContent">FIGURECONTENT</hole>
      </dptemplate>
    </dplibrary>
  </body>
</html>
```

- 4 To add fixed text to the template, place it in the desired location and use standard HTML tagging. This example adds text that appears above the data types table in the generated report. Only the `<dptemplate name="LookupTable">` portion of the file is shown.

```
<dptemplate name="LookupTable">
  <hole id="Content">LUT_CONTENT</hole>
  <p><scan>This lookup table block contains the following
    data types:</scan></p>
  <hole id="LUTDataTypes">DATA_TYPE</hole>
  <hole id="FootNoteContent">LUT_FOOTNOTE_CONTENT</hole>
</dptemplate>
```

- 5 Save the file.
- 6 At the MATLAB command line, zip the template folder into a template package. For this example, the template package is zipped to the `CustomTemplate.htmtx` file.

```
zipTemplate('CustomTemplate');
```

- 7 To use the saved template for your report, specify the template source in your report generator program.

```
lutable = LookupTable();  
lutable.TemplateSrc = 'CustomTemplate';
```

**See Also**

`mlreportgen.report.BaseTable` | `mlreportgen.report.Figure` |  
`slreportgen.finder.BlockFinder` | `slreportgen.finder.BlockResult` |  
`slreportgen.finder.DiagramElementFinder` |  
`slreportgen.finder.DiagramElementResult` | `slreportgen.utils.isLookupTable`

**Topics**

“Report Generation for Simulink and Stateflow Elements” on page 1-9

“What Is a Reporter?”

“Templates”

**Introduced in R2018a**

# slreportgen.report.MATLABFunction class

**Package:** slreportgen.report

MATLAB Function block or Stateflow MATLAB function reporter

## Description

Create a MATLAB Function block or Stateflow MATLAB function reporter.

---

**Note** To use a MATLABFunction reporter in a report, you must create the report using the slreportgen.report.Report class.

---

## Construction

`reporter = MATLABFunction()` creates an empty MATLABFunction reporter based on a default template. Use its properties to specify the Simulink MATLAB Function block or a Stateflow MATLAB function on which to report and specify report options.

`reporter = MATLABFunction(mlfcnobj)` creates a MATLABFunction reporter for the specified mlfcnobj. This reporter adds this default information to the generated report:

- Simulink MATLAB Function block properties or Stateflow MATLAB function properties, depending on whether the MATLAB function is a block or object
- Function input and output argument summary
- MATLAB code used by the MATLAB function to compute its outputs from its inputs

Use the reporter properties to include other information, such as detailed argument properties, function symbol properties, and supporting functions information.

`reporter = MATLABFunction(Name, Value)` sets MATLABFunction reporter properties using name-value pairs. You can specify multiple name-value pair arguments in any order. Enclose each property name in single or double quotes.

## Input Arguments

### **mlfcnobj** — Simulink MATLAB Function block or Stateflow MATLAB function

Simulink MATLAB Function block path | Simulink MATLAB Function block handle | Stateflow MATLAB function handle

See the Object property.

## Properties

### **Object** — Simulink MATLAB Function block or Stateflow MATLAB function

MATLAB Function Function block path | MATLAB Function block handle | Stateflow MATLAB function handle

Simulink MATLAB function block or Stateflow MATLAB function whose properties to report, specified as a path or handle.

**Note** If you use a finder to find a Lookup Table block and add it directly to a report, this LookupTable reporter is used rather than a Simulink object property reporter.

---

### **IncludeObjectProperties** – Include object properties

true (default) | false

Whether to include object properties, specified as a logical. If true, the report includes a table of the properties of a MATLAB function. If false, the property table is not included.

### **ObjectPropertiesReporter** – Object properties reporter

BaseTable reporter (default) | custom reporter

Object properties reporter, specified as a BaseTable reporter or a custom reporter. The MATLABFunction reporter uses the specified reporter to create a table of the properties of the MATLAB function. The properties to be reported depend on whether the MATLAB function is a Simulink MATLAB Function block or a Stateflow MATLAB function.

For a Simulink MATLAB Function block, these properties are reported:

- Update Method
- Sample Time
- Support variable-size arrays
- Saturate on integer overflow
- Treat these inherited Simulink signal types as fi objects
- MATLAB Function block fimath
- Input fi math
- Description

For a Stateflow MATLAB function, these properties are reported:

- Saturate on integer overflow
- MATLAB function fimath
- Input fi math
- Description

To customize the appearance of the function property table and its title, customize the default BaseTable reporter or replace it with a custom version of the BaseTable reporter. To customize the title of the function property table, specify the contents in the Title property of the default or replacement reporter. The content you specify is placed at the front of the default title.

### **IncludeArgumentSummary** – Include summary of arguments

true (default) | false

Whether to include an argument summary table, specified as a logical. If true, the report includes a summary table of the argument properties of the MATLAB Function. If false, the report does not include an argument summary table.

### **ArgumentSummaryProperties** – Argument properties to include

cell array of character vectors

Argument properties to include in the MATLAB Function argument summary table, specified as a cell array of property name character vectors. The default properties included in the summary are Name, Scope, Port, Compiled Type, and Compiled Size. If the model is not already compiled, the MATLABFunction reporter compiles the model to obtain Compile Type and Compiled Size data. When the report is closed, the model is uncompiled.

The value of this property can be any combination of these MATLAB Function argument properties.

- Name
- Scope
- Port
- Compiled Type (Data Type)
- Compiled Size (Size)
- Complexity
- Description
- Maximum (Max Value)
- Minimum (Min Value)
- Tunable
- Variable Size

### **ArgumentSummaryReporter — Summary of arguments reporter**

BaseTable reporter (default) | custom reporter

Argument summary reporter, specified as a BaseTable reporter or a custom reporter. The MATLABFunction reporter uses the specified reporter to create a table of the arguments of the MATLAB function. The table includes a summary of arguments and the argument properties specified by ArgumentSummaryProperties.

To customize the appearance of the argument summary table and the content of its title, customize the default BaseTable reporter or replace it with a custom version of the BaseTable reporter.

### **IncludeArgumentProperties — Include details for each argument**

false (default) | true

Whether to include a property table with detailed information for each argument, specified as a logical. If false, the report does not include a property table for each argument in the MATLAB function. If true, the report includes the property table, and if the model is not already compiled, the MATLABFunction reporter compiles the model. When the report is closed, the model is uncompiled.

### **ArgumentPropertiesReporter — Argument properties reporter**

BaseTable (default) | custom reporter

Argument properties reporter, specified as a BaseTable reporter or a custom reporter. The MATLABFunction reporter uses the specified reporter to create a table of the argument properties of the MATLAB function. The MATLABFunction reporter creates and uses a copy of this reporter for each argument property to be reported.

To customize the appearance of the argument property table, customize the default BaseTable reporter or replace it with a customized version of the BaseTable reporter.

**IncludeFunctionScript — Include function script**`true (default) | false`

Whether to include the function script, specified as a logical. If `true`, the report includes the MATLAB function script that computes the output of the MATLAB Function block from its inputs. If `false`, the report does not include the function script.

**FunctionScript — Function script code format**`mlreportgen.dom.Paragraph (default)`

Function script code format, specified as an `mlreportgen.dom.Paragraph` paragraph. To customize the function script format, such as the font family, font size, alignment, and other attributes, set its `mlreportgen.dom.Paragraph` properties. The `mlreportgen.dom.Paragraph` does not control whether the reporter uses colors to highlight script syntax. See the `HighlightScriptSyntax` to control the highlighting.

To customize the appearance of the script, modify the properties of the default paragraph or replace the paragraph with another paragraph object. If you add content to the default or replacement paragraph, that content is placed before the MATLAB Function script in the generated report.

**FunctionScriptTitle — Function script title**`mlreportgen.dom.paragraph`

Function script title, specified as an `mlreportgen.dom.Paragraph`, which contains the title for the section that contains the MATLAB function script. The default title is the MATLAB Function block name followed by "Function Script," which is appended to the paragraph. For example, for a MATLAB Function block named "Covariance Derivative," the title is "Covariance Derivative Function Script." The properties of the paragraph specify the appearance of the script title.

To customize the title appearance, modify the properties of the default paragraph object or replace it with another paragraph object. If you add content to the default or replacement title paragraph, the content you specify is placed at the front of the default title.

**HighlightScriptSyntax — Highlight script syntax keywords**`true (default) | false`

Highlight script syntax keywords, specified as a logical. If `true`, the report uses colors to highlight script syntax keywords. If `false`, the report does not highlight keywords.

**IncludeFunctionSymbolData — Include function symbol data**`false (default) | true`

Whether to include function symbol data, specified as a logical.

If `false`, the report does not include information about the symbols that appear in the main MATLAB function script. If `true`, the report includes the symbol data information. If the `IncludeSupportingFunctions` property is also `true`, the report also contains information on symbols that appear in supporting functions. Function symbol data is reported only if the source Object property of this reporter is a Simulink MATLAB Function block.

---

**Note** If you include function symbol data, report generation can be slower than if you do not include it.

---



**FunctionSymbolReporter — Function symbol data reporter**

BaseTable (default) | custom reporter

Function symbol data reporter, specified as an `mlreportgen.report.BaseTable`. This reporter is the basis for reporting the properties of symbols that appear in the main script of the MATLAB function. The symbols in the generated report are grouped by types, which are variable, operation, and function-call site. The table that lists each symbol type appears after the function properties. The symbol tables list all the symbols of particular types that appear in the main function or specific supporting functions. The report includes supporting function symbols only if

`IncludeSupportingFunctions` is `true`. The `MATLABFunction` reporter creates a copy of this reporter for each symbol to be reported and uses the copy to report on the symbol properties. The reported function and symbol properties are:

Function Properties	Description
Function Name	Name of the function
Function ID	ID of the function. Simulink assigns a unique ID to every MATLAB Function in a model and to every supporting function. A built-in or user-defined supporting function uses its same ID, regardless of how many functions it supports in a given model.
Path	Path of the function, which is the model path of the MATLAB Function block or Stateflow block that contains it. The path of a supporting function is the path of the MATLAB file that defines it.

Variable Properties	Description
Name	Name of the variable
Data type	Data type and the size of the variable
Start position	Line and column number of the first character of the variable name in the script in which it appears

Operation Properties	Description
Type	Character or characters that represent the operation type. For example, +
Data type	Data type and size of the value produced by the operation
Start position	Line and column number of the first character of the operation in the script in which it appears

Function Call Site Properties	Description
Name	Name of the called function
Data type	Data type and the size of the value returned by the called function
ID	ID of the called function

Function Call Site Properties	Description
Start position	Line and column number of the first character of the call site in the script in which it appears

To customize the appearance of the function symbol data tables, customize the default `BaseTable` reporter or replace it with a customized version of the `BaseTable` reporter. If you specify `Title` property content in the default or replacement `BaseTable` reporter, that content is placed at the front of the default title in the generated report.

### **IncludeSupportingFunctions — Include supporting functions**

`false` (default) | `true`

Whether to include supporting functions, specified as a logical. If `false`, the report does not include a list of functions invoked directly or indirectly by the function script. If `true`, the report includes the supporting functions.

### **SupportingFunctionsType — Supporting function types**

cell array of character vectors (default)

Supporting function types to be reported, specified as a cell array. The cell array can include one or both of these character vectors.

- 'MATLAB' — Include only MATLAB supporting functions
- 'user-defined' — Include only user-defined supporting functions

### **SupportingFunctionsReporter — Supporting functions reporter**

`BaseTable` (default) | custom reporter

Supporting functions reporter, specified as an `BaseTable`, which lists the functions invoked directly or indirectly by the MATLAB function script. The functions are sorted by function name.

To customize the appearance of the supporting functions table, customize the default `BaseTable` reporter or replace it with a customized version of the `BaseTable` reporter.

### **TemplateSrc — Source of template for this reporter**

[] (default) | character vector | string scalar | reporter or report | DOM document or document part

Source of the template for this reporter, specified in one of these ways:

- Character vector or string scalar that specifies the path of the file that contains the template for this reporter
- Reporter or report whose template is used for this reporter or whose template library contains the template for this reporter
- DOM document or document part whose template is used for this reporter or whose template library contains the template for this reporter

The specified template must be the same type as the report to which this reporter is appended. For example, for a Microsoft Word report, `TemplateSrc` must be a Word reporter template. If the `TemplateSrc` property is empty, this reporter uses the default reporter template for the output type of the report.

### **TemplateName — Name of template for this reporter**

character vector | string scalar

Name of the template for this reporter, specified as a character vector or string scalar. The template for this reporter must be in the template library of the template source (TemplateSrc) for this reporter.

### LinkTarget — Hyperlink target for this reporter

[] (default) | character vector | string scalar | mlreportgen.dom.LinkTarget object

Hyperlink target for this reporter, specified as a character vector or string scalar that specifies the link target ID, or an mlreportgen.dom.LinkTarget object. A character vector or string scalar value is converted to a LinkTarget object. The link target immediately precedes the content of this reporter in the output report.

## Methods

createTemplate	Create MATLAB Function reporter template
customizeReporter	Create custom MATLAB Function reporter class
getClassFolder	MATLAB Function reporter class definition file location

## Inherited Methods

copy	Create copy of reporter object and make deep copies of property values that reference a reporter, ReporterLayout, or DOM object
getImpl	Get implementation of reporter

## Examples

### Add MATLAB Function Properties, Arguments, and Function Script

Create a PDF report generator that uses the MATLABFunction reporter. This reporter includes information about the MATLAB Function block in the sldemo\_radar\_eml model. The report, by default, includes a table of object properties, a table of input and output arguments, and the function script.

```
import slreportgen.report.*
import mlreportgen.report.*

model_name = 'sldemo_radar_eml';
load_system(model_name);
mlfcnobj = 'sldemo_radar_eml/MATLAB Function';

rpt = slreportgen.report.Report('output','pdf');
chapter = Chapter(mlfcnobj);
rptr = MATLABFunction(mlfcnobj);
add(chapter,rptr);
add(rpt,chapter);

close(rpt);
close_system(model_name);
rptview(rpt);
```

The first page of the generated report is shown.

## Chapter 1. sldemo\_radar\_eml/ MATLAB Function

**Table 1.1. MATLAB Function Object Properties**

Property	Value
Update Method	INHERITED
Sample Time	
Support variable-size arrays	0
Saturate on integer overflow	1
Treat these inherited Simulink signal types as fi objects	Fixed-point
MATLAB Function block fimath	Other:UserSpecified
Input fi math	fimath(... 'RoundMode', 'floor',... 'OverflowMode', 'wrap',... 'ProductMode', 'KeepLSB', 'ProductWordLength', 32,... 'SumMode', 'KeepLSB', 'SumWordLength', 32,... 'CastBeforeSum', true)
Description	

**Table 1.2. MATLAB Function Argument Summary**

Name	Scope	Port	Compiled Type	Compiled Size
meas	Input	1	unknown	
residual	Output	1	unknown	
deltat	Parameter	NaN	unknown	
xhatOut	Output	2	unknown	

### MATLAB Function Function Script

```
function [residual, xhatOut] = EXT_KALMAN(meas, deltat)
%EXT_KALMAN Radar Data Processing Tracker Using an Extended Kalman Filter
%
% This program is executed as a MATLAB function block in the
% sldemo_radar Simulink model. The estimated and actual positions are
% saved to the workspace and are plotted at the end of the simulation by
% the program aero_radplot (called from the simulation automatically).
%
```

### Add MATLAB Function Argument Details

Create an HTML report generator that uses the MATLABFunction reporter and includes MATLAB Function argument details. Use the IncludeArgumentProperties property to include a table for each MATLAB Function block input and output argument. This example uses the sldemo\_radar\_eml model.

```
import slreportgen.report.*
import mlreportgen.report.*

model_name = 'sldemo_radar_eml';
load_system(model_name);
mlfcnobj = 'sldemo_radar_eml/MATLAB Function';
```

```

rpt = slreportgen.report.Report('output','html');
chapter = Chapter(mlfcnobj);
rptr = MATLABFunction(mlfcnobj);
rptr.IncludeArgumentProperties = true;

add(chapter,rptr);
add(rpt,chapter);

close(rpt);
close_system(model_name);
rptview(rpt);

```

This portion of the generated report shows some of the argument detail tables.

**Table 1.2. MATLAB Function Argument Summary**

Name	Scope	Port	Compiled Type	Compiled Size
meas	Input	1	unknown	
residual	Output	1	unknown	
deltat	Parameter	NaN	unknown	
xhatOut	Output	2	unknown	

**Table 1.3. meas Argument Properties**

Property	Value
Name	meas
Scope	Input
Port	1
Compiled Type	unknown

### Change the MATLAB Function Script Formatting and Title

Create an HTML report generator that uses the `MATLABFunction` reporter and sets the appearance of the function script and title. Change the appearance of the MATLAB Function block function script section of the report. Use the `FunctionScriptTitle` property to specify the text to add at the front of the default function script table title. Create and use a DOM paragraph to change the font, font size, and color of the script. This example uses the `sldemo_radar_eml` model.

```
import slreportgen.report.*
import mlreportgen.report.*

model_name = 'sldemo_radar_eml';
load_system(model_name);
mlfcnobj = 'sldemo_radar_eml/MATLAB Function';

rpt = slreportgen.report.Report('output', 'html');
chapter = Chapter(mlfcnobj);
rptr = MATLABFunction(mlfcnobj);

paraTitle = mlreportgen.dom.Paragraph('SCRIPT: ');
rptr.FunctionScriptTitle = paraTitle;

paraScript = mlreportgen.dom.Paragraph;
paraScript.FontFamilyName = 'Arial';
paraScript.FontSize = '12pt';
paraScript.Color = 'blue';
rptr.FunctionScript = paraScript;

add(chapter, rptr);
add(rpt, chapter);

close(rpt);
close_system(model_name);
rptview(rpt);
```

This section of the generated report shows "SCRIPT:" added to the title and 12pt Arial blue font.

**SCRIPT: MATLAB Function Function Script**

```

function [residual, xhatOut] = EXTKALMAN(meas, deltat)
%EXTKALMAN Radar Data Processing Tracker Using an Extended Kalman Filter
%
% This program is executed as a MATLAB function block in the
% sldemo_radar Simulink model. The estimated and actual positions are
% saved to the workspace and are plotted at the end of the simulation by
% the program aero_radplot (called from the simulation automatically).
%
%
% Copyright 1990-2013 The MathWorks, Inc.

% Initialization
persistent P;
persistent xhat
if isempty(P)
    xhat = [0.001; 0.01; 0.001; 400];
    P = zeros(4);
end

% Radar update time deltat is inherited from model workspace

% 1. Compute Phi, Q, and R
Phi = [1 deltat 0 0; 0 1 0 0; 0 0 1 deltat; 0 0 0 1];
Q = diag([0 .005 0 .005]);
R = diag([300^2 0.001^2]);

% 2. Propagate the covariance matrix:
P = Phi*P*Phi' + Q;

```

**See Also**

[MATLAB Function](#) | [mlreportgen.dom.Paragraph](#) | [mlreportgen.report.BaseTable](#) | [slreportgen.finder.BlockFinder](#) | [slreportgen.finder.BlockResult](#) | [slreportgen.finder.DiagramElementFinder](#) | [slreportgen.finder.DiagramElementResult](#) | [slreportgen.finder.StateflowDiagramElementFinder](#) | [slreportgen.utils.isMATLABFunction](#)

**Topics**

“Report on MATLAB Function” on page 4-13

“Report Generation for Simulink and Stateflow Elements” on page 1-9

“What Is a Reporter?”

**Introduced in R2018a**

## slreportgen.report.ModelConfiguration class

**Package:** slreportgen.report

Model configuration set reporter

### Description

Use an object of the slreportgen.report.ModelConfiguration class to report on the active configuration set of a model.

---

**Note** To use an slreportgen.report.ModelConfiguration reporter in a report, you must create the report using the slreportgen.report.Report class or subclass.

---

The slreportgen.report.ModelConfiguration class is a handle class.

### Class Attributes

HandleCompatible true

For information on class attributes, see “Class Attributes”.

## Creation

### Description

reporter = slreportgen.report.ModelConfiguration() creates an empty ModelConfiguration reporter object based on the default template. You must specify a model for which to report the active configuration set by setting the Model property. Use other properties to specify report options.

reporter = slreportgen.report.ModelConfiguration(model) creates a ModelConfiguration reporter object and sets the Model property to the specified model.

reporter = slreportgen.report.ModelConfiguration(Name,Value) sets the reporter properties using name-value pairs. You can specify multiple name-value pair arguments in any order. Enclose each property name in single or double quotes.

## Properties

### Model — Name or handle of model

[] (default) | string scalar | character vector | handle

Name or handle of open or loaded Simulink model, specified as a string scalar, character vector, or handle.

### Title — Configuration set title

[] (default) | character vector | string scalar | mlreportgen.dom.Text object | mlreportgen.dom.InternalLink object | mlreportgen.dom.ExternalLink object



Configuration set title, specified as a character vector, string scalar, `mlreportgen.dom.Text` object, `mlreportgen.dom.InternalLink` object, or `mlreportgen.dom.ExternalLink` object.

If the `FormatPolicy` property is set to "Inline Text" and the `Title` property is set to:

In both cases, to format the title, use the `TextFormatter` property of this `ModelConfiguration` reporter.

If you do not specify a title, the title is the model name followed by `Configuration Set`. For example:

```
sf_car Configuration Set
```

### **FormatPolicy — Format for reporting configuration set**

"Auto" (default) | "Table" | "Paragraph" | "Inline Text"

Format for reporting the configuration set, specified as one of these strings or character vectors:

### **TableReporter — Table formatter**

`mlreportgen.report.BaseTable` object

Table formatter for the tables that the `ModelConfiguration` reporter generates, specified as an `mlreportgen.report.BaseTable` object. The default value of this property is a `BaseTable` object with the `TableStyleName` property set to the `ModelConfigurationTable` style, which is defined in the default template for a `ModelConfiguration` reporter. To customize the appearance of the table, modify the properties of the default `BaseTable` object or replace the object with your own `BaseTable` object. If you add content to the `Title` property, the content appears in front of the table title in the generated report.

### **ParagraphFormatter — Paragraph formatter**

`mlreportgen.dom.Paragraph` object

Paragraph formatter for any model configuration content that is generated as a paragraph, specified as an `mlreportgen.dom.Paragraph` object. The default value of this property is a `Paragraph` object with the `StyleName` property set to the `ModelConfigurationParagraph` style, which is defined in the default template for a `ModelConfiguration` reporter. To customize the appearance of the paragraph, modify the properties of the default `Paragraph` object or replace the object with your own `Paragraph` object. If you add content to the paragraph object, the content appears in front of the model configuration content in the generated report.

### **TextFormatter — Text formatter**

`mlreportgen.dom.Text` object

Text formatter for any model configuration content that is generated as inline text, specified as an `mlreportgen.dom.Text` object. By default, the value of this property is an empty `Text` object. To customize the appearance of the text, modify the properties of the default `mlreportgen.dom.Text` object or replace the object with a customized `mlreportgen.dom.Text` object. If you add content to the `Text` object, the content appears in front of the model configuration content in the generated report.

### **MaxCols — Maximum number of columns in value tables**

32 (default) | positive integer

Maximum number of table columns in value tables, specified as a positive integer. If a property value is reported using a table and the number of columns is greater than the value of the `MaxCols` property, the table is sliced vertically. Slicing divides the table into multiple tables.

**DepthLimit — Maximum number of nested levels to report**

10 (default) | nonnegative integer

Maximum number of nested levels in the structured object hierarchy to report, specified as a nonnegative integer. The top level of the hierarchy is the configuration set object (`Simulink.ConfigSet`). Levels less than or equal to the value of `DepthLimit` are flattened into a sequence of interlinked tables. Levels greater than the depth limit are not reported. If you set the `DepthLimit` property to 0, hierarchically structured types are not expanded.

**ObjectLimit — Maximum number of nested objects to report**

200 (default) | positive integer

Maximum number of objects in an object hierarchy to report, specified as a positive integer.

**IncludeTitle — Whether to include the configuration set title**

true (default) | false

Whether to include the configuration set title, specified as `true` or `false`.

When `IncludeTitle` is `true`, the configuration set title (the content of the `Title` property) is included in the:

The configuration set title is always included in the title for the paragraph or table that contains the configuration set components, regardless of the value of the `IncludeTitle` property.

**ShowDataType — Whether a title includes data type**

false (default) | true

Whether a title includes the data type of the value that the title describes, specified as `true` or `false`.

Data Types: `logical`

**ShowEmptyValues — Whether to show configuration properties that have empty values**

true (default) | false

Whether to show configuration set or component properties that have empty values, specified as a `true` or `false`.

Data Types: `logical`

**ShowDefaultValues — Whether to show configuration properties that use default values**

true (default) | false

Whether to show configuration set or component properties that use the default values, specified as `true` or `false`.

Data Types: `logical`

**PropertyFilterFcn — Function or expression to filter configuration properties from a report**

[] (default) | function handle | string scalar | character vector

Function or expression to filter configuration set and component object properties from a report, specified as a function handle, string scalar, or character vector. Specify a function as a function handle. Specify an expression as a string scalar or character vector. If `PropertyFilterFcn` is empty, all properties are included in the report.

If you provide a function handle, the associated function must:

For example, this code uses the `PropertyFilterFcn` property to prevent the display of the `Name` and `Description` properties:

```
import slreportgen.report.*

rpt = slreportgen.report.Report("MyReport", "pdf");
open(rpt);

model = "sf_car";
load_system(model);
reporter = ModelConfiguration(model);

filterFcnHandle = @(variableName,variableObject,propertyName) ...
    (propertyName == "Name") || ...
    (propertyName == "Description");

reporter.PropertyFilterFcn = filterFcnHandle;
append(rpt,reporter);
close(rpt);
rptview(rpt);
```

If you provide a string scalar or a character vector, it must contain an expression. The expression:

For example, this code uses the `PropertyFilterFcn` property to prevent the display of the `Name` and `Description` properties:

```
import slreportgen.report.*

rpt = slreportgen.report.Report("MyReport", "pdf");
open(rpt);

model = "sf_car";
load_system(model);
reporter = ModelConfiguration(model);

filterStr = "isFiltered = " + ...
    "strcmp(propertyName, 'Name') || strcmp(propertyName, 'Description');";
reporter.PropertyFilterFcn = filterStr;

append(rpt,reporter);
close(rpt);
rptview(rpt);
```

### **NumericFormat — Format or precision used to display noninteger numeric values**

[ ] (default) | string scalar | character vector | positive integer

Format or precision used to display noninteger numeric values, specified as a string scalar, character vector, or positive integer.

Specify the format as a string scalar or a character vector. See the `formatSpec` argument on the `sprintf` reference page.

Specify the precision as a positive integer. See the `precision` argument on the `num2str` reference page.

Example: `"%.2f"` displays double values with two digits to the right of the decimal place.

Example: `2` displays a maximum number of two significant digits.

### TemplateSrc — Source of template for this reporter

[ ] (default) | character vector | string scalar | reporter or report | DOM document or document part

Source of the template for this reporter, specified in one of these ways:

The specified template must be the same type as the report to which this reporter is appended. For example, for a Microsoft Word report, `TemplateSrc` must be a Word reporter template. If the `TemplateSrc` property is empty, this reporter uses the default reporter template for the output type of the report.

### TemplateName — Name of template for this reporter

character vector | string scalar

Name of the template for this reporter, specified as a character vector or string scalar. The template for this reporter must be in the template library of the template source (`TemplateSrc`) for this reporter.

### LinkTarget — Hyperlink target for this reporter

[ ] (default) | character vector | string scalar | `mlreportgen.dom.LinkTarget` object

Hyperlink target for this reporter, specified as a character vector or string scalar that specifies the link target ID, or an `mlreportgen.dom.LinkTarget` object. A character vector or string scalar value is converted to a `LinkTarget` object. The link target immediately precedes the content of this reporter in the output report.

## Methods

### Public Methods

<code>slreportgen.report.ModelConfiguration.createTemplate</code>	Create model configuration reporter template
<code>slreportgen.report.ModelConfiguration.customizeReporter</code>	Create custom model configuration reporter class
<code>slreportgen.report.ModelConfiguration.getClassFolder</code>	Get location of model configuration reporter class definition file
<code>copy</code>	Create copy of reporter object and make deep copies of property values that reference a reporter, <code>ReporterLayout</code> , or DOM object
<code>getConfigSet</code>	Get active configuration set from model configuration reporter
<code>getImpl</code>	Get implementation of reporter

## Examples

## Report Active Model Configuration Set

Use an object of the `slreportgen.report.ModelConfiguration` class to report on the active model configuration set.

Import the MATLAB Report and Simulink Report API packages so that you do not have to use long, fully qualified class names.

```
import mlreportgen.report.*
import slreportgen.report.*
```

Create a Simulink report.

```
rpt = slreportgen.report.Report("MyReport", "pdf");
open(rpt);
```

Create a chapter for the active model configuration set.

```
chapter = Chapter();
chapter.Title = "Active Model Configuration Set";
```

Load a model.

```
model = "sf_car";
load_system(model);
```

Create an `slreportgen.report.ModelConfiguration` object to report on the active configuration set of the model.

```
reporter = ModelConfiguration(model);
```

Append the reporter to the chapter and chapter to the report.

```
append(chapter, reporter);
append(rpt, chapter);
```

Close and view the report

```
close(rpt);
rptview(rpt);
```

## See Also

`Simulink.ConfigSet`

## Topics

“Report Generation for Simulink and Stateflow Elements” on page 1-9

“What Is a Reporter?”

“Manage Configuration Sets for a Model”

## Introduced in R2020b

## slreportgen.report.ModelVariable class

**Package:** slreportgen.report

Model variable reporter

### Description

Reporter for a Simulink model variable.

---

**Note** To use an `slreportgen.report.ModelVariable` reporter in a report, you must create the report using the `slreportgen.report.Report` class or subclass.

---

The `slreportgen.report.ModelVariable` class is a handle class.

### Class Attributes

`HandleCompatible` true

For information on class attributes, see “Class Attributes”.

## Creation

### Description

You do not create an `slreportgen.report.ModelVariable` object explicitly. To get the `slreportgen.report.ModelVariable` object for a found model variable:

- Use the `find` or `next` method of the `slreportgen.finder.ModelVariableFinder` object to get the `slreportgen.finder.ModelVariableResult` object for the found variable.
- Call the `getReporter` method of the `slreportgen.finder.ModelVariableResult` object to get the `slreportgen.report.ModelVariable` object.

You can customize the reporting of a model variable by setting the properties of the `slreportgen.report.ModelVariable` object.

## Properties

### Variable — Simulink.VariableUsage object

`Simulink.VariableUsage`

The `Simulink.VariableUsage` object that corresponds to the variable to report on. The object contains the name of the variable, the source of the variable, and the blocks that use the variable. This property is read-only.

### ModelBlockPath — Path of Model block that sets the value of variable

`[]` | character vector

Path of the Model block that sets the variable value, specified as a character vector.

Suppose that a referenced model uses a model argument to set a block parameter value. If a model has multiple instances of the referenced model, the model variable finder returns multiple instances of the variable that is associated with the model argument. The `ModelBlockPath` property uniquely identifies the instance of the variable by providing the path to the model block that set its value. If a variable is not associated with a model argument in a referenced model, the `ModelBlockPath` is empty. For more information about referenced models and instance-specific parameters, see “Parameterize Instances of a Reusable Referenced Model”.

### ShowUsedBy — Whether to include blocks that use this variable

`true` (default) | `false`

Whether to include a list of blocks that use this variable, specified as `true` or `false`. If the `FormatPolicy` property has a value of "Inline Text", the list of blocks is not included in the report, regardless of the value of the `ShowUsedBy` property.

If the report includes reported content for a block in the Used By list, clicking the hyperlink for the block takes you to the content. See “Generate Report of Model Variables, Diagrams, and Blocks” on page 7-86.

### ShowWorkspaceInfo — Whether to include workspace

`true` (default) | `false`

Whether to include the workspace that the variable is resolved in, specified as `true` or `false`. If the `FormatPolicy` property has a value of "Inline Text", the workspace is not included in the report, regardless of the value of the `ShowWorkspaceInfo` property.

### ListFormatter — List formatter

`mlreportgen.dom.UnorderedList` (default) | `mlreportgen.dom.OrderedList`

List formatter that formats the list of blocks that use the variable, specified as an `mlreportgen.dom.UnorderedList` object or an `mlreportgen.dom.OrderedList` object. To customize how the list is formatted, modify the list object properties or replace the list object with a customized list object that does not contain list items.

### FormatPolicy — Format of variable values

"Auto" (default) | "Table" | "Paragraph" | "Inline Text"

Format of the variable values, specified as one of these strings or character vectors:

### TableReporter — Table reporter

`mlreportgen.report.BaseTable`

Table reporter used to format the value of the variable, specified as an `mlreportgen.report.BaseTable` object. To customize the appearance of the table, modify the properties of the default table reporter or replace it with a customized table reporter. If you add content to the `Title` property of the default or customized table reporter, the content appears in front of the table title in the generated report.

### ParagraphFormatter — Paragraph formatter

`mlreportgen.dom.Paragraph` object

Paragraph formatter to format the value of a model variable, specified as an `mlreportgen.dom.Paragraph` object. To customize the appearance of the paragraph, modify the properties of the `mlreportgen.dom.Paragraph` object or replace the object with a customized

`mlreportgen.dom.Paragraph` object. If you add content to the default or replacement paragraph object, the content appears in front of the variable content in the generated report.

### **TextFormatter — Text formatter**

`mlreportgen.dom.Text` object

Text formatter to format the name and value of the model variable when the text is in line with the surrounding text, specified as an `mlreportgen.dom.Text` object. To customize the appearance of the text, modify the properties of the default `mlreportgen.dom.Text` object or replace the object with a customized `mlreportgen.dom.Text` object. If you add content to the default or replacement text object, the content appears in front of the variable content in the generated report.

### **MaxCols — Maximum number of table columns to display**

32 (default) | positive integer

Maximum number of table columns to display, specified as a positive integer. For array variables reported using a table, if the number of columns is greater than the value of the `MaxCols` property, the table is sliced vertically. Slicing divides the table into multiple tables.

### **DepthLimit — Maximum number of nested levels to report**

10 (default) | nonnegative integer

Maximum number of levels to report for a variable that is a structured object or an array of structured objects, specified as a nonnegative integer. Levels less than or equal to the value of `DepthLimit` are flattened into a sequence of interlinked tables (see the `FormatPolicy` property). Levels greater than the depth limit are not reported. If you set the `DepthLimit` property to 0, structured objects are not expanded.

### **ObjectLimit — Maximum number of nested objects to report**

200 (default) | positive integer

Maximum number of objects in an object hierarchy to report, specified as a positive integer.

### **IncludeTitle — Whether to include title**

`true` (default) | `false`

Whether to include a title, specified as `true` or `false`. The title contains the variable name and optionally, the data type. If `IncludeTitle` is `true`, the title is included. By default, the title includes only the name of the variable. To include the data type of the variable, set the `ShowDataType` property to `true`.

### **Title — Title of variable to report**

[] (default) | character vector | string scalar | `mlreportgen.dom.Text` object | `mlreportgen.dom.InternalLink` object | `mlreportgen.dom.ExternalLink` object

Title of variable to report, specified as a character vector, string scalar, `mlreportgen.dom.Text` object, `mlreportgen.dom.InternalLink` object, or `mlreportgen.dom.ExternalLink` object.

If the `FormatPolicy` property is set to "Inline Text" and the `Title` property is set to:

In both cases, to format the title, use the `TextFormatter` property of this `ModelVariable` reporter.

If you do not specify the `Title` property, the title is the variable name.



**ShowDataType — Whether to show data type of variable in title**

false (default) | true

Whether to show the data type of the variable in the title, specified as true or false.

**ShowEmptyValues — Whether to show properties that have empty values**

true (default) | false

Whether to show properties that have empty values, specified as a true or false. The ShowEmptyValues property applies only to MATLAB object, Simulink object, and Stateflow object variables.

**ShowDefaultValues — Whether to show properties that use default values**

true (default) | 0

Whether to show properties that use the default value, specified as true or false. The ShowDefaultValues property applies only to MATLAB object, Simulink object, and Stateflow object variables.

**PropertyFilterFcn — Function or expression to filter properties of a reported model variable**

[] (default) | function handle | string scalar | character vector

Function or expression to filter the properties of a reported model variable from a report. Specify a function as a function handle. Specify an expression as a string scalar or character vector. This property applies only to a variable that contains an object. If you do not provide PropertyFilterFcn, all properties of the model variable are included in the report.

If you provide a function handle, the associated function must:

For example, this code prevents the display of the Description and Complexity properties of a Simulink.Parameter object.

```
import slreportgen.finder.*
import slreportgen.report.*

rpt = slreportgen.report.Report('modelvarrpt','pdf');
model_name = load_system('sldemo_mdref_datamngt');

finder = slreportgen.finder.ModelVariableFinder(model_name);

while hasNext(finder)
    result = next(finder);
    varRptr = getReporter(result);
    varRptr.PropertyFilterFcn = @varPropertyFilter;
    add(rpt,varRptr);
end

close(rpt);

close_system(model_name);
rptview(rpt);

function tf = varPropertyFilter(~, variableObject,propertyName)
if isa(variableObject, 'Simulink.Parameter')
    tf = (propertyName == "Description") || ...
        (propertyName == "Complexity");
else
    tf = false;
```

```
end
end
```

If you provide a string scalar or a character vector, it must contain an expression. The expression:

For example, this code filters the `CoderInfo` property of a `Simulink.Parameter` object from the report.

```
import slreportgen.finder.*
import slreportgen.report.*

rpt = slreportgen.report.Report('modelvarrpt','pdf');

model_name = load_system('sldemo_mdlref_datamngt');

finder = slreportgen.finder.ModelVariableFinder(model_name);

while hasNext(finder)
    result = next(finder);
    varRptr = getReporter(result);

    varRptr.PropertyFilterFcn = "isFiltered = " + ...
        "isa(variableObject, 'Simulink.Parameter') && " + ...
        "propertyName == 'CoderInfo';";
    add(rpt,varRptr);
end

close(rpt);

close_system(model_name);
rptview(rpt);
```

### **NumericFormat** — Format or precision used to display noninteger numeric values

"%.2f" (default) | string scalar | character vector | positive integer

Format or precision used to display noninteger numeric values.

Specify the format as a string scalar or a character vector. See the `formatSpec` argument on the `sprintf` reference page.

Specify the precision as a positive integer. See the `precision` argument on the `num2str` reference page.

Example: "%.2f" displays double values with two digits to the right of the decimal place.

Example: 2 displays a maximum number of two significant digits.

### **TemplateSrc** — Source of template for this reporter

[] (default) | character vector | string scalar | reporter or report | DOM document or document part

Source of the template for this reporter, specified in one of these ways:

The specified template must be the same type as the report to which this reporter is appended. For example, for a Microsoft Word report, `TemplateSrc` must be a Word reporter template. If the `TemplateSrc` property is empty, this reporter uses the default reporter template for the output type of the report.

**TemplateName — Name of template for this reporter**

character vector | string scalar

Name of the template for this reporter, specified as a character vector or string scalar. The template for this reporter must be in the template library of the template source (`TemplateSrc`) for this reporter.

**LinkTarget — Hyperlink target for this reporter**[] (default) | character vector | string scalar | `mlreportgen.dom.LinkTarget` object

Hyperlink target for this reporter, specified as a character vector or string scalar that specifies the link target ID, or an `mlreportgen.dom.LinkTarget` object. A character vector or string scalar value is converted to a `LinkTarget` object. The link target immediately precedes the content of this reporter in the output report.

**Methods****Public Methods**

<code>getVariableName</code>	Get name of variable from model variable reporter
<code>getVariableValue</code>	Get value of variable from model variable reporter
<code>slreportgen.report.ModelVariable.createTemplate</code>	Create model variable reporter template
<code>slreportgen.report.ModelVariable.customizeReporter</code>	Create custom model variable reporter class
<code>slreportgen.report.ModelVariable.getClassFolder</code>	Get location of model variable reporter class definition file
<code>copy</code>	Create copy of reporter object and make deep copies of property values that reference a reporter, <code>ReporterLayout</code> , or DOM object
<code>getImpl</code>	Get implementation of reporter

**Examples****Customize the Formatting of Model Variables in a Report**

Use the properties of an `slreportgen.report.ModelVariable` object to customize the formatting of a variable.

```
% Create a Report
rpt = slreportgen.report.Report("MyReport","pdf");

% Create a Chapter
chapter = mlreportgen.report.Chapter();
chapter.Title = "Model Variable Reporter Example";

% Load the model
model_name = "sf_car";
load_system(model_name);

% Find the variables in the model
finder = slreportgen.finder.ModelVariableFinder(model_name);

while hasNext(finder)
```

```

    result = next(finder);

    % Get the ModelVariable reporter for the result
    % Customize the formatting of numbers
    reporter = getReporter(result);
    reporter.NumericFormat = "%.4f";

    % Add the reporter to the chapter
    add(chapter,reporter);
end
% Add chapter to the report
add(rpt,chapter);

% Close the report and open the viewer
close(rpt);
rptview(rpt);

```

### Generate Report of Model Variables, Diagrams, and Blocks

Generate a report that includes:

- A chapter for the model variables
- A chapter for each model diagram, with a section for the blocks in the diagram

Each block name in the Used By list for a model variable is a hyperlink to the corresponding content reported for the block.

```

% Create a Report
rpt = slreportgen.report.Report("MyReport","pdf");

% Load the model
model_name = "sf_car";
load_system(model_name);

% Create a Chapter for the Variables
chapter = mlreportgen.report.Chapter();
chapter.Title = sprintf("Model Variable Report for the %s model",model_name);

% Find the variables in the model
finder = slreportgen.finder.ModelVariableFinder(model_name);

% Report on the variables
while hasNext(finder)
    result = next(finder);
    reporter = getReporter(result);
    add(chapter,reporter);
end
add(rpt,chapter);

% Add diagrams to the report
finder = slreportgen.finder.DiagramFinder(model_name);
while hasNext(finder)
    result = next(finder);
    ch = mlreportgen.report.Chapter(result.Name);
    add(ch, result);
    % Add a section for the blocks in the diagram

```

```
sect = mlreportgen.report.Section("Title", "Blocks");
blFinder = slreportgen.finder.BlockFinder(result.Object);
while hasNext(blFinder)
    blockresult = next(blFinder);
    add(sect, blockresult);
end
add(ch, sect);
add(rpt, ch);
end

% Close and view the report
close(rpt);
rptview(rpt);
```

## See Also

Simulink.VariableUsage | slreportgen.finder.ModelVariableFinder |  
slreportgen.finder.ModelVariableResult | slreportgen.report.BusObject

## Topics

“Report Generation for Simulink and Stateflow Elements” on page 1-9  
“What Is a Reporter?”

## Introduced in R2019b

## slreportgen.report.Notes class

**Package:** slreportgen.report slreportgen.report slreportgen.report  
slreportgen.report

Simulink or Stateflow diagram notes reporter

### Description

Create a reporter that reports on Simulink or Stateflow diagram notes.

---

**Note** To use a Notes reporter in a report, you must create the report using the `slreportgen.report.Report` class.

---

The `slreportgen.report.Notes` class is a `handle` class.

### Class Attributes

`HandleCompatible` true

For information on class attributes, see “Class Attributes”.

## Creation

### Description

`notes = slreportgen.report.Notes()` creates an empty `slreportgen.report.Notes` reporter.

`notes = slreportgen.report.Notes(source)` creates an `slreportgen.report.Notes` reporter for the system specified by `source` and sets the `Source` property to `source`.

`slreportgen.report.Notes(Name, Value)` sets the reporter properties using name-value pairs. You can specify multiple name-value pair arguments in any order. Enclose each property name in single or double quotes.

## Properties

### Source — Source from which to extract notes

`[]` (default) | string scalar | character vector | `handle` | `slreportgen.finder.DiagramResult` object

Source from which to extract notes, specified as a string scalar, character vector, `handle`, or `slreportgen.finder.DiagramResult` object. The source can be a model, subsystem, Stateflow chart, Stateflow truth table, or Stateflow state transition table.

### NoteType — Type of notes

'None' (default) | 'Internal' | 'External' | 'Inherit'

Type of notes, specified as one of the values in this table:

Value	Description
'Internal'	Note content is included with the model and is saved in a <code>.mldatx</code> file.
'External'	Note content is external to the model and is specified by a URL.
'Inherited'	Note content originates from the ancestors of the specified diagram source.
'None'	The diagram does not have notes.

This property is read-only.

### ReportOnInheritNoteType — Whether to report on notes with type inherit

false (default) | true

Whether to report on notes that have a `NoteType` of 'Inherit', specified as true or false. The reported note content is based on the parent note type as described in this table.

Parent Note Type	Reported Note Content
'Internal'	Link to parent note content
'External'	Link to external content specified by the parent note
'None'	Empty content

### TemplateSrc — Source of template for this reporter

[] (default) | character vector | string scalar | reporter or report | DOM document or document part

Source of the template for this reporter, specified in one of these ways:

The specified template must be the same type as the report to which this reporter is appended. For example, for a Microsoft Word report, `TemplateSrc` must be a Word reporter template. If the `TemplateSrc` property is empty, this reporter uses the default reporter template for the output type of the report.

### TemplateName — Name of template for this reporter

character vector | string scalar

Name of the template for this reporter, specified as a character vector or string scalar. The template for this reporter must be in the template library of the template source (`TemplateSrc`) for this reporter.

### LinkTarget — Hyperlink target for this reporter

[] (default) | character vector | string scalar | `mlreportgen.dom.LinkTarget` object

Hyperlink target for this reporter, specified as a character vector or string scalar that specifies the link target ID, or an `mlreportgen.dom.LinkTarget` object. A character vector or string scalar value is converted to a `LinkTarget` object. The link target immediately precedes the content of this reporter in the output report.

## Methods

### Public Methods

exportToHTML	htmlContent = exportToHTML(notes) exports the internal notes associated with the specified slreportgen.report.Notes reporter to a string of HTML. If the NoteType of the note reporter is not 'Internal', the exportToHTML method ends with an error.
exportToHTMLFile	fullFileName = exportToHTMLFile(notes, filename) exports the internal notes associated with the specified slreportgen.report.Notes reporter to the specified HTML file. If the NoteType of the note reporter is not 'Internal' or if the HTML file name exists, the exportToHTMLFile method ends with an error.
getURL	getURL(notes) returns the URL of the external notes associated with the specified slreportgen.report.Notes reporter as a string. If the NoteType of the note reporter is not 'External', the getURL method ends with an error.
slreportgen.report.Notes.createTemplate	Copy the default slreportgen.report.Notes reporter template.
slreportgen.report.Notes.customizeReporter	Subclass the slreportgen.report.Notes class for customization.
copy	Create copy of reporter object and make deep copies of property values that reference a reporter, ReporterLayout, or DOM object
slreportgen.report.Notes.getClassFolder	Get the location of the folder that contains the slreportgen.report.Notes reporter class definition file.
getImpl	impl = getImpl(notes, report) returns the DOM object used to implement the notes reporter in the specified report. Examining the implementation file can help you to debug report generation problems.

## Examples

### Include Model Notes in a Report

This example reports the notes for the slreportgendemo\_autotrans model, which is a version of the sldemo\_autotrans model with notes. This example reports the notes for the overall model. For an example that reports the diagram and notes for each subsystem of the model, see “Report Model Notes” on page 4-39.



The example creates a chapter for the notes and includes the model notes in the chapter by adding an `slreportgen.report.Notes` reporter to the chapter.

```
model = "slreportgendemo_autotrans";
open_system(model);

import mlreportgen.report.*
import slreportgen.report.*

rpt = slreportgen.report.Report(model + "_Notes_Report", "pdf");
open(rpt);

ch = Chapter("Title", model + " Notes");
notes = Notes(model);
add(ch,notes);
add(rpt,ch);

close(rpt);
rptview(rpt);
```

## See Also

`slreportgen.finder.ChartDiagramFinder` | `slreportgen.finder.DiagramResult` |  
`slreportgen.finder.SystemDiagramFinder`

## Topics

“Report Model Notes” on page 4-39

“Report Generation for Simulink and Stateflow Elements” on page 1-9

“What Is a Reporter?”

## Introduced in R2020a

## slreportgen.report.Report class

**Package:** slreportgen.report

Superclass for report creation

### Description

slreportgen.report.Report is a container for a report based on Simulink reporters and DOM objects. Use this object to generate an HTML, PDF, or Word report based on templates in a template library.

---

**Note** Use objects of this type, instead of mlreportgen.report.Report, to create Simulink reports, which are reports that use Simulink reporters to generate content. You can also use MATLAB reporters and DOM objects to generate Simulink report content.

---

### Construction

report = slreportgen.report.Report() returns a report object report with the default report type (PDF) and a default file name (untitled.pdf).

report = slreportgen.report.Report(path) uses the specified output path for the report.

report = slreportgen.report.Report(path,type) creates the specified type of report.

report = slreportgen.report.Report(path,type,template) uses the specified template.

report = slreportgen.report.Report(Name,Value) sets properties using name-value pairs. You can specify multiple name-value pair arguments in any order. Enclose each property name in single quotes.

### Input Arguments

#### path — Report output path

untitled.pdf (default) | string | character array

See the OutputPath property.

#### type — Report output type

'pdf' (default) | 'html' | 'html-file' | 'docx'

See the Type property.

#### template — Report template

string | character array

See the TemplatePath property.

## Properties

### OutputPath — Report document output path

string | character array

Report document output path, specified as a string or character array. The path is the location in the file system where the report output document is stored. The path can be a full path or a path relative to the current MATLAB folder, for example, 'C:/myreports/reportA.docx' or 'reportA'. If the file name does not have a file extension corresponding to the Type property, the appropriate file extension is added.

---

**Note** Generating a PDF report on a cloud drive, such as MATLAB Drive™, can result in an error that is caused by file contention between the report generation software and the cloud drive synchronization software. To avoid this error, generate reports on a local drive that does not synchronize with the cloud. Consider writing a script that generates a report on a local drive and then copies the report to the cloud drive.

---

### Type — Output type

string | character vector | 'docx' | 'pdf'

Output type, specified as one of these values.

- 'HTML' — HTML report packaged as a zipped file containing the HTML file, images, style sheet, and JavaScript files of the report.
- 'HTML-FILE' — HTML report as a single HTML file containing the text, style sheet, JavaScript, and base64-encoded images of the report
- 'PDF' — PDF file
- 'DOCX' — Microsoft Word document

If you specify a template using the TemplatePath property, the value for Type must match the template type.

### Layout — Page layout options

mlreportgen.report.ReportLayout object

Page layout options for this report, specified as a report layout object. See mlreportgen.report.ReportLayout.

### Locale — Locale or language

[] (default) | string | character array

Locale or language, specified as the ISO 639-1 two-letter language code of the locale for which this report is to be generated. The default [] specifies the language of the system locale, for example, English on an English system. The Report API uses the language code to translate chapter title prefixes to the language of the specified locale. Translations are provided for the following locales: af, ca, cs, da, de, el, en, es, et, eu, fi, fr, hu, id, it, ja, ko, nl, nn, no, pl, pt, ro, ru, sk, sl, sr, sv, tr, uk, xh, and zh. If you specify an unsupported locale, the English version is used. See ISO 639-1 codes.

### TemplatePath — Location of template

string | character array

Location of template used to format the report, specified as a string or character array. Use this property to specify a custom template for this report.

**Document — Underlying DOM document object**

DOM document object

This read-only property is an `mlreportgen.dom.Document` that is used to generate the content of the report.

**Context — Container for keys and values**

map object

This read-only property is a `containers.Map` object that contains information for generating the report, such as the hierarchical level of the current report section.

**Debug — Debug mode**

`false` (default) | `true`

Debug mode, specified as a logical. If you set `Debug` to `true`, the temporary files for the report are stored in a subfolder of the report folder. In debug mode, these files are not deleted when the report is closed.

**CompileModelBeforeReporting — Compile Simulink model**

`true` (default) | `false`

Whether to compile the Simulink model before reporting, specified as a logical. If this property is `true` and the model is not already compiled, it compiles when you add a reporter that reports on that model to this report. If the model cannot be compiled, report generation terminates. If this property is `false`, report generation proceeds without compiling the model.

## Methods

This class uses the same methods as the MATLAB version. Instead of using `mlreportgen` in the class name, use `slreportgen`. See `mlreportgen.report.Report` for a list of methods.

## Copy Semantics

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

## Compatibility Considerations

**add method is not recommended**

Starting in R2020b, use the `append` method instead of the `add` method to add content to objects of these Report API classes:

- `slreportgen.report.Report`
- `mlreportgen.report.Chapter`
- `mlreportgen.report.Section`

To add content to a DOM API object, such as an `mlreportgen.dom.Paragraph` object, continue to use the `append` method of the DOM object. The advantage of using `append` to add content to Report API objects is that you use the same method name as you use to add content to DOM API objects.

There are no plans to remove the add methods of the Report, Chapter, or Section classes. Report API programs that use the add methods will continue to run.

To update existing code, replace the method name add with append as shown by the examples in the table.

Not Recommended	Recommended
<pre>import mlreportgen.report.* import mlreportgen.dom.*  rpt = slreportgen.report.Report("myrpt", "pdf"); ch = Chapter("My Chapter"); sect = Section("My Section"); para = Paragraph("My Content "); append(para, "more Content"); add(sect, para); add(ch, sect); add(rpt, ch);  close(rpt); rptview(rpt);</pre>	<pre>import mlreportgen.report.* import mlreportgen.dom.*  rpt = slreportgen.report.Report("myrpt", "pdf"); ch = Chapter("My Chapter"); sect = Section("My Section"); para = Paragraph("My Content "); append(para, "more Content"); append(sect, para); append(ch, sect); append(rpt, ch);  close(rpt); rptview(rpt);</pre>

## See Also

slreportgen.finder.AnnotationFinder | slreportgen.finder.BlockFinder |  
slreportgen.finder.ChartDiagramFinder | slreportgen.finder.DiagramElementFinder  
| slreportgen.finder.DiagramFinder | slreportgen.finder.StateFinder |  
slreportgen.finder.StateflowDiagramElementFinder |  
slreportgen.finder.SystemDiagramFinder | slreportgen.report.Diagram |  
slreportgen.report.SimulinkObjectProperties |  
slreportgen.report.StateflowObjectProperties

## Topics

“Report Generation for Simulink and Stateflow Elements” on page 1-9  
“What Is a Reporter?”

## Introduced in R2017b

## slreportgen.report.RptFile class

**Package:** slreportgen.report

Create Report Explorer-based reporter

### Description

Use the `RptFile` reporter to include the content generated by a Report Explorer setup (`.rpt`) file in a Report API report. When added to a report, the `RptFile` reporter:

- 1 Executes the specified Report Explorer setup file to generate a DocBook XML rendition of the Report Explorer report
- 2 Uses a modified version of the Report Explorer Docbook-to-DOM conversion template to convert the XML to a set of DOM objects (see “Manage Report Conversion Templates”)
- 3 Adds the DOM content to the Report API report.

---

**Note** Use a Block Loop rather than a Chart Loop component in your report setup file to report on Stateflow charts. See “Report on Stateflow Dialog Snapshots” on page 7-101.

---

The `slreportgen.report.RptFile` class is a handle class.

### Creation

#### Description

`reporter = RptFile()` creates an empty Report Explorer-based `RptFile` reporter. Before adding the reporter to a report, your report program must set the reporter's `SetupFile` property to the path of a Report Explorer setup (`.rpt`) file. Otherwise, an error occurs.

By default, the `RptFile` reporter uses a conversion template that is a slightly modified version of the Report Explorer's default conversion template for the report output type. For example, if the report output type is PDF, the reporter uses a slightly modified version of the default template for the Report Explorer's PDF (`from template`) output type.

You can use a custom conversion template to customize the reporter output. Use the reporter's `createTemplate` method to create a copy of one of the reporter's default output-type-specific conversion templates for customization. To use the customized template, set the `RptFile` reporter's `TemplateSrc` property to the path of the customized template.

`reporter = RptFile(SetupFile)` creates a `RptFile` reporter based on the specified Report Explorer setup file (`.rpt` file). See the `SetupFile` property.

`reporter = RptFile(Name,Value)` sets properties using name-value pairs. You can specify multiple name-value pair arguments in any order. Enclose each property name in single quotes.

## Properties

### SetupFile — Report Explorer setup file path

character array | string

Report Explorer setup file path, specified as a character array or string. Do not use form-based reports for setup files that you use with the RptFile reporter. The Report API report to which the setup file is added overrides the output type of the setup file.

#### Attributes:

GetAccess	public
SetAccess	public

Data Types: string | character array

### Model — Model name

character array | string

Model name, specified as a character array or string, of the model for which the specified SetupFile is executed. If the setup file contains a Model Loop, the RptFile reporter sets its value to the value of this property. An error occurs if the setup file does not contain a Model Loop or contains multiple model loops.

#### Attributes:

GetAccess	public
SetAccess	public

Data Types: character array | string

### System — System path

character array | string | slreportgen.finder.DiagramResult object

System path, specified as a character array, string, or slreportgen.finder.DiagramResult object. If the setup file contains a System Loop, the RptFile reporter sets the System Loop's value to the value of this property if it is a character or string. If the value is a DiagramResult object, the reporter sets the System Loop to the value of the result's Path property. An error occurs if the setup file does not contain a System Loop or contains multiple system loops.

#### Attributes:

GetAccess	public
SetAccess	public

Data Types: character array | string | object

### Block — Block path

character array | string | slreportgen.finder.DiagramElementResult object | slreportgen.finder.BlockResult object

Block path, specified as a character array or string, slreportgen.finder.DiagramElementResult object, or slreportgen.finder.BlockResult object for a block. If the setup file contains a Block Loop, the RptFile reporter sets the Block Loop's value to the value of this property if it is a character or string. If the value is an slreportgen.finder.BlockResult object, the reporter uses the value of the object's BlockPath

property. If the value is a `DiagramElementResult` object, the reporter uses the value of the object's `DiagramPath` and `Name` properties to determine the full path. An error occurs if the setup file does not contain a Block Loop or contains multiple block loops.

---

**Note** Use a Block Loop component in your setup file to report on Stateflow charts. See “Report on Stateflow Dialog Snapshots” on page 7-101.

---

**Attributes:**

<code>GetAccess</code>	<code>public</code>
<code>SetAccess</code>	<code>public</code>

Data Types: `character array | string | object`

**TemplateSrc — Source of conversion template**

`[] | string | character array`

Source of conversion template to be used by this reporter to convert the setup file's XML output to DOM objects. An empty value specifies use of the default template for the output type of the report to be generated. A string or character array value specifies the path of a customized version of the default template for the output type to be generated.

**Attributes:**

<code>GetAccess</code>	<code>public</code>
<code>SetAccess</code>	<code>public</code>

Data Types: `character array | string`

**TemplateName — Name of template for this reporter**

`character array | string`

Name of template for this reporter, specified as a character array or string. By default this property specifies `RptFile`, the name of the reporter's default template. This default template resides in the template library of its default conversion template along with other templates used to convert Report Explorer XML components to DOM objects. The default reporter template contains a single hole named `Content` to be filled with the DOM content converted from the XML content generated by the setup. If you change the name of this template, you must set this property to the new name. You can modify the template itself, but the modified template must contain a hole named `Content`.

**Attributes:**

<code>GetAccess</code>	<code>public</code>
<code>SetAccess</code>	<code>public</code>

Data Types: `character array | string`

**LinkTarget — Hyperlink target for content created by this reporter**

`string | character array | mlreportgen.dom.LinkTarget object`

Hyperlink target for this reporter, specified as a string or character array that specifies the link target ID, or an `mlreportgen.dom.LinkTarget` object. A string or character array value is converted to a `LinkTarget` object. The link target object immediately precedes the content of this reporter in the output report.



**Attributes:**

GetAccess	public
SetAccess	public

Data Types: string | character array | object

**Methods****Public Methods**

slreportgen.report.RptFile.createTemplate	Create Report Explorer-based (RptFile) reporter template
slreportgen.report.RptFile.customizeReporter	Create custom Report Explorer-based reporter class
slreportgen.report.RptFile.getClassFolder	Report Explorer-based reporter class definition file location
copy	Create copy of reporter object and make deep copies of property values that reference a reporter, ReporterLayout, or DOM object
getImpl	Get implementation of reporter

**Examples****Create a RptFile Reporter**

Create an RptFile reporter without specifying a setup file. Then, use the SetupFile property to specify the Report Explorer setup file.

```
reporter = slreportgen.report.RptFile();
reporter.SetupFile = "my_setup_file.rpt"
```

**Report on a Documentation Block**

Use the RptFile reporter to report on a Documentation block in the sldemo\_fuelsys Simulink model.

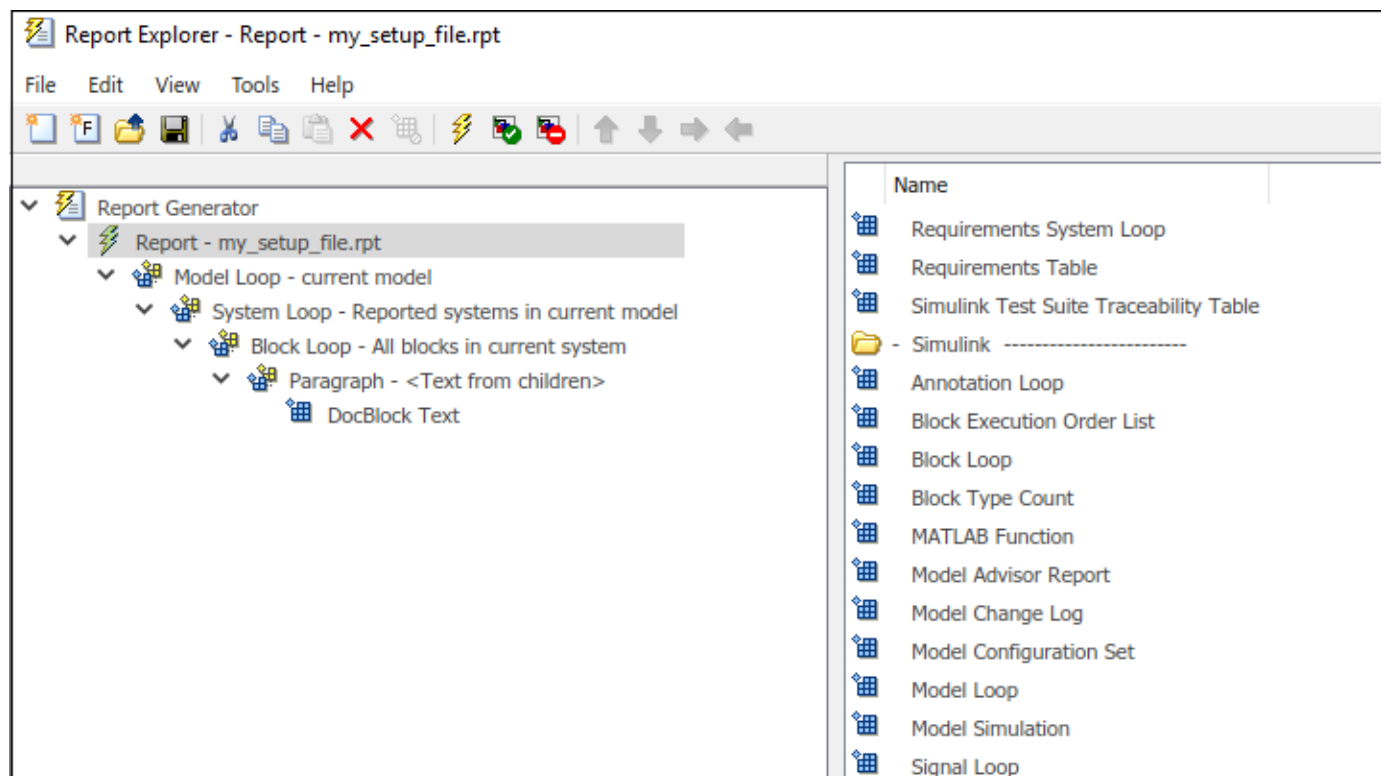
The RptFile reporter uses a Report Explorer setup file to obtain information about the Documentation block.

---

**Note** Before you run this example, use the Report Explorer to create a setup file named my\_setup\_file.rpt. The setup file for this example contains a hierarchy of a Model Loop, System Loop, Block Loop, Paragraph, and Documentation components as shown. Select the components from the middle pane.

- Model Loop, System Loop, and Block Loop components are in the Simulink folder.
- Paragraph component is in the Formatting folder.
- Documentation block component is in the Simulink Blocks folder.

For more information on setting up a setup file for this example, see “Create a Report Setup File”.



Use this script to generate a report that includes information about the properties of the Sensor Info Documentation block in the ToController system of the sldemo\_fuelsys model.

```

model = "sldemo_fuelsys";
load_system(model)

rpt = slreportgen.report.Report("MyReport","pdf");
chap = mlreportgen.report.Chapter("Report on a DocBlock");

rptFile = slreportgen.report.RptFile("my_setup_file.rpt");
rptFile.Model = model;
rptFile.System = "sldemo_fuelsys/To Controller";
rptFile.Block = "sldemo_fuelsys/To Controller/Sensor Info";

add(chap,rptFile);
add(rpt,chap);

close(rpt);
rptview(rpt);

```

## Chapter 1. Report on a DocBlock

### Measurement Descriptions

=====

Throttle Angle (degrees):  
min=3, max=90, precision=0.1

Engine Speed (radians/sec.):  
min=0, max=620, precision=.1

Ego Sensor (volts):  
min=0, max=12, precision=.05

Manifold Pressure (bar):  
min=0.001, max=1, precision=.001

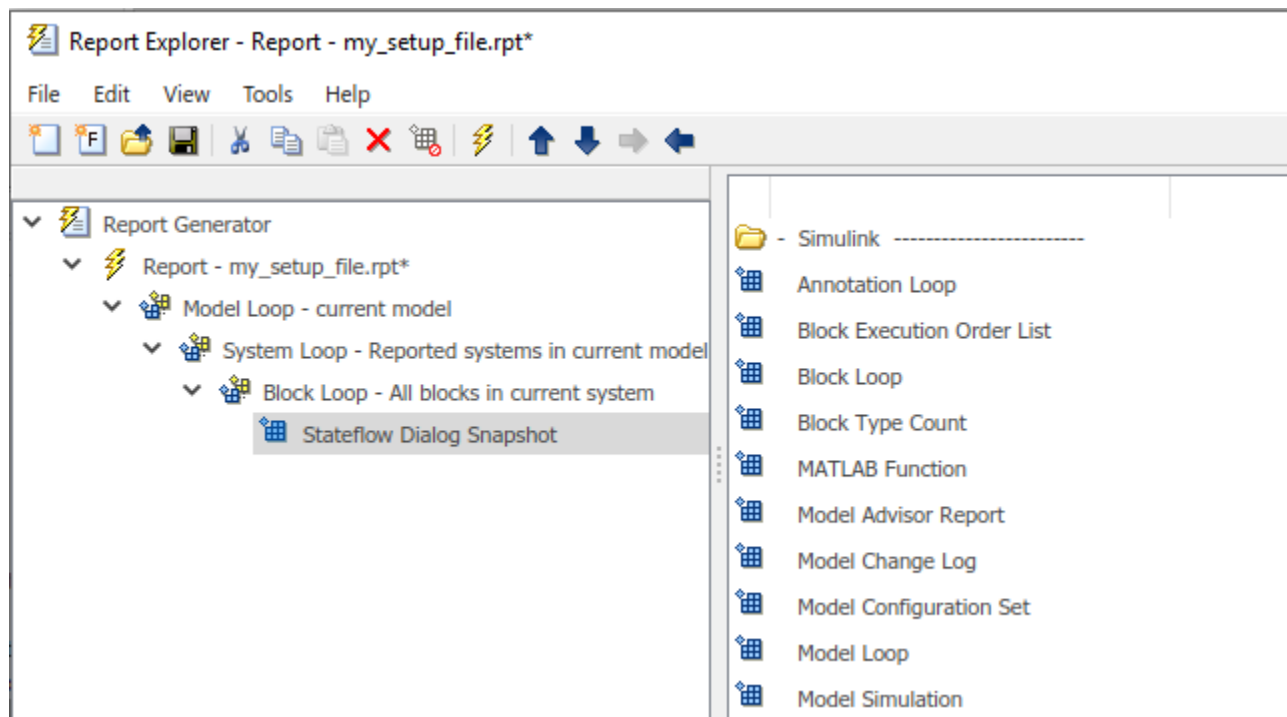
### Report on Stateflow Dialog Snapshots

To use `slreportgen.report.RptFile` to report on Stateflow dialog snapshots, use a Block Loop in the Report Explorer setup file.

Use the Report Explorer to create a setup file named `my_setup_file.rpt`. The setup file for this example contains a hierarchy consisting of a Model Loop, System Loop, Block Loop, and Stateflow Dialog Snapshot component. Select the components from the middle pane.

- The Model Loop, System Loop, and Block Loop components are in the Simulink folder.
- The Stateflow Dialog Snapshot component is in the Stateflow folder.

For more information on setting up a setup file, see “Create a Report Setup File”.



Create a Simulink report.

```
rpt = slreportgen.report.Report("MyReport", "pdf");
open(rpt);
```

Load a model.

```
model = "sf_car";
load_system(model);
```

Create a chapter.

```
chap = mlreportgen.report.Chapter();
chap.Title = strcat(model, ": Stateflow Dialog Snapshots");
```

Find all the systems in the model.

```
sys_finder = slreportgen.finder.SystemDiagramFinder(model);
systems = find(sys_finder);
```

Find all the blocks in the current system. Use the report setup file to report on Stateflow dialog snapshots.

```
for system = systems
    blk_finder = slreportgen.finder.BlockFinder(system);
    blocks = find(blk_finder);

    for block = blocks
        if slreportgen.utils.isValidSlSystem(block.Object) && ...
            ~isempty(slreportgen.utils.block2chart(block.Object))
            rptFile = slreportgen.report.RptFile("my_setup_file.rpt");
            rptFile.Model = model;
```

```
        rptFile.System = system;  
        rptFile.Block = block;  
        add(chap,rptFile);  
    end  
end  
end
```

Add the chapter to the report.

```
add(rpt, chap);
```

Close and view the report.

```
close(rpt);  
rptview(rpt);
```

## See Also

[mlreportgen.report.RptFile](#) | [slreportgen.report.Report](#)

## Topics

“Use Simulink Report Explorer Components in a Report API Report” on page 4-20

“Working with the Report Explorer”

**Introduced in R2019a**

## slreportgen.report.SimulinkObjectProperties class

**Package:** slreportgen.report

Simulink object properties reporter

### Description

The `SimulinkObjectProperties` reporter generates tables that list the properties and property values of Simulink objects.

---

**Note** To use a Simulink object properties reporter in a report, you must create the report using the `slreportgen.report.Report` class.

---

### Construction

`reporter = SimulinkObjectProperties()` creates an empty Simulink object properties reporter. Use the properties of this reporter to specify reporting on these Simulink objects:

- Object whose properties to report
- Properties to report
- Format of the reported properties

`reporter = SimulinkObjectProperties(obj)` creates a reporter that generates a table listing the property values of the specified Simulink object.

To specify a list of properties to include in the generated properties table, use the `Properties` property of the reporter. If you do not specify any properties, the reporter includes a default set of properties based on the object type. For example, the property table for a block includes the properties set by its parameter dialog box.

To customize the format of the generated property table., use the `PropertyTable` property

---

**Note** This reporter compiles the model containing the object to be reported if the model is not already compiled. Compiling the model is necessary to propagate values to properties that are unspecified when the model has not been compiled. The model is uncompiled when you close the report that contains the generated property table.

---

`reporter = SimulinkObjectProperties(Name, Value)` sets properties using name-value pairs. You can specify multiple name-value pair arguments in any order. Enclose each property name in single quotes.

### Input Arguments

#### **obj** — Simulink object

path string or character vector | object handle

See Object property.

## Properties

### Object — Simulink object to report

path string or character vector | object handle

Simulink object whose properties to report, specified as a path to or handle of the specified object. The `Object` value must be one of these types of objects:

- model
- block
- annotation
- port
- line
- line segment

### PropertyTable — Object properties table reporter

mlreportgen.report.BaseTable reporter

Object properties table reporter, specified as an `mlreportgen.report.BaseTable` reporter. The object properties reporter uses the base table reporter to format object properties. If this property is initially empty, the object properties reporter sets the property to a default property table reporter. To customize the property table formatting, set this property to a base table reporter that meets your formatting requirements.

### ShowPromptNames — Whether to show property prompt names

true (default) | false

Choice to display property names as dialog box prompts, specified as a logical. If `true` and the property appears on the dialog box of the object, the table lists its dialog box prompt instead of its property name. Otherwise, the generated property table lists the property using its property name.

### ShowEmptyValues — Whether to show properties with empty values

false (default) | true

Choice to show properties with empty values.

Whether to show properties with empty values, specified as a logical. If `false`, the generated object properties table omits object properties whose value is empty. If `true`, the table includes properties whose value is empty.

### Properties — Names of properties to be reported

cell array of strings or character vectors

Names of object properties to be reported, specified as a cell array.

A cell array of names of object properties to be reported, specified as a cell array of strings or character vectors. If you do not specify any properties, the reporter determines a set of properties to report.

### TemplateSrc — Source of template for this reporter

[] (default) | character vector | string scalar | reporter or report | DOM document or document part

Source of the template for this reporter, specified in one of these ways:

- Character vector or string scalar that specifies the path of the file that contains the template for this reporter
- Reporter or report whose template is used for this reporter or whose template library contains the template for this reporter
- DOM document or document part whose template is used for this reporter or whose template library contains the template for this reporter

The specified template must be the same type as the report to which this reporter is appended. For example, for a Microsoft Word report, `TemplateSrc` must be a Word reporter template. If the `TemplateSrc` property is empty, this reporter uses the default reporter template for the output type of the report.

### TemplateName — Name of template for this reporter

character vector | string scalar

Name of the template for this reporter, specified as a character vector or string scalar. The template for this reporter must be in the template library of the template source (`TemplateSrc`) for this reporter.

### LinkTarget — Hyperlink target for this reporter

[] (default) | character vector | string scalar | `mlreportgen.dom.LinkTarget` object

Hyperlink target for this reporter, specified as a character vector or string scalar that specifies the link target ID, or an `mlreportgen.dom.LinkTarget` object. A character vector or string scalar value is converted to a `LinkTarget` object. The link target immediately precedes the content of this reporter in the output report.

## Methods

<code>createTemplate</code>	Create Simulink object properties reporter template
<code>customizeReporter</code>	Create custom Simulink object properties class
<code>getClassFolder</code>	Simulink object properties class definition file location

## Inherited Methods

<code>copy</code>	Create copy of reporter object and make deep copies of property values that reference a reporter, <code>ReporterLayout</code> , or DOM object
<code>getImpl</code>	Get implementation of reporter

## Examples

### Add Properties Table to Report

Use the `SimulinkObjectProperties` reporter to add a properties table for the `vdp` model to the report.

```
import slreportgen.report.*
import mlreportgen.report.*
model_name = 'vdp';
load_system(model_name);
```



```
rpt = slreportgen.report.Report('output', 'pdf');
chapter = Chapter(model_name);
rptr = SimulinkObjectProperties(model_name);

add(chapter, rptr);
add(rpt, chapter);
close(rpt);
close_system(model_name);
rptview(rpt);
```

### Specify Object Properties for Report Table

Add a properties table to a report and include properties for a model line segment only.

```
import slreportgen.report.*
model_name = 'vdp';
rpt = slreportgen.report.Report('output', 'pdf');
chapter = Chapter(model_name);

load_system(model_name);
ph = get_param('vdp/Mu', 'PortHandles');
outPort = ph.Outport;
line = get_param(outPort, 'Line');
rptr = SimulinkObjectProperties(line);
rptr.Properties = {'Parent', 'SourcePort', 'StorageClass'};

add(chapter, rptr);
add(rpt, chapter);
close(rpt);
close_system(model_name);
rptview(rpt);
```

### See Also

[mlreportgen.report.BaseTable](#) | [slreportgen.report.Report](#)

### Topics

“Report Generation for Simulink and Stateflow Elements” on page 1-9  
“What Is a Reporter?”

### Introduced in R2017b

## slreportgen.report.StateflowObjectProperties class

**Package:** slreportgen.report

Stateflow object properties reporter

### Description

The `StateflowObjectProperties` reporter generates tables that list the properties and property values of Stateflow objects.

---

**Note** To use a Stateflow object properties reporter in a report, you must create the report using the `slreportgen.report.Report` class.

---

### Construction

`reporter = StateflowObjectProperties()` creates an empty Stateflow object properties reporter. Use the properties of this reporter to specify reporting on these Stateflow objects:

- Object whose properties to report
- Properties to report
- Format of the reported properties

`reporter = StateflowObjectProperties(obj)` creates a reporter that generates a table listing the property values of the specified Stateflow object. The properties included by default depend on the object type. For example, a chart table includes the state and data of the chart. To specify a custom list of properties to be included in the generated property table, use the `Properties` property. Use the `PropertyTable` property to customize the format of the generated property table.

---

**Note** This reporter compiles the model containing the object to be reported if the model is not already compiled. Compiling the model is necessary to propagate values to properties that are unspecified when the model has not been compiled. The model is in an uncompiled state when you close the report that contains the generated property table.

---

`reporter = StateflowObjectProperties(Name, Value)` sets properties using name-value pairs. You can specify multiple name-value pair arguments in any order. Enclose each property name in single quotes.

### Input Arguments

**obj** — Stateflow object

Stateflow object handle

See Object property.

## Properties

### Object — Stateflow object to report

Stateflow object handle

Stateflow object whose properties to report, specified as the path string or character vector or as the handle of the specified object.

### PropertyTable — Object properties table reporter

mlreportgen.report.BaseTable reporter

Object properties table reporter, specified as an mlreportgen.report.BaseTable reporter. The object properties reporter uses the base table reporter to format object properties. If this property is initially empty, the object properties reporter sets the property to a default property table reporter. To customize the property table formatting, set this property to a base table reporter that meets your formatting requirements.

### ShowEmptyValues — Whether to show properties with empty values

false (default) | true

Choice to show properties with empty values.

Whether to show properties with empty values, specified as a logical. If false, the generated object properties table omits object properties whose value is empty. If true, the table includes properties whose value is empty.

### Properties — Names of properties to be reported

cell array of character vectors | cell array of strings

Names of object properties to be reported, specified as a cell array of strings or character vectors.

A cell array of names of object properties to be reported, specified as a cell array of strings or character vectors. If you do not specify any properties, the reporter determines a set of properties to report.

### TemplateSrc — Source of template for this reporter

[] (default) | character vector | string scalar | reporter or report | DOM document or document part

Source of the template for this reporter, specified in one of these ways:

- Character vector or string scalar that specifies the path of the file that contains the template for this reporter
- Reporter or report whose template is used for this reporter or whose template library contains the template for this reporter
- DOM document or document part whose template is used for this reporter or whose template library contains the template for this reporter

The specified template must be the same type as the report to which this reporter is appended. For example, for a Microsoft Word report, TemplateSrc must be a Word reporter template. If the TemplateSrc property is empty, this reporter uses the default reporter template for the output type of the report.

### TemplateName — Name of template for this reporter

character vector | string scalar

Name of the template for this reporter, specified as a character vector or string scalar. The template for this reporter must be in the template library of the template source (`TemplateSrc`) for this reporter.

### LinkTarget — Hyperlink target for this reporter

[ ] (default) | character vector | string scalar | `mlreportgen.dom.LinkTarget` object

Hyperlink target for this reporter, specified as a character vector or string scalar that specifies the link target ID, or an `mlreportgen.dom.LinkTarget` object. A character vector or string scalar value is converted to a `LinkTarget` object. The link target immediately precedes the content of this reporter in the output report.

## Methods

<code>createTemplate</code>	Create Stateflow object properties reporter template
<code>customizeReporter</code>	Create custom Stateflow object properties class
<code>getClassFolder</code>	Stateflow object properties class definition file location

## Inherited Methods

<code>copy</code>	Create copy of reporter object and make deep copies of property values that reference a reporter, <code>ReporterLayout</code> , or DOM object
<code>getImpl</code>	Get implementation of reporter

## Examples

### Add Stateflow Chart Properties Table to Report

Add a table that reports the properties of the `shift_model` chart in `sf_car` model.

```
import slreportgen.report.*
import mlreportgen.report.*
import slreportgen.utils.*

model_name = 'sf_car';
load_system(model_name);

rpt = slreportgen.report.Report('output', 'pdf');
chapter = Chapter(model_name);
chart = block2chart('sf_car/shift_logic');
rptr = StateflowObjectProperties(chart);

add(chapter, rptr);
add(rpt, chapter);
close(rpt);
close_system(model_name);
rptview(rpt);
```

## See Also

`mlreportgen.report.BaseTable` | `slreportgen.report.Report`

**Topics**

“Report Generation for Simulink and Stateflow Elements” on page 1-9  
“What Is a Reporter?”

**Introduced in R2017b**

## slreportgen.report.SystemHierarchy class

**Package:** slreportgen.report

System hierarchy reporter

### Description

Creates a system hierarchy reporter that generates a nested list of the subsystems of a Simulink model or subsystem in a report.

---

**Note** To use a system hierarchy reporter in a report, you must create the report using the `slreportgen.report.Report` class or subclass.

---

The `slreportgen.report.SystemHierarchy` class is a `handle` class.

### Class Attributes

`HandleCompatible` true

For information on class attributes, see “Class Attributes”.

## Creation

### Description

`rptr = slreportgen.report.SystemHierarchy()` creates an empty system hierarchy reporter based on a default template. Customize the content and format of the generated list by using the reporter properties. Before you add the reporter to a report, you must specify a model or subsystem in the `Source` property of the reporter. Adding an empty reporter to a report produces an error.

`rptr = slreportgen.report.SystemHierarchy(source)` creates a system hierarchy reporter for the model or subsystem specified by `source`. See the `Source` property.

`rptr = slreportgen.report.SystemHierarchy(Name,Value)` sets the reporter properties using name-value pairs. You can specify multiple name-value pair arguments in any order. Enclose each property name in single or double quotes.

## Properties

### Source — Simulink model or subsystem

`[]` (default) | string scalar | character vector | handle

Simulink model or subsystem, specified as a string scalar or character vector that contains the path to the model or subsystem, or as a handle to the model or subsystem.

### MaxAncestorLevel — Maximum number of levels of ancestors to include

`Inf` (default) | nonnegative integer scalar

Maximum number of levels of ancestors of the source subsystem to include in the generated list, specified as a nonnegative integer scalar. For example, if `MaxAncestorLevel` is 2, the list includes the source and up to two levels of ancestors. If `MaxAncestorLevel` is `Inf`, the default value, the list includes all ancestors. If `MaxAncestorLevel` is zero, the list does not include ancestors.

### **MaxDescendantLevel — Maximum number of levels of descendants to include**

`Inf` (default) | nonnegative integer scalar

Maximum number of levels of descendants of the source model or subsystem to include in the generated list, specified as a nonnegative integer scalar. For example, if `MaxDescendantLevel` is 2, the list includes the source and up to two levels of descendants. If `MaxDescendantLevel` is `Inf`, the default value, the list includes all descendants. If `MaxDescendantLevel` is zero, the list does not include descendants.

### **IncludePeers — Whether to include peers of the subsystem**

`true` (default) | `false`

Whether to include peers of the source subsystem in the generated list, specified as `true` or `false`.

### **EmphasizeSource — Whether to emphasize the source**

`true` (default) | `false`

Whether to emphasize the source model or subsystem in the generated list, specified as `true` or `false`. If `EmphasizeSource` is `true`, the name of the source model or subsystem is formatted according to the `TextFormatter` property. Otherwise, it is formatted like the other items in the list.

### **ListFormatter — List formatter**

`mlreportgen.dom.UnorderedList` (default) | `mlreportgen.dom.OrderedList`

List formatter that formats the generated list, specified as an `mlreportgen.dom.UnorderedList` object or an `mlreportgen.dom.OrderedList` object. To customize the list formatting, modify the list object properties or replace the list object with a customized list object that does not contain list items.

### **SourceTextFormatter — Text formatter for highlighting the name of the source**

`mlreportgen.dom.Text`

Text formatter object that formats the name of the source model or subsystem in the generated list, specified as an `mlreportgen.dom.Text` object. This property applies only if the `EmphasizeSource` property is `true`. The initial value of the `SourceTextFormatter` property is an `mlreportgen.dom.Text` object with the `Bold` and `Italic` properties set to `true`. To customize the appearance of the name in the generated list, modify the `mlreportgen.dom.Text` object properties or replace the object with a customized `mlreportgen.dom.Text` object. If you add text to the default or replacement text object, the text appears in front of the source name in the generated report.

### **IncludeMaskedSubsystems — Whether to include masked subsystems**

`false` (default) | `true`

Whether the generated list of descendants of the source system includes masked subsystems, specified as `true` or `false`. If `IncludeMaskedSubsystems` is `true`, the list includes masked subsystems and their descendant subsystems, as long as the number of levels below the source subsystem is less than or equal to the value of the `MaxDescendantLevel` property.

To enable the system hierarchy reporter to link masked subsystems to the corresponding diagrams in the report, in the diagram reporter, set the `MaskedSystemLinkPolicy` property to 'system'.

### **IncludeReferencedModels – Whether to include referenced models**

`true` (default) | `false`

Whether the generated list of descendants of the source system includes referenced models, specified as `true` or `false`. If `IncludeReferencedModels` is `true`, the list includes referenced models and their descendant subsystems, as long as the number of levels below the source subsystem is less than or equal to the value of the `MaxDescendantLevel` property.

### **IncludeSimulinkLibraryLinks – Whether to include Simulink library links**

`true` (default) | `false`

Whether the generated list of descendants of the source system includes subsystems that link to a Simulink library subsystem, specified as `true` or `false`. The list includes a linked subsystem or one of its descendant subsystems, only if all of these conditions are true:

### **IncludeUserLibraryLinks – Whether to include library links to user-defined libraries**

`true` (default) | `false`

Whether the generated list of descendants of the source system includes subsystems that link to a user-defined library subsystem, specified as `true` or `false`. The list includes a linked subsystem, or one of its descendant subsystems, only if all of these conditions are true:

### **IncludeVariants – Variants to include**

"Active" (default) | "All" | "ActivePlusCode"

Variants of a variant block to include in the generated list of descendants of the source system, specified as one of the values in the table. You can specify the value as a string scalar or a character vector.

Value	Description
"Active"	Active variants (default)
"All"	All variants
"ActivePlusCode"	Active variants and code variants

The list includes the variants only if the number of levels below the source subsystem is less than or equal to the value of the `MaxDescendantLevel` property.

### **TemplateSrc – Source of template for this reporter**

[] (default) | character vector | string scalar | reporter or report | DOM document or document part

Source of the template for this reporter, specified in one of these ways:

The specified template must be the same type as the report to which this reporter is appended. For example, for a Microsoft Word report, `TemplateSrc` must be a Word reporter template. If the `TemplateSrc` property is empty, this reporter uses the default reporter template for the output type of the report.



**TemplateName — Name of template for this reporter**

character vector | string scalar

Name of the template for this reporter, specified as a character vector or string scalar. The template for this reporter must be in the template library of the template source (`TemplateSrc`) for this reporter.

**LinkTarget — Hyperlink target for this reporter**[] (default) | character vector | string scalar | `mlreportgen.dom.LinkTarget` object

Hyperlink target for this reporter, specified as a character vector or string scalar that specifies the link target ID, or an `mlreportgen.dom.LinkTarget` object. A character vector or string scalar value is converted to a `LinkTarget` object. The link target immediately precedes the content of this reporter in the output report.

**Methods****Public Methods**

<code>slreportgen.report.SystemHierarchy.createTemplate</code>	Create system hierarchy reporter template
<code>slreportgen.report.SystemHierarchy.customizeReporter</code>	Create custom system hierarchy reporter class
<code>slreportgen.report.SystemHierarchy.getClassFolder</code>	Get location of system hierarchy reporter class definition file
<code>copy</code>	Create copy of reporter object and make deep copies of property values that reference a reporter, <code>ReporterLayout</code> , or DOM object
<code>getImpl</code>	Get implementation of reporter

**Examples****Include System Hierarchy of a Model in a Report**

Include the system hierarchy of the `fuelsys` model in a report by adding an `slreportgen.report.SystemHierarchy` reporter to a report generation program. Generate the model diagrams by adding an `slreportgen.finder.DiagramFinder` object. The system hierarchy reporter generates links from subsystems in the nested list to the corresponding diagrams.

```
% Import the API packages
import slreportgen.report.*
import mlreportgen.report.*
import mlreportgen.dom.*

% Load the model
model = 'fuelsys';
load_system(model);

% Create a report
rpt = slreportgen.report.Report('output','pdf');

% Create a chapter reporter
chapter = Chapter("System Hierarchy for the " + model + " Model");
```

```

% Create a SystemHierarchy reporter for the model
rptr = SystemHierarchy(model);

% Add the SystemHierarchy reporter to the chapter.
% Add the chapter to the report
add(chapter, rptr);
add(rpt, chapter);

% Find the diagrams for the subsystems
finder = slreportgen.finder.DiagramFinder(model);
while hasNext(finder)
    result = next(finder);
    ch = Chapter(result.Name);
    add(ch, result);
    add(rpt, ch);
end
















% Close and view the output report
close(rpt);
close_system(model);
rptview(rpt);

```

Here is the system hierarchy in the generated report:

---

## Chapter 1. System Hierarchy for the fuelsys Model

-  **fuelsys**
  -  engine gas dynamics
    -  Mixing & Combustion
    -  Throttle & Manifold
      -  Intake Manifold
      -  Throttle
  -  fuel rate controller
    -  Airflow calculation
    -  Fuel Calculation
      -  Switchable Compensation
    -  Sensor correction and Fault Redundancy
      -  MAP Estimate
      -  Speed Estimate
      -  Throttle Estimate
  -  control logic

To see the diagram corresponding to a subsystem, click the subsystem in the list.

## **See Also**

`slreportgen.finder.DiagramFinder`

## **Topics**

“Report Systems Hierarchically” on page 4-25

“Report Generation for Simulink and Stateflow Elements” on page 1-9

“What Is a Reporter?”

**Introduced in R2019b**

## slreportgen.report.SystemIO class

**Package:** slreportgen.report

Simulink system input and output signal reporter

### Description

Create a reporter that reports on signals entering or leaving a Simulink model or subsystem.

---

**Note** To use a `SystemIO` reporter in a report, you must create the report using the `slreportgen.report.Report` class.

---

The `slreportgen.report.SystemIO` class is a `handle` class.

### Class Attributes

<code>HandleCompatible</code>	<code>true</code>
-------------------------------	-------------------

For information on class attributes, see “Class Attributes”.

## Creation

### Description

`slreportgen.report.SystemIO()` creates an empty `slreportgen.report.SystemIO` reporter object. Use the `Object` property to specify the model or subsystem to be reported. By default, the reporter generates summary tables for the system inputs and outputs. The input summary table lists the sources of the input signals. The output summary table lists the destinations of the output signals. The reporter also generates a signal details section that lists the properties of the input and output ports. Use the reporter properties to customize the content and appearance of the generated report. For example, use the `DetailsReporter` property to customize the content and appearance of the signal details section.

`slreportgen.report.SystemIO(object)` creates a reporter for the model or subsystem specified by `object`. See the `Object` property.

`slreportgen.report.SystemIO(Name, Value)` sets the reporter properties using name-value pairs. You can specify multiple name-value pair arguments in any order. Enclose each property name in single or double quotes.

## Properties

### Object — Model or subsystem to be reported

`string scalar` | `character vector` | `handle` | `slreportgen.finder.DiagramResult` object | `slreportgen.finder.BlockResult` object

Simulink model or subsystem to be reported, specified as a string scalar or character vector that contains the path to the model or subsystem, as a handle to the model or subsystem, as an

`slreportgen.finder.DiagramResult` object, or as an `slreportgen.finder.BlockResult` object.

### **InputSummaryProperties — List of properties to report for each input**

string array | cell array of character vectors

List of properties to report for each input, specified as a string array or a cell array of character vectors. You can include these properties in the list:

By default, the list includes `Source`, `Name`, and `DataType`.

### **OutputSummaryProperties — List of properties to report for each output**

string array | cell array of character vectors

List of properties to report for each output, specified as a string array or a cell array of character vectors. You can include these properties in the list:

By default, the list includes `Destination`, `Name`, and `DataType`.

### **ShowInputSummary — Whether to show input summary table**

true (default) | false

Whether to show a table that summarizes the inputs to the subsystem or model, specified as `true` or `false`.

### **ShowOutputSummary — Whether to show output summary table**

true (default) | false

Whether to show a table that summarizes the outputs from the subsystem or model, specified as `true` or `false`.

### **ShowDetails — Whether to show details for each input and output**

true (default) | false

Whether to show details for each input or output, specified as `true` or `false`. If `ShowDetails` is `true`, the reporter inserts `slreportgen.report.SimulinkObjectProperties` reporters after the input and output summary tables. If `Object` is a model, details about the input or output blocks are included. If `Object` is a subsystem, details about the input or output ports are included. The port numbers in the summary tables link to the corresponding `SimulinkObjectProperties` reporter for that port.

### **ShowEmptyColumns — Whether to show empty columns in summary tables**

false (default) | true

Whether to show empty columns in summary tables, specified as `true` or `false`. If `ShowEmptyColumns` is `true`, the summary tables include columns that do not have data.

### **InputSummaryReporter — Table formatter for input summary tables**

`mlreportgen.report.BaseTable` object

Table formatter for input summary tables, specified as an `mlreportgen.report.BaseTable` reporter. The default value is a `BaseTable` reporter. To customize the appearance of the table, modify the properties of the default table reporter or replace it with a customized table reporter. If

you add content to the `Title` property of the default or customized table reporter, the content appears in front of the table title in the generated report.

### **OutputSummaryReporter — Table formatter for output summary tables**

`mlreportgen.report.BaseTable` object

Table formatter for output summary tables, specified as an `mlreportgen.report.BaseTable` reporter. The default value is a `BaseTable` reporter. To customize the appearance of the table, modify the properties of the default table reporter or replace it with a customized table reporter. If you add content to the `Title` property of the default or customized table reporter, the content appears in front of the table title in the generated report.

### **DetailsReporter — Formatter for detail tables**

`slreportgen.report.SimulinkObjectProperties` object

Formatter for detail tables, specified as an `slreportgen.report.SimulinkObjectProperties` reporter. The default value is an `SimulinkObjectProperties` reporter. To customize the appearance of the detail tables, modify the properties of the default `SimulinkObjectProperties` reporter or replace it with a customized `SimulinkObjectProperties` reporter.

### **ListFormatter — List formatter for source and destination lists**

`mlreportgen.dom.UnorderedList` object | `mlreportgen.dom.OrderedList` object

List formatter for the source and destination lists, specified as an `mlreportgen.dom.UnorderedList` or `mlreportgen.dom.OrderedList` object. The source list is the list of blocks to which an input signal is connected. The destination list is the list of blocks to which an output signal is connected. The default formatter is an `UnorderedList` object. To customize the appearance of the list, modify the properties of the default list formatter or replace it with a customized list object that does not contain any children.

### **TemplateSrc — Source of template for this reporter**

`[]` (default) | character vector | string scalar | reporter or report | DOM document or document part

Source of the template for this reporter, specified in one of these ways:

The specified template must be the same type as the report to which this reporter is appended. For example, for a Microsoft Word report, `TemplateSrc` must be a Word reporter template. If the `TemplateSrc` property is empty, this reporter uses the default reporter template for the output type of the report.

### **TemplateName — Name of template for this reporter**

character vector | string scalar

Name of the template for this reporter, specified as a character vector or string scalar. The template for this reporter must be in the template library of the template source (`TemplateSrc`) for this reporter.

### **LinkTarget — Hyperlink target for this reporter**

`[]` (default) | character vector | string scalar | `mlreportgen.dom.LinkTarget` object

Hyperlink target for this reporter, specified as a character vector or string scalar that specifies the link target ID, or an `mlreportgen.dom.LinkTarget` object. A character vector or string scalar value is converted to a `LinkTarget` object. The link target immediately precedes the content of this reporter in the output report.

## Methods

### Public Methods

slreportgen.report.SystemIO.createTemplate	Copy the default slreportgen.report.SystemIO reporter template
slreportgen.report.SystemIO.customizeReporter	Create subclass of slreportgen.report.SystemIO class
slreportgen.report.SystemIO.getClassFolder	Get location of folder that contains the slreportgen.report.SystemIO class definition file
copy	Create copy of reporter object and make deep copies of property values that reference a reporter, ReporterLayout, or DOM object
getImpl	Get implementation of reporter

## Examples

### Report on Inputs and Outputs of a Model

This example uses an `slreportgen.report.SystemIO` reporter to report on the inputs and outputs of a model and its subsystems.

```

model_name = "f14";
load_system(model_name);
% Create a Simulink report
rpt = slreportgen.report.Report("SystemIO_example", "docx");

% Create finder to find all diagrams in model
finder = slreportgen.finder.DiagramFinder(model_name);

% Report inputs and outputs of each diagram
ch = mlreportgen.report.Chapter("Diagrams");
while hasNext(finder)
    result = next(finder);
    if strcmpi(result.Type, "Simulink.SubSystem") ...
        || strcmpi(result.Type, "Simulink.BlockDiagram")
        sect = mlreportgen.report.Section(result.Name);
        add(sect, result);
        % Create SystemIO reporter and add to report
        ioRptr = slreportgen.report.SystemIO(result);
        add(sect, ioRptr);

    add(ch, sect);
end
end

% Add chapter to report and close report
add(rpt, ch);
close(rpt);
rptview(rpt);

```

## Tips

- The input and output signal properties reported by the SystemIO reporter correspond to Simulink properties, which you can query by using `get_param`. For example, the `DataType` and `Dimensions` properties correspond to the Simulink `CompiledPortDataType` and `CompiledPortDimensions` properties of the port handles.
- For bus signals, Simulink determines the values of the `CompiledPortDataType` and `CompiledPortDimensions` properties based on whether the signal is a nonvirtual or virtual bus.

## See Also

`slreportgen.finder.BlockResult` | `slreportgen.finder.DiagramResult`

## Topics

“Report System Inputs and Outputs” on page 4-34

“Report Generation for Simulink and Stateflow Elements” on page 1-9

“What Is a Reporter?”

## Introduced in R2020a



# slreportgen.report.TestSequence class

**Package:** slreportgen.report

Test Sequence block reporter

## Description

Use an object of the `slreportgen.report.TestSequence` class to report on a Simulink Test Sequence block. Using a Test Sequence block in a Simulink model requires Simulink Test.

By default, a `TestSequence` reporter generates:

- Property tables for all of the symbols — input, output, local, constant, parameter, and data store memory
- A nested list for the step hierarchy
- Details for each step including the step description, When condition, action statements, and a table of transition conditions and next steps

If the Test Sequence block uses scenarios, the `TestSequence` reporter includes the scenario parameter in the parameters table and a list for each scenario in the step hierarchy. In the report, scenarios are identified by the scenario icon. The active scenario is also identified by the word **Active** followed by the active scenario icon. For example:

 **Scenario 2 (Active ⚡)**

The reporter adds a note to the report when the active scenario is controlled in the workspace. In this case, the active scenario is not identified in the report.

Use the `TestSequence` reporter properties to filter the content and customize the content formatting.

---

**Note** To use an `slreportgen.report.TestSequence` reporter in a report, you must create the report using the `slreportgen.report.Report` class or subclass.

---

The `slreportgen.report.TestSequence` class is a `handle` class.

### Class Attributes

`HandleCompatible` true

For information on class attributes, see “Class Attributes”.

## Creation

### Description

`reporter = slreportgen.report.TestSequence()` creates an empty `TestSequence` reporter object based on the default template. You must specify a Test Sequence block to report by setting the `Object` property. Use other properties to specify report options.

`reporter = slreportgen.report.TestSequence(testSeqObject)` creates a `TestSequence` reporter and sets the `Object` property to the specified Test Sequence block.

`reporter = slreportgen.report.TestSequence(Name, Value)` sets the reporter properties using name-value pairs. You can specify multiple name-value pair arguments in any order. Enclose each property name in single or double quotes.

### Properties

#### Object — Test Sequence block to report

[] (default) | character vector | string scalar | handle | `slreportgen.finder.BlockResult` | `slreportgen.finder.DiagramElementResult`

Test Sequence block to report, specified as one of these types of values:

#### IncludeSymbols — Whether to include symbols

true (default) | false

Whether to include symbols in the report, specified as `true` or `false`. If `IncludeSymbols` is `true`, the report includes property tables for the symbols. In the input and output symbol property tables, the symbols are sorted by the port number. In the local, constant, parameter, and data store memory tables, the symbols are sorted by the symbol name.

If the model was compiled before report generation, the values in the property tables are the values after compilation. Otherwise, the property values are the values before compilation and a note at the end of the Symbols section states that the model was not compiled. By default, models are compiled during report generation. You can control whether a model is compiled during report generation by setting the `CompileModelBeforeReporting` property of the `slreportgen.report.Report` object that includes this reporter.

Data Types: logical

#### IncludeStepHierarchy — Whether to include step hierarchy

true (default) | false

Whether to include the step hierarchy, specified as `true` or `false`. If `IncludeStepHierarchy` is `true`, the report includes the step hierarchy as a nested list. The step name in the list links to the corresponding step content in the report.

Data Types: logical

#### IncludeStepContent — Whether to include step content

true (default) | false

Whether to include step content, specified as `true` or `false`. If `IncludeStepContent` is `true`, the report includes the content for each step. In the reported step content, the step name links to the

step hierarchy in the report. You can filter the reported step content by using the `IncludeStepDescription`, `IncludeStepWhenCondition`, `IncludeStepAction`, `IncludeStepTransitions`, and `IncludeStepRequirements` properties.

Data Types: `logical`

#### **IncludeStepDescription — Whether to include step description**

`true` (default) | `false`

Whether to include the step description in the content reported for a step, specified as `true` or `false`.

Data Types: `logical`

#### **IncludeStepWhenCondition — Whether to include step When condition**

`true` (default) | `false`

Whether to include the step When condition in the content reported for a step, specified as `true` or `false`. The When condition is the condition that activates a When decomposition child step.

Data Types: `logical`

#### **IncludeStepAction — Whether to include step actions**

`true` (default) | `false`

Whether to include the step actions in the content reported for a step, specified as `true` or `false`.

Data Types: `logical`

#### **IncludeStepTransitions — Whether to include step transitions table**

`true` (default) | `false`

Whether to include the step transitions table in the content reported for a step, specified as `true` or `false`. The step transitions table contains the transition conditions and the next steps.

Data Types: `logical`

#### **IncludeStepRequirements — Whether to include step requirements**

`false` (default) | `true`

Whether to include a link to the step requirements in the content reported for a step, specified as `true` or `false`. Linking to the step requirements requires Simulink Requirements.

Data Types: `logical`

#### **TableReporter — Table formatter**

`mlreportgen.report.BaseTable` object

Table formatter for tables generated by this reporter, specified as an `mlreportgen.report.BaseTable` object. The default value of this property is a `BaseTable` object with the `TableStyleName` property set to the `TestSequenceTable` style, which is defined in the default template for a `TestSequence` reporter. To customize the appearance of the table, modify the properties of the default `BaseTable` object or replace it with your own `BaseTable` object. If you add content to the `Title` property of the `BaseTable` object, the content appears in front of the table title in the generated report.

#### **ListFormatter — Formatter for step hierarchy list**

`mlreportgen.dom.UnorderedList` object | `mlreportgen.dom.OrderedList` object

Formatter for the step hierarchy list, specified as an `mlreportgen.dom.UnorderedList` object or `mlreportgen.dom.OrderedList` object. The `UnorderedList` or `OrderedList` object must not contain list items.

The default value of this property is an `UnorderedList` object with the `StyleName` property set to the `TestSequenceList` style, which is defined in the default template for a `TestSequence` reporter. To customize the appearance of the list, modify the properties of the default `UnorderedList` object or replace the object with your own `UnorderedList` or `OrderedList` object.

### TemplateSrc — Source of template for this reporter

[ ] (default) | character vector | string scalar | reporter or report | DOM document or document part

Source of the template for this reporter, specified in one of these ways:

The specified template must be the same type as the report to which this reporter is appended. For example, for a Microsoft Word report, `TemplateSrc` must be a Word reporter template. If the `TemplateSrc` property is empty, this reporter uses the default reporter template for the output type of the report.

### TemplateName — Name of template for this reporter

character vector | string scalar

Name of the template for this reporter, specified as a character vector or string scalar. The template for this reporter must be in the template library of the template source (`TemplateSrc`) for this reporter.

### LinkTarget — Hyperlink target for this reporter

[ ] (default) | character vector | string scalar | `mlreportgen.dom.LinkTarget` object

Hyperlink target for this reporter, specified as a character vector or string scalar that specifies the link target ID, or an `mlreportgen.dom.LinkTarget` object. A character vector or string scalar value is converted to a `LinkTarget` object. The link target immediately precedes the content of this reporter in the output report.

## Methods

### Public Methods

<code>slreportgen.report.TestSequence.createTemplate</code>	Create Test Sequence block reporter template
<code>slreportgen.report.TestSequence.customizeReporter</code>	Create custom Test Sequence block reporter class
<code>slreportgen.report.TestSequence.getClassFolder</code>	Get location of Test Sequence block reporter class definition file
<code>copy</code>	Create copy of reporter object and make deep copies of property values that reference a reporter, <code>ReporterLayout</code> , or DOM object
<code>getImpl</code>	Get implementation of reporter

## Examples

### Report on a Test Sequence Block

Use an `slreportgen.report.TestSequence` object to report on a Test Sequence block.

Import the MATLAB Report and Simulink Report API packages so that you do not have to use long, fully qualified class names.

```
import mlreportgen.report.*
import slreportgen.report.*
```

Create a Simulink report.

```
rpt = slreportgen.report.Report("myTestSequenceReport", "pdf");
```

Load a model that has a test harness.

```
model_name = "sltestTestSequenceExample";
load_system(model_name);
```

Find and load the test harness that contains the Test Sequence block to report.

```
harness = sltest.harness.find(strcat(model_name, "/shift_controller"));
sltest.harness.load(harness.ownerFullPath, harness.name);
testSeqObj = strcat(harness.name, "/Test Sequence");
```

Create a chapter for the Test Sequence block.

```
chapter = Chapter(testSeqObj);
```

Create a reporter for the Test Sequence block.

```
rptr = TestSequence(testSeqObj);
```

Append the reporter to the chapter and the chapter to the report.

```
append(chapter, rptr);
append(rpt, chapter);
```

Close the report, test harness, and model. View the report.

```
close(rpt);
sltest.harness.close(harness.ownerFullPath, harness.name);
close_system(model_name);
rptview(rpt);
```

## See Also

Test Sequence

## Topics

“Report Generation for Simulink and Stateflow Elements” on page 1-9

“What Is a Reporter?”

“Test Sequence Editor” (Simulink Test)

## Introduced in R2020b

## slreportgen.report.TruthTable class

**Package:** slreportgen.report

Truth table reporter

### Description

Create a Simulink truth table block or Stateflow truth table object reporter.

---

**Note** To use a TruthTable reporter in a report, you must create the report using the slreportgen.report.Report class.

---

### Construction

`rptr = TruthTable()` creates an empty TruthTable reporter. Use its properties to specify the truth table on which to report and specify report options and format.

`rptr = TruthTable(truthtableobj)` creates a TruthTable reporter for the truth table specified by `truthtableobj`, which can be either a block or a Stateflow object. By default, the reporter generates a table of the conditions and actions of the truth table.

`rptr = TruthTable(Name,Value)` creates a truth table reporter with additional options specified by one or more `Name,Value` pair arguments. `Name` is a property name and `Value` is the corresponding value. `Name` must appear inside single quotes ( ' ') or double quotes ( " "). You can specify several name-value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

### Input Arguments

#### **truthtableobj** — Truth table object

Simulink Truth Table block path | Simulink Truth Table block handle | Stateflow Truth Table handle

See the Object property.

### Properties

#### **Object** — Truth table block or object

Simulink Truth Table block path | Simulink Truth Table block handle | Stateflow Truth Table handle

Simulink Truth Table block or Stateflow truth table object, specified as a path or handle.

#### **IncludeConditionTableHeader** — Include headers in condition table

true (default) | false

Specify whether to include headers in the truth table condition table, specified as a logical. If this property is true, the report includes column headers that identify the contents of the table columns.

#### **IncludeConditionTableRowNumber** — Include row numbers in condition table

true (default) | false

Specify whether to include row numbers in the truth table condition table, specified as a logical. If this property is `true`, each row of the condition table starts with a row number.

**IncludeConditionTableConditionCol — Include condition column in condition table**

`true` (default) | `false`

Specify whether to include the condition column in the truth table condition table, specified as a logical. If this property is `true`, the report includes the conditions that trigger the decisions.

**IncludeConditionTableDescriptionCol — Include description column in condition table**

`true` (default) | `false`

Specify whether to include the description column in the truth table condition table, specified as a logical. If this property is `true`, the report includes descriptions of the truth table conditions.

**ConditionTableReporter — Reporter for truth table condition table**

`BaseTable reporter` (default) | `custom reporter`

Reporter used by the `TruthTable` reporter to create the truth table condition table. This property is set by default to an instance of a `BaseTable` reporter.

Use the associated `BaseTable` reporter properties to customize the appearance of the condition table. If the condition table is too wide to fit legibly on a page, use the `MaxCols` property of the `BaseTable` reporter to generate the condition table as a set of table slices that fit legibly.

---

**Note** The `TruthTable` reporter always repeats the first two columns of the condition table in each slice. It does not use the `RepeatCols` property of the `BaseTable` or custom reporter.

---

**IncludeActionTableHeader — Include headers in action table**

`true` (default) | `false`

Specify whether to include headers in the truth table action table, specified as a logical. If this property is `true`, the action table includes the column headers, such as "Description" that identify the contents of each column.

**IncludeActionTableRowNumber — Include row numbers in action table**

`true` (default) | `false`

Specify whether to include row numbers in the truth table action table, specified as a logical. If this property is `true`, each row of the action table starts with a row number.

**IncludeActionTableActionCol — Include action column in action table**

`true` (default) | `false`

Specify whether to include the action column in the truth table action table, specified as a logical. If this property is `true`, each row of the action table lists the executable action statements for each action.

**IncludeActionTableDescriptionCol — Include description column in action table**

`true` (default) | `false`

Specify whether to include the description column in the truth table action table, specified as a logical. If this property is `true`, each row of the action table contains a description of the corresponding action.

**ActionTableReporter — Reporter for truth table action table**

BaseTable reporter (default) | custom reporter

Reporter used by the TruthTable reporter to create the truth table's action table. This property is set by default to an instance of a BaseTable reporter. You can customize the appearance of the action table by changing the properties of this table reporter or by replacing it with a customized version of a BaseTable reporter.

**TemplateSrc — Source of template for this reporter**

[] (default) | character vector | string scalar | reporter or report | DOM document or document part

Source of the template for this reporter, specified in one of these ways:

- Character vector or string scalar that specifies the path of the file that contains the template for this reporter
- Reporter or report whose template is used for this reporter or whose template library contains the template for this reporter
- DOM document or document part whose template is used for this reporter or whose template library contains the template for this reporter

The specified template must be the same type as the report to which this reporter is appended. For example, for a Microsoft Word report, TemplateSrc must be a Word reporter template. If the TemplateSrc property is empty, this reporter uses the default reporter template for the output type of the report.

**TemplateName — Name of template for this reporter**

character vector | string scalar

Name of the template for this reporter, specified as a character vector or string scalar. The template for this reporter must be in the template library of the template source (TemplateSrc) for this reporter.

**LinkTarget — Hyperlink target for this reporter**

[] (default) | character vector | string scalar | mlreportgen.dom.LinkTarget object

Hyperlink target for this reporter, specified as a character vector or string scalar that specifies the link target ID, or an mlreportgen.dom.LinkTarget object. A character vector or string scalar value is converted to a LinkTarget object. The link target immediately precedes the content of this reporter in the output report.

**Methods**

createTemplate	Create truth table template
customizeReporter	Create custom truth table reporter class
getClassFolder	Location of truth table class definition file

**Inherited Methods**

copy	Create copy of reporter object and make deep copies of property values that reference a reporter, ReporterLayout, or DOM object
------	---



getImpl	Get implementation of reporter
---------	--------------------------------

## Examples

### Add Truth Table to a PDF Report

```
import slreportgen.report.*
import mlreportgen.report.*

model_name = 'sf_climate_control';
load_system(model_name)

rpt = slreportgen.report.Report('output','pdf');
truthtableobj = 'sf_climate_control/ClimateController';

chapter = Chapter(truthtableobj);
rptr = TruthTable(truthtableobj);
rptr.IncludeConditionTableRowNumber = false;
add(chapter,rptr)
add(rpt,chapter)

close(rpt)
close_system(model_name)
rptview(rpt)
```

## Chapter 1. sf\_climate\_control/ClimateController

**Table 1.1. Condition Table for ClimateController**

Description	Condition	D1	D2	D3	D4
Hot	$t > T\_thresh$	T	T	-	-
Dry	$h < H\_thresh$	T	-	T	-
	<b>Actions:</b>	<b>CoolOn,HumidOn</b>	<b>CoolOn</b>	<b>HeatOn,HumidOn</b>	<b>HeatOn</b>

**Table 1.2. Action Table for ClimateController**

#	Description	Action
1	Turn On Cooling (This implicitly reduces humidity)	CoolOn: cooler = 1; heater = 0; humidifier = 0;
2	Turn On Heater (This implicitly reduces humidity)	HeatOn: heater = 1; cooler = 0; humidifier = 0;
3	Turn On Humidifier	HumidOn: humidifier = 1;

### Slice Truth Table Condition Table

```
import slreportgen.report.*
import mlreportgen.report.*

model_name = 'sf_climate_control';
load_system(model_name)

rpt = slreportgen.report.Report('output','pdf');
truthtableobj = 'sf_climate_control/ClimateController';

chapter = Chapter(truthtableobj);
rptr = TruthTable(truthtableobj);
rptr.IncludeConditionTableRowNumber = false;
rptr.ConditionTableReporter.MaxCols = 4;
add(chapter,rptr)
add(rpt,chapter)

close(rpt)
close_system(model_name)
rptview(rpt)
```

## Chapter 1. sf\_climate\_control/ClimateController

**Table 1.1. Condition Table for ClimateController**

From column 1 to column 2, from column 3 to column 4.

Description	Condition	D1	D2
Hot	$t > T_{\text{thresh}}$	T	T
Dry	$h < H_{\text{thresh}}$	T	-
	<b>Actions:</b>	<b>CoolOn,HumidOn</b>	<b>CoolOn</b>

From column 1 to column 2, from column 5 to column 6.

Description	Condition	D3	D4
Hot	$t > T_{\text{thresh}}$	-	-
Dry	$h < H_{\text{thresh}}$	T	-
	<b>Actions:</b>	<b>HeatOn,HumidOn</b>	<b>HeatOn</b>

**Table 1.2. Action Table for ClimateController**

#	Description	Action
1	Turn On Cooling (This implicitly reduces humidity)	CoolOn: cooler = 1; heater = 0; humidifier = 0;
2	Turn On Heater (This implicitly reduces humidity)	HeatOn: heater = 1; cooler = 0; humidifier = 0;
3	Turn On Humidifier	HumidOn: humidifier = 1;

### See Also

mlreportgen.report.BaseTable | mlreportgen.utils.TableSlice |  
mlreportgen.utils.TableSlicer | slreportgen.finder.BlockFinder |  
slreportgen.finder.BlockResult | slreportgen.finder.DiagramElementFinder |  
slreportgen.finder.DiagramElementResult |  
slreportgen.finder.StateflowDiagramElementFinder |  
slreportgen.utils.isTruthTable

**Introduced in R2018b**

# slreportgen.finder.AnnotationFinder class

**Package:** slreportgen.finder

Find Simulink annotation objects

## Description

Find annotation objects in a Simulink or Stateflow diagram.

## Construction

`finder = AnnotationFinder(diagram)` creates a finder that finds by default all annotations in the specified diagram. To constrain the search to specific types of annotations, use the properties of this finder.

---

**Note** This finder provides two ways to get search results:

- 1 To return the search results as an array, use the `find` method. Add the results directly to a report or process the results in a `for` loop.
- 2 To iterate through the results one at a time, use the `hasNext` and `next` methods in a `while` loop.

Neither option has a performance advantage.

---

`finder = AnnotationFinder(Name, Value)` sets properties using name-value pairs. You can specify multiple name-value pair arguments in any order. Enclose each property name in single or double quotes.

## Input Arguments

### **diagram** — Diagram to search

path | handle | chart ID | chart object

See Container property.

## Properties

### **Container** — Diagram to search

path | handle | chart ID | chart object

Diagram in which to search, specified as one of these values:

- Handle to a Simulink block
- Path to a Simulink block
- Handle to a Stateflow chart block
- Path to a Stateflow chart block
- Stateflow chart ID

- Stateflow chart object

### Properties — Properties of objects to find

cell array

Properties of objects to find, specified as a cell array of name-value pairs. The finder returns only objects that have the specified properties with the specified values.

Example: `finder.Properties = {'Gain','5'}`

### Methods

`results = find(finder)` finds annotations in the diagram specified by the finder. This method returns the annotations it finds wrapped in result objects of type `slreportgen.finder.DiagramElementResult`. To add tables of the annotation properties, add the results objects directly to the report or add them to a reporter that you then add to a report. The reports to which you can add the results of this method must be reports of type `slreportgen.report.Report`.

`tf = hasNext(finder)` determines if the diagram that the finder searches contains at least one annotation. If the diagram has at least one annotation, the `hasNext` method queues that annotation as the next annotation that the `next` method will return. The `hasNext` method then returns `true`. Use the `next` method to obtain that annotation. On subsequent calls, the `hasNext` method determines if the diagram has an annotation that the `next` method has not yet retrieved. It queues the annotation for the `next` method to retrieve and returns `true`. If there are no more annotations to be retrieved, this method returns `false`. To search a diagram progressively for annotations, use the `hasNext` method with the `next` method in a while loop.

`result = next(finder)` returns the next search result in the result queue that the `hasNext` method created. This method returns the annotation that it finds wrapped in a result object of type `slreportgen.finder.DiagramElementResult`. To add tables of the annotation properties, add the results objects directly to the report or add them to a reporter that you then add to a report. The reports to which you can add the results of this method must be of type `slreportgen.report.Report`.

### Copy Semantics

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

### Examples

#### Find Annotations in a Model

Create a report that finds annotations in the `sf_car` model.

```
import mlreportgen.report.*
import slreportgen.report.*
import slreportgen.finder.*

model_name = 'sf_car';
load_system(model_name);

rpt = slreportgen.report.Report('output','pdf');
```

```
add(rpt, TitlePage("Title",...
    sprintf('Annotations in %s Model',model_name)));
add(rpt, TableOfContents);

diagFinder = SystemDiagramFinder(model_name);
diagrams = find(diagFinder);
while hasNext(diagFinder)
    diagram = next(diagFinder);
    annotFinder = AnnotationFinder(diagram.Object);
    annotations = find(annotFinder);
    if ~isempty(annotations)
        chapter = Chapter("Title",diagram.Name);
        add(chapter, diagram);
        sect = Section("Title","Annotations");
        add(sect,annotations);
        add(chapter,sect);
        add(rpt,chapter);
    end
end

close(rpt);
close_system(model_name);
rptview(rpt);
```

## See Also

[slreportgen.finder.BlockFinder](#) | [slreportgen.finder.DiagramElementResult](#) |  
[slreportgen.finder.SystemDiagramFinder](#) | [slreportgen.report.Diagram](#) |  
[slreportgen.report.Report](#) | [slreportgen.report.SimulinkObjectProperties](#)

## Topics

“Report Generation for Simulink and Stateflow Elements” on page 1-9  
“What Is a Reporter?”

## Introduced in R2017b

## slreportgen.finder.BlockFinder class

**Package:** slreportgen.finder

Find Simulink blocks

### Description

Finds blocks in a Simulink diagram.

### Construction

`finder = BlockFinder(diagram)` creates a finder that finds by default all types of blocks in the specified Simulink block diagram. To constrain the search to specific types of blocks, use the properties of the finder.

---

**Note** This finder provides two ways to get search results:

- 1 To return the search results as an array, use the `find` method. Add the results directly to a report or process the results in a `for` loop.
- 2 To iterate through the results one at a time, use the `hasNext` and `next` methods in a `while` loop.

Neither option has a performance advantage.

---

`finder = BlockFinder(Name, Value)` sets properties using name-value pairs. You can specify multiple name-value pair arguments in any order. Enclose each property name in single quotes.

### Input Arguments

**diagram** — Block diagram to search

path | handle

See Container property.

### Properties

**Container** — Diagram to search

path | handle

Diagram in which to search, specified as one of these values:

- Handle to a Simulink model or subsystem
- Path to a Simulink model or subsystem

**BlockTypes** — Types of blocks to find

string | character array | string array | cell array of character arrays

Type of block to find, such as Gain, specified as a string or character array, or a set of block types to find, specified as a string array or a cell array of character arrays.



**IncludeCommented — Include commented-out blocks**`false (default) | true`

Whether to include commented-out blocks in the search results, specified as a logical. If `false`, commented-out blocks are excluded from the search results.

**IncludeVariants — Variants to include**`"Active" (default) | "All" | "ActivePlusCode"`

Variants of a variant block to include in the search results, specified as one of the values in the table. You can specify the value as a string scalar or a character vector.

Value	Description
"Active"	Active variants (default)
"All"	All variants
"ActivePlusCode"	Active variants and code variants

**Properties — Properties of objects to find**`cell array`

Properties of objects to find, specified as a cell array of name-value pairs. The finder returns only objects that have the specified properties with the specified values.

Example: `finder.Properties = {'Gain','5'}`

**Methods**

`results = find(finder)` finds blocks in the diagram specified by the finder. This method returns the blocks it finds wrapped in result objects of type `slreportgen.finder.BlockResult`. To add tables of the block properties, add the results objects directly to the report or add them to a reporter that you then add to a report. The reports to which you can add the results of this method must be of reports of type `slreportgen.report.Report`

`tf = hasNext(finder)` determines if the diagram that the finder searches contains at least one block. If the diagram has at least one block, the `hasNext` method queues that block as the next block that the `next` method will return. The `hasNext` method then returns `true`. Use the `next` method to obtain that block. On subsequent calls, the `hasNext` method determines if the diagram has a block that the `next` method has not yet retrieved. It queues the block for the `next` method to retrieve and returns `true`. If there are no more blocks to be retrieved, this method returns `false`. To search a diagram progressively for blocks, use the `hasNext` method with the `next` method in a while loop.

`result = next(finder)` returns the next search result in the result queue that the `hasNext` method created. This method returns the block that it finds wrapped in a result object of type `slreportgen.finder.BlockResult`. To add tables of the block properties, add the result object to the report directly or add it to a reporter that you then add to a report. The reports to which you can add the results of this method must be of type `slreportgen.report.Report`.

**Copy Semantics**

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

## Examples

### Find Inport and Output Blocks in a Model

Find Inport and Output blocks in the `sf_car` model.

```
import mlreportgen.report.*
import slreportgen.report.*
import slreportgen.finder.*

model_name = 'sf_car';
load_system(model_name)
rpt = slreportgen.report.Report('output','pdf');

add(rpt,TitlePage("Title",...
    sprintf('I/O Blocks in %s Model',model_name)));
add(rpt,TableOfContents);

diagFinder = SystemDiagramFinder(model_name);
diagFinder.IncludeRoot = false;
while hasNext(diagFinder)
    diagram = next(diagFinder);
    chapter = Chapter("Title",diagram.Name);
    add(chapter,diagram)
    sect = Section("Title","Inport Blocks");
    ioFinder = BlockFinder(diagram.Object);
    ioFinder.BlockTypes = "Inport";
    blocks = find(ioFinder);
    for block = blocks
        add(sect,block)
    end
    add(chapter,sect);
    sect = Section("Title","Outport Blocks");
    ioFinder = BlockFinder(diagram.Object);
    ioFinder.BlockTypes = "Outport";
    outblocks = find(ioFinder);
    for block = outblocks
        add(sect,block)
    end
    add(chapter,sect)
    add(rpt,chapter)
end
close(rpt)
close_system(model_name)
rptview(rpt)
```

### See Also

`slreportgen.finder.BlockResult` | `slreportgen.finder.DiagramElementFinder` |  
`slreportgen.finder.DiagramFinder` | `slreportgen.finder.SystemDiagramFinder` |  
`slreportgen.report.Diagram` | `slreportgen.report.Report` |  
`slreportgen.report.SimulinkObjectProperties` |  
`slreportgen.report.SimulinkObjectProperties`

### Topics

“Report Generation for Simulink and Stateflow Elements” on page 1-9  
 “What Is a Reporter?”

**Introduced in R2017b**

## slreportgen.finder.BlockResult class

**Package:** slreportgen.finder

Create block finder result object

### Description

Block search result object for a block in a Simulink diagram.

### Construction

`result = BlockResult(block)` creates a search result object for a block found by a `BlockFinder`. The `result` object contains the Simulink block.

---

**Note** The `simulink.finder.BlockFinder` `find` method creates objects of this type for each block that it finds. You do not need to create this object yourself.

---

`finder = BlockResult(Name, Value)` sets properties using name-value pairs. You can specify multiple name-value pair arguments in any order. Enclose each property name in single quotes.

### Input Arguments

#### **block** — Simulink or Stateflow block

block path | block handle

Simulink block, specified as a path or block handle to that block.

### Properties

#### **Object** — Block handle

handle

This read-only property contains a handle to the block returned in this result.

#### **Name** — Name of block

string

This read-only property specifies the name of the block returned in this result.

#### **Type** — Block type

string

This read-only property specifies the type of the block returned in this result.

Example: "Gain"

#### **DiagramPath** — Path of block

string

This read-only property returns the path of the block returned in this result.

**Tag — Additional information**

string | character vector | object | ...

Additional information to add to this result. You can set it to any type of value.

**Methods**

getDiagramReporter	Returns Diagram reporter for this block result
getReporter	Get block reporter

**Copy Semantics**

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

**See Also**

slreportgen.finder.BlockFinder | slreportgen.report.Report

**Topics**

“Report Generation for Simulink and Stateflow Elements” on page 1-9

“What Is a Reporter?”

**Introduced in R2017b**

## slreportgen.finder.ChartDiagramFinder class

**Package:** slreportgen.finder

Create Stateflow chart finder

### Description

Finds Stateflow charts.

### Construction

`finder = ChartDiagramFinder(container)` creates a finder that finds by default all uncommented Stateflow chart diagrams in the specified `container`. To constrain the search to specific types of diagrams, use the properties of this finder.

---

**Note** This finder can operate in either `find` or `iterator` mode. In `find` mode, use its `find` method to return the results of a search as an array of results. In `iterator` mode, use its `hasNext` and `next` methods to return the results of a search one-by-one. When searching in models that have many model references, use `iterator` mode. `Iterator` mode closes a model after compiling and searching it, whereas `find` mode keeps all the models that it searches open. Having many open models can consume all system memory and slow report generation. `Iterator` mode is slower than `find` mode, so use `find` mode to search models that reference few or no other models.

---

`finder = ChartDiagramFinder(Name, Value)` sets properties using name-value pairs. You can specify multiple name-value pair arguments in any order. Enclose each property name in single quotes.

### Input Arguments

#### Container — Chart container to search

`path` | `handle` | `chart ID` | `chart object`

See `Container` property.

### Properties

#### Container — Chart container to search

`path` | `handle` | `chart ID` | `chart object`

Chart container in which to search, specified as one of these values:

- Handle of a Stateflow chart block
- Path to a Stateflow chart block
- Stateflow chart ID
- Stateflow chart object

#### SearchDepth — Depth of system diagram search

`inf` (default) | positive integer

Depth of system diagram search, specified as `inf` or a positive integer. `SearchDepth` specifies how many levels deep to search a diagram container for diagrams. To search all levels, use `inf`.

### **IncludeMaskedSubsystems — Search masked subsystems**

`true` (default) | `false`

Choice to search masked subsystems, specified as a logical. If this property is `true`, the finder searches masked Subsystem blocks in the diagram container. It searches to the specified `SearchDepth` and includes the diagrams it finds in the search results.

### **IncludeReferencedModels — Search model references**

`true` (default) | `false`

Choice to search referenced models, specified as a logical. If this property is `true`, the finder searches models referenced in the diagram container. It searches to the specified `SearchDepth` and includes the diagrams it finds in the search results.

### **IncludeSimulinkLibraryLinks — Search Simulink library links**

`true` (default) | `false`

Choice to search Simulink library links, specified as a logical. If both this property and `IncludeMaskedSubsystems` are `true`, the finder searches links in the diagram container to both Subsystem and masked Subsystem blocks in Simulink libraries. It searches to the specified `SearchDepth` and includes the diagrams it finds in the search results. If this property is `true`, but `IncludeMaskedSubsystems` is `false`, the finder searches only links to Subsystem blocks in Simulink libraries.

### **IncludeUserLibraryLinks — Search user library links**

`true` (default) | `false`

Choice to search user library links, specified as a logical. If this property is `true` and the `IncludeMaskedSubsystems` property is `true`, the finder searches links in the diagram container to Subsystem and masked Subsystem blocks in user libraries. It searches to the specified `SearchDepth` and includes the diagrams it finds in the search results. If this property is `true`, but the `IncludeMaskedSubsystems` property is `false`, the finder searches only links to Subsystem blocks in user libraries.

### **IncludeCommented — Include commented-out charts**

`false` (default) | `true`

Choice to include commented-out charts in the search results, specified as a logical. If `false`, commented-out charts are excluded from the search results.

### **IncludeVariants — Include diagram variants**

`string` | `character vector`

Variants to search for diagrams, specified as a string or character vector. The default value is `Active`. Valid values are:

- `All` — All variants
- `Active` — Only active variants
- `ActivePlusCode` — All active variants and code variants

**Properties — Properties of objects to find**

cell array

Properties of objects to find, specified as a cell array of name-value pairs. The finder returns only objects that have the specified properties with the specified values.

Example: `finder.Properties = {'Gain','5'}`

**AutoCloseModel — Whether to close models**

true (default) | false

Whether to close models, specified as true or false. If true, the next method of the finder closes the currently open model before moving to the next model to search. Closing models prevents excessive consumption of memory when searching a model that references many models.

---

**Note** The find method of the finder ignores this property and leaves all referenced models open. For this reason, you should not use the find method to search models with many model references.

---

**Methods**

`results = find(finder)` finds chart diagrams in the container specified by the finder. The finder is an `slreportgen.finder.ChartDiagramFinder` object. `results` is an array of `slreportgen.finder.DiagramResult` objects, each of which contains a chart diagram found by this method. Adding this array to a report or reporter adds images of the charts that it contains. The reports to which you can add the results of this method are reports of type `slreportgen.report.Report` or another reporter object, such as an `slreportgen.report.Chapter` reporter.

`tf = hasNext(finder)` determines if the container that the finder searches contains at least one chart. If the container has at least one chart, the `hasNext` method queues that chart as the next chart that the `next` method will return. The `hasNext` method then returns true. Use the `next` method to obtain that chart. On subsequent calls, the `hasNext` method determines if the container has a chart that the `next` has not yet retrieved. It queues the chart for the `next` method to retrieve and returns true. If there are no more charts exist to be retrieved, this method returns false. To search a container progressively for charts, use the `hasNext` method with the `next` method in a while loop.

---

**Note** If the current result is the last result in the search queue for the current chart and the `AutoCloseModel` property is true, this method closes the current chart before it opens the next chart. Although this increases search time, it reduces memory consumption when searching a chart that references many other charts. If your chart does not reference many other charts, to speed up the search, set the `AutoCloseModel` property to false or use the `find` method.

---

`result = next(result)` returns the next search result in the result queue that the `hasNext` method created. The search result contains the resulting chart. Adding this `result` object to a report or reporter adds a Diagram reporter for the chart.

**Copy Semantics**

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



## Examples

### Find Stateflow Charts

Create a report that includes images of all Stateflow charts in the `sf_car` model. Use a separate chapter for each chart.

```
import mlreportgen.report.*
import slreportgen.report.*
import slreportgen.finder.*

model_name = 'sldemo_fuelsys';
load_system(model_name);
rpt = slreportgen.report.Report('output','pdf');

add(rpt, TitlePage('Title', sprintf('%s Charts',...
    model_name)));
add(rpt, TableOfContents);
chapter = Chapter('Root System');
add(chapter, Diagram(model_name));
add(rpt,chapter);

chapter = Chapter('Charts');
finder = ChartDiagramFinder(model_name);
results = find(finder);
for result = results
    section = Section('Title',result.Name);
    add(section,result);
    add(chapter,section);
end
add(rpt, chapter);

close(rpt);
close_system(model_name);
rptview(rpt);
```

### See Also

[slreportgen.finder.DiagramElementFinder](#) | [slreportgen.finder.DiagramFinder](#) | [slreportgen.finder.DiagramResult](#) | [slreportgen.finder.StateFinder](#) | [slreportgen.finder.StateflowDiagramElementFinder](#) | [slreportgen.report.Report](#) | [slreportgen.report.StateflowObjectProperties](#)

### Topics

“Report Generation for Simulink and Stateflow Elements” on page 1-9

“What Is a Reporter?”

### Introduced in R2017b

## slreportgen.finder.DataDictionaryFinder class

**Package:** slreportgen.finder slreportgen.finder slreportgen.finder  
slreportgen.finder

**Superclasses:** mlreportgen.finder.Finder

Find data dictionaries

### Description

Use an object of the `slreportgen.finder.DataDictionaryFinder` class to find Simulink data dictionaries.

The `slreportgen.finder.DataDictionaryFinder` class is a `handle` class.

### Class Attributes

`HandleCompatible` `true`

For information on class attributes, see “Class Attributes”.

## Creation

### Description

`finder = slreportgen.finder.DataDictionaryFinder()` creates a data dictionary finder and sets the `Container` property to `'MATLABPath'`.

You can constrain the search by setting the properties of the finder. Use the methods of the finder to perform the search.

---

**Note** This finder provides two ways to get search results:

- 1 To return the search results as an array, use the `find` method. Add the results directly to a report or process the results in a `for` loop.
- 2 To iterate through the results one at a time, use the `hasNext` and `next` methods in a `while` loop.

Neither option has a performance advantage.

---

`finder = slreportgen.finder.DataDictionaryFinder(searchFolder)` creates a data dictionary finder and sets the `Container` property to the folder or folders specified by `searchFolder`.

`finder = slreportgen.finder.DataDictionaryFinder(Name, Value)` sets properties using name-value pairs. You can specify multiple name-value pair arguments in any order. Enclose each property name in single or double quotes.

## Properties

### Container — Folders to search for data dictionaries

string array | character vector | cell array of character vectors

Folders to search for data dictionaries, specified as a string array, character vector, or cell array of character vectors. Strings and character vectors can include the \* and the \*\* wildcards. Characters next to the \*\* wildcard must be file separators. For example, to find all data dictionaries in the exampleFolder folder and its subfolders, set Container to "exampleFolder\\*\*". If Container is set to 'MATLABPath', the finder searches for data dictionaries in the current folder and all folders on the MATLAB path.

### Name — Data dictionary to find

string scalar | character vector

Data dictionary to find, specified as a string scalar or character vector. The Name property value can include the \* wildcard. For example, to find all data dictionaries that begin with sldemo\_fuelsys\_dd, set the Name property to "sldemo\_fuelsys\_dd\*". The name or expression specified in this property must have no file name extension or the extension .sldd.

### Properties — Properties of data dictionaries to find

{ } (default) | cell array

Properties of the data dictionaries to find, specified as a cell array of name-value pairs. Use the Properties property to filter the finder results by the data dictionary properties. The finder searches the folders specified by the Container property for data dictionaries with names that match the Name property and that have the specified properties values. For data dictionary properties, see Simulink.data.Dictionary. For example, to return only the data dictionaries that have access to the base workspace, set Properties to {'HasAccessToBaseWorkspace', true}.

## Methods

### Public Methods

find	<p><code>results = find(finder)</code> finds data dictionaries in a model or subsystem according to the constraints specified by the finder. The variables are returned as an array of <code>slreportgen.finder.DataDictionaryResult</code> objects.</p> <p>Append all of the results directly to a report or process the results in a <code>for</code> loop. In the loop, you can customize the content and formatting for a data dictionary by setting the properties of the reporter for the data dictionary. Get the reporter by using the <code>getReporter</code> method of the <code>slreportgen.finder.DataDictionaryResult</code> object that contains the data dictionary.</p>
------	--

hasNext	<code>tf = hasNext(finder)</code> returns <code>true</code> if a data dictionary returned by the finder is available for the <code>next</code> method to retrieve. If the finder does not return any data dictionaries, or if all dictionaries have been retrieved, <code>hasNext</code> returns <code>false</code> . Use <code>hasNext</code> and <code>next</code> to iterate through the data dictionaries in a <code>while</code> loop. In the loop, you can customize the content and formatting for a data dictionary by setting properties of the reporter for the data dictionary. Get the reporter by using the <code>getReporter</code> method of the <code>slreportgen.finder.DataDictionaryResult</code> object.
next	<code>result = next(finder)</code> returns the next data dictionary as an <code>slreportgen.finder.DataDictionaryResult</code> object. Use <code>hasNext</code> and <code>next</code> to iterate through found data dictionaries in a <code>while</code> loop. In the loop, you can customize the content and formatting for a data dictionary by setting the properties of the reporter for the data dictionary. Get the reporter by using the <code>getReporter</code> method of the <code>slreportgen.finder.DataDictionaryResult</code> object.

## Examples

### Find and Report on Data Dictionary

To report on data dictionaries, create an `slreportgen.DataDictionaryFinder` object. Use the object properties to constrain the search and the methods to get the results.

Import the MATLAB Report and Simulink Report API packages so that you do not have to use long, fully qualified class names.

```
import mlreportgen.report.*
import slreportgen.finder.*
import slreportgen.report.*
```

Create a Simulink report and append a table of contents to the report.

```
rpt = slreportgen.report.Report("MyReport", "html-file");
append(rpt, TableOfContents);
```

Create a Simulink data dictionary finder to search the entire MATLAB path.

```
f = DataDictionaryFinder();
```

Constrain the finder to find only data dictionaries that have names that begin with `sldemo_fuelsys_dd`.

```
f.Name = "sldemo_fuelsys_dd*";
```

Create a chapter for the data dictionaries.

```
ch = Chapter("Data Dictionaries");
```

For each found dictionary, create a section and append it to the chapter.

```
while hasNext(f)
    result = next(f);
    s = Section(result.Name);
    append(s,result);
    append(ch,s);
end
```

Append the chapter to the report. Close and view the report.

```
append(rpt,ch);
close(rpt);
rptview(rpt);
```

## See Also

[slreportgen.finder.DataDictionaryResult](#) | [slreportgen.report.DataDictionary](#)

## Topics

“What Is a Data Dictionary?”

“Report Generation for Simulink and Stateflow Elements” on page 1-9

## Introduced in R2020b

## slreportgen.finder.DataDictionaryResult class

**Package:** slreportgen.finder slreportgen.finder

Data dictionary search result object

### Description

An object of the `slreportgen.finder.DataDictionaryResult` class represents a result of a search for data dictionaries. You can append a `DataDictionaryResult` object directly to a report. Alternatively, you can use the `getReporter` method to access the `slreportgen.report.DataDictionary` reporter for the result and then customize the reporter and append it to the report.

The `slreportgen.finder.DataDictionaryResult` class is a `handle` class.

### Class Attributes

`HandleCompatible` `true`

For information on class attributes, see “Class Attributes”.

### Creation

You do not create an `slreportgen.finder.DataDictionaryResult` object explicitly. The `slreportgen.finder.DataDictionaryFinder` `find` and `next` methods create an `slreportgen.finder.DataDictionaryResult` object for each data dictionary that is found.

### Properties

#### Object — Full path of data dictionary

`string scalar`

Full path of the data dictionary represented by this result, specified as a string scalar. This property is read-only.

#### Name — File name of data dictionary

`string scalar`

File name of data dictionary represented by this result, specified as a string scalar. This property is read-only.

#### Tag — Additional information

`string` | `character vector` | `number` | ...

Additional information to save with this result. You can set this property to any type of value.

## Methods

### Public Methods

getReporter	reporter = getReporter(dictionaryResult) returns the slreportgen.report.DataDictionary reporter object for the specified data dictionary result. Use the reporter to customize the information reported and the formatting of the information.
-------------	--

## Examples

### Customize Reporter for Data Dictionary Result

Use the getReporter method of an slreportgen.finder.DataDictionaryResult object to access the data dictionary reporter for the result. Then, customize the reporter by setting its properties.

Import the MATLAB and Simulink Report API packages so that you do not have to use long, fully qualified class names.

```
import mlreportgen.report.*
import slreportgen.report.*
```

Create a Simulink report and add a table of contents.

```
rpt = slreportgen.report.Report("MyReport", "html-file");
append(rpt, TableOfContents);
```

Create a dictionary finder that searches the entire MATLAB path. Find only data dictionaries with names that begin with sldemo\_fuelysy\_dd.

```
f = slreportgen.finder.DataDictionaryFinder();
f.Name = "sldemo_fuelysy_dd*";
```

Create a chapter with a section for each data dictionary result. For each result, customize the reporter to use a list for referenced dictionaries.

```
ch = Chapter("Data Dictionaries");
while hasNext(f)
    result = next(f);
    s = Section(result.Name);
    rptr = getReporter(result);
    rptr.ReferencedDictionaryPolicy = "List";
    append(s, rptr);
    append(ch, s);
end
```

Add the chapter to the report. Close and view the report.

```
append(rpt, ch);
close(rpt);
rptview(rpt);
```

The report has a section for each of the found dictionaries, `sldemo_fuelsys_dd` and `sldemo_fuelsys_dd_controller`. The `sldemo_fuelsys_dd` dictionary references two dictionaries, which are reported as a list. There is no Design Data Summary section for the `sldemo_fuelsys_dd` dictionary because all of its entries are owned by the referenced dictionaries.



## Chapter 1. Data Dictionaries

### 1.1. `sldemo_fuelsys_dd.sldd`

#### Referenced Dictionaries

- [sldemo\\_fuelsys\\_dd\\_controller.sldd](#)
- [sldemo\\_fuelsys\\_dd\\_plant.sldd](#)

### 1.2. `sldemo_fuelsys_dd_controller.sldd`

#### Design Data Summary

Table 1.1. `sldemo_fuelsys_dd_controller.sldd` Design Data

Name	Value	Class	LastModified	LastModifiedBy	Status	DataSource
<a href="#">RampRateKiX</a>	See Details	Simulink.Parameter	2013-10-22 08:05	The MathWorks, Inc.	Unchanged	sldemo_fuelsys_dd_controller.sldd
<a href="#">RampRateKiY</a>	See Details	Simulink.Parameter	2013-10-22 08:05	The MathWorks, Inc.	Unchanged	sldemo_fuelsys_dd_controller.sldd
<a href="#">hys</a>	25	Simulink.Parameter	2013-10-22 08:05	The MathWorks, Inc.	Unchanged	sldemo_fuelsys_dd_controller.sldd
<a href="#">zero_thresh</a>	250	Simulink.Parameter	2013-10-22 08:05	The MathWorks, Inc.	Unchanged	sldemo_fuelsys_dd_controller.sldd
<a href="#">st_range</a>	0.0001	double	2013-10-22 08:06	The MathWorks, Inc.	Unchanged	sldemo_fuelsys_dd_controller.sldd
<a href="#">PumpCon</a>	See Details	Simulink.Parameter	2013-10-22 08:33	The MathWorks, Inc.	Unchanged	sldemo_fuelsys_dd_controller.sldd
<a href="#">RampRateKiZ</a>	See Details	Simulink.Parameter	2013-10-22 08:33	The MathWorks, Inc.	Unchanged	sldemo_fuelsys_dd_controller.sldd
<a href="#">sensors</a>	See Details	Simulink.Signal	2013-10-22 08:34	The MathWorks, Inc.	Unchanged	sldemo_fuelsys_dd_controller.sldd
<a href="#">fuel_mode</a>	See Details	Simulink.Signal	2013-10-22 08:34	The MathWorks, Inc.	Unchanged	sldemo_fuelsys_dd_controller.sldd
<a href="#">PressVect</a>	See Details	Simulink.Parameter	2015-12-16 23:09	The MathWorks, Inc.	Unchanged	sldemo_fuelsys_dd_controller.sldd
<a href="#">SpeedVect</a>	See Details	Simulink.Parameter	2015-12-16 23:10	The MathWorks, Inc.	Unchanged	sldemo_fuelsys_dd_controller.sldd
<a href="#">ThrotVect</a>	See Details	Simulink.Parameter	2015-12-16 23:10	The MathWorks, Inc.	Unchanged	sldemo_fuelsys_dd_controller.sldd

## See Also

`slreportgen.finder.DataDictionaryFinder` | `slreportgen.report.DataDictionary`

## Topics

“What Is a Data Dictionary?”

“Report Generation for Simulink and Stateflow Elements” on page 1-9

## Introduced in R2020b



# slreportgen.finder.DiagramElementFinder class

**Package:** slreportgen.finder

Create diagram element finder object

## Description

Finds elements in a Simulink block or Stateflow chart diagram.

## Construction

`finder = DiagramElementFinder(diagram)` creates a finder that finds elements of a Simulink block or Stateflow chart diagram. By default this finder finds blocks, annotations, lines, states, and other elements in the diagram. Use the properties of the finder to constrain the search to specific types of elements.

---

**Note** This finder provides two ways to get search results:

- 1 To return the search results as an array, use the `find` method. Add the results directly to a report or process the results in a `for` loop.
- 2 To iterate through the results one at a time, use the `hasNext` and `next` methods in a `while` loop.

Neither option has a performance advantage.

---

`finder = DiagramElementFinder(Name, Value)` sets properties using name-value pairs. You can specify multiple name-value pair arguments in any order. Enclose each property name in single quotes.

## Input Arguments

### **diagram** — Diagram to search

path | handle | chart ID | chart object

See Container property.

## Properties

### **Container** — Diagram to search

path | handle | chart ID | chart object

Diagram in which to search, specified as one of these values:

- Handle to a Simulink block
- Path to a Simulink block
- Handle to a Stateflow chart block
- Path to a Stateflow chart block

- Stateflow chart ID
- Stateflow chart object

### Types — Types of diagram elements to find

string | character array | array of strings | cell array of character arrays

Types of Simulink or Stateflow diagram elements to find, specified as a string, character array, array of strings, or a cell array of character arrays. If the type is an array, it specifies a set of element types. The default is `All` or `all`, which finds all elements in all diagrams. Use one of these values to constrain the search to specific diagram element types. You can use either the fully qualified name or the short name.

Fully Qualified Name	Short Name
All	all
Simulink.Annotation	annotation
Simulink.Block	block
Simulink.Segment	line
Simulink.Port	port
Stateflow.Annotation	sf_annotation
Stateflow.Box	box
Stateflow.EMFunction	emfunction
Stateflow.Function	function
Stateflow.Junction	junction
Stateflow.SLFunction	slfunction
Stateflow.State	state
Stateflow.Transition	transition
Stateflow.TruthTable	truthtable

### IncludeCommented — Include commented-out diagram elements

false (default) | true

Choice to include commented-out diagram elements in the search results, specified as a logical. If false, commented-out elements are excluded from the search results.

### IncludeVariants — Include diagram variants

string | character vector

Variants to search for diagrams, specified as a string or character vector. The default value is `Active`. Valid values are:

- `All` — All variants
- `Active` — Only active variants
- `ActivePlusCode` — All active variants and code variants

### Properties — Properties of objects to find

cell array

Properties of objects to find, specified as a cell array of name-value pairs. The finder returns only objects that have the specified properties with the specified values.

Example: `finder.Properties = {'Gain','5'}`

## Methods

`results = find(finder)` finds diagram elements in the diagram specified by the finder. This method returns the diagram elements it finds wrapped in result objects of type `slreportgen.finder.DiagramElementResult`. To add tables of the diagram element properties, add the results objects directly to the report or add them to a reporter that you then add to a report. The reports to which you can add the results of this method must be reports of type `slreportgen.report.Report`.

`tf = hasNext(finder)` determines if the diagram that the finder searches contains at least one element. If the diagram has at least one element, the `hasNext` method queues that element as the next element that the `next` method will return. The `hasNext` method then returns `true`. Use the `next` method to obtain that element. On subsequent calls, the `hasNext` method determines if the diagram has an element that the `next` method has not yet retrieved. It queues the element for the `next` method to retrieve and returns `true`. If there are no more elements exist to be retrieved, this method returns `false`. To search a diagram progressively for elements, use the `hasNext` method with the `next` method in a `while` loop.

`result = next(finder)` returns the next search result in the result queue that the `hasNext` method created. This method returns the diagram element that it finds wrapped in a result object of type `slreportgen.finder.DiagramElementResult`. To add tables of the diagram element properties, add the results objects directly to the report or add them to a reporter that you then add to a report. The reports to which you can add the results of this method must be of type `slreportgen.report.Report`.

## Copy Semantics

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

## Examples

### Find Block, Annotation, and Line Elements

Find the block, annotation, and line diagram elements to a search depth of 1 in the `f14` model.

```
import mlreportgen.report.*
import slreportgen.report.*
import slreportgen.finder.*

model_name = 'f14';
load_system(model_name);

rpt = slreportgen.report.Report('output','pdf');
add(rpt, TitlePage("Title",sprintf('%s Model',...
    model_name)));
add(rpt, TableOfContents);

diagFinder = SystemDiagramFinder("Container", ...
    model_name,"SearchDepth",1);
```

```
while hasNext(diagFinder)
    system = next(diagFinder);
    chapter = Chapter("Title",system.Name);
    add(chapter,system);
    sect = Section("Title","Diagram Elements");
    elemFinder = DiagramElementFinder("Container", ...
        system.Object, "Types",...
        ["block" "annotation" "line"]);
    elems = find(elemFinder);
    for elem = elems
        add(sect, elem);
    end
    add(chapter, sect);
    add(rpt, chapter);
end

close(rpt);
close_system(model_name);
rptview(rpt);
```

## See Also

slreportgen.finder.AnnotationFinder | slreportgen.finder.BlockFinder |  
slreportgen.finder.ChartDiagramFinder | slreportgen.finder.DiagramElementResult  
| slreportgen.finder.DiagramFinder | slreportgen.finder.StateFinder |  
slreportgen.finder.StateflowDiagramElementFinder |  
slreportgen.finder.SystemDiagramFinder | slreportgen.report.Diagram |  
slreportgen.report.Report

## Topics

“Report Generation for Simulink and Stateflow Elements” on page 1-9  
“What Is a Reporter?”

## Introduced in R2017b

# slreportgen.finder.DiagramFinder class

**Package:** slreportgen.finder

Create finder for diagrams

## Description

Finds Simulink diagrams and Stateflow charts.

## Construction

`finder = DiagramFinder(container)` creates a finder that finds by default all uncommented Simulink block diagrams and Stateflow chart diagrams in the specified `container`. To constrain the search to include specific types of diagrams, use the properties of this finder.

---

**Note** This finder can operate in either `find` or `iterator` mode. In `find` mode, use its `find` method to return the results of a search as an array of results. In `iterator` mode, use its `hasNext` and `next` methods to return the results of a search one-by-one. When searching in models that have many model references, use `iterator` mode. `Iterator` mode closes a model after compiling and searching it, whereas `find` mode keeps all the models that it searches open. Having many open models can consume all system memory and slow report generation. `Iterator` mode is slower than `find` mode, so use `find` mode to search models that reference few or no other models.

---

`finder = DiagramFinder(Name, Value)` sets properties using name-value pairs. You can specify multiple name-value pair arguments in any order. Enclose each property name in single quotes.

## Input Arguments

### Container — Model container to search

path | handle | chart ID | chart object

See Container property.

## Properties

### Container — Model container to search

path | handle | chart ID | chart object

Model container in which to search, specified as one of these values:

- Handle of a Simulink model, subsystem, or model block
- Path to a Simulink model, subsystem, or model block
- Handle of a Stateflow chart block
- Path to a Stateflow chart block
- Stateflow chart ID
- Stateflow chart object

**SearchDepth — Depth of system diagram search**`inf (default) | positive integer`

Depth of system diagram search, specified as `inf` or a positive integer. `SearchDepth` specifies how many levels deep to search a diagram container for diagrams. To search all levels, use `inf`.

**IncludeMaskedSubsystems — Search masked subsystems**`true (default) | false`

Choice to search masked subsystems, specified as a logical. If this property is `true`, the finder searches masked Subsystem blocks in the diagram container. It searches to the specified `SearchDepth` and includes the diagrams it finds in the search results.

**IncludeReferencedModels — Search model references**`true (default) | false`

Choice to search referenced models, specified as a logical. If this property is `true`, the finder searches models referenced in the diagram container. It searches to the specified `SearchDepth` and includes the diagrams it finds in the search results.

**IncludeSimulinkLibraryLinks — Search Simulink library links**`true (default) | false`

Choice to search Simulink library links, specified as a logical. If both this property and `IncludeMaskedSubsystems` are `true`, the finder searches links in the diagram container to both Subsystem and masked Subsystem blocks in Simulink libraries. It searches to the specified `SearchDepth` and includes the diagrams it finds in the search results. If this property is `true`, but `IncludeMaskedSubsystems` is `false`, the finder searches only links to Subsystem blocks in Simulink libraries.

**IncludeUserLibraryLinks — Search user library links**`true (default) | false`

Choice to search user library links, specified as a logical. If this property is `true` and the `IncludeMaskedSubsystems` property is `true`, the finder searches links in the diagram container to Subsystem and masked Subsystem blocks in user libraries. It searches to the specified `SearchDepth` and includes the diagrams it finds in the search results. If this property is `true`, but the `IncludeMaskedSubsystems` property is `false`, the finder searches only links to Subsystem blocks in user libraries.

**IncludeCommented — Include commented-out diagrams**`false (default) | true`

Whether to include commented-out diagrams in the search results, specified as a logical. If `false`, commented-out diagrams are excluded from the search results.

**IncludeVariants — Include diagram variants**`string | character vector`

Variants to search for diagrams, specified as a string or character vector. The default value is `Active`. Valid values are:

- `All` — All variants
- `Active` — Only active variants

- `ActivePlusCode` — All active variants and code variants

### Properties — Properties of objects to find

cell array

Properties of objects to find, specified as a cell array of name-value pairs. The finder returns only objects that have the specified properties with the specified values.

Example: `finder.Properties = {'Gain','5'}`

### AutoCloseModel — Whether to close models

`true` (default) | `false`

Whether to close models, specified as `true` or `false`. If `true`, the `next` method of the finder closes the currently open model before moving to the next model to search. Closing models prevents excessive consumption of memory when searching a model that references many models.

---

**Note** The `find` method of the finder ignores this property and leaves all referenced models open. For this reason, you should not use the `find` method to search models with many model references.

---

## Methods

`results = find(finder)` finds diagrams in the specified container. The `finder` is an `slreportgen.finder.DiagramFinder` object. `results` is an array of `slreportgen.finder.DiagramResult` objects, each of which contains a diagram found by this method. Adding this array to a report or a reporter adds images of all the diagrams that it contains. The reports to which you can add the `results` of this method are reports of type `slreportgen.report.Report` or another reporter object, such as an `slreportgen.report.Chapter` reporter.

---

**Note** The `find` method opens and compiles a top-level model and all models it references. This method leaves all the models open at the conclusion of a search, which can slow reporting on models that contain many model references. To avoid this slowdown, use the `hasNext` and `next` methods to search such a model.

---

`tf = hasNext(finder)` determines if the container that the finder searches contains at least one diagram. If the container has at least one diagram, the `hasNext` method queues that diagram as the next diagram that the `next` method will return. The `hasNext` method then returns `true`. Use the `next` method to obtain that diagram. On subsequent calls, the `hasNext` method determines if the container has a diagram that the `next` has not yet retrieved. It queues the diagram for the `next` method to retrieve and returns `true`. If there are no more diagrams to be retrieved, this method returns `false`. To search a container progressively for diagrams, use the `hasNext` method with the `next` method in a `while` loop.

---

**Note** If the current result is the last result in the search queue for the current model and the `AutoCloseModel` property is `true`, this method closes the current model before it opens the next model. Although this increases search time, it reduces memory consumption when searching a top model that references many other models. If your model does not reference many other models, to speed up the search, set the `AutoCloseModel` property to `false` or use the `find` method.

---

`result = next(finder)` returns the next search result in the result queue that the `hasNext` method created. The search result contains the resulting diagram. Adding this `result` object to a report or reporter adds a Diagram reporter for the diagram.

## Copy Semantics

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

## Examples

### Find All Block Diagrams and Stateflow Charts

Create a report that includes an image of all diagrams in the `sf_car` model. Use a separate chapter for each diagram. Although the model used in this example does not contain model references, the example uses iterator mode to illustrate its syntax.

```
import mlreportgen.report.*
import slreportgen.report.*
import slreportgen.finder.*

model_name = 'sf_car';
load_system(model_name);
rpt = slreportgen.report.Report('output','pdf');
add(rpt, TitlePage('Title', sprintf('%s Systems',...
    model_name)));

finder = DiagramFinder(model_name);
while hasNext(finder)
    add(rpt,next(finder));
end

close(rpt);
close_system(model_name);
rptview(rpt);
```

### Find All Diagrams in a Subsystem

Open the `sf_car` model and find all the diagrams in its `Engine` subsystem. Use either the path to the subsystem or its handle. You can then include the results in your report.

```
sf_car

% Use path
enginePath = "sf_car/Engine";
finder = slreportgen.finder.DiagramFinder(enginePath);
results = find(finder);

% or use handle
engineHandle = get_param("sf_car/Engine","Handle");
finder = slreportgen.finder.DiagramFinder(engineHandle);
results_enginehandle = find(finder);
```



## Find Diagram Elements with Specific Property Value

To find elements with specific property values, use an object of the `slreportgen.finder.DiagramElementFinder` class. Open the `f14` model and find all Gain blocks with a value of `Zw`.

```
model = 'f14';  
load_system(model);  
finder = slreportgen.finder.DiagramElementFinder(model)  
finder.Properties = {'Gain','Zw'};  
results = find(finder);
```

## See Also

`slreportgen.finder.ChartDiagramFinder` | `slreportgen.finder.DiagramElementFinder`  
| `slreportgen.finder.DiagramResult` |  
`slreportgen.finder.StateflowDiagramElementFinder` |  
`slreportgen.finder.SystemDiagramFinder` | `slreportgen.report.Diagram` |  
`slreportgen.report.Report`

## Topics

“Report Generation for Simulink and Stateflow Elements” on page 1-9  
“What Is a Reporter?”

## Introduced in R2017b

## slreportgen.finder.DiagramElementResult class

**Package:** slreportgen.finder

Create diagram element finder result object

### Description

Diagram element search result object for an element in a Simulink or Stateflow diagram.

### Construction

`result = DiagramElementResult(elem)` creates a search result object for a diagram element. The `result` object contains the specified Simulink or Stateflow diagram element.

---

**Note** The find methods of diagram element finders create and return instances of this `slreportgen.finder.DiagramElementResult` object. You do not need to create instances yourself.

---

`finder = DiagramElementResult(Name, Value)` sets properties using name-value pairs. You can specify multiple name-value pair arguments in any order. Enclose each property name in single quotes.

### Input Arguments

**elem — Simulink or Stateflow diagram element**

diagram element path | diagram element handle

Simulink or Stateflow diagram element, specified as a path or handle to the element.

### Properties

**Object — Diagram element handle**

handle

This read-only property contains a handle to the diagram element returned in this result.

**Name — Name of diagram element**

string

This read-only property specifies the name of the diagram element returned in this result.

**Type — Diagram element type**

string

This read-only property specifies the type of the diagram element returned in this result.

Example: "Simulink.BlockDiagram"

**DiagramPath — Path of diagram that contains element**

string

This read-only property returns the path of the diagram that contains the element returned in this result.

### Tag — Additional information

string | character vector | object | ...

Additional information to add to this result. You can set it to any type of value.

## Methods

getDiagramReporter	Returns Diagram reporter for diagram element result
getReporter	Get diagram element reporter

## Copy Semantics

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

## Examples

### Create Diagram Element Result Object for Subsystem

Load the `sf_car` model and save the result of searching for diagram elements in its Engine subsystem. Both code examples produce the same result object. Use the second approach if you want to customize how the results display in your report.

```
import slreportgen.finder.*
load_system('sf_car')
result = DiagramElementResult('sf_car/Engine')
```

% or

```
load_system('sf_car')
finder = DiagramElementFinder();
result = find(finder,'sf_car/Engine')
```

Include the result directly in an `slreportgen.report.Report` report or, add it to a reporter that you then add to the report.

## See Also

`slreportgen.finder.DiagramElementFinder` | `slreportgen.report.Report`

### Topics

“Report Generation for Simulink and Stateflow Elements” on page 1-9  
 “What Is a Reporter?”

### Introduced in R2017b

## slreportgen.finder.DiagramResult class

**Package:** slreportgen.finder

Create diagram result finder object

### Description

Diagram search result object for a Simulink or Stateflow diagram.

`finder = DiagramResult(Name, Value)` sets properties using name-value pairs. You can specify multiple name-value pair arguments in any order. Enclose each property name in single quotes.

### Construction

`result = DiagramResult(diagram)` creates a search result object for a Simulink or Stateflow diagram. The `result` object contains the specified Simulink or Stateflow diagram. Diagram finder find methods create and return instances of this object for each diagram that they find. You do not need to create this object yourself.

### Input Arguments

#### **diagram — Simulink or Stateflow diagram**

diagram path | diagram handle

Simulink or Stateflow diagram, specified by its path or its handle.

### Properties

#### **Object — Diagram handle**

handle

This read-only property contains a handle to the diagram returned by this result.

#### **Name — Name of model or block**

string

This read-only property specifies the name of the model or block that contains the diagram returned by this result.

#### **Type — Diagram type**

string

This read-only property specifies the type of the diagram returned by this result.

Example: "Simulink.BlockDiagram"

#### **Path — Path to diagram**

string

This read-only property returns the path of the container of the diagram returned by this result. An example of a container of a diagram is a subsystem block.



## slreportgen.finder.ModelVariableFinder class

**Package:** slreportgen.finder slreportgen.finder slreportgen.finder  
slreportgen.finder

**Superclasses:** mlreportgen.finder.Finder

Finds variables used by a Simulink model

### Description

Find variables used by a Simulink model.

The `slreportgen.finder.ModelVariableFinder` class is a `handle` class.

### Class Attributes

`HandleCompatible` `true`

For information on class attributes, see “Class Attributes”.

## Creation

### Description

`finder = slreportgen.finder.ModelVariableFinder(container)` creates a finder that finds variables used in the specified `container`, which can be a Simulink model or subsystem. See the `Container` property. You can constrain the search by setting the properties of the finder. Use the methods of the finder to perform the search.

---

**Note** This finder provides two ways to get search results:

- 1 To return the search results as an array, use the `find` method. Add the results directly to a report or process the results in a `for` loop.
- 2 To iterate through the results one at a time, use the `hasNext` and `next` methods in a `while` loop.

Neither option has a performance advantage.

---

`finder = slreportgen.finder.ModelVariableFinder(Name, Value)` sets properties using name-value pairs. You can specify multiple name-value pair arguments in any order. Enclose each property name in single or double quotes.

## Properties

### Container — Model or subsystem to search

`string scalar` | `character vector` | `handle`

Model or subsystem to search, specified as a string scalar or character vector that contains the path to the model or subsystem, or as a handle to the model or subsystem.

**Regexp — Regular expression matching**`"off" (default) | "on"`

Regular expression matching, specified as "off" or "on". If `Regexp` is "off", regular expression matching is not enabled. If `Regexp` is "on", regular expression matching is enabled for the values of the `Name`, `SourceType`, and `Users` properties. For example, this code finds variables that start with `vehicle`.

```
finder = slreportgen.finder.ModelVariableFinder('sf_car');
finder.Regexp = "on";
finder.Name = "^vehicle";
```

See “Regular Expressions”.

**SearchMethod — Compile status**`"compiled" (default) | "cached"`

Compile status, specified as one of the values in the table.

Value	Description
"compiled"	Get up-to-date results by compiling models before the search. (default)
"cached"	Get results more quickly by using data cached during the previous compilation.

**SearchReferencedModels — Whether to search referenced models**`"on" (default) | "off"`

Whether to search for variables in referenced models, specified as one of the values in the table.

Value	Description
"on"	Search for variables in referenced models. (default)
"off"	Do not search for variables in referenced models.

**Name — Name of variable to search for**`[] (default) | character vector | string scalar`

Name of variable to search for, specified as a character vector or string scalar. If the `Regexp` property is set to "on", the value of `Name` can be a regular expression. If the `Name` property is empty, the finder does not search based on the variable name.

Example: "vehicledata"

Example: "^vehicle"

**SourceType — Source of variable definitions**`[] (default) | "base workspace" | "model workspace" | "mask workspace" | "data dictionary"`

Source of variable definitions, specified as one of these values:

If you set `SourceType`, the finder returns variables only from the specified source. If the `Regex` property is set to "on", the value of `SourceType` can be a regular expression. If the `SourceType` property is empty, the finder does not filter the search results by the source.

Example: `finder.SourceType = "model workspace"` returns all variables defined in the model workspace.

Example: `finder.SourceType = "(base|mask) workspace"` returns all variables defined in the base workspace or the mask workspace if the `Regex` property is set to "On".

Example: `finder.SourceType = "\\w* workspace"` returns all variables defined in the base, mask, or model workspace if the `Regex` property is set to "On".

### Users — Names of blocks to search for variables

[ ] (default) | character vector | string scalar | array of character vectors | string array

Names of blocks to search for variables. Specify one block as a character vector or string scalar. Specify multiple blocks as an array of character vectors or a string array. The finder returns variables used by one or more of the specified blocks. If you do not set the `Users` property, the finder searches the entire model or subsystem. If the `Regex` property is set to `true`, you can set the `Users` property to a regular expression.

For example, to find all variables in `MyModel` that are used by either the `Gain1` block or the `Gain2` block, you can specify both blocks in the `Users` property.

```
myFinder.Users = ["myModel/Gain1", "myModel/Gain2"];
```

Alternatively, you can use a regular expression that matches both block names.

```
myFinder.Regexp = "on";
myFinder.Users = "Gain(1|2)";
```

### LookUnderMasks — Whether to search masked subsystems

"all" (default) | "none"

Whether to search for variables in masked subsystems, specified as one of the values in the table.

Value	Description
"all"	Search for variables in masked subsystems. (default)
"none"	Do not search for variables in masked subsystems.

### FollowLibraryLinks — Whether to follow library links

"on" (default) | "off"

Whether to follow library links when searching for variables, specified as one of the values in the table.

Value	Description
"on"	Follow links into library blocks. Library links are treated as subsystems. (default)
"off"	Do not follow links into library blocks. Library links are treated as blocks.



**IncludeInactiveVariants – Whether to include variables of inactive variant systems**

"off" (default) | "on"

Whether to include variables of inactive variant systems, specified as one of the values in the table.

Value	Description
"off"	Do not include variables used by inactive variant systems. (default)
"on"	Include variables used by inactive variant systems. Variables in inactive variants are only found if the <b>Variant activation time</b> configuration parameter of the containing Variant Subsystem or Variant Model block is set to <code>code compile</code> or <code>update diagram analyze all choices</code> . To include variables in Model blocks that are inactive systems, the <code>SearchReferencedModels</code> property of this finder must also be set to "on".

**Properties – Properties of Simulink.VariableUsage objects to find**

{ } (default) | cell array

Properties of `Simulink.VariableUsage` objects to find, specified as a cell array of name-value pairs. The finder returns only variables whose associated `Simulink.VariableUsage` object has the specified property values.

Example: `finder.Properties = {'SourceType', 'base workspace'}`

**Methods****Public Methods**

<code>find</code>	<p><code>results = find(finder)</code> finds variables in a model or subsystem according to the constraints specified by the finder. The variables are returned as an array of <code>slreportgen.finder.ModelVariableResult</code> objects.</p> <p>Add all of the results directly to a report or process the results in a <code>for</code> loop. In the loop, you can customize the content and formatting for a variable by setting properties of the reporter for the variable. Get the reporter by using the <code>getReporter</code> method of the <code>slreportgen.finder.ModelVariableResult</code> object that contains the variable.</p>
-------------------	--

hasNext	<p><code>tf = hasNext(finder)</code> returns <code>true</code> if a variable returned by the finder is available for the <code>next</code> method to retrieve. If the finder does not return any variables, or if all variables have been retrieved, <code>hasNext</code> returns <code>false</code>. Use <code>hasNext</code> and <code>next</code> to iterate through the variables in a <code>while</code> loop. In the loop, you can customize the content and formatting for a variable by setting properties of the reporter for the variable. Get the reporter by using the <code>getReporter</code> method of the <code>slreportgen.finder.ModelVariableResult</code> object.</p>
next	<p><code>result = next(finder)</code> returns the next variable as an <code>slreportgen.finder.ModelVariableResult</code> object. Use <code>hasNext</code> and <code>next</code> to iterate through found variables in a <code>while</code>-loop. In the loop, you can customize the content and formatting for a variable by setting properties of the reporter for the variable. Get the reporter by using the <code>getReporter</code> method of the <code>slreportgen.finder.ModelVariableResult</code> object.</p>

## Examples

### Add Model Variables to a Report

Find the variables in a model and add the results directly to a report. Specify that the finder includes variables in masked systems.

```
% Create a Simulink Report
rpt = slreportgen.report.Report("MyReport","pdf");

% Create a Chapter
chapter = mlreportgen.report.Chapter();
chapter.Title = "Model Variable Finder Example";

% Load the model
model_name = "sf_car";
load_system(model_name)

% Create a variable finder and set its properties
finder = slreportgen.finder.ModelVariableFinder(model_name);
finder.LookUnderMasks = "all";

% Find variables used by the model
results = find(finder);

% Add the results to the chapter
add(chapter,results);
```

```
% Add chapter to the report
add(rpt,chapter);

% Close the report and open the viewer
close(rpt);
rptview(rpt);
```

## Customize the Formatting of Model Variables in a Report

Customize the formatting of model variables in a report by iterating through the search results and setting properties of the model variable reporter for each result.

```
% Create a Report
rpt = slreportgen.report.Report("MyReport","pdf");

% Create a Chapter
chapter = mlreportgen.report.Chapter();
chapter.Title = "Model Variable Reporter Example";

% Load the model
model_name = "sf_car";
load_system(model_name);

% Find the variables in the model
finder = slreportgen.finder.ModelVariableFinder(model_name);

while hasNext(finder)
    result = next(finder);

    % Get the ModelVariable reporter for the result
    % Customize the formatting of numbers
    reporter = getReporter(result);
    reporter.NumericFormat = "%.4f";

    % Add the reporter to the chapter
    add(chapter,reporter);
end
% Add chapter to the report
add(rpt,chapter);

% Close the report and open the viewer
close(rpt);
rptview(rpt);
```

## See Also

Simulink.VariableUsage | Simulink.findVars |  
slreportgen.finder.ModelVariableResult | slreportgen.report.BusObject |  
slreportgen.report.ModelVariable

## Topics

“Report Generation for Simulink and Stateflow Elements” on page 1-9  
“What Is a Reporter?”

**Introduced in R2019b**

# slreportgen.finder.ModelVariableResult class

**Package:** slreportgen.finder

Model variable search result object

## Description

Model variable search result object for a variable used in a Simulink model or subsystem.

The slreportgen.finder.ModelVariableResult class is a handle class.

## Class Attributes

HandleCompatible true

For information on class attributes, see “Class Attributes”.

## Creation

You do not create an slreportgen.finder.ModelVariableResult object explicitly. The slreportgen.finder.ModelVariableFinder find or next methods create an slreportgen.finder.ModelVariableResult object for each variable that is found.

## Properties

### Object — Simulink.VariableUsage object

Simulink.VariableUsage object

Simulink.VariableUsage object for the variable represented by this result. This property is read-only.

### Name — Name of variable

character vector

Name of the variable represented by this result, specified as a character vector. This property is read-only.

### Source — Source of variable definition

character vector

Source of the variable definition, specified as a character vector. This property is read-only. This table shows example values.

Value	Description
'base workspace'	MATLAB base workspace
'MyModel'	Model workspace for MyModel
'MyModel/Mask1'	Mask workspace for a masked block

Value	Description
'my_data_dictionary.slidd'	The data dictionary my_data_dictionary.slidd.

**SourceType — Type of defining workspace**

character vector

Type of the workspace that defines the variable, specified as one of these character vectors:

This property is read-only.

**Users — Blocks that use the variable**

cell array

Blocks that use the variable, specified as a cell array of character vectors. This property is read-only.

**ModelBlockPath — Path of Model block that set the variable value**

character vector

Path of the Model block that set the variable value, specified as a character vector. This property is read-only.

Suppose that a referenced model uses a model argument to set a block parameter value. If a model has multiple instances of the referenced model, the model variable finder returns multiple instances of the variable that is associated with the model argument. The `ModelBlockPath` property uniquely identifies the instance of the variable by providing the path to the Model block that set its value. If a variable is not associated with a model argument in a referenced model, the `ModelBlockPath` is empty. For more information about referenced models and instance-specific parameters, see “Parameterize Instances of a Reusable Referenced Model”.

**Tag — Additional information**

string | character vector | number | ...

Additional information to save with this result. You can set it to any type of value.

**Methods****Public Methods**

`getReporter`      Get reporter for model variable search result  
`getVariableID`    Get unique ID of model variable  
`getVariableValue` Get value of variable from model variable search result

**Examples****Customize the Formatting of Model Variables in a Report**

Customize the formatting of model variables in a report by iterating through the search results and setting properties of the model variable reporter for each result.

```
% Create a Report
rpt = slreportgen.report.Report("MyReport", "pdf");
```

```
% Create a Chapter
chapter = mlreportgen.report.Chapter();
chapter.Title = "Model Variable Reporter Example";

% Load the model
model_name = "sf_car";
load_system(model_name);

% Find the variables in the model
finder = slreportgen.finder.ModelVariableFinder(model_name);

while hasNext(finder)
    result = next(finder);

    % Get the ModelVariable reporter for the result
    % Customize the formatting of numbers
    reporter = getReporter(result);
    reporter.NumericFormat = "%.4f";

    % Add the reporter to the chapter
    add(chapter,reporter);
end
% Add chapter to the report
add(rpt,chapter);

% Close the report and open the viewer
close(rpt);
rptview(rpt);
```

## See Also

Simulink.VariableUsage | slreportgen.finder.ModelVariableFinder |  
slreportgen.report.ModelVariable

## Topics

"Report Generation for Simulink and Stateflow Elements" on page 1-9  
"What Is a Reporter?"

## Introduced in R2019b

## slreportgen.finder.StateFinder class

**Package:** slreportgen.finder

Find Stateflow states

### Description

Finds Stateflow states.

### Construction

`finder = StateFinder(diagram)` creates a finder that finds by default all uncommented Stateflow states in the specified chart `diagram`. To constrain the search to specific types of chart diagrams, use the properties of this finder.

---

**Note** This finder provides two ways to get search results:

- 1 To return the search results as an array, use the `find` method. Add the results directly to a report or process the results in a `for` loop.
- 2 To iterate through the results one at a time, use the `hasNext` and `next` methods in a `while` loop.

Neither option has a performance advantage.

---

`finder = StateFinder(Name, Value)` sets properties using name-value pairs. You can specify multiple name-value pair arguments in any order. Enclose each property name in single quotes.

### Input Arguments

#### **diagram** — Diagram to search

path | handle | chart ID | chart object

See Container property.

### Properties

#### **Container** — Chart diagram to search

path | handle | chart ID | chart object

Chart diagram in which to search, specified as one of these values:

- Handle to a Stateflow chart block
- Path to a Stateflow chart block
- Stateflow chart ID
- Stateflow chart object

#### **IncludeCommented** — Include commented-out states

false (default) | true



Choice to include commented-out states in the search results, specified as a logical. If `false`, commented-out states are excluded from the search results.

### Properties — Properties of states to find

cell array

Properties of states to find, specified as a cell array of name-value pairs. The finder returns only states that have the specified properties with the specified values.

Example: `finder.Properties = {'ArrowSize','5'}`

## Methods

`results = find(finder)` finds states in the chart diagram specified by the finder. This method returns the states it finds wrapped in result objects of type `slreportgen.finder.DiagramElementResult`. To add tables of the state properties, add the results objects directly to the report or add them to a reporter that you then add to a report. The reports to which you can add the results of this method must be reports of type `slreportgen.report.Report`.

`tf = hasNext(finder)` determines if the chart diagram that the finder searches contains at least one state. If the chart diagram has at least one state, the `hasNext` method queues that state as the next state that the `next` method will return. The `hasNext` method then returns `true`. Use the `next` method to obtain that state. On subsequent calls, the `hasNext` method determines if the chart diagram has a state that the `next` method has not yet retrieved. It queues the state for the next method to retrieve and returns `true`. If there are no more states to be retrieved, this method returns `false`. To search a chart diagram progressively for states, use the `hasNext` method with the `next` method in a while loop.

`result = next(finder)` returns the next search result in the result queue that the `hasNext` method created. This method returns the state that it finds wrapped in a result object of type `slreportgen.finder.DiagramElementResult`. To add tables of the state properties, add the results objects directly to the report or add them to a reporter that you then add to a report. The reports to which you can add the results of this method must be of type `slreportgen.report.Report`.

## Copy Semantics

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

## Examples

### Find Stateflow States

Create a report that includes properties of all the Stateflow states in the `shift_logic` chart of the `sf_car` model.

```
import mlreportgen.report.*
import slreportgen.report.*
import slreportgen.finder.*
```

```
model_name = 'sf_car';
load_system(model_name);
```

```
rpt = slreportgen.report.Report('output','pdf');
open(rpt)

add(rpt, TitlePage("Title",...
    sprintf('States in %s Model',model_name)));
add(rpt, TableOfContents);

chartFinder = ChartDiagramFinder(model_name);
charts = find(chartFinder);
while hasNext(chartFinder)
    diagram = next(chartFinder);
    stFinder = StateFinder(diagram.Object);
    states = find(stFinder);
    if ~isempty(states)
        chapter = Chapter("Title",diagram.Name);
        add(chapter,diagram)
        for state = states
            sect = Section("Title","States");
            add(sect,states)
        end
        add(chapter,sect)
        add(rpt,chapter)
    end
end

close(rpt)
close_system(model_name)
rptview(rpt)
```

## See Also

slreportgen.finder.ChartDiagramFinder | slreportgen.finder.DiagramElementFinder  
| slreportgen.finder.DiagramElementResult |  
slreportgen.finder.StateflowDiagramElementFinder | slreportgen.report.Report |  
slreportgen.report.StateflowObjectProperties

## Topics

“Report Generation for Simulink and Stateflow Elements” on page 1-9  
“What Is a Reporter?”

## Introduced in R2017b

# slreportgen.finder.StateflowDiagramElementFinder class

**Package:** slreportgen.finder

Find Stateflow diagram elements

## Description

StateflowDiagramElementFinder creates a finder object that finds elements in a Stateflow chart diagram.

## Construction

`finder = StateflowDiagramElementFinder(diagram)` creates a finder that finds elements of a Stateflow chart diagram. By default this finder finds states, transitions, truth tables, and other elements in the specified Stateflow chart diagram. Use the properties of the finder to constrain the search to specific types of elements.

---

**Note** This finder provides two ways to get search results:

- 1 To return the search results as an array, use the `find` method. Add the results directly to a report or process the results in a `for` loop.
- 2 To iterate through the results one at a time, use the `hasNext` and `next` methods in a `while` loop.

Neither option has a performance advantage.

---

`finder = StateflowDiagramElementFinder(Name, Value)` sets properties using name-value pairs. You can specify multiple name-value pair arguments in any order. Enclose each property name in single quotes.

## Input Arguments

**diagram — Chart Diagram to search**

path | handle | chart ID | chart object

See Container property.

## Properties

**Container — Chart diagram to search**

path | handle | chart ID | chart object

Chart diagram in which to search, specified as one of these values:

- Handle to a Stateflow chart block
- Path to a Stateflow chart block

- Stateflow chart ID
- Stateflow chart object

### Types — Types of diagram elements to find

string | character array | array of strings | cell array of character arrays

Types of Stateflow diagram elements to find, specified as a string, character array, array of strings, or a cell array of character arrays. If the type is an array, it specifies a set of element types. The default is `All` or `all`, which finds all elements in all diagrams. Use one of these values to constrain the search to specific diagram element types. You can use either the fully qualified name or the short name.

Fully Qualified Name	Short Name
All	all
Simulink.Annotation	annotation
Simulink.Block	block
Simulink.Segment	line
Simulink.Port	port
Stateflow.Annotation	sf_annotation
Stateflow.Box	box
Stateflow.EMFunction	emfunction
Stateflow.Function	function
Stateflow.Junction	junction
Stateflow.SLFunction	slfunction
Stateflow.State	state
Stateflow.Transition	transition
Stateflow.TruthTable	truthtable

### IncludeCommented — Include commented-out chart elements

false (default) | true

Whether to include commented-out chart elements in the search results, specified as a logical. If false, commented-out elements are excluded from the search results.

### Properties — Properties of elements to find

cell array

Properties of objects to find, specified as a cell array of name-value pairs. The finder returns only elements that have the specified properties with the specified values.

Example: `finder.Properties = {'ArrowSize','5'}`

### Methods

`results = find(finder)` finds Stateflow chart diagram elements in the diagram specified by the finder. This method returns the chart diagram elements it finds wrapped in result objects of type `slreportgen.finder.DiagramElementResult`. To add tables of the chart diagram element properties, add the results objects directly to the report or add them to a reporter that you then add

to a report. The reports to which you can add the results of this method must be reports of type `slreportgen.report.Report`.

`tf = hasNext(finder)` determines if the chart diagram that the finder searches contains at least one element. If the chart diagram has at least one element, the `hasNext` method queues that element as the next element that the `next` method will return. The `hasNext` method then returns `true`. Use the `next` method to obtain that element. On subsequent calls, the `hasNext` method determines if the chart diagram has an element that the `next` method has not yet retrieved. It queues the element for the `next` method to retrieve and returns `true`. If there are no more elements to be retrieved, this method returns `false`. To search a chart diagram progressively for elements, use the `hasNext` method with the `next` method in a while loop.

`result = next(finder)` returns the next search result in the result queue that the `hasNext` method created. This method returns the chart diagram element that it finds wrapped in a result object of type `slreportgen.finder.DiagramElementResult`. To add tables of the chart diagram element properties, add the results objects directly to the report or add them to a reporter that you then add to a report. The reports to which you can add the results of this method must be of type `slreportgen.report.Report`.

## Copy Semantics

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

## Examples

### Find Stateflow States and Transitions

Create a report that finds Stateflow states and transitions in the `fuelsys` model

```
import mlreportgen.report.*
import slreportgen.report.*
import slreportgen.finder.*
model_name = 'fuelsys';
load_system(model_name)

rpt = slreportgen.report.Report('output','pdf');
add(rpt, TitlePage('Title',sprintf('%s Charts',...
    model_name)))
add(rpt, TableOfContents)

chartFinder = ChartDiagramFinder(model_name);
while hasNext(chartFinder)
    chart = next(chartFinder);
    chapter = Chapter("Title",chart.Name);
    add(chapter, chart)
    sect = Section("Title","States");
    stateFinder = StateFinder(chart.Object);
    states = find(stateFinder);
    for state = states
        add(sect,state)
    end
    add(chapter,sect)

    sect = Section("Title","Transitions");
    transitionFinder = StateflowDiagramElementFinder...
```

```
        ('Container',chart.Object, 'Types', 'transition');
    transitions = find(transitionFinder);
    for transition = transitions
        add(sect,transition)
    end
    add(chapter,sect)
    add(rpt, chapter)
end

close(rpt)
close_system(model_name)
rptview(rpt)
```

### **See Also**

slreportgen.finder.AnnotationFinder | slreportgen.finder.ChartDiagramFinder |  
slreportgen.finder.DiagramElementResult | slreportgen.finder.StateFinder |  
slreportgen.report.Diagram | slreportgen.report.Report |  
slreportgen.report.StateflowObjectProperties

### **Topics**

“Report Generation for Simulink and Stateflow Elements” on page 1-9  
“What Is a Reporter?”

### **Introduced in R2017b**

# slreportgen.finder.SystemDiagramFinder class

**Package:** slreportgen.finder

Create block diagram finder

## Description

Create Simulink block diagram finder.

## Construction

`finder = SystemDiagramFinder(container)` creates a finder that finds by default all uncommented Simulink block diagrams in the specified container, which can be a Simulink model or subsystem. To constrain the search to specific types of models or subsystems, use the properties of the finder.

---

**Note** This finder can operate in either find or iterator mode. In find mode, use its `find` method to return the results of a search as an array of results. In iterator mode, use its `hasNext` and `next` methods to return the results of a search one-by-one. When searching in models that have many model references, use iterator mode. Iterator mode closes a model after compiling and searching it, whereas find mode keeps all the models that it searches open. Having many open models can consume all system memory and slow report generation. Iterator mode is slower than find mode, so use find mode to search models that reference few or no other models.

---

`finder = SystemDiagramFinder(Name, Value)` sets properties using name-value pairs. You can specify multiple name-value pair arguments in any order. Enclose each property name in single quotes.

## Input Arguments

### Container — Diagram container to search

path | handle | chart ID | chart object

See Container property.

## Properties

### Container — Diagram container to search

path | handle | chart ID | chart object

System container in which to search, specified as one of these values:

- Handle of a Simulink model or subsystem
- Path to a Simulink model or subsystem

### SearchDepth — Depth of system diagram search

inf (default) | positive integer

Depth of system diagram search, specified as `inf` or a positive integer. `SearchDepth` specifies how many levels deep to search a diagram container for diagrams. To search all levels, use `inf`.

#### **IncludeMaskedSubsystems — Search masked subsystems**

`true` (default) | `false`

Choice to search masked subsystems, specified as a logical. If this property is `true`, the finder searches masked Subsystem blocks in the diagram container. It searches to the specified `SearchDepth` and includes the diagrams it finds in the search results.

#### **IncludeReferencedModels — Search model references**

`true` (default) | `false`

Choice to search referenced models, specified as a logical. If this property is `true`, the finder searches models referenced in the diagram container. It searches to the specified `SearchDepth` and includes the diagrams it finds in the search results.

#### **IncludeSimulinkLibraryLinks — Search Simulink library links**

`true` (default) | `false`

Choice to search Simulink library links, specified as a logical. If both this property and `IncludeMaskedSubsystems` are `true`, the finder searches links in the diagram container to both Subsystem and masked Subsystem blocks in Simulink libraries. It searches to the specified `SearchDepth` and includes the diagrams it finds in the search results. If this property is `true`, but `IncludeMaskedSubsystems` is `false`, the finder searches only links to Subsystem blocks in Simulink libraries.

#### **IncludeUserLibraryLinks — Search user library links**

`true` (default) | `false`

Choice to search user library links, specified as a logical. If this property is `true` and the `IncludeMaskedSubsystems` property is `true`, the finder searches links in the diagram container to Subsystem and masked Subsystem blocks in user libraries. It searches to the specified `SearchDepth` and includes the diagrams it finds in the search results. If this property is `true`, but the `IncludeMaskedSubsystems` property is `false`, the finder searches only links to Subsystem blocks in user libraries.

#### **IncludeRoot — Include root diagram**

`true` (default) | `false`

Choice to include the root diagram in the search results, specified as a logical. If `true` and the top-level diagram container is a model, the model block diagram is included in the search results. Otherwise, the search results omit the model block diagram.

#### **IncludeCommented — Include commented-out diagrams**

`false` (default) | `true`

Choice to include commented-out diagrams in the search results, specified as a logical. If `false`, commented-out diagrams are excluded from the search results.

#### **IncludeVariants — Include diagram variants**

`string` | character vector

Variants to search for diagrams, specified as a string or character vector. The default value is `Active`. Valid values are:



- `All` — All variants
- `Active` — Only active variants
- `ActivePlusCode` — All active variants and code variants

### Properties — Properties of objects to find

cell array

Properties of objects to find, specified as a cell array of name-value pairs. The finder returns only objects that have the specified properties with the specified values.

Example: `finder.Properties = {'Gain','5'}`

### AutoCloseModel — Whether to close models

`true` (default) | `false`

Whether to close models, specified as `true` or `false`. If `true`, the `next` method of the finder closes the currently open model before moving to the next model to search. Closing models prevents excessive consumption of memory when searching a model that references many models.

---

**Note** The `find` method of the finder ignores this property and leaves all referenced models open. For this reason, you should not use the `find` method to search models with many model references.

---

## Methods

`results = find(finder)` finds block diagrams in the container specified by the finder. The finder is an `sreportgen.finder.SystemDiagramFinder` object. `results` is an array of `sreportgen.finder.DiagramResult` objects, each of which contains a block diagram found by this method. Adding the array to a report or reporter adds images of all the block diagrams it contains. The reports to which you can add the `results` of this method are reports of type `sreportgen.report.Report` or another reporter object, such as an `sreportgen.report.Chapter` reporter.

`tf = hasNext(finder)` determines if the container that the finder searches contains at least one diagram. If the container has at least one diagram, the `hasNext` method queues that diagram as the next diagram that the `next` method will return. The `hasNext` method then returns `true`. Use the `next` method to obtain that diagram. On subsequent calls, the `hasNext` method determines if the container has a diagram that the `next` has not yet retrieved. It queues the diagram for the `next` method to retrieve and returns `true`. If there are no more diagrams to be retrieved, this method returns `false`. To search a container progressively for diagrams, use the `hasNext` method with the `next` method in a while loop.

---

**Note** If the current result is the last result in the search queue for the current model and the `AutoCloseModel` property is `true`, this method closes the current model before it opens the next model. Although this increases search time, it reduces memory consumption when searching a top model that references many other models. If your model does not reference many other models, to speed up the search, set the `AutoCloseModel` property to `false` or use the `find` method.

---

`result = next(finder)` returns the next search result in the result queue that the `hasNext` method created. The search result contains the resulting diagram. Adding this `result` object to a report or reporter adds a `Diagram` reporter for the diagram.

## Copy Semantics

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

## Examples

### Find Block Diagrams

Create a report that finds block diagrams in the `sf_car` model.

```
import mlreportgen.report.*
import slreportgen.report.*
import slreportgen.finder.*
model_name = 'sf_car';
load_system(model_name);

rpt = slreportgen.report.Report('output','pdf');
add(rpt, TitlePage('Title', sprintf('%s Systems',...
    model_name)));
add(rpt,TableOfContents);
finder = SystemDiagramFinder(model_name);
results = find(finder);
for result = results
    chapter = Chapter('Title',result.Name);
    add(chapter,result);
    add(rpt,chapter);
end

close(rpt);
close_system(model_name);
rptview(rpt);
```

### See Also

[slreportgen.finder.DiagramElementFinder](#) | [slreportgen.finder.DiagramFinder](#) | [slreportgen.finder.DiagramResult](#) | [slreportgen.report.Diagram](#) | [slreportgen.report.Report](#) | [slreportgen.report.SimulinkObjectProperties](#)

### Topics

“Report Generation for Simulink and Stateflow Elements” on page 1-9

“What Is a Reporter?”

### Introduced in R2017b

# slreportgen.webview.ExportOptions class

**Package:** slreportgen.webview

Options for exporting Simulink model to web view

## Description

Use an `slreportgen.webview.ExportOptions` object to specify the items to export to a web view or an embedded web view report.

## Construction

`exportopts = slreportgen.webview.ExportOptions(wvdoc)` creates an export options object, `exportopts`, for the `wvdoc` web view or embedded web view document. `wvdoc` is created using `slreportgen.webview.WebViewDocument` or `slreportgen.webview.EmbeddedWebViewDocument`. When you create a Web view or embedded Web view document, an export options object is created automatically.

## Input Arguments

### **wvdoc — Web view object**

`slreportgen.webview.WebViewDocument` or  
`slreportgen.webview.EmbeddedWebViewDocument` object

Web view document, specified as a Web view or embedded Web view object.

## Output Arguments

### **exportopts — Export options**

class

Export options, returned as a class containing properties that specify what to export to a web view document.

## Properties

### **Diagrams — Diagram or diagrams to export to the web view**

character vector

Diagram on page 7-190 or diagrams to export to the web view, specified as a character vector.

### **SearchScope — System and subsystem levels to export for the specified**

`Current` and `Below` (default) | `All` | `Current` and `Above`

System and subsystem levels to export, specified as one of:

- `Current` and `Below` — Current specified system or subsystem and all of its dependents.
- `Current` — Current specified system or subsystem
- `Current` and `Above` — Current specified system or subsystem and all of its antecedents

- All — All systems and subsystems in the model

**IncludeMaskedSubsystems — Option to export masked subsystems**`False` (default) | `True`

Option to export masked subsystems, specified as a logical. If you set `IncludeMaskedSystems` to `False`, no masked subsystems are exported. If you set it to `True`, all masked subsystems are exported.

**IncludeReferencedModels — Option to export referenced models**`False` (default) | `True`

Option to export referenced models, specified as a logical. If you set `IncludeReferencedModels` to `False`, no masked subsystems are exported. If you set it to `True`, all masked subsystems are exported.

**IncludeSimulinkLibraryLinks — Option to export Simulink library links**`False` (default) | `True`

Option to export Simulink library links, specified as a logical. If you set `IncludeSimulinkLibraryLinks` to `False`, no library links are exported. If you set it to `True`, all library links are exported.

**IncludeUserLibraryLinks — Option to export user-defined library links**`False` (default) | `True`

Option to export user-defined library links, specified as a logical. If you set `IncludeUserLibraryLinks` to `False`, no user-defined library links are exported. If you set it to `True`, all user-defined library links are exported.

**FilterCallback — Function to determine whether to export a system and its descendants**

function name or handle

Function to determine whether to export a system and its descendants, specified as a function name or function handle. The function must return `True` to include the system or `False` to exclude the system.

**More About****Diagram**

Diagram refers to a Simulink model, subsystem, or Stateflow chart.

**See Also**

`slreportgen.webview.EmbeddedWebViewDocument` |  
`slreportgen.webview.WebViewDocument` | `slwebview`

**Introduced in R2017a**

# slreportgen.webview.WebViewDocument class

**Package:** slreportgen.webview

Create a web view document generator

## Description

Creates a document generator that generates an HTML document containing a web view of one or more Simulink models.

## Construction

`wvdocgen = slreportgen.webview.WebViewDocument(docname,model)` creates a document generator that generates an HTML document at the specified location containing a web view of the specified model. Use the generator's `fill` method to generate the document.

`wvdocgen = slreportgen.webview.WebViewDocument(docname,model1,model2,...,modeln)` creates a document generator that includes two or more models in the web view that it creates. This constructor assigns an array of default `slreportgen.webview.ExportOptions` objects to the generator's `ExportOptions` property, one for each of the models to be included in the generated document's web view. You can use the objects to specify custom export options for the models to be exported.

`wvdocgen = slreportgen.webview.WebViewDocument(docname,{model1,model2,...,modeln})` assigns a default `slreportgen.webview.ExportOptions` object to the generator's `ExportOptions` property that applies to all of the models to be exported.

`wvdocgen = slreportgen.webview.WebViewDocument(docname)` creates a web view document generator for an initially unspecified model or set of models. Use the `Systems` property of the generator's `ExportOptions` property to specify the model or models to be included in the web view that it generates.

## Input Arguments

**docname** — Name of output document file and folder

character vector

Name of the zip file and/or folder containing the report generated by this generator. Use this generator's `PackageType` property to specify whether to package the generated report as a file or a folder or both. If you specify an extension, the extension must be `.html`. If you do not specify an extension, the report generator appends `.html`.

**model** — Name of Simulink model to export

character vector

Name of the Simulink model to export to an interactive HTML Web view, specified as a character vector.

## Output Arguments

### **wvdoc — Web view document generator**

`slreportgen.Web view.WebViewDocument` object

## Properties

### **CurrentHoldID — Identifier of current hole in document**

character vector

Identifier of current hole in document. This is a read-only property.

### **ExportOptions — Web view export options**

`slreportgen.webview.ExportOptions` (default)

An array of `slreportgen.webview.ExportOptions` objects, one for each model or set of models to be exported. The generator's constructor sets this property with default values for the model or models you specify. Use the properties of the `ExportOptions` object or objects to customize export of the models to the generated web view. For example, you can specify additional models to include or whether to include the block diagrams of masked subsystems and library blocks.

### **ForceOverwrite — Overwrite existing file or folder**

`True` (default) | `False`

Whether to overwrite an existing report with the same name. `True` overwrites the existing report. `False` generates the report under a new name.

### **OpenStatus — Status of the web view document**

`unopened` (default) | `opened`

### **OutputPath — Path of the document output directory**

current working directory (default)

Path of the document output directory.

### **PackageType — Packaging for files generated**

`'both'` (default) | `'zipped'` | `'unzipped'`

Packaging to use for output document, specified as one of these character vectors:

- `'zipped'` — Creates a zip file with an `.htmx` extension
- `'unzipped'` — Creates a folder of files
- `'both'` — Creates both zipped and unzipped output

### **TemplatePath — Path of the template to use to generated this document**

character vector

Path to the HTML template to use to generate this report. The template has an `.htmx` extension. This property points by default to a default template. To use a custom template, set this property to the path of the custom template.

### **TitleBarText — Text for HTML browser title bar**

character vector

Text to display in the title bar of the HTML browser displaying the generated web view document. The default text is "Simulink Web View - Created by Simulink Report Generator."

## Methods

Method	Purpose
fill	Invokes web view generator's hole-filling methods (e.g., fillslwebview) to fill the holes in the web view document's template
fillslwebview	Fills template's slwebview hole with a web view
getExportModels	Names of models to be included in the web view
getExportDiagrams	Paths and handles of block diagrams to be included in the web view
getExportSimulinkSubSystems	Paths and handles of subsystem blocks to be included in this web view
getExportStateflowCharts	Paths and handles of
getExportStateflowDiagrams	Array of Stateflow diagram paths

## Examples

### Export Model to a Web View

```
import slreportgen.webview.*
open_system('f14')
d = WebViewDocument('f14WebView', 'f14');
fill(d);
rptview(d);
```

### Export Multiple Models to a Web View

The export options in this example allow you to view the subsystem that implements the Simulink library block, Band-Limited White Noise, in the f14 model and the Stateflow chart that implements the Engine block in the sf\_car model. If the example did not enable the export options, the subsystem and chart would appear only as blocks in the exported web view.

```
import slreportgen.webview.*
open_system('f14');
open_system('sf_car');
wvdoc = WebViewDocument(...
    'myWebview', 'f14', 'sf_car');
opts = wvdoc.ExportOptions;

f14opts = opts(1);
f14opts.IncludeMaskedSubsystems = true;
f14opts.IncludeSimulinkLibraryLinks = true;

sfcaropts = opts(2);
sfcaropts.IncludeMaskedSubsystems = true;

fill(wvdoc)
rptview(wvdoc);
```

### **Export Selected Systems to a Web View**

This example exports f14's root system and Aircraft Dynamics Model subsystem.

```
import slreportgen.webview.*
open_system('f14');
wvdoc = WebViewDocument('myWebView', ...
    {'f14', 'f14/Aircraft Dynamics Model'});
wvdoc.ExportOptions.SearchScope = 'Current';
fill(wvdoc)
rptview(wvdoc);
```

### **Export Subsystem's Ancestors to a Web View**

This example exports f14's root system and Controller subsystem.

```
import slreportgen.webview.*
open_system('f14');
wvdoc = WebViewDocument(...
    'mydoc', 'f14/Controller');
wvdoc.ExportOptions.SearchScope = 'CurrentAndAbove';
fill(wvdoc)
rptview(wvdoc);
```

### **See Also**

[slreportgen.webview.EmbeddedWebViewDocument](#) | [slreportgen.webview.ExportOptions](#)

**Introduced in R2017a**



# Functions

---

## getDiagramReporter

**Class:** slreportgen.finder.BlockResult

**Package:** slreportgen.finder

Returns Diagram reporter for this block result

### Syntax

```
reporter = getDiagramReporter(result)
```

### Description

`reporter = getDiagramReporter(result)` returns a diagram reporter if the block result contains a subsystem or chart block. The reporter generates a snapshot of the block's diagram or chart, respectively. If the block result contains any other type of block, this method returns empty, []. To include a diagram of the subsystem or chart block search result in a report, add this reporter to the report, either directly or via a Chapter or Section reporter.

### Input Arguments

**result — Block result object**

BlockResult object

BlockResult object, which is the output of the `slreportgen.finder.BlockFinder` class.

### Output Arguments

**reporter — Diagram reporter object**

slreportgen.report.Diagram object | []

Diagram reporter object, returned as an `slreportgen.report.Diagram` or empty, []. If the result contains a subsystem or chart block, this result returns a Diagram reporter that generates a snapshot of the block's block diagram or chart, respectively. Otherwise, it returns empty, [].

### Examples

#### Add Block Diagram to Report

Add a subsystem snapshot and property table of the Controller block subsystem of the f14 model to a report.

```
model_name = 'f14';
load_system(model_name)
import slreportgen.report.*
import slreportgen.finder.*
import mlreportgen.report.*

rpt = slreportgen.report.Report('output', 'pdf');
chapter = Chapter();
```

```
chapter.Title = 'Block Diagram Reporter Example';

blkFinder = BlockFinder(model_name);
blocks = find(blkFinder);
for block = blocks
    if block.Name == "Controller"
        rptr = getDiagramReporter(block);
        section = Section("Title", ...
            strep(block.Name, newline, ' '));
        add(section,rptr);
        add(section,block);
        add(chapter,section);
    end
end
add(rpt,chapter)
rptview(rpt)
```

# Chapter 1. Block Diagram Reporter Example

## 1.1. Controller

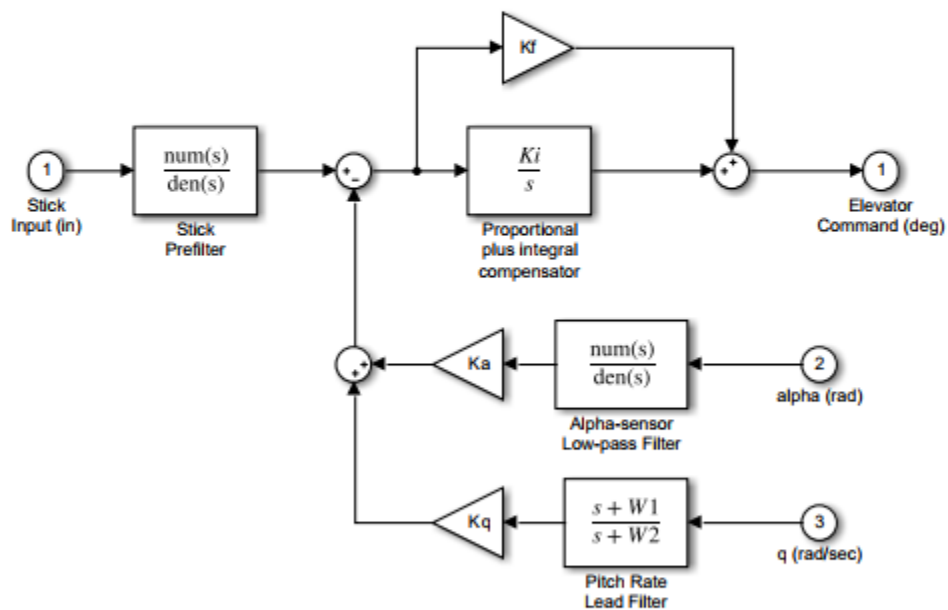


Figure 1.1. Controller

Table 1.1. f14/Controller Properties

Property	Value
Type	System

### See Also

`slreportgen.finder.BlockFinder` | `slreportgen.report.Diagram`

Introduced in R2018b

# getReporter

**Class:** `slreportgen.finder.BlockResult`

**Package:** `slreportgen.finder`

Get block reporter

## Syntax

```
reporter = getReporter(blockResult)
```

## Description

`reporter = getReporter(blockResult)` returns a reporter that generates a properties table for the block in the result.

## Input Arguments

**blockResult** — **Block result object**

`slreportgen.finder.BlockResult` object

Search result object for a block, specified as an `slreportgen.finder.BlockResult` object.

## Output Arguments

**reporter** — **Simulink object properties reporter**

`slreportgen.report.SimulinkObjectProperties` object

Simulink object properties reporter, returned as an `slreportgen.report.SimulinkObjectProperties` object. This reporter generates a properties table for the block.

## See Also

`slreportgen.finder.BlockFinder` | `slreportgen.report.SimulinkObjectProperties`

**Introduced in R2017b**

## getDiagramReporter

**Class:** `slreportgen.finder.DiagramElementResult`

**Package:** `slreportgen.finder`

Returns Diagram reporter for diagram element result

### Syntax

```
reporter = getDiagramReporter(result)
```

### Description

`reporter = getDiagramReporter(result)` returns a reporter that generates a snapshot of the element returned in the diagram element result, or empty, []. If the result contains a diagram element, such as a Simulink block, or Stateflow chart or subchart, that contains a diagram, this method returns a reporter that generates a snapshot of the diagram. Otherwise, it returns empty, []. For example, this method returns a diagram reporter for chart and subchart results but [] for state results, which do not contain diagrams. To include the diagram of an applicable search result in a report, add this reporter to the report, either directly or via a Chapter or Section reporter.

### Input Arguments

**result** — Diagram element result object

`DiagramElementResult` object

`DiagramElementResult` object, which is the output of the `slreportgen.finder.DiagramElementFinder` class.

### Output Arguments

**reporter** — Diagram reporter object

`slreportgen.report.Diagram` object | []

Diagram reporter object, returned as an `slreportgen.report.Diagram` object, depending on the search result. If the search result is an object that contains a diagram, such as a subsystem block or a subchart, the output is a diagram reporter. Otherwise, it is empty, []. If not empty, the diagram reporter generates a snapshot image of the diagram element.

### Examples

#### Add Stateflow Function Diagram to Report

The `sf_car` model uses a Simulink Function, which is a function that uses a Simulink subsystem to compute its inputs from its outputs. In addition to the properties of the function, include the subsystem block diagram in the report.

```
import slreportgen.report.*
import slreportgen.finder.*
import mlreportgen.report.*
```

```
model = "sf_car";
load_system(model)

rpt = slreportgen.report.Report('output','pdf');
chapter = Chapter();
chapter.Title = 'Diagram Element Reporter Example';

diagFinder = DiagramFinder(model);
diagrams = find(diagFinder);
for diag = diagrams
    elemFinder = DiagramElementFinder(diag);
    elemFinder.Types = "slfunction";
    elems = find(elemFinder);
    for elem = elems
        section = Section("Title", ...
            strep(elem.Type, newline, ' '));
        rptr = getDiagramReporter(elem);
        if ~isempty(rptr)
            add(section,rptr)
        end
        r = getReporter(elem);
        add(section,elem)
        add(chapter,section)
    end
end

add(rpt,chapter)
close(rpt)
rptview(rpt)
```

# Chapter 1. Diagram Element Reporter Example

## 1.1. Stateflow.SLFunction

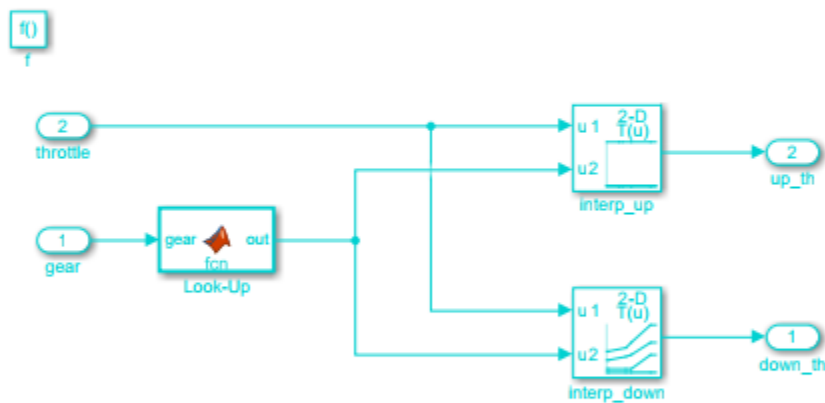


Figure 1.1. calc\_th

**Table 1.1. sf\_car/shift\_logic/selection\_state/calc\_th Properties**

Property	Value
Type	SLFunction
LabelString	[down_th,up_th] = calc_th(gear,throttle)
SimulinkSubSystem	selection_state.calc_th

### See Also

`slreportgen.finder.DiagramElementFinder` | `slreportgen.report.Diagram`

**Introduced in R2018b**



# getReporter

**Class:** `slreportgen.finder.DiagramElementResult`

**Package:** `slreportgen.finder`

Get diagram element reporter

## Syntax

```
reporter = getReporter(result)
```

## Description

`reporter = getReporter(result)` returns a reporter that generates a properties table for the diagram element returned in this result. A report or reporter to which a diagram element result is added invokes this method to generate a properties table for the element in the result. If you need to customize a properties table, you can invoke this method to get the properties reporter. Then, modify the reporter and add the modified properties reporter to the target report or reporter.

## Input Arguments

**result** — **Diagram element finder object**

finder object

DiagramElementFinder object, which you create using `slreportgen.finder.DiagramElementFinder`.

## Output Arguments

**reporter** — **Diagram element reporter object**

`slreportgen.finder.SimulinkObjectProperties` reporter.

Diagram element reporter object, returned as an `slreportgen.finder.SimulinkObjectProperties` object for block diagram elements. For Stateflow chart elements, the returned object is an `slreportgen.finder.StateflowObjectProperties` reporter.

## See Also

`slreportgen.finder.DiagramElementResult` |  
`slreportgen.report.SimulinkObjectProperties` |  
`slreportgen.report.StateflowObjectProperties`

**Introduced in R2017b**

## getReporter

**Class:** `slreportgen.finder.DiagramResult`

**Package:** `slreportgen.finder`

Get diagram reporter

### Syntax

```
reporter = getReporter(result)
```

### Description

`reporter = getReporter(result)` returns an `slreportgen.report.Diagram` reporter for the diagram returned by this report. The diagram reporter adds a snapshot of the diagram to a report. A report or reporter to which you add a diagram result invokes this method to create an image of the diagram that the result contains. If you need to change the size or otherwise customize the diagram image, you can invoke this method to get the diagram reporter. Then, modify the reporter and add the modified diagram reporter to a destination report or reporter.

### Input Arguments

**result — Diagram finder object**

`finder object`

DiagramFinder object, which you create using the `slreportgen.finder.DiagramFinder` class.

### Output Arguments

**reporter — Diagram reporter object**

`slreportgen.report.Diagram` object

Diagram reporter object, returned as an `slreportgen.report.Diagram` object. The Diagram reporter generates a snapshot image of the diagram.

### See Also

`slreportgen.finder.DiagramFinder` | `slreportgen.report.Diagram`

**Introduced in R2017b**

# getReporter

**Class:** slreportgen.finder.ModelVariableResult

**Package:** slreportgen.finder

Get reporter for model variable search result

## Syntax

```
reporter = getReporter(variableResult)
```

## Description

reporter = getReporter(variableResult) returns an slreportgen.report.ModelVariable object for a model variable search result.

## Input Arguments

**variableResult** — Result of model variable search

slreportgen.finder.ModelVariableResult object

Result of a search using the find or next method of an slreportgen.finder.ModelVariableFinder object.

## Output Arguments

**reporter** — Reporter for model variable

slreportgen.report.ModelVariable object

Reporter that includes information about a model variable in a report. Customize the content and formatting of the information for a variable by setting properties of the reporter.

## Examples

### Customize the Formatting of a Model Variable in a Report

Customize the formatting of a model variable in a report by setting the reporter properties.

```
% Create a Report
rpt = slreportgen.report.Report("MyReport", "pdf");

% Create a Chapter
chapter = mlreportgen.report.Chapter();
chapter.Title = "Model Variable Reporter Example";

% Load the model
model_name = "sf_car";
load_system(model_name);
```

```
% Find the variables in the model
finder = slreportgen.finder.ModelVariableFinder(model_name);

while hasNext(finder)
    result = next(finder);

    % Get the ModelVariable reporter for the result
    % Customize the formatting of numbers
    reporter = getReporter(result);
    reporter.NumericFormat = "%.4f";

    % Add the reporter to the chapter
    add(chapter,reporter);
end
% Add chapter to the report
add(rpt,chapter);

% Close the report and open the viewer
close(rpt);
rptview(rpt);
```

## See Also

slreportgen.finder.ModelVariableFinder | slreportgen.finder.ModelVariableResult  
| slreportgen.report.ModelVariable

## Topics

“Report Generation for Simulink and Stateflow Elements” on page 1-9  
“What Is a Reporter?”

## Introduced in R2019b

## getVariableID

**Class:** slreportgen.finder.ModelVariableResult

**Package:** slreportgen.finder

Get unique ID of model variable

### Syntax

```
varID = getVariableID(variableResult)
```

### Description

`varID = getVariableID(variableResult)` returns a string that uniquely identifies the variable represented by the model variable search result. This ID is the default value of the `LinkTarget` property of the `slreportgen.report.ModelVariable` reporter for the variable. Therefore, you can use the ID to generate a link to the reported content for the variable.

### Input Arguments

**variableResult** — Result that represents a model variable

`slreportgen.finder.ModelVariableResult` object

Result of a search using the `find` or `next` method of an `slreportgen.finder.ModelVariableFinder` object.

### Output Arguments

**varID** — Variable ID

string scalar

Unique ID for the model variable, returned as a string scalar.

### Examples

#### Create a List of Variables with Links to Content

You can use the variable ID returned by the `getVariableID` method to create a link to the reported content for the variable. This example generates a report of the variables used by the `sf_car` model. The list of variables at the beginning of the report provides links to the reported content for the variables.

```
% Create a Report
rpt = slreportgen.report.Report("MyReport", "pdf");

% Load the model
model_name = "sf_car";
load_system(model_name);
```

```
% Create a Chapter
chapter = mlreportgen.report.Chapter();
chapter.Title = sprintf("Variables Used in the %s model",model_name);

% Find the variables in the model
finder = slreportgen.finder.ModelVariableFinder(model_name);
results = find(finder);

% Create a list of the variables with links to the reported variable content
ul = mlreportgen.dom.OrderedList;
for r = results
    varname = r.Name;
    %get ID that is used for the link target for this variable
    varid = getVariableID(r);
    link = mlreportgen.dom.InternalLink(varid,varname);
    li = mlreportgen.dom.ListItem();
    append(li,link);
    append(ul,li);
end
add(chapter,ul);

% Add reporters for the variables to report
for r = results
    % Get the ModelVariable reporter for the result
    % Customize the formatting of numbers
    reporter = getReporter(r);
    reporter.NumericFormat = "%.4f";

    % Add the reporter to the chapter
    add(chapter,reporter);
end
add(rpt,chapter);

% Close the report and open the viewer
close(rpt);
rptview(rpt);
```

## See Also

[slreportgen.finder.ModelVariableFinder](#) | [slreportgen.finder.ModelVariableResult](#) | [slreportgen.report.ModelVariable](#)

## Topics

“Report Generation for Simulink and Stateflow Elements” on page 1-9  
“What Is a Reporter?”

## Introduced in R2019b

# getVariableValue

**Class:** slreportgen.finder.ModelVariableResult

**Package:** slreportgen.finder

Get value of variable from model variable search result

## Syntax

```
value = getVariableValue(variableResult)
```

## Description

`value = getVariableValue(variableResult)` returns the value of the variable represented by the model variable search result.

## Input Arguments

**variableResult** — Result that represents a model variable

slreportgen.finder.ModelVariableResult object

Result of a search using the `find` or `next` method of an `slreportgen.finder.ModelVariableFinder` object.

## Examples

### Use getVariableValue to Identify Simulink.Bus Objects

If the `getVariableValue` method returns a `Simulink.Bus` object, use an `slreportgen.report.BusObject` object instead of an `slreportgen.report.ModelVariable` object to report on the bus object.

```
mdl = "sldemo_bus_arrays";
load_system(mdl);

rpt = slreportgen.report.Report("ExampleBusReport", "pdf");

% Find variables used by the model
f = slreportgen.finder.ModelVariableFinder(mdl);
results = find(f);

for r = results
    % If the result represents a Bus object, add a Bus object reporter to the
    % report
    if isa(getVariableValue(r), "Simulink.Bus")
        reporter = slreportgen.report.BusObject(r);
        % Add the reporter to a chapter in the report
        ch = slreportgen.report.Chapter(reporter.Name);
        add(ch, reporter);
        add(rpt, ch);
    end
end
```

```
        end
    end

    % Close and view the report
    close(rpt);
    rptview(rpt);
```

## See Also

[slreportgen.finder.ModelVariableFinder](#) | [slreportgen.finder.ModelVariableResult](#)  
| [slreportgen.report.BusObject](#) | [slreportgen.report.ModelVariable](#)

## Topics

“Report Generation for Simulink and Stateflow Elements” on page 1-9  
“What Is a Reporter?”

**Introduced in R2019b**



# slreportgen.report.BusObject.createTemplate

**Class:** slreportgen.report.BusObject

**Package:** slreportgen.report

Create bus object reporter template

## Syntax

```
template = slreportgen.report.BusObject.createTemplate(templatePath,type)
```

## Description

`template = slreportgen.report.BusObject.createTemplate(templatePath,type)` creates a copy of the bus object reporter template specified by `type` at the `templatePath` location. You can use the copied template as a starting point to design a custom bus object template for your report.

## Input Arguments

**templatePath — Path and file name of new template**

character vector | string scalar

Path and file name of the new template, specified as a character vector or string scalar.

**type — Type of template**

'html' | 'html-file' | 'docx' | 'pdf'

Type of template, specified as 'html', 'html-file', 'docx', or 'pdf'.

## Output Arguments

**template — Path and file name of template copy**

string scalar

Path and file name of the copy of the template, returned as a string scalar. The file name extension of the template is assigned based on the specified template type. For example, if the `type` argument is 'pdf', the file name extension is `.pdftx`.

## Examples

### Create a Bus Object Reporter Template

Create a copy of the HTML template for the bus object reporter and save it with the name `myBusTemplate` in the `mytemplates` folder.

```
template = slreportgen.report.BusObject.createTemplate...  
('mytemplates/myBusObjectTemplate','html');
```

After you modify the template, you can use it by setting the `TemplateSrc` property of the reporter.

**See Also**

`slreportgen.report.BusObject` | `slreportgen.report.Report`

**Introduced in R2019b**

# slreportgen.report.BusObject.customizeReporter

**Class:** slreportgen.report.BusObject

**Package:** slreportgen.report

Create custom bus object reporter

## Syntax

```
reporter = slreportgen.report.BusObject.customizeReporter(classpath)
```

## Description

`reporter = slreportgen.report.BusObject.customizeReporter(classpath)` creates a reporter class definition file that defines a subclass of `slreportgen.report.BusObject`. The `slreportgen.report.BusObject.customizeReporter` method creates the reporter class definition file at the location specified by `classpath`. The method also copies the default reporter templates to the `resources/templates` subfolder of the folder that contains the class definition file. You can use the class definition file as a starting point to design a custom bus object reporter class for your report.

## Input Arguments

**classpath — Path and name of new class definition file**

string scalar | character vector

Path and name of new class definition file, specified as a string scalar or character vector.

You can specify a relative path or an absolute path. For example, this code creates `MyClass.m` in the subfolder `myFolder` of the current folder.

```
slreportgen.report.BusObject.customizeReporter("myFolder/MyClass")
```

To create the reporter class in a class folder, precede the class name with the `@` character. Do not specify the `.m` extension. For example, this code creates `MyClass.m` in the subfolder `myFolder/@MyClass` in the current folder.

```
slreportgen.report.BusObject.customizeReporter("myFolder/@MyClass")
```

See “Folders Containing Class Definitions”.

To create the reporter class in a class package, precede the folder name with the `+` character. For example, this code creates a bus object reporter in the `myOrg` package folder in the current folder.

```
slreportgen.report.BusObject.customizeReporter("+myOrg/@MyClass")
```

## Output Arguments

**reporter — Path and file name of new bus object reporter class**

string scalar

Path and file name of the new bus object reporter class, returned as a string scalar.

## Examples

### Create Custom Bus Object Reporter

Create a custom bus object reporter, MyBus, and the associated default templates in the subfolder MyFolder of the current working folder.

```
slreportgen.report.BusObject.customizeReporter('MyFolder/MyBus')
```

```
ans =
```

```
    "MyFolder\MyBus.m"
```

### See Also

`slreportgen.report.BusObject` | `slreportgen.report.Report`

**Introduced in R2019b**

# slreportgen.report.BusObject.getClassFolder

**Class:** slreportgen.report.BusObject

**Package:** slreportgen.report

Bus object reporter class definition file location

## Syntax

```
path = slreportgen.report.BusObject.getClassFolder()
```

## Description

`path = slreportgen.report.BusObject.getClassFolder()` returns the path of the folder that contains the `slreportgen.report.BusObject` class definition file.

## Output Arguments

**path** — Bus object reporter class definition file location

character vector

slreportgen.report.BusObject class definition file location, returned as a character vector.

## Examples

### Get Bus Object Reporter Class Folder

Get the location of the folder that contains the bus object reporter class definition.

```
path = slreportgen.report.BusObject.getClassFolder()
```

## See Also

slreportgen.report.BusObject | slreportgen.report.Report

**Introduced in R2019b**

## slreportgen.report.DataDictionary.createTemplate

**Class:** slreportgen.report.DataDictionary

**Package:** slreportgen.report

Copy the default slreportgen.report.DataDictionary reporter template

### Syntax

```
template = slreportgen.report.DataDictionary.createTemplate(templatePath,  
type)
```

### Description

template = slreportgen.report.DataDictionary.createTemplate(templatePath, type) creates a copy of the slreportgen.report.DataDictionary reporter template for the report type specified by type at the location specified by templatePath. You can use the copy of the template as a starting point to design a custom slreportgen.report.DataDictionary template for your report.

### Input Arguments

**templatePath — Path and file name of new template**

character vector | string scalar

Path and file name of the new template, specified as a character vector or string scalar.

**type — Type of template**

'html' | 'html-file' | 'docx' | 'pdf'

Type of template, specified as 'html', 'html-file', 'docx', or 'pdf'.

### Output Arguments

**template — Path and file name of template copy**

string scalar

Path and file name of the copy of the template, returned as a string scalar. The file name extension of the template is assigned based on the specified template type. For example, if the type argument is 'pdf', the file name extension is .pdftx.

### Examples

#### Create a Data Dictionary Reporter Template

Create a copy of the HTML template for the slreportgen.report.DataDictionary reporter and save it with the name myDataDictionaryTemplate in the mytemplates folder.

```
template = slreportgen.report.DataDictionary.createTemplate...  
    ('mytemplates/myDataDictionaryTemplate', 'html');
```

After you modify the template, you can use it by setting the `TemplateSrc` property of a data dictionary reporter to the path of the template file.

## See Also

`slreportgen.report.DataDictionary` | `slreportgen.report.Report`

**Introduced in R2020b**

## slreportgen.report.DataDictionary.customizeReporter

**Class:** slreportgen.report.DataDictionary

**Package:** slreportgen.report

Create subclass of slreportgen.report.DataDictionary class

### Syntax

```
reporter = slreportgen.report.DataDictionary.customizeReporter(classpath)
```

### Description

`reporter = slreportgen.report.DataDictionary.customizeReporter(classpath)` creates a reporter class definition file that defines a subclass of `slreportgen.report.DataDictionary` at the location specified by `classpath`. The method also copies the default reporter templates to the `resources/templates` subfolder of the folder that contains the class definition file. You can use the class definition file as a starting point to design a custom data dictionary reporter class for your report.

### Input Arguments

**classpath — Path and name of new class definition file**

string scalar | character vector

Path and name of new class definition file, specified as a string scalar or character vector.

You can specify a relative path or an absolute path. For example, this code creates `MyClass.m` in the subfolder `myFolder` of the current folder.

```
slreportgen.report.DataDictionary.customizeReporter("myFolder/MyClass")
```

To create the reporter class in a class folder, precede the class name with the `@` character. Do not specify the `.m` extension. For example, this code creates `MyClass.m` in the subfolder `myFolder/@MyClass` in the current folder.

```
slreportgen.report.DataDictionary.customizeReporter("myFolder/@MyClass")
```

See “Folders Containing Class Definitions”.

To create the reporter class in a class package, precede the folder name with the `+` character. For example, this code creates a data dictionary reporter in the `myOrg` package folder in the current folder.

```
slreportgen.report.DataDictionary.customizeReporter("+myOrg/@MyClass")
```

### Output Arguments

**reporter — Path and file name of new reporter class**

string scalar



Path and file name of the new reporter class, returned as a string scalar.

## Examples

### Create Custom Data Dictionary Reporter

Create a custom data dictionary reporter, `myDataDictionary`, and the associated default templates in the subfolder `MyFolder` of the current working folder.

```
slreportgen.report.DataDictionary.customizeReporter('MyFolder/myDataDictionary')
```

```
ans =
```

```
    "MyFolder\myDataDictionary.m"
```

### See Also

`slreportgen.report.DataDictionary` | `slreportgen.report.Report`

**Introduced in R2020b**

## **slreportgen.report.DataDictionary.getClassFolder**

**Class:** slreportgen.report.DataDictionary

**Package:** slreportgen.report

Get location of folder that contains the slreportgen.report.DataDictionary class definition file

### **Syntax**

```
path = slreportgen.report.DataDictionary.getClassFolder()
```

### **Description**

path = slreportgen.report.DataDictionary.getClassFolder() returns the path of the folder that contains the slreportgen.report.DataDictionary class definition file.

### **Output Arguments**

**path** — slreportgen.report.DataDictionary class definition file location

character vector

slreportgen.report.DataDictionary class definition file location, returned as a character vector.

### **Examples**

#### **Get Data Dictionary Reporter Class Folder**

Get the location of the folder that contains the data dictionary reporter class definition file.

```
path = slreportgen.report.DataDictionary.getClassFolder()
```

### **See Also**

slreportgen.report.DataDictionary | slreportgen.report.Report

**Introduced in R2020b**

# slreportgen.report.Diagram.createTemplate

**Class:** slreportgen.report.Diagram

**Package:** slreportgen.report

Create diagram template

## Syntax

```
template = slreportgen.report.Diagram.createTemplate(templatePath,type)
```

## Description

`template = slreportgen.report.Diagram.createTemplate(templatePath,type)` creates a copy of the default diagram template specified by `type` at the location specified by `templatePath`. Use the copied template as a starting point to design a custom diagram template for your report.

## Input Arguments

**templatePath — Path and file name of new template**

character vector | string scalar

Path and file name of the new template, specified as a character vector or string scalar.

**type — Type of template**

'html' | 'html-file' | 'docx' | 'pdf'

Type of template, specified as 'html', 'html-file', 'docx', or 'pdf'.

## Output Arguments

**template — Path and file name of template copy**

string scalar

Path and file name of the copy of the template, returned as a string scalar. The file name extension of the template is assigned based on the specified template type. For example, if the `type` argument is 'pdf', the file name extension is `.pdf.tx`.

## Examples

### Create a Report Template

Before you run this example, create a copy of the default HTML diagram template in the `mytemplates` folder. Name the copied template `myDiagramReporter.htm`. To use the new template, assign its path to the `slreportgen.report.Diagram.TemplateSrc` property.

```
import slreportgen.report.*
rpt = Report('My Report','html');
load_system('sf_car')
diagram = Diagram('sf_car');
```

```
template = Diagram.createTemplate('mytemplates\myDiagram', 'html');  
diagram.TemplateSrc = template;
```

**See Also**

`slreportgen.report.Diagram` | `slreportgen.report.Report`

**Introduced in R2017b**

# slreportgen.report.Diagram.customizeReporter

**Class:** slreportgen.report.Diagram

**Package:** slreportgen.report

Create custom diagram reporter class

## Syntax

```
reporter = slreportgen.report.Diagram.customizeReporter(classpath)
```

## Description

`reporter = slreportgen.report.Diagram.customizeReporter(classpath)` creates a diagram class definition file that is a subclass of `slreportgen.report.Diagram`. The file is created at the specified `classpath` location. The `customizeReporter` method also copies the default diagram templates to the `<classpath>/resources/template` folder. You can use the new class definition file as a starting point to design a custom diagram class for your report.

## Input Arguments

**classpath — Location of custom diagram class**

current working folder (default) | string | character array

Location of custom diagram class, specified as a string or character array. The `classpath` argument also supports specifying a folder with `@` before the class name.

## Output Arguments

**reporter — Diagram reporter path**

string

Diagram reporter path, returned as the string specifying the path to the derived report class file.

## Examples

### Create Custom Diagram Reporter

Create a custom diagram reporter and its associated default templates. The derived class file is created at the specified path relative to the current working folder. In this case, the path to the `MyDiagram.m` class file is `<current working folder>/newDiagram/@MyDiagram/MyDiagram.m`. The default diagram templates are in the `<current working folder>/newDiagram/@MyDiagram/resources/templates` folder.

```
import slreportgen.report.*
Diagram.customizeReporter('newDiagram/@MyDiagram');
```

After editing this new class file and loading a model, you can use the new diagram reporter.

```
sf_car;
diagram = MyDiagram('sf_car');
```

**See Also**

`slreportgen.report.Diagram` | `slreportgen.report.Report`

**Introduced in R2017b**

## slreportgen.report.Diagram.getClassFolder

**Class:** slreportgen.report.Diagram

**Package:** slreportgen.report

Diagram class definition file location

### Syntax

```
path = slreportgen.report.Diagram.getClassFolder()
```

### Description

`path = slreportgen.report.Diagram.getClassFolder()` returns the path of the folder that contains the diagram class definition file.

### Output Arguments

**path — Diagram class definition file location**

character array

Diagram class definition file location, returned as a character array.

### See Also

slreportgen.report.Diagram | slreportgen.report.Report

**Introduced in R2017b**

## getSnapshotImage

**Class:** `slreportgen.report.Diagram`

**Package:** `slreportgen.report`

Diagram snapshot image file location

### Syntax

```
path = getSnapshotImage(diag,rpt)
```

### Description

`path = getSnapshotImage(diag, rpt)` takes the snapshot of the diagram specified by the `slreportgen.report.Diagram` reporter (`diag`). It creates an image file and returns the path of that file. `rpt` is the report into which the diagram is added. This method gives you access to the image file so you can place it in specific locations of the report, such as on a title page. By changing the report layout and then adding this image, you can control the image layout.

---

**Note** If you use this method, set the `Diagram Scaling` property to `custom` or `zoom`. If you use `auto` scaling, the image does not scale to fit on the page.

---

### Input Arguments

#### **diag** — Dialog reporter

variable name

Dialog reporter, specified as the variable name of the `Diagram` class. For example,

```
rpt = slreportgen.report.Report
diag = slreportgen.report.Diagram...
      ("f14/Aircraft Dynamics Model");
getSnapshotImage(diag,rpt)
```

#### **rpt** — Report name

variable name

Name of the report into which the diagram will go, specified as the report variable name.

### Output Arguments

#### **path** — Location of snapshot image file

string

Location of snapshot image file, returned as a string. The location is a temporary folder that is deleted when the report is closed. To retain the folder, set the `Debug` property of `slreportgen.report.Report`.



## **See Also**

`slreportgen.report.Diagram` | `slreportgen.report.Report`

**Introduced in R2018b**

## slreportgen.report.DocBlock.createTemplate

**Class:** slreportgen.report.DocBlock

**Package:** slreportgen.report

Create DocBlock reporter template

### Syntax

```
template = slreportgen.report.DocBlock.createTemplate(templatePath,type)
```

### Description

`template = slreportgen.report.DocBlock.createTemplate(templatePath,type)` creates a copy of the DocBlock reporter template specified by `type` at the `templatePath` location. You can use the copied template as a starting point to design a custom DocBlock template for your report.

### Input Arguments

**templatePath — Path and file name of new template**

character vector | string scalar

Path and file name of the new template, specified as a character vector or string scalar.

**type — Type of template**

'html' | 'html-file' | 'docx' | 'pdf'

Type of template, specified as 'html', 'html-file', 'docx', or 'pdf'.

### Output Arguments

**template — Path and file name of template copy**

string scalar

Path and file name of the copy of the template, returned as a string scalar. The file name extension of the template is assigned based on the specified template type. For example, if the `type` argument is 'pdf', the file name extension is `.pdftx`.

### Examples

#### Create a DocBlock Reporter Template

Create a copy of the HTML template for the DocBlock reporter and save it with the name `myDocBlockTemplate` in the `mytemplates` folder.

```
template = slreportgen.report.DocBlock.createTemplate...  
('mytemplates/myDocBlockTemplate','html');
```

After you modify the template, you can use it by setting the `TemplateSrc` property of the reporter.

## **See Also**

`slreportgen.report.DocBlock` | `slreportgen.report.Report`

### **Topics**

“Modify Styles in a Microsoft Word Template”

“Modify Styles in HTML Templates”

“Modify Styles in PDF Templates”

### **Introduced in R2019b**

# slreportgen.report.DocBlock.customizeReporter

**Class:** slreportgen.report.DocBlock

**Package:** slreportgen.report

Create custom DocBlock reporter class

## Syntax

```
reporter = slreportgen.report.DocBlock.customizeReporter(classpath)
```

## Description

`reporter = slreportgen.report.DocBlock.customizeReporter(classpath)` creates a class definition file that defines a subclass of `slreportgen.report.DocBlock` at the location specified by `classpath`. The method also copies the default reporter templates to the `resources/templates` subfolder of the folder that contains the class definition file. You can use the class definition file as a starting point to design a custom DocBlock reporter class for your report.

## Input Arguments

**classpath — Path and name of new class definition file**

string scalar | character vector

Path and name of new class definition file, specified as a string scalar or character vector.

You can specify a relative path or an absolute path. For example, this code creates `MyClass.m` in the subfolder `myFolder` of the current folder.

```
slreportgen.report.DocBlock.customizeReporter("myFolder/MyClass")
```

To create the reporter class in a class folder, precede the class name with the `@` character. Do not specify the `.m` extension. For example, this code creates `MyClass.m` in the subfolder `myFolder/@MyClass` in the current folder.

```
slreportgen.report.DocBlock.customizeReporter("myFolder/@MyClass")
```

See “Folders Containing Class Definitions”.

To create the reporter class in a class package, precede the folder name with the `+` character. For example, this code creates a DocBlock reporter in the `myOrg` package folder in the current folder.

```
slreportgen.report.DocBlock.customizeReporter("+myOrg/@DocBlock");
```

## Output Arguments

**reporter — Path and file name of the new DocBlock reporter class**

string scalar

Path and file name of the new DocBlock reporter class, returned as a string scalar.

## Examples

### Create Custom DocBlock Reporter

Create a custom DocBlock reporter, MyDocBlock, and its associated default templates in the subfolder MyFolder of the current working folder.

```
slreportgen.report.DocBlock.customizeReporter('MyFolder/MyDocBlock')
```

```
ans =
```

```
    "MyFolder\MyDocBlock.m"
```

After editing this new class file, you can use it as your DocBlock reporter.

```
rptr = MyDocBlock();
```

### Create Custom DocBlock Reporter in a Class Folder Inside a Package Folder

Create a custom DocBlock reporter and its associated default templates in a class folder that is a subfolder of a package folder.

```
slreportgen.report.DocBlock.customizeReporter("+MyPackage/@MyDocBlock")
```

```
ans =
```

```
    "+MyPackage\@MyDocBlock\MyDocBlock.m"
```

After editing this new class file, you can use it as your DocBlock reporter.

```
rptr = MyPackage.MyDocBlock();
```

### See Also

[slreportgen.report.DocBlock](#) | [slreportgen.report.Report](#)

**Introduced in R2019b**

## slreportgen.report.DocBlock.getClassFolder

**Class:** slreportgen.report.DocBlock

**Package:** slreportgen.report

Get location of DocBlock reporter class definition file

### Syntax

```
path = slreportgen.report.DocBlock.getClassFolder()
```

### Description

`path = slreportgen.report.DocBlock.getClassFolder()` returns the path of the folder that contains the `slreportgen.report.DocBlock` class definition file.

### Output Arguments

**path** — Location of the DocBlock reporter class definition file

character vector

Location of the `slreportgen.report.DocBlock` class definition file, returned as a character vector.

### Examples

#### Get DocBlock Reporter Class Folder

Get the location of the folder that contains the DocBlock reporter class definition.

```
path = slreportgen.report.DocBlock.getClassFolder()
```

### See Also

`slreportgen.report.DocBlock` | `slreportgen.report.Report`

**Introduced in R2019b**

# slreportgen.report.ElementDiagram.createTemplate

**Class:** slreportgen.report.ElementDiagram

**Package:** slreportgen.report

Create element diagram template

## Syntax

```
template = slreportgen.report.ElementDiagram.createTemplate(templatePath,  
type)
```

## Description

`template = slreportgen.report.ElementDiagram.createTemplate(templatePath, type)` creates a copy of the default element diagram reporter template specified by `type` at the location specified by `templatePath`. Use the copied template as a starting point to design a custom diagram template for your report.

## Input Arguments

### templatePath — Location of reporter template

string | character vector | character array | template source object

Location of the reporter template, specified as a character vector, character array, or template source object.

### type — Type of template

'html' | 'html-file' | 'docx' | 'pdf'

Type of template, specified as 'html', 'html-file', 'docx', or 'pdf'.

## Output Arguments

### template — Template name

string

Name of template, returned as the path and file name of the template. The template file name extension is assigned based on the specified output type. For example, for PDF output, the template name has a .pdftx file extension.

## Examples

### Create an Element Diagram Template

```
import slreportgen.report.*  
rpt = Report('My Report', 'html');  
load_system('sf_car')
```

```
diagram = slreportgen.report.ElementDiagram('sf_car');  
template = slreportgen.report.ElementDiagram.createTemplate...  
    ('mytemplates\myElemDiagram','html');  
diagram.TemplateSrc = template;
```

**See Also**

`slreportgen.report.ElementDiagram` | `slreportgen.report.Report`

**Introduced in R2018b**



# slreportgen.report.ElementDiagram.customizeReporter

**Class:** slreportgen.report.ElementDiagram

**Package:** slreportgen.report

Create custom element diagram reporter class

## Syntax

```
customRptrPath = slreportgen.report.ElementDiagram.customizeReporter(
classpath)
```

## Description

`customRptrPath = slreportgen.report.ElementDiagram.customizeReporter(classpath)` creates an empty element diagram class definition file that is a subclass of `slreportgen.report.ElementDiagram`. The file is created at the specified `classpath` location. The `customizeReporter` method also copies the default element diagram templates to the `<classpath>/resources/template` folder. You can use the new class definition file as a starting point to design a custom element diagram class for your report.

## Input Arguments

**classpath — Location of custom element diagram class**

current working folder (default) | string | character array

Location of custom element diagram class, specified as a string or character array. The `classpath` argument also supports specifying a folder with `@` before the class name. For example, both of these are valid paths:

- `slreportgen.report.ElementDiagram.customizeReporter("path_folder/MyClassA.m")`
- `slreportgen.report.ElementDiagram.customizeReporter("+package/@MyClassB")`

## Output Arguments

**customRptrPath — Path of custom element diagram reporter**

string

Path of the custom element diagram reporter classdef file that defines the custom element diagram reporter, specified as a string.

## Examples

### Create Custom Element Diagram Reporter

Create a custom element diagram reporter and its associated default templates. The derived class file is created at the specified path relative to the current working folder. In this case, the path to the

MyElemDiagram.m class file is <current working folder>/newElemDiagram/@MyElemDiagram/MyElemDiagram.m. The default diagram templates are in the <current working folder>/newDiagram/@MyElemDiagram/resources/templates folder.

```
import slreportgen.report.*  
ElementDiagram.customizeReporter('newElemDiagram/@MyElemDiagram');
```

After editing this new class file and loading a model, you can use the new diagram reporter.

```
sf_car;  
diagram = MyElemDiagram('sf_car');
```

## **See Also**

slreportgen.report.Report

**Introduced in R2018b**

# slreportgen.report.ElementDiagram.getClassFolder

**Class:** slreportgen.report.ElementDiagram

**Package:** slreportgen.report

Element diagram class definition file location

## Syntax

```
path = slreportgen.report.ElementDiagram.getClassFolder()
```

## Description

`path = slreportgen.report.ElementDiagram.getClassFolder()` returns the path of the folder that contains the element diagram class definition file.

## Output Arguments

**path** — Element diagram class definition file location

character array

Element diagram class definition file location, returned as a character array.

## See Also

slreportgen.report.Diagram | slreportgen.report.ElementDiagram |  
slreportgen.report.Report

**Introduced in R2018b**

## getSnapshotImage

**Class:** slreportgen.report.ElementDiagram

**Package:** slreportgen.report

Element diagram snapshot image file location

### Syntax

```
path = getSnapshotImage(elemdiag,rpt)
```

### Description

`path = getSnapshotImage(elemdiag,rpt)` generates the image of the element diagram this reporter would generate if it were added to the report (`rpt`). This method returns the path of the generated image. Use this method to take snapshots of element diagrams without having to add the `ElementDiagram` reporter to a report. For example, you can use this method to set the `Image` property of a `TitlePage` reporter to a snapshot of an element diagram.

---

**Note** If you use this method, set the `ElementDiagram Scaling` property to `custom` or `zoom`. If you use `auto` scaling, the image does not scale to fit on the page.

---

### Input Arguments

**elemdiag — Element diagram reporter**

ElementDiagram object

Element diagram reporter, specified as an `ElementDiagram` class object. For example,

```
load_system('f14')
rpt = slreportgen.report.Report;
eldiag = slreportgen.report.ElementDiagram...
    ("f14/Aircraft Dynamics Model");
getSnapshotImage(eldiag,rpt);
```

**rpt — Report class object**

Report class

Report class object used to generate the diagram image

### Output Arguments

**path — Location of snapshot image file**

string

Location of snapshot image file, returned as a string. The location is a temporary folder that is deleted when the report is closed. To retain the folder, set the `Debug` property of `slreportgen.report.Report`.

## **See Also**

`slreportgen.report.ElementDiagram` | `slreportgen.report.Report`

**Introduced in R2018b**

# slreportgen.report.ExecutionOrder.createTemplate

**Class:** slreportgen.report.ExecutionOrder

**Package:** slreportgen.report

Create execution order reporter template

## Syntax

```
template = slreportgen.report.ExecutionOrder.createTemplate(templatePath,  
type)
```

## Description

`template = slreportgen.report.ExecutionOrder.createTemplate(templatePath, type)` creates a copy of the `slreportgen.report.ExecutionOrder` reporter template for the report type specified by `type` at the location specified by `templatePath`. You can use the copied template as a starting point to design a custom execution order reporter template for your report.

## Input Arguments

**templatePath — Path and file name of new template**

character vector | string scalar

Path and file name of the new template, specified as a character vector or string scalar.

**type — Type of template**

'html' | 'html-file' | 'docx' | 'pdf'

Type of template, specified as 'html', 'html-file', 'docx', or 'pdf'.

## Output Arguments

**template — Path and file name of template copy**

string scalar

Path and file name of the copy of the template, returned as a string scalar. The file name extension of the template is assigned based on the specified template type. For example, if the `type` argument is 'pdf', the file name extension is `.pdf.tx`.

## Examples

### Create an Execution Order Reporter Template

Create a copy of the HTML template for the `slreportgen.report.ExecutionOrder` reporter and save it with the name `myExecutionOrderTemplate` in the `mytemplates` folder.

```
template = slreportgen.report.ExecutionOrder.createTemplate...  
( 'mytemplates/myExecutionOrder', 'html' );
```

After you modify the template, you can use it by setting the `TemplateSrc` property of an `ExecutionOrder` reporter to the path of the template file.

### **See Also**

`slreportgen.report.ExecutionOrder`

**Introduced in R2020b**

# slreportgen.report.ExecutionOrder.customizeReporter

**Class:** slreportgen.report.ExecutionOrder

**Package:** slreportgen.report

Create custom execution order reporter class

## Syntax

```
reporter = slreportgen.report.ExecutionOrder.customizeReporter(classpath)
```

## Description

`reporter = slreportgen.report.ExecutionOrder.customizeReporter(classpath)` creates a class definition file that defines a subclass of `slreportgen.report.ExecutionOrder` at the location specified by `classpath`. This method also copies the default reporter templates to the `resources/templates` subfolder of the folder that contains the class definition file. You can use the class definition file as a starting point to design a custom execution order reporter class for your report.

## Input Arguments

**classpath — Path and name of new class definition file**

string scalar | character vector

Path and name of new class definition file, specified as a string scalar or character vector.

You can specify a relative path or an absolute path. For example, this code creates `MyClass.m` in the subfolder `myFolder` of the current folder.

```
slreportgen.report.ExecutionOrder.customizeReporter("myFolder/MyClass")
```

To create the reporter class in a class folder, precede the class name with the `@` character. Do not specify the `.m` extension. For example, this code creates `MyClass.m` in the subfolder `myFolder/@MyClass` in the current folder.

```
slreportgen.report.ExecutionOrder.customizeReporter("myFolder/@MyClass")
```

See “Folders Containing Class Definitions”.

To create the reporter class in a class package, precede the folder name with the `+` character. For example, this code creates an execution order reporter in the `myOrg` package folder in the current folder.

```
slreportgen.report.ExecutionOrder.customizeReporter("+myOrg/@MyClass");
```

## Output Arguments

**reporter — Path and file name of new execution order reporter class**

string scalar



Path and file name of the new execution order reporter class, returned as a string scalar.

## Examples

### Create Custom Execution Order Reporter

Create a custom execution order reporter, `MyExecutionOrder`, and its associated default templates in the subfolder `MyFolder` of the current working folder.

```
slreportgen.report.ExecutionOrder.customizeReporter('MyFolder/MyExecutionOrder')
```

```
ans =
```

```
    "MyFolder\MyExecutionOrder.m"
```

### See Also

`slreportgen.report.ExecutionOrder`

**Introduced in R2020b**

## **slreportgen.report.ExecutionOrder.getClassFolder**

**Class:** slreportgen.report.ExecutionOrder

**Package:** slreportgen.report

Get location of execution order reporter class definition file

### **Syntax**

```
path = slreportgen.report.ExecutionOrder.getClassFolder()
```

### **Description**

`path = slreportgen.report.ExecutionOrder.getClassFolder()` returns the path of the folder that contains the `slreportgen.report.ExecutionOrder` class definition file.

### **Output Arguments**

**path** — Location of the execution order reporter class definition file

character vector

Location of the `slreportgen.report.ExecutionOrder` class definition file, returned as a character vector.

### **Examples**

#### **Get Execution Order Reporter Class Folder**

Get the folder that contains the `slreportgen.report.ExecutionOrder` reporter class definition.

```
path = slreportgen.report.ExecutionOrder.getClassFolder();
```

### **See Also**

`slreportgen.report.ExecutionOrder`

**Introduced in R2020b**

# slreportgen.report.LookupTable.createTemplate

**Class:** slreportgen.report.LookupTable

**Package:** slreportgen.report

Create Simulink lookup table block reporter template

## Syntax

```
template = slreportgen.report.LookupTable.createTemplate(templatePath,type)
```

## Description

`template = slreportgen.report.LookupTable.createTemplate(templatePath,type)` creates a copy of the LookupTable reporter template specified by `type` at the `templatePath` location. You can use the copied template as a starting point to design a custom LookupTable reporter template for your report.

## Input Arguments

**templatePath — Path and file name of new template**

character vector | string scalar

Path and file name of the new template, specified as a character vector or string scalar.

**type — Type of template**

'html' | 'html-file' | 'docx' | 'pdf'

Type of template, specified as 'html', 'html-file', 'docx', or 'pdf'.

## Output Arguments

**template — Path and file name of template copy**

string scalar

Path and file name of the copy of the template, returned as a string scalar. The file name extension of the template is assigned based on the specified template type. For example, if the `type` argument is 'pdf', the file name extension is `.pdftx`.

## Examples

### Create LookupTable Reporter Template

Before you run this example, create a copy of the default HTML LookupTable template and save it in a `mytemplates` folder. Name the copied template `myLUtable.htm`. Edit the template as desired. To use the new template for the lookup table, assign its path to the `TemplateSrc` property of `slreportgen.report.LookupTable`.

```
import mlreportgen.report.*
import slreportgen.report.*
```

```
rpt = Report('My Report', 'html');  
lutable = LookupTable();  
template = LookupTable.createTemplate('mytemplates\myLUTable', 'html');  
lutable.TemplateSrc = template;
```

## See Also

slreportgen.report.LookupTable | slreportgen.report.Report

## Topics

“Modify Styles in a Microsoft Word Template”

“Modify Styles in HTML Templates”

“Modify Styles in PDF Templates”

## Introduced in R2018a

# slreportgen.report.LookupTable.customizeReporter

**Class:** slreportgen.report.LookupTable

**Package:** slreportgen.report

Create custom LookupTable reporter class

## Syntax

```
reporter = slreportgen.report.LookupTable.customizeReporter(classpath)
```

## Description

`reporter = slreportgen.report.LookupTable.customizeReporter(classpath)` creates a LookupTable reporter class definition file that is a subclass of `slreportgen.report.LookupTable`. The file is created at the specified `classpath` location. The `LookupTable.customizeReporter` method also copies the default LookupTable reporter templates to the `<classpath>/resources/template` folder. You can use the class definition file as a starting point to design a custom LookupTable reporter class for your report.

## Input Arguments

**classpath — Location of custom lookup table reporter class**

current working folder (default) | string | character array

Location of custom lookup table reporter class, specified as a string or character array. The `classpath` argument also supports specifying a folder with `@` before the class name.

## Output Arguments

**reporter — Lookup table reporter path**

string

Lookup table reporter path, returned as the string specifying the path to the derived report class file.

## Examples

### Create Custom Lookup Table Reporter

Create a custom lookup table reporter and its associated default templates. The derived class file is created at the specified path relative to the current working folder. In this case, the path to the `myLUTable.m` class file is `<current working folder>/newLUTable/@myLUTable/myLUTable.m`. The default lookup table reporter templates are in the `<current working folder>/newLUTable/@myLUTable/resources/templates` folder.

```
import mlreportgen.report.*
import slreportgen.report.*
LookupTable.customizeReporter('newLUTable/@myLUTable');
```

After editing this new class file, you can use it as your `LookupTable` reporter.

```
lutable = myLUTable();
```

**See Also**

`slreportgen.report.LookupTable` | `slreportgen.report.Report`

**Introduced in R2018a**

# slreportgen.report.LookupTable.getClassFolder

**Class:** slreportgen.report.LookupTable

**Package:** slreportgen.report

Lookup Table reporter class definition file location

## Syntax

```
path = slreportgen.report.LookupTable.getClassFolder()
```

## Description

`path = slreportgen.report.LookupTable.getClassFolder()` returns the path of the folder that contains the LookupTable class definition file.

## Output Arguments

**path — Lookup Table class definition file location**

character array

Lookup Table class definition file location, returned as a character array.

## See Also

slreportgen.report.LookupTable | slreportgen.report.Report

**Introduced in R2018a**

# slreportgen.report.MATLABFunction.createTemplate

**Class:** slreportgen.report.MATLABFunction

**Package:** slreportgen.report

Create MATLAB Function reporter template

## Syntax

```
template = slreportgen.report.MATLABFunction.createTemplate(templatePath,  
type)
```

## Description

`template = slreportgen.report.MATLABFunction.createTemplate(templatePath, type)` creates a copy of the MATLAB Function reporter template specified by `type` at the `templatePath` location. You can use the copied template as a starting point to design a custom MATLAB Function template for your report.

## Input Arguments

**templatePath** — Path and file name of new template

character vector | string scalar

Path and file name of the new template, specified as a character vector or string scalar.

**type** — Type of template

'html' | 'html-file' | 'docx' | 'pdf'

Type of template, specified as 'html', 'html-file', 'docx', or 'pdf'.

## Output Arguments

**template** — Path and file name of template copy

string scalar

Path and file name of the copy of the template, returned as a string scalar. The file name extension of the template is assigned based on the specified template type. For example, if the `type` argument is 'pdf', the file name extension is `.pdf.tx`.

## Examples

### Create MATLAB Function Reporter Template

Create a copy of the HTML template for the MATLAB Function reporter.

```
template = slreportgen.report.MATLABFunction.createTemplate...  
( 'mytemplates\myMFunction', 'html' );
```



After you modify the template, use it by setting the `TemplateSrc` property of the reporter.

```
rptr = slreportgen.report.MATLABFunction;  
rptr.TemplateSrc = template;
```

## See Also

`slreportgen.report.MATLABFunction` | `slreportgen.report.Report`

## Topics

“Modify Styles in a Microsoft Word Template”

“Modify Styles in HTML Templates”

“Modify Styles in PDF Templates”

## Introduced in R2018a

# slreportgen.report.MATLABFunction.customizeReporter

**Class:** slreportgen.report.MATLABFunction

**Package:** slreportgen.report

Create custom MATLAB Function reporter class

## Syntax

```
reporter = slreportgen.report.MATLABFunction.customizeReporter(classpath)
```

## Description

`reporter = slreportgen.report.MATLABFunction.customizeReporter(classpath)` creates a MATLAB Function reporter class definition file that is a subclass of `slreportgen.report.MATLABFunction`. The file is created at the specified `classpath` location. The `MATLABFunction.customizeReporter` method also copies the default MATLAB Function reporter templates to the `<classpath>/resources/template` folder. You can use the class definition file as a starting point to design a custom MATLAB Function reporter class for your report.

## Input Arguments

**classpath** — Location of custom MATLAB Function reporter class

current working folder (default) | string | character array

Location of custom MATLAB Function reporter class, specified as a string or character array. The `classpath` argument also supports specifying a folder with `@` before the class name.

## Output Arguments

**reporter** — MATLAB Function reporter path

string

MATLAB Function reporter path, returned as the string specifying the path to the derived report class file.

## Examples

### Create Custom MATLAB Function Reporter

Create a custom MATLAB Function reporter and its associated default templates. The derived class file is created at the specified path relative to the current working folder. In this case, the path to the `myMFunction.m` class file is `<current working folder>/newMFunction/@myMFunction/myMFunction.m`. The default MATLAB Function reporter templates are in the `<current working folder>/newMFunction/@myMFunction/resources/templates` folder.

```
import mlreportgen.report.*
import slreportgen.report.*
MATLABFunction.customizeReporter('newMFunction/@myMFunction');
```

After editing this new class file, you can use it as your MATLAB Function reporter.

```
mfunction = myMFunction();
```

## See Also

[slreportgen.report.MATLABFunction](#) | [slreportgen.report.Report](#)

**Introduced in R2018a**

## **slreportgen.report.MATLABFunction.getClassFolder**

**Class:** `slreportgen.report.MATLABFunction`

**Package:** `slreportgen.report`

MATLAB Function reporter class definition file location

### **Syntax**

```
path = slreportgen.report.MATLABFunction.getClassFolder()
```

### **Description**

`path = slreportgen.report.MATLABFunction.getClassFolder()` returns the path of the folder that contains the MATLAB Function class definition file.

### **Output Arguments**

**path** — **MATLAB Function class definition file location**

character array

MATLAB Function class definition file location, returned as a character array.

### **See Also**

`slreportgen.report.MATLABFunction` | `slreportgen.report.Report`

**Introduced in R2018a**

# slreportgen.report.ModelConfiguration.createTemplate

**Class:** slreportgen.report.ModelConfiguration

**Package:** slreportgen.report

Create model configuration reporter template

## Syntax

```
template = slreportgen.report.ModelConfiguration.createTemplate(templatePath,  
type)
```

## Description

`template = slreportgen.report.ModelConfiguration.createTemplate(templatePath, type)` creates a copy of the `slreportgen.report.ModelConfiguration` reporter template for the report type specified by `type` at the location specified by `templatePath`. You can use the copied template as a starting point to design a custom model configuration reporter template for your report.

## Input Arguments

**templatePath — Path and file name of new template**

character vector | string scalar

Path and file name of the new template, specified as a character vector or string scalar.

**type — Type of template**

'html' | 'html-file' | 'docx' | 'pdf'

Type of template, specified as 'html', 'html-file', 'docx', or 'pdf'.

## Output Arguments

**template — Path and file name of template copy**

string scalar

Path and file name of the copy of the template, returned as a string scalar. The file name extension of the template is assigned based on the specified template type. For example, if the `type` argument is 'pdf', the file name extension is `.pdf.tx`.

## Examples

### Create a Simulink Model Configuration Reporter Template

Create a copy of the HTML template for the `slreportgen.report.ModelConfiguration` reporter and save it with the name `myModelConfigurationTemplate` in the `mytemplates` folder.

```
template = slreportgen.report.ModelConfiguration.createTemplate...  
    ('mytemplates/myModelConfigurationTemplate', 'html');
```

After you modify the template, you can use it by setting the `TemplateSrc` property of a model configuration reporter to the path of the template file.

### **See Also**

`slreportgen.report.ModelConfiguration`

**Introduced in R2020b**

# slreportgen.report.ModelConfiguration.customizeReporter

**Class:** slreportgen.report.ModelConfiguration

**Package:** slreportgen.report

Create custom model configuration reporter class

## Syntax

```
reporter = slreportgen.report.ModelConfiguration.customizeReporter(classpath)
```

## Description

`reporter = slreportgen.report.ModelConfiguration.customizeReporter(classpath)` creates a class definition file that defines a subclass of `slreportgen.report.ModelConfiguration` at the location specified by `classpath`. This method also copies the default reporter templates to the `resources/templates` subfolder of the folder that contains the class definition file. You can use the class definition file as a starting point to design a custom model configuration reporter class for your report.

## Input Arguments

**classpath — Path and name of new class definition file**

string scalar | character vector

Path and name of new class definition file, specified as a string scalar or character vector.

You can specify a relative path or an absolute path. For example, this code creates `MyClass.m` in the subfolder `myFolder` of the current folder.

```
slreportgen.report.ModelConfiguration.customizeReporter("myFolder/MyClass")
```

To create the reporter class in a class folder, precede the class name with the `@` character. Do not specify the `.m` extension. For example, this code creates `MyClass.m` in the subfolder `myFolder/@MyClass` in the current folder.

```
slreportgen.report.ModelConfiguration.customizeReporter("myFolder/@MyClass")
```

See “Folders Containing Class Definitions”.

To create the reporter class in a class package, precede the folder name with the `+` character. For example, this code creates a model configuration reporter in the `myOrg` package folder in the current folder.

```
slreportgen.report.ModelConfiguration.customizeReporter("+myOrg/@MyClass");
```

## Output Arguments

**reporter — Path and file name of new model configuration reporter class**

string scalar

Path and file name of the new model configuration reporter class, returned as a string scalar.

## Examples

### Create Custom Model Configuration Reporter

Create a custom model configuration reporter, `MyModelConfiguration`, and its associated default templates in the subfolder `MyFolder` of the current working folder.

```
slreportgen.report.ModelConfiguration.customizeReporter('MyFolder/MyModelConfiguration')
```

```
ans =
```

```
    "MyFolder\MyModelConfiguration.m"
```

### See Also

`slreportgen.report.ModelConfiguration`

**Introduced in R2020b**



# slreportgen.report.ModelConfiguration.getClassFolder

**Class:** slreportgen.report.ModelConfiguration

**Package:** slreportgen.report

Get location of model configuration reporter class definition file

## Syntax

```
path = slreportgen.report.ModelConfiguration.getClassFolder()
```

## Description

path = slreportgen.report.ModelConfiguration.getClassFolder() returns the path of the folder that contains the slreportgen.report.ModelConfiguration class definition file.

## Output Arguments

**path** — Location of the model configuration reporter class definition file

character vector

Location of the slreportgen.report.ModelConfiguration class definition file, returned as a character vector.

## Examples

### Get Model Configuration Reporter Class Folder

Get the folder that contains the model configuration reporter class definition.

```
path = slreportgen.report.ModelConfiguration.getClassFolder()
```

## See Also

slreportgen.report.ModelConfiguration

**Introduced in R2020b**

## getConfigSet

**Class:** `slreportgen.report.ModelConfiguration`

**Package:** `slreportgen.report`

Get active configuration set from model configuration reporter

### Syntax

```
configSetObj = getConfigSet(reporter)
```

### Description

`configSetObj = getConfigSet(reporter)` returns the `Simulink.ConfigSet` object that represents the active model configuration set to be reported by the specified `slreportgen.report.ModelConfiguration` reporter.

### Input Arguments

**reporter** — Model configuration reporter

`slreportgen.report.ModelConfiguration` object

Model configuration reporter, specified as an `slreportgen.report.ModelConfiguration` object.

### Output Arguments

**configSetObj** — Model configuration set object

`Simulink.ConfigSet` object

Model configuration set object, specified as a `Simulink.ConfigSet` object.

### Examples

#### Get Active Configuration Set to be Reported

Create an `slreportgen.report.ModelConfiguration` reporter for the `sf_car` model and then get the active configuration set object associated with the reporter.

```
model = "sf_car";  
load_system(model);  
reporter = slreportgen.report.ModelConfiguration(model);  
configSet = getConfigSet(reporter)
```

```
configSet =
```

```
    Simulink.ConfigSet
```

### See Also

`Simulink.ConfigSet` | `slreportgen.report.ModelConfiguration`

**Introduced in R2020b**

# slreportgen.report.ModelVariable.createTemplate

**Class:** slreportgen.report.ModelVariable

**Package:** slreportgen.report

Create model variable reporter template

## Syntax

```
template = slreportgen.report.ModelVariable.createTemplate(templatePath,type)
```

## Description

`template = slreportgen.report.ModelVariable.createTemplate(templatePath,type)` creates a copy of the `slreportgen.report.ModelVariable` reporter template for the report type specified by `type` at the location specified by `templatePath`. You can use the copied template as a starting point to design a custom model variable reporter template for your report.

## Input Arguments

**templatePath — Path and file name of new template**

character vector | string scalar

Path and file name of the new template, specified as a character vector or string scalar.

**type — Type of template**

'html' | 'html-file' | 'docx' | 'pdf'

Type of template, specified as 'html', 'html-file', 'docx', or 'pdf'.

## Output Arguments

**template — Path and file name of template copy**

string scalar

Path and file name of the copy of the template, returned as a string scalar. The file name extension of the template is assigned based on the specified template type. For example, if the `type` argument is 'pdf', the file name extension is `.pdftx`.

## Examples

### Create a Simulink Model Variable Reporter Template

Create a copy of the HTML template for the `slreportgen.report.ModelVariable` reporter and save it with the name `myModelVariableTemplate` in the `mytemplates` folder.

```
template = slreportgen.report.ModelVariable.createTemplate...  
('mytemplates/myModelVariableTemplate','html');
```

After you modify the template, you can use it by setting the `TemplateSrc` property of the model variable reporter.

## See Also

`slreportgen.finder.ModelVariableFinder` | `slreportgen.finder.ModelVariableResult`  
| `slreportgen.report.ModelVariable`

## Topics

“Report Generation for Simulink and Stateflow Elements” on page 1-9  
“What Is a Reporter?”

## Introduced in R2019b

# slreportgen.report.ModelVariable.customizeReporter

**Class:** slreportgen.report.ModelVariable

**Package:** slreportgen.report

Create custom model variable reporter class

## Syntax

```
reporter = slreportgen.report.ModelVariable.customizeReporter(classpath)
```

## Description

`reporter = slreportgen.report.ModelVariable.customizeReporter(classpath)` creates a class definition file that defines a subclass of `slreportgen.report.ModelVariable` at the location specified by `classpath`. This method also copies the default reporter templates to the `resources/templates` subfolder of the folder that contains the class definition file. You can use the class definition file as a starting point to design a custom model variable reporter class for your report.

## Input Arguments

**classpath — Path and name of new class definition file**

string scalar | character vector

Path and name of new class definition file, specified as a string scalar or character vector.

You can specify a relative path or an absolute path. For example, this code creates `MyClass.m` in the subfolder `myFolder` of the current folder.

```
slreportgen.report.ModelVariable.customizeReporter("myFolder/MyClass")
```

To create the reporter class in a class folder, precede the class name with the `@` character. Do not specify the `.m` extension. For example, this code creates `MyClass.m` in the subfolder `myFolder/@MyClass` in the current folder.

```
slreportgen.report.ModelVariable.customizeReporter("myFolder/@MyClass")
```

See “Folders Containing Class Definitions”.

To create the reporter class in a class package, precede the folder name with the `+` character. For example, this code creates a model variable reporter in the `myOrg` package folder in the current folder.

```
slreportgen.report.ModelVariable.customizeReporter("+myOrg/@MyClass");
```

## Output Arguments

**reporter — Path and file name of new model variable reporter class**

string scalar

Path and file name of the new model variable reporter class, returned as a string scalar.

## Examples

### Create Custom Model Variable Reporter

Create a custom model variable reporter, `MyModelVariable`, and its associated default templates in the subfolder `MyFolder` of the current working folder.

```
slreportgen.report.ModelVariable.customizeReporter('MyFolder/MyModelVariable')
```

```
ans =
```

```
    "MyFolder\MyModelVariable.m"
```

### See Also

[Simulink.VariableUsage](#) | [slreportgen.finder.ModelVariableFinder](#) | [slreportgen.finder.ModelVariableResult](#)

### Topics

“Report Generation for Simulink and Stateflow Elements” on page 1-9

“What Is a Reporter?”

### Introduced in R2019b

## slreportgen.report.ModelVariable.getClassFolder

**Class:** slreportgen.report.ModelVariable

**Package:** slreportgen.report

Get location of model variable reporter class definition file

### Syntax

```
path = slreportgen.report.ModelVariable.getClassFolder()
```

### Description

`path = slreportgen.report.ModelVariable.getClassFolder()` returns the path of the folder that contains the `slreportgen.report.ModelVariable` class definition file.

### Output Arguments

**path** — Location of the model variable reporter class definition file

character vector

Location of the `slreportgen.report.ModelVariable` class definition file, returned as a character vector.

### Examples

#### Get Model Variable Reporter Class Folder

Get the folder that contains the model variable reporter class definition.

```
path = slreportgen.report.ModelVariable.getClassFolder()
```

### See Also

[Simulink.VariableUsage](#) | [slreportgen.finder.ModelVariableResult](#) | [slreportgen.finder.ModelVariableResult](#)

### Topics

“Report Generation for Simulink and Stateflow Elements” on page 1-9

“What Is a Reporter?”

**Introduced in R2019b**



# getVariableName

**Class:** slreportgen.report.ModelVariable

**Package:** slreportgen.report

Get name of variable from model variable reporter

## Syntax

```
name = getVariableName(reporter)
```

## Description

`name = getVariableName(reporter)` returns the name of the model variable to be reported by the specified model variable reporter.

## Input Arguments

**reporter** — Model variable reporter

slreportgen.report.Modelvariable object

Model variable reporter, specified as an slreportgen.report.ModelVariable object.

## Output Arguments

**name** — Name of model variable

character vector

Name of model variable, returned as a character vector.

## Examples

### Get Name of Model Variable from Reporter

After you get the reporter for a model variable result, you can use the `getVariableName` method to get the variable name from the reporter.

```
...
finder = slreportgen.finder.ModelVariableFinder(model);

while hasNext(finder)
    result = next(finder);

    % Get the ModelVariable reporter for the result
    % Get the variable name
    reporter = getReporter(result);
    name = getVariableName(reporter);
    ...
    % Add the reporter to the chapter
    add(chapter, reporter);
```

end  
...

### See Also

`slreportgen.finder.ModelVariableFinder` | `slreportgen.finder.ModelVariableResult`  
| `slreportgen.report.ModelVariable`

### Topics

“Report Generation for Simulink and Stateflow Elements” on page 1-9  
“What Is a Reporter?”

**Introduced in R2019b**

# getVariableValue

**Class:** slreportgen.report.ModelVariable

**Package:** slreportgen.report

Get value of variable from model variable reporter

## Syntax

```
value = getVariableValue(reporter)
```

## Description

`value = getVariableValue(reporter)` returns the value of the model variable to be reported by the specified model variable reporter.

## Input Arguments

**reporter** — Model variable reporter

slreportgen.report.Modelvariable object

Model variable reporter, specified as an slreportgen.report.ModelVariable.

## Examples

### Get Value of Model Variable from Model Variable Reporter

After you get the reporter for a model variable result, you can use the `getVariableValue` method to get the variable value from the reporter.

```
...
finder = slreportgen.finder.ModelVariableFinder(model);

while hasNext(finder)
    result = next(finder);

    % Get the ModelVariable reporter for the result
    % Get the variable value
    reporter = getReporter(result);
    value = getVariableValue(reporter);
    ...
    % Add the reporter to the chapter
    add(chapter, reporter);
end
...
```

## See Also

slreportgen.finder.ModelVariableFinder | slreportgen.finder.ModelVariableResult  
| slreportgen.report.ModelVariable

**Topics**

“Report Generation for Simulink and Stateflow Elements” on page 1-9  
“What Is a Reporter?”

**Introduced in R2019b**

# slreportgen.report.Notes.createTemplate

**Class:** slreportgen.report.Notes

**Package:** slreportgen.report

Copy default slreportgen.report.Notes reporter template

## Syntax

```
template = slreportgen.report.Notes.createTemplate(templatePath,type)
```

## Description

`template = slreportgen.report.Notes.createTemplate(templatePath,type)` creates a copy of the `slreportgen.report.Notes` reporter template for the report type specified by `type` at the location specified by `templatePath`. You can use the copy of the template as a starting point to design a custom `slreportgen.report.Notes` template for your report.

## Input Arguments

**templatePath — Path and file name of new template**

character vector | string scalar

Path and file name of the new template, specified as a character vector or string scalar.

**type — Type of template**

'html' | 'html-file' | 'docx' | 'pdf'

Type of template, specified as 'html', 'html-file', 'docx', or 'pdf'.

## Output Arguments

**template — Path and file name of template copy**

string scalar

Path and file name of the copy of the template, returned as a string scalar. The file name extension of the template is assigned based on the specified output type. For example, if `type` is 'pdf', the file name extension is `.pdftx`.

## Examples

### Create a Notes Reporter Template

Create a copy of the HTML template for the `slreportgen.report.Notes` reporter and save it with the name `myNotesTemplate` in the `mytemplates` folder.

```
template = slreportgen.report.Notes.createTemplate...  
('mytemplates/myNotesTemplate','html');
```

Modify the template copy and then set the `TemplateSrc` property of the `Notes` reporter to the template copy.

**See Also**

`slreportgen.report.Notes` | `slreportgen.report.Report`

**Introduced in R2020a**

# slreportgen.report.RptFile.createTemplate

**Class:** slreportgen.report.RptFile

**Package:** slreportgen.report

Create Report Explorer-based (RptFile) reporter template

## Syntax

```
template = slreportgen.report.RptFile.createTemplate(templatePath,type)
```

## Description

`template = slreportgen.report.RptFile.createTemplate(templatePath,type)` creates a copy of the default Report Explorer-based reporter (RptFile) template specified by `type` at the `templatePath` location . You can use the copied template as a starting point to design a custom RptFile reporter template for your report.

## Input Arguments

**templatePath — Path and file name of new template**

character vector | string scalar

Path and file name of the new template, specified as a character vector or string scalar.

**type — Type of template**

'html' | 'html-file' | 'docx' | 'pdf'

Type of template, specified as 'html', 'html-file', 'docx', or 'pdf'.

## Output Arguments

**template — Path and file name of template copy**

string scalar

Path and file name of the copy of the template, returned as a string scalar. The file name extension of the template is assigned based on the specified template type. For example, if the `type` argument is 'pdf', the file name extension is `.pdftx`.

## Examples

### Create Report Explorer-based Reporter Template

Copy the template file of the desired output type. In this example the copied template file is named `myrptfile.htm` and is saved in a folder named `mytemplates`. To use the new template for the RptFile reporter, assign its path to the RptFile `TemplateSrc` property.

```
template = RptFile.createTemplate('mytemplates\myrptfile','html');  
rptfile.TemplateSrc = template;
```

## **See Also**

`slreportgen.report.Report` | `slreportgen.report.RptFile`

## **Topics**

“Modify Styles in a Microsoft Word Template”

“Modify Styles in HTML Templates”

“Modify Styles in PDF Templates”

**Introduced in R2019a**



# slreportgen.report.Notes.customizeReporter

**Class:** slreportgen.report.Notes

**Package:** slreportgen.report

Create subclass of slreportgen.report.Notes class

## Syntax

```
reporter = slreportgen.report.Notes.customizeReporter(classpath)
```

## Description

`reporter = slreportgen.report.Notes.customizeReporter(classpath)` creates a reporter class definition file that defines a subclass of `slreportgen.report.Notes` at the location specified by `classpath`. The method also copies the default reporter templates to the `resources/templates` subfolder of the folder that contains the class definition file. You can use the class definition file as a starting point to design a custom notes reporter class for your report.

## Input Arguments

**classpath — Path and name of new class definition file**

string scalar | character vector

Path and name of new class definition file, specified as a string scalar or character vector.

You can specify a relative path or an absolute path. For example, this code creates `MyClass.m` in the subfolder `myFolder` of the current folder.

```
slreportgen.report.Notes.customizeReporter("myFolder/MyClass")
```

To create the reporter class in a class folder, precede the class name with the `@` character. Do not specify the `.m` extension. For example, this code creates `MyClass.m` in the subfolder `myFolder/@MyClass` in the current folder.

```
slreportgen.report.Notes.customizeReporter("myFolder/@MyClass")
```

See “Folders Containing Class Definitions”.

To create the reporter class in a class package, precede the folder name with the `+` character. For example, this code creates a notes reporter in the `myOrg` package folder in the current folder.

```
slreportgen.report.Notes.customizeReporter("+myOrg/@MyClass")
```

## Output Arguments

**reporter — Path and file name of new reporter class**

string scalar

Path and file name of the new reporter class, returned as a string scalar.

## Examples

### Create a Custom Notes Reporter

Create a custom notes reporter, `myNotes`, and the associated default templates in the subfolder `MyFolder` of the current working folder.

```
slreportgen.report.Notes.customizeReporter("MyFolder/myNotes")
```

```
ans =
```

```
    "MyFolder\myNotes.m"
```

### See Also

`slreportgen.report.Notes` | `slreportgen.report.Report`

**Introduced in R2020a**

# slreportgen.report.Notes.getClassFolder

**Class:** slreportgen.report.Notes

**Package:** slreportgen.report

Get location of folder that contains slreportgen.report.Notes reporter class definition file

## Syntax

```
path = slreportgen.report.Notes.getClassFolder()
```

## Description

path = slreportgen.report.Notes.getClassFolder() returns the path of the folder that contains the slreportgen.report.Notes class definition file.

## Output Arguments

**path — slreportgen.report.Notes class definition file location**

character vector

slreportgen.report.Notes class definition file location, returned as a character vector.

## Examples

### Get Notes Reporter Class Folder

Get the location of the folder that contains the notes reporter class definition.

```
path = slreportgen.report.Notes.getClassFolder()
```

## See Also

slreportgen.report.Notes | slreportgen.report.Report

**Introduced in R2020a**

## slreportgen.report.RptFile.customizeReporter

**Class:** slreportgen.report.RptFile

**Package:** slreportgen.report

Create custom Report Explorer-based reporter class

### Syntax

```
reporter = slreportgen.report.RptFile.customizeReporter(classpath)
```

### Description

`reporter = slreportgen.report.RptFile.customizeReporter(classpath)` creates a Report Explorer-based reporter (`RptFile`) class definition file that is a subclass of `slreportgen.report.RptFile`. The file is created at the specified `classpath` location. The `RptFile.customizeReporter` method also copies the default `RptFile` templates to the `<classpath>/resources/template` folder. You can use the new class definition file as a starting point to design a custom Report Explorer-based reporter class for your report.

### Input Arguments

**classpath — Location of custom Report Explorer-based reporter class**

current working folder (default) | string | character array

Location of custom Report Explorer-based reporter class, specified as a string or character array. The `classpath` argument also supports specifying a folder with `@` before the class name.

### Output Arguments

**reporter — Report Explorer-based reporter path**

string

Report Explorer-based reporter path, returned as a string specifying the path to the derived report class file.

### Examples

#### Create Custom Report Explorer-based Reporter

Create a custom Report Explorer-based reporter and its associated default templates. In this example, the derived class file is created at the specified path under the current working folder. In this example, the path to the `MyRptExplRptr.m` class file is `<current working folder>/new_rptexpl_rptr/@MyRptExplRptr/MyRptExplRptr.m`. The default `RptFile` templates are in the `<current working folder>/new_rptexpl_rptr/@RptExplRptr/resources/templates` folder.

```
import slreportgen.report.*
RptFile.customizeReporter('new_rptexpl_rptr/@MyRptExplRptr');
```

After editing this new class file, you can use it as your RptFile reporter.

```
rptr = MyRptExplRptr();
```

### **See Also**

[slreportgen.report.Report](#) | [slreportgen.report.RptFile](#)

**Introduced in R2019a**

## **slreportgen.report.RptFile.getClassFolder**

**Class:** slreportgen.report.RptFile

**Package:** slreportgen.report

Report Explorer-based reporter class definition file location

### **Syntax**

```
path = slreportgen.report.RptFile.getClassFolder()
```

### **Description**

`path = slreportgen.report.RptFile.getClassFolder()` returns the path of the folder that contains the Report Explorer-based reporter class definition file.

### **Output Arguments**

**path — Report Explorer-based reporter class definition file location**

character array

Report Explorer-based reporter class definition file location, returned as a character array.

### **See Also**

slreportgen.report.Report | slreportgen.report.RptFile

**Introduced in R2019a**

# slreportgen.report.SimulinkObjectProperties.createTemplate

**Class:** slreportgen.report.SimulinkObjectProperties

**Package:** slreportgen.report

Create Simulink object properties reporter template

## Syntax

```
template = slreportgen.report.SimulinkObjectProperties.createTemplate(  
templatePath,type)  
output_args = createTemplate(input_args,Name,Value)
```

## Description

`template = slreportgen.report.SimulinkObjectProperties.createTemplate(templatePath,type)` creates a copy of the default Simulink object properties template specified by `type` at the location specified by `templatePath`. To design a custom Simulink object properties template for your report, use the copied template as a starting point.

`output_args = createTemplate(input_args,Name,Value)` <verb phase> with additional options specified by one or more `Name,Value` pair arguments.

## Input Arguments

### **templatePath** — Path and file name of new template

character vector | string scalar

Path and file name of the new template, specified as a character vector or string scalar.

### **type** — Type of template

'html' | 'html-file' | 'docx' | 'pdf'

Type of template, specified as 'html', 'html-file', 'docx', or 'pdf'.

## Output Arguments

### **template** — Path and file name of template copy

string scalar

Path and file name of the copy of the template, returned as a string scalar. The file name extension of the template is assigned based on the specified template type. For example, if the `type` argument is 'pdf', the file name extension is `.pdf.tx`.

## Examples

### Create a Report Template

Before you run this example, create a copy of the default HTML Simulink object properties template in the `mytemplates` folder. Name the copied template `myObjectsReporter.htm`. Edit the template as desired. To use the new template, assign its path to the `slreportgen.report.SimulinkObjectProperties.TemplateSrc` property.

```
import slreportgen.report.*
import mlreportgen.report.*
rpt = slreportgen.report.Report('My Report','html');
model_name = 'vdp');
load_system(model_name);

chapter = Chapter(model_name);
chart = block2chart('sf_car/shift_logic');
rptr = SimulinkObjectProperties(model_name);
template = SimulinkObjectProperties.createTemplate...
    ('mytemplates\myObjsReporter','html');
rptr.TemplateSrc = template;
```

### See Also

`slreportgen.report.SimulinkObjectProperties`

### Topics

“Modify Styles in a Microsoft Word Template”  
“Modify Styles in HTML Templates”  
“Modify Styles in PDF Templates”

### Introduced in R2017b



# slreportgen.report.StateflowObjectProperties.customizeReporter

**Class:** slreportgen.report.StateflowObjectProperties

**Package:** slreportgen.report

Create custom Simulink object properties class

## Syntax

```
reporter = slreportgen.report.SimulinkObjectProperties.customizeReporter(
classpath)
```

## Description

reporter = slreportgen.report.SimulinkObjectProperties.customizeReporter(classpath) creates a Simulink object properties page class definition file that is a subclass of slreportgen.report.SimulinkObjectProperties. The file is created at the specified classpath location. The SimulinkObjectProperties.customizeReporter method also copies the default Simulink object properties templates to the <classpath>/resources/template folder. To design a custom Simulink object properties class for your report, use the new class definition file as a starting point.

## Input Arguments

**classpath — Location of custom Simulink object properties class**

current working folder (default) | string | character array

Location of custom Simulink object properties class, specified as a string or character array. The classpath argument also supports specifying a folder with @ before the class name.

## Output Arguments

**reporter — Simulink object properties reporter path**

string

Simulink object properties reporter path, returned as the string specifying the path to the derived report class file.

Simulink object properties

## Examples

### Create Custom Simulink Object Properties Reporter

Create a custom Simulink object properties reporter and its associated default templates. The derived class file is created at the specified path relative to the current working folder. In this case, the path to the MySFObjProps.m class file is <current working folder>/newSFObjProps/

@MySF0bjProps/MySF0bjProps.m. The default title page templates are in the <current working folder>/newTitlePage/@MySF0bjProps/resources/templates folder.

```
import slreportgen.report.*
StateflowObjectProperties.customizeReporter...
    ('newSF0bjProps/@MySF0bjProps');
```

After editing this new class file, you can use it as your Simulink object properties reporter.

```
objprop = MySF0bjProps();
```

## **See Also**

slreportgen.report.Report | slreportgen.report.SimulinkObjectProperties

**Introduced in R2017b**

# slreportgen.report.SimulinkObjectProperties.getClassFolder

**Class:** slreportgen.report.SimulinkObjectProperties

**Package:** slreportgen.report

Simulink object properties class definition file location

## Syntax

```
path =  
getClassFolderslreportgen.report.SimulinkObjectProperties.getClassFolder()
```

## Description

```
path =  
getClassFolderslreportgen.report.SimulinkObjectProperties.getClassFolder()  
returns the path of the folder that contains the Simulink object properties class definition file.
```

## Output Arguments

**path** — Simulink object properties class definition file location

character array

Simulink object properties class definition file location, returned as a character array.

## See Also

slreportgen.report.Report | slreportgen.report.SimulinkObjectProperties

**Introduced in R2017b**

# slreportgen.report.StateflowObjectProperties.createTemplate

**Class:** slreportgen.report.StateflowObjectProperties

**Package:** slreportgen.report

Create Stateflow object properties reporter template

## Syntax

```
template = slreportgen.report.StateflowObjectProperties.createTemplate(  
templatePath,type)
```

## Description

`template = slreportgen.report.StateflowObjectProperties.createTemplate(templatePath,type)` creates a copy of the default Stateflow object properties template specified by `type` at the location specified by `templatePath`. Use the copied template as a starting point to design a custom Stateflow object properties template for your report.

## Input Arguments

**templatePath — Path and file name of new template**

character vector | string scalar

Path and file name of the new template, specified as a character vector or string scalar.

**type — Type of template**

'html' | 'html-file' | 'docx' | 'pdf'

Type of template, specified as 'html', 'html-file', 'docx', or 'pdf'.

## Output Arguments

**template — Path and file name of template copy**

string scalar

Path and file name of the copy of the template, returned as a string scalar. The file name extension of the template is assigned based on the specified template type. For example, if the `type` argument is 'pdf', the file name extension is `.pdf.tx`.

## Examples

### Create a Report Template

Before you run this example, create a copy of the default HTML Stateflow object properties template in a folder named `mytemplates` and name the copied template `myObjectsReporter.htm`. Edit the template as desired. To use the new template, assign its path to the `slreportgen.report.StateflowObjectProperties.TemplateSrc` property.

```
import slreportgen.report.*
import mlreportgen.report.*
rpt = slreportgen.report.Report('My Report','html');
model_name = 'sf_car');
load_system(model_name);

chapter = Chapter(model_name);
chart = block2chart('sf_car/shift_logic');
rptr = StateflowObjectProperties(chart);
template = StateflowObjectProperties.createTemplate...
('mytemplates\myObjReporter','html');
rptr.TemplateSrc = template;
```

## See Also

slreportgen.report.StateflowObjectProperties

**Introduced in R2017b**

# slreportgen.report.SimulinkObjectProperties.customizeReporter

**Class:** slreportgen.report.SimulinkObjectProperties

**Package:** slreportgen.report

Create custom Stateflow object properties class

## Syntax

```
reporter = slreportgen.report.StateflowObjectProperties.customizeReporter(  
classpath)
```

## Description

`reporter = slreportgen.report.StateflowObjectProperties.customizeReporter(classpath)` creates a Stateflow object properties page class definition file that is a subclass of `slreportgen.report.SimulinkObjectProperties`. The file is created at the specified `classpath` location. The `SimulinkObjectProperties.customizeReporter` method also copies the default Stateflow object properties templates to the `<classpath>/resources/template` folder. To design a custom Stateflow object properties class for your report, use the new class definition file as a starting point.

## Input Arguments

**classpath — Location of custom Stateflow object properties class**

current working folder (default) | string | character array

Location of custom Stateflow object properties class, specified as a string or character array. The `classpath` argument also supports specifying a folder with `@` before the class name.

## Output Arguments

**reporter — Stateflow object properties reporter path**

string

Stateflow object properties reporter path, returned as the string specifying the path to the derived report class file.

## Examples

### Create Custom Stateflow Object Properties Reporter

Create a custom Stateflow object properties reporter and its associated default templates. The derived class file is created at the specified path relative to the current working folder. In this case, the path to the `MySFObjProps.m` class file is `<current working folder>/newSFObjProps/@MySFObjProps/MySFObjProps.m`. The default title page templates are in the `<current working folder>/newTitlePage/@MySFObjProps/resources/templates` folder.

```
import slreportgen.report.*
StateflowObjectProperties.customizeReporter...
    ('newSFObjProps/@MySFObjProps');
```

After editing this new class file, you can use it as your Stateflow object properties reporter.

```
objprop = MySFObjProps();
```

## See Also

[slreportgen.report.Report](#) | [slreportgen.report.StateflowObjectProperties](#)

**Introduced in R2017b**

## **slreportgen.report.StateflowObjectProperties.getClassFolder**

**Class:** slreportgen.report.StateflowObjectProperties

**Package:** slreportgen.report

Stateflow object properties class definition file location

### **Syntax**

```
path = slreportgen.report.StateflowObjectProperties.getClassFolder()
```

### **Description**

`path = slreportgen.report.StateflowObjectProperties.getClassFolder()` returns the path of the folder that contains the Stateflow object properties class definition file.

### **Output Arguments**

**path** — Stateflow object properties class definition file location

character array

Stateflow object properties class definition file location, returned as a character array.

### **See Also**

`slreportgen.report.Report` | `slreportgen.report.StateflowObjectProperties`

**Introduced in R2017b**



# slreportgen.report.SystemHierarchy.createTemplate

**Class:** slreportgen.report.SystemHierarchy

**Package:** slreportgen.report

Create system hierarchy reporter template

## Syntax

```
template = slreportgen.report.SystemHierarchy.createTemplate(templatePath,  
type)
```

## Description

`template = slreportgen.report.SystemHierarchy.createTemplate(templatePath, type)` creates a copy of the `slreportgen.report.SystemHierarchy` reporter template for the report type specified by `type` at the location specified by `templatePath`. You can use the copied template as a starting point to design a custom system hierarchy template for your report.

## Input Arguments

**templatePath** — Path and file name of new template

character vector | string scalar

Path and file name of the new template, specified as a character vector or string scalar.

**type** — Type of template

'html' | 'html-file' | 'docx' | 'pdf'

Type of template, specified as 'html', 'html-file', 'docx', or 'pdf'.

## Output Arguments

**template** — Path and file name of template copy

string scalar

Path and file name of the copy of the template, returned as a string scalar. The file name extension of the template is assigned based on the specified template type. For example, if the `type` argument is 'pdf', the file name extension is `.pdf.tx`.

## Examples

### Create a System Hierarchy Reporter Template

Create a copy of the HTML template for the system hierarchy reporter and save it with the name `mySysHierarchyTemplate` in the `mytemplates` folder.

```
template = slreportgen.report.SystemHierarchy.createTemplate...  
    ('mytemplates/mySysHierarchyTemplate', 'html');
```

After you modify the template, you can use it by setting the TemplateSrc property of the reporter.

**See Also**

slreportgen.report.Report | slreportgen.report.SystemHierarchy

**Introduced in R2019b**

# slreportgen.report.SystemHierarchy.customizeReporter

**Class:** slreportgen.report.SystemHierarchy

**Package:** slreportgen.report

Create custom system hierarchy reporter class

## Syntax

```
reporter = slreportgen.report.SystemHierarchy.customizeReporter(classpath)
```

## Description

`reporter = slreportgen.report.SystemHierarchy.customizeReporter(classpath)` creates a class definition file that defines a subclass of `slreportgen.report.SystemHierarchy` at the location specified by `classpath`. This method also copies the default reporter templates to the `resources/templates` subfolder of the folder that contains the class definition file. You can use the class definition file as a starting point to design a custom system hierarchy reporter class for your report.

## Input Arguments

**classpath — Path and name of new class definition file**

string scalar | character vector

Path and name of new class definition file, specified as a string scalar or character vector.

You can specify a relative path or an absolute path. For example, this code creates `MyClass.m` in the subfolder `myFolder` of the current folder.

```
slreportgen.report.SystemHierarchy.customizeReporter("myFolder/MyClass")
```

To create the reporter class in a class folder, precede the class name with the `@` character. Do not specify the `.m` extension. For example, this code creates `MyClass.m` in the subfolder `myFolder/@MyClass` of the current folder.

```
slreportgen.report.SystemHierarchy.customizeReporter("myFolder/@MyClass")
```

See “Folders Containing Class Definitions”.

To create the reporter class in a class package, precede the folder name with the `+` character. For example, this code creates a system hierarchy reporter in the `myOrg` package folder in the current folder.

```
slreportgen.report.SystemHierarchy.customizeReporter("+myOrg/@SystemHierarchy");
```

## Output Arguments

**reporter — Path to new system hierarchy reporter class**

string scalar

Path and file name of the new system hierarchy reporter class, returned as a string scalar.

## Examples

### Create Custom System Hierarchy Reporter Class

Create a custom system hierarchy reporter class, `MySystemHierarchy`, and its associated default templates in the subfolder `MyFolder` of the current working folder.

```
slreportgen.report.SystemHierarchy.customizeReporter('MyFolder/MySystemHierarchy')
```

```
ans =
```

```
    "MyFolder\MySystemHierarchy.m"
```

After editing this new class file, you can use it as your system hierarchy reporter.

```
rptr = MySystemHierarchy;
```

### See Also

`slreportgen.report.Report` | `slreportgen.report.SystemHierarchy`

**Introduced in R2019b**

# slreportgen.report.SystemHierarchy.getClassFolder

**Class:** slreportgen.report.SystemHierarchy

**Package:** slreportgen.report

Get location of system hierarchy reporter class definition file

## Syntax

```
path = slreportgen.report.SystemHierarchy.getClassFolder()
```

## Description

path = slreportgen.report.SystemHierarchy.getClassFolder() returns the path of the folder that contains the slreportgen.report.SystemHierarchy class definition file.

## Output Arguments

**path** — Location of the system hierarchy reporter class definition file

character vector

Location of the slreportgen.report.SystemHierarchy class definition file, returned as a character vector.

## Examples

### Get System Hierarchy Reporter Class Folder

Get the location of the folder that contains the system hierarchy reporter class definition.

```
path = slreportgen.report.SystemHierarchy.getClassFolder()
```

## See Also

slreportgen.report.Report | slreportgen.report.SystemHierarchy

**Introduced in R2019b**

## slreportgen.report.SystemIO.createTemplate

**Class:** slreportgen.report.SystemIO

**Package:** slreportgen.report

Copy the default slreportgen.report.SystemIO reporter template

### Syntax

```
template = slreportgen.report.SystemIO.createTemplate(templatePath,type)
```

### Description

`template = slreportgen.report.SystemIO.createTemplate(templatePath,type)` creates a copy of the `slreportgen.report.SystemIO` reporter template for the report type specified by `type` at the location specified by `templatePath`. You can use the copy of the template as a starting point to design a custom `slreportgen.report.SystemIO` template for your report.

### Input Arguments

**templatePath — Path and file name of new template**

character vector | string scalar

Path and file name of the new template, specified as a character vector or string scalar.

**type — Type of template**

'html' | 'html-file' | 'docx' | 'pdf'

Type of template, specified as 'html', 'html-file', 'docx', or 'pdf'.

### Output Arguments

**template — Path and file name of template copy**

string scalar

Path and file name of the copy of the template, returned as a string scalar. The file name extension of the template is assigned based on the specified template type. For example, if the `type` argument is 'pdf', the file name extension is `.pdftx`.

### Examples

#### Create a System Input and Output Reporter Template

Create a copy of the HTML template for the `slreportgen.report.SystemIO` reporter and save it with the name `mySystemIOTemplate` in the `mytemplates` folder.

```
template = slreportgen.report.SystemIO.createTemplate...  
('mytemplates/mySystemIOTemplate','html');
```

After you modify the template, you can use it by setting the `TemplateSrc` property of the reporter.

### **See Also**

`slreportgen.report.Report` | `slreportgen.report.SystemIO`

**Introduced in R2020a**

## slreportgen.report.SystemIO.customizeReporter

**Class:** slreportgen.report.SystemIO

**Package:** slreportgen.report

Create subclass of slreportgen.report.SystemIO class

### Syntax

```
reporter = slreportgen.report.SystemIO.customizeReporter(classpath)
```

### Description

`reporter = slreportgen.report.SystemIO.customizeReporter(classpath)` creates a reporter class definition file that defines a subclass of `slreportgen.report.SystemIO` at the location specified by `classpath`. The method also copies the default reporter templates to the `resources/templates` subfolder of the folder that contains the class definition file. You can use the class definition file as a starting point to design a custom system input and output reporter class for your report.

### Input Arguments

**classpath — Path and name of new class definition file**

string scalar | character vector

Path and name of new class definition file, specified as a string scalar or character vector.

You can specify a relative path or an absolute path. For example, this code creates `MyClass.m` in the subfolder `myFolder` of the current folder.

```
slreportgen.report.SystemIO.customizeReporter("myFolder/MyClass")
```

To create the reporter class in a class folder, precede the class name with the `@` character. Do not specify the `.m` extension. For example, this code creates `MyClass.m` in the subfolder `myFolder/@MyClass` in the current folder.

```
slreportgen.report.SystemIO.customizeReporter("myFolder/@MyClass")
```

See “Folders Containing Class Definitions”.

To create the reporter class in a class package, precede the folder name with the `+` character. For example, this code creates a system input and output reporter in the `myOrg` package folder in the current folder.

```
slreportgen.report.SystemIO.customizeReporter("+myOrg/@MyClass")
```

### Output Arguments

**reporter — Path and file name of new reporter class**

string scalar

Path and file name of the new reporter class, returned as a string scalar.



## Examples

### Create Custom System Input and Output Reporter

Create a custom system input and output reporter, `mySystemIO`, and the associated default templates in the subfolder `MyFolder` of the current working folder.

```
slreportgen.report.SystemIO.customizeReporter('MyFolder/mySystemIO')
```

```
ans =
```

```
    "MyFolder\mySystemIO.m"
```

### See Also

`slreportgen.report.Report` | `slreportgen.report.SystemIO`

**Introduced in R2020a**

## **slreportgen.report.SystemIO.getClassFolder**

**Class:** slreportgen.report.SystemIO

**Package:** slreportgen.report

Get location of folder that contains the slreportgen.report.SystemIO class definition file

### **Syntax**

```
path = slreportgen.report.SystemIO.getClassFolder()
```

### **Description**

path = slreportgen.report.SystemIO.getClassFolder() returns the path of the folder that contains the slreportgen.report.SystemIO class definition file.

### **Output Arguments**

**path — slreportgen.report.SystemIO class definition file location**

character vector

slreportgen.report.SystemIO class definition file location, returned as a character vector.

### **Examples**

#### **Get System Input and Output Reporter Class Folder**

Get the location of the folder that contains the system input and output reporter class definition.

```
path = slreportgen.report.SystemIO.getClassFolder()
```

### **See Also**

slreportgen.report.Report | slreportgen.report.SystemIO

**Introduced in R2020a**

# slreportgen.report.TestSequence.createTemplate

**Class:** slreportgen.report.TestSequence

**Package:** slreportgen.report

Create Test Sequence block reporter template

## Syntax

```
template = slreportgen.report.TestSequence.createTemplate(templatePath,type)
```

## Description

`template = slreportgen.report.TestSequence.createTemplate(templatePath,type)` creates a copy of the `slreportgen.report.TestSequence` reporter template for the report type specified by `type` at the location specified by `templatePath`. You can use the copied template as a starting point to design a custom Test Sequence block reporter template for your report.

## Input Arguments

**templatePath — Path and file name of new template**

character vector | string scalar

Path and file name of the new template, specified as a character vector or string scalar.

**type — Type of template**

'html' | 'html-file' | 'docx' | 'pdf'

Type of template, specified as 'html', 'html-file', 'docx', or 'pdf'.

## Output Arguments

**template — Path and file name of template copy**

string scalar

Path and file name of the copy of the template, returned as a string scalar. The file name extension of the template is assigned based on the specified template type. For example, if the `type` argument is 'pdf', the file name extension is `.pdftx`.

## Examples

### Create a Test Sequence Block Reporter Template

Create a copy of the HTML template for the `slreportgen.report.TestSequence` reporter and save it with the name `myTestSequenceTemplate` in the `mytemplates` folder.

```
template = slreportgen.report.TestSequence.createTemplate...
('mytemplates/myTestSequenceTemplate','html');
```

After you modify the template, you can use it by setting the `TemplateSrc` property of a `TestSequence` reporter to the path of the template file.

**See Also**

`slreportgen.report.TestSequence`

**Introduced in R2020b**

# slreportgen.report.TestSequence.customizeReporter

**Class:** slreportgen.report.TestSequence

**Package:** slreportgen.report

Create custom Test Sequence block reporter class

## Syntax

```
reporter = slreportgen.report.TestSequence.customizeReporter(classpath)
```

## Description

`reporter = slreportgen.report.TestSequence.customizeReporter(classpath)` creates a class definition file that defines a subclass of `slreportgen.report.TestSequence` at the location specified by `classpath`. This method also copies the default reporter templates to the `resources/templates` subfolder of the folder that contains the class definition file. You can use the class definition file as a starting point to design a custom Test Sequence block reporter class for your report.

## Input Arguments

**classpath — Path and name of new class definition file**

string scalar | character vector

Path and name of new class definition file, specified as a string scalar or character vector.

You can specify a relative path or an absolute path. For example, this code creates `MyClass.m` in the subfolder `myFolder` of the current folder.

```
slreportgen.report.TestSequence.customizeReporter("myFolder/MyClass")
```

To create the reporter class in a class folder, precede the class name with the `@` character. Do not specify the `.m` extension. For example, this code creates `MyClass.m` in the subfolder `myFolder/@MyClass` in the current folder.

```
slreportgen.report.TestSequence.customizeReporter("myFolder/@MyClass")
```

See “Folders Containing Class Definitions”.

To create the reporter class in a class package, precede the folder name with the `+` character. For example, this code creates a Test Sequence block reporter in the `myOrg` package folder in the current folder.

```
slreportgen.report.TestSequence.customizeReporter("+myOrg/@MyClass");
```

## Output Arguments

**reporter — Path and file name of new Test Sequence block reporter class**

string scalar

Path and file name of the new Test Sequence block reporter class, returned as a string scalar.

## Examples

### Create Custom Test Sequence Block Reporter

Create a custom Test Sequence block reporter, `MyTestSequence`, and its associated default templates in the subfolder `MyFolder` of the current working folder.

```
slreportgen.report.TestSequence.customizeReporter('MyFolder/MyTestSequence')
```

```
ans =
```

```
    "MyFolder\MyTestSequunce.m"
```

### See Also

`slreportgen.report.TestSequence`

**Introduced in R2020b**

# slreportgen.report.TestSequence.getClassFolder

**Class:** slreportgen.report.TestSequence

**Package:** slreportgen.report

Get location of Test Sequence block reporter class definition file

## Syntax

```
path = slreportgen.report.TestSequence.getClassFolder()
```

## Description

`path = slreportgen.report.TestSequence.getClassFolder()` returns the path of the folder that contains the `slreportgen.report.TestSequence` class definition file.

## Output Arguments

**path** — Location of the Test Sequence block reporter class definition file

character vector

Location of the `slreportgen.report.TestSequence` class definition file, returned as a character vector.

## Examples

### Get Test Sequence Block Reporter Class Folder

Get the folder that contains the Test Sequence block reporter class definition.

```
path = slreportgen.report.TestSequence.getClassFolder()
```

## See Also

`slreportgen.report.TestSequence`

**Introduced in R2020b**

## slreportgen.report.TruthTable.createTemplate

**Class:** slreportgen.report.TruthTable

**Package:** slreportgen.report

Create truth table template

### Syntax

```
template = slreportgen.report.TruthTable.createTemplate(templatePath,type)
```

### Description

`template = slreportgen.report.TruthTable.createTemplate(templatePath,type)` creates a copy of the default `TruthTable` reporter template specified by `type` at the location specified by `templatePath`. To design a custom truth table template for your report, use the copied template as a starting point.

### Input Arguments

#### **templatePath** — Location of reporter template

string | character vector | character array | template source object

Location of the reporter template, specified as a character vector, character array, or template source object.

#### **type** — Type of template

'html' | 'html-file' | 'docx' | 'pdf'

Type of template, specified as 'html', 'html-file', 'docx', or 'pdf'.

### Output Arguments

#### **template** — Template name

string

Name of template, returned as the path and file name of the template. The template file name extension is assigned based on the specified output type. For example, for PDF output, the template name has a `.pdftx` file extension.

### Examples

#### **Create a Truth Table Template**

```
import slreportgen.report.*
rpt = Report('My Report','html');
load_system('sf_climate_control')

template = slreportgen.report.TruthTable.createTemplate...
    ('mytemplates\myTruthTable','html');
```



```
trtable = slreportgen.report.TruthTable...  
    ('sf_climate_control/ClimateController');  
trtable.TemplateSrc = template;
```

**See Also**

slreportgen.report.Report | slreportgen.report.TruthTable

**Introduced in R2018b**

## slreportgen.report.TruthTable.customizeReporter

**Class:** slreportgen.report.TruthTable

**Package:** slreportgen.report

Create custom truth table reporter class

### Syntax

```
customRptrPath = slreportgen.report.TruthTable.customizeReporter(classpath)
```

### Description

`customRptrPath = slreportgen.report.TruthTable.customizeReporter(classpath)` creates an empty truth table class definition file that is a subclass of `slreportgen.report.TruthTable`. The file is created at the specified `classpath` location. The `customizeReporter` method also copies the default truth table templates to the `<classpath>/resources/template` folder. To design a custom truth table class for your report, you can use the new class definition file as a starting point.

### Input Arguments

**classpath** — Location of custom truth table class

current working folder (default) | string | character array

Location of custom truth table class, specified as a string or character array. The `classpath` argument also supports specifying a folder with `@` before the class name. For example, both of these paths are valid:

- `slreportgen.report.TruthTable.customizeReporter("path_folder/MyClassA.m")`
- `slreportgen.report.TruthTable.customizeReporter("+package/@MyClassB")`

### Output Arguments

**customRptrPath** — Path of custom truth table reporter

string

Path of the class definition file that defines the custom truth table reporter, specified as a string.

### Examples

#### Create Custom Truth Table Reporter

Create a custom `TruthTable` reporter and its associated default templates. The derived class file is created at the specified path relative to the current working folder. In this case, the path to the `MyTruthTable.m` class file is `<current working folder>/newTruthTable/@MyTruthTable/MyTruthTable.m`. The default diagram templates are in the `<current working folder>/newTruthTable/@MyTruthTable/resources/templates` folder.

```
import slreportgen.report.*  
TruthTable.customizeReporter('newTruthTable/@MyTruthTable');
```

After editing this new class file and loading a model, you can use the new diagram reporter.

```
sf_car;  
diagram = MyTruthTable('sf_car');
```

## See Also

[slreportgen.report.Report](#) | [slreportgen.report.TruthTable](#)

**Introduced in R2018b**

## **slreportgen.report.TruthTable.getClassFolder**

**Class:** slreportgen.report.TruthTable

**Package:** slreportgen.report

Location of truth table class definition file

### **Syntax**

```
path = slreportgen.report.TruthTable.getClassFolder()
```

### **Description**

`path = slreportgen.report.TruthTable.getClassFolder()` returns the path of the folder that contains the truth table class definition file.

### **Output Arguments**

**path — Truth table class definition file location**

character array

Truth table class definition file location, returned as a character array.

### **See Also**

`slreportgen.report.Report` | `slreportgen.report.TruthTable`

**Introduced in R2018b**

# slreportgen.utils.block2chart

**Package:** slreportgen.utils

Get Stateflow chart object from containing block

## Syntax

```
charts = slreportgen.utils.block2chart(subsystems)
```

## Description

`charts = slreportgen.utils.block2chart(subsystems)` returns an array of Stateflow chart objects that are contained in the specified subsystems.

## Examples

### Obtain Chart Object

```
load_system('sldemo_fuelsys')
slreportgen.utils.block2chart...
('sldemo_fuelsys/fuel_rate_control/control_logic')
```

## Input Arguments

### **subsystems** — Subsystem blocks

array of subsystem block handles | array of paths to subsystem blocks

Subsystem blocks, specified as an array of subsystem block handles or an array of handles or paths to the subsystem blocks.

## Output Arguments

### **charts** — Stateflow chart objects

array of Stateflow chart objects

Stateflow chart objects, returned as an array. The returned Stateflow charts are contained in the specified subsystems.

## See Also

`slreportgen.utils.getSfHandle`

**Introduced in R2018b**

## **slreportgen.utils.compileModel**

**Package:** slreportgen.utils

Compile model

### **Syntax**

```
slreportgen.utils.compileModel(modelname)
```

### **Description**

`slreportgen.utils.compileModel(modelname)` compiles the model specified in the `modelname` input. If `modelname` is a Simulink block or Stateflow object, the model that contains the block or object is compiled.

### **Input Arguments**

#### **Input Arguments**

##### **modelname — Model to compile**

character array | string

Model to compile, specified as a character array or string of the model handle, block handle, or Stateflow object. If `modelname` is an element in the model or chart, this utility compiles the containing model or chart.

### **See Also**

`slreportgen.utils.isModelCompiled` | `slreportgen.utils.uncompileModel`

**Introduced in R2018b**

# slreportgen.utils.getCurrentEditorView

Get current editor view area

## Syntax

```
viewArea = slreportgen.utils.getCurrentEditorView()
```

## Description

`viewArea = slreportgen.utils.getCurrentEditorView()` returns the current Simulink Editor view area as a 1-by-4 array of doubles. The first two values of the array are the x and y coordinates, in pixels, of the top left corner of the diagram area in the Simulink Editor coordinate space. The last two values are the width and height, in pixels.

## Examples

### Take a Snapshot of the Current Editor View

This example creates an `slreportgen.report.Diagram` reporter that takes a snapshot of the current editor view.

Open a model in Simulink.

```
f14
```

In the Simulink Editor, zoom in on the area of the diagram that you want to capture in the report.

Set up the report and create an `slreportgen.report.Diagram` reporter to take a snapshot of the top-level diagram. Specify that you want to capture the current editor view in the diagram snapshot.

```
import slreportgen.report.*
import slreportgen.utils.*

rpt = Report('output', 'pdf');

diag = Diagram('f14');
diag.SnapshotArea = getCurrentEditorView();
add(rpt,diag);

close(rpt);
rptview(rpt);
```

## See Also

`slreportgen.report.Diagram`

**Introduced in R2020a**

## slreportgen.utils.getDisplayIcon

**Package:** slreportgen.utils

Get Simulink or Stateflow icon file name

### Syntax

```
displayIcon = slreportgen.utils.getDisplayIcon(obj)
```

### Description

`displayIcon = slreportgen.utils.getDisplayIcon(obj)` returns the icon image file name for a Simulink handle or Stateflow object.

### Examples

#### Get Icon Path for Stateflow Chart

```
load_system('sf_car')
chart = find(slroot, '-isa', ...
    'Stateflow.Chart', 'Name', 'shift_logic');
iconPath = slreportgen.utils.getDisplayIcon(chart)

iconPath =

    "C:\Program Files\MATLAB\R2018b\toolbox\...
    shared\dastudio\resources\Chart.png"
```

### Input Arguments

**obj** — Simulink or Stateflow object

string | character array

Simulink handle or Stateflow object, specified as a string or character array of its path or handle.

### Output Arguments

**displayIcon** — Icon image file name

string

Icon image file name, returned as a string.

### See Also

**Introduced in R2018b**



# slreportgen.utils.getModelHandle

**Package:** slreportgen.utils

Get Simulink model

## Syntax

```
modelHandle = slreportgen.utils.getModelHandle(obj)
```

## Description

`modelHandle = slreportgen.utils.getModelHandle(obj)` returns the handle of a Simulink model or Stateflow object.

## Examples

### Obtain Simulink Model Handle

```
load_system('f14')
modelHandle_blk = slreportgen.utils.getModelHandle('f14/Controller')

modelHandle_blk =

    2.0002
```

### Obtain Stateflow Chart Handle

```
load_system('sf_car');
chart = find(slroot, '-isa', ...
    'Stateflow.Chart', 'Name', 'shift_logic');
chart_handle = slreportgen.utils.getModelHandle(chart)

chart_handle =

    186.0001
```

## Input Arguments

### **obj** — Simulink or Stateflow object

string | character array

Simulink or Stateflow object, specified as a string or character array of the object path or object handle. This utility returns the handle of the specified model or chart. If you specify an element in the model or chart, this utility returns the handle of the containing model or chart.

## Output Arguments

### **modelHandle** — Model handle

double

Model handle, returned as a double.

**See Also**

`slreportgen.utils.getSlsfHandle`

**Introduced in R2018b**

# slreportgen.utils.getObjectID

**Package:** slreportgen.utils

Generate link target ID for Simulink or Stateflow object

## Syntax

```
id = slreportgen.utils.getObjectID(obj)
id = slreportgen.utils.getObjectID(obj, "Hash", false)
```

## Description

`id = slreportgen.utils.getObjectID(obj)` generates a link target ID for the specified Simulink or Stateflow object. The ID is hashed so that it does not exceed the 40-character limit imposed on Microsoft Word bookmarks.

---

**Note** The `slreportgen.report.Diagram` and `slreportgen.report.ElementDiagram` reporters use this utility to generate IDs for element hyperlinks in diagrams generated for HTML and PDF reports. You can use this function to generate corresponding link targets for the diagram elements.

To create a link target for a Truth Table block in a Simulink diagram, specify the Truth Table block path and not the `Stateflow.TruthTable` object.

---

`id = slreportgen.utils.getObjectID(obj, "Hash", false)` does not hash the generated ID.

## Examples

### Obtain Object ID from Handle

```
load_system('f14')
modelHandle = slreportgen.utils.getSfHandle('f14');
objID = slreportgen.utils.getObjectID(modelHandle)
```

```
objID =
```

```
    '8bc7ba92e180202ffc5ce217625c6563'
```

## Input Arguments

**obj** — Simulink or Stateflow object

object path | object handle

Simulink or Stateflow object, specified by its path or handle, for which to generate a link target ID.

## Output Arguments

### **id — Simulink or Stateflow link target ID**

character vector

Simulink or Stateflow link target ID, returned as a character vector. You use this character vector as the anchor ID for linking.

### **See Also**

`mlreportgen.dom.LinkTarget` | `slreportgen.report.Diagram` |  
`slreportgen.report.ElementDiagram`

**Introduced in R2018b**

# slreportgen.utils.getResolvedParamValue

**Package:** slreportgen.utils

Evaluate parameter value expression

## Syntax

```
param_value = slreportgen.utils.getResolvedParamValue(block,parameter)
```

## Description

`param_value = slreportgen.utils.getResolvedParamValue(block,parameter)` returns the value of a block parameter. The value of the parameter is an evaluated MATLAB expression that includes references to workspace variables. This utility produces an `slreportgen:UnresolvableExpression` error if it cannot find expression variables in the MATLAB or model workspace.

## Examples

### Obtain Parameter Value

```
load_system('f14')
numval = slreportgen.utils.getResolvedParamValue...
('f14/Actuator Model','Numerator')
```

```
numval =
```

```
1
```

## Input Arguments

### **block** — Block name

string | character array

Block name, specified as a string or character array of the block name or block handle.

### **parameter** — Block parameter name

string | character array

Block parameter name, specified as a string or character array.

## Output Arguments

### **param\_value** — Value of block parameter

depends on block

Value of block parameter. The data type of the returned value depends on the specific parameter.

**See Also**

`slreportgen.utils.traceSignal`

**Introduced in R2018b**

# slreportgen.utils.getSfHandle

Get Simulink handle or Stateflow object

## Syntax

```
sisfhandle = slreportgen.utils.getSfHandle(path_id)
```

## Description

`sisfhandle = slreportgen.utils.getSfHandle(path_id)` returns the Simulink or Stateflow object handle of the input Simulink path, Simulink identifier (SID), or Stateflow numeric ID.

## Examples

### Obtain Simulink Model and Object Handles

```
load_system('f14')
modelHandle = slreportgen.utils.getSfHandle('f14')
blockHandle = slreportgen.utils.getSfHandle('f14/Controller')
SIDHandle = slreportgen.utils.getSfHandle('f14:3')
```

```
modelHandle =
```

```
    2.0001
```

```
blockHandle =
```

```
    38.0001
```

```
SIDHandle =
```

```
    6.0001
```

### Obtain Stateflow Chart Handle

```
load_system('sf_car')
chart = find(slroot, '-isa',...
    'Stateflow.Chart', 'Name', 'shift_logic');
slreportgen.utils.getSfHandle(chart.Id);
```

This example returns the list of Stateflow chart properties to the workspace. The Stateflow handle is equivalent to the Stateflow object. To use a Stateflow handle, assign it to a variable.

## Input Arguments

### **path\_id** – Simulink or Stateflow object path or ID

string | character array

Simulink or Stateflow object path or ID, specified as a string or character array. This utility returns the handle of the specified `path_id` object.

## **Output Arguments**

**sisfhandle** — Handle of object

double

Handle of the specified Simulink or Stateflow object, returned as a double.

## **See Also**

`getSimulinkBlockHandle` | `slreportgen.utils.getModelHandle`

**Introduced in R2018b**



# slreportgen.utils.hasDiagram

**Package:** slreportgen.utils

Check if object has diagram

## Syntax

```
tf = slreportgen.utils.hasDiagram(obj)
```

## Description

`tf = slreportgen.utils.hasDiagram(obj)` returns 1 (true) if the input object has a diagram and 0 (false) if the object does not have a diagram.

These types of objects have diagrams:

- Simulink block diagram
- Simulink graphical subsystem
- Stateflow chart
- Stateflow Simulink function
- Subcharted Stateflow state
- Subcharted Stateflow function
- Subcharted Stateflow box

You can use this function to check if an object has a diagram before you try to use the object with an `slreportgen.report.Diagram` reporter.

## Examples

### Check Whether Object Has a Diagram

```
load_system('f14');  
tf = slreportgen.utils.hasDiagram('f14/Aircraft Dynamics Model')  
  
tf =  
  
    logical  
  
     1
```

## Input Arguments

### obj — Object to check

string scalar | character vector | handle

Object to check for a diagram, specified as a character vector or string scalar that contains the Simulink path of the object or as a handle to the object.

## Output Arguments

### **tf** — Whether object has a diagram

1 (true) | 0 (false)

Whether the input object has a diagram, returned as 1 (true) if the input object has a diagram or 0 (false) if the input object does not have a diagram.

### See Also

`slreportgen.report.Diagram` | `slreportgen.utils.getModelHandle`

**Introduced in R2020b**

# slreportgen.utils.isBusSelector

**Package:** slreportgen.utils

Check if Bus Selector block

## Syntax

```
tf = slreportgen.utils.isBusSelector(obj)
```

## Description

`tf = slreportgen.utils.isBusSelector(obj)` tests if the input `obj` is a Simulink Bus Selector block.

## Examples

### Test If Object Is a Bus Selector

```
load_system('sldemo_fuelsys')
tf = slreportgen.utils.isBusSelector...
    ('sldemo_fuelsys/fuel_rate_control/Bus Selector1')

tf =

    logical

     1
```

In this case, the `Bus Selector1` block in the `sldemo_fuelsys` model is a Bus Selector block.

## Input Arguments

### **obj** — Input object to test for being a Bus Selector block

string | character array | object handle

Input object to test for being a Bus Selector block, specified as a string or character array of the object path or handle.

## Output Arguments

### **tf** — Whether input is a Bus Selector block

true | false

Whether input is a Bus Selector block, returned as 1 (true) if the input is a Bus Selector. Otherwise, it returns 0 (false).

## See Also

Bus Selector

**Introduced in R2019a**

# slreportgen.utils.isCommented

**Package:** slreportgen.utils

Check if object is commented out

## Syntax

```
tf = slreportgen.utils.isCommented(obj)
```

## Description

`tf = slreportgen.utils.isCommented(obj)` tests whether the input Simulink or Stateflow object is commented out.

## Examples

### Check If System Is Commented Out

```
load_system('sf_car')
tf = slreportgen.utils.isCommented('sf_car/Vehicle')

tf =

    logical

     0
```

In this case, the `sf_car` model is not commented out.

## Input Arguments

**obj** — Object to check for being commented out

string | character array

Simulink or Stateflow object to check for being commented out, specified as a string or character array of the Simulink object path or handle or Stateflow object.

## Output Arguments

**tf** — Whether input is a masked system

true | false

Whether input is a masked system, returned as 1 (true) if the input system is masked. Otherwise, it returns 0 (false).

## See Also

**Introduced in R2018b**

## slreportgen.utils.isDocBlock

**Package:** slreportgen.utils

Check if DocBlock

### Syntax

```
tf = slreportgen.utils.isDocBlock(obj)
```

### Description

`tf = slreportgen.utils.isDocBlock(obj)` tests if the input `obj` is a Simulink DocBlock block.

### Examples

#### Test If Object Is a DocBlock

```
load_system('sldemo_fuelsys')
tf = slreportgen.utils.isDocBlock ...
    ('sldemo_fuelsys/To Controller/Sensor Info')

tf =

    logical

     1
```

In this case, the Sensor Info block in the `sldemo_fuelsys` model is a DocBlock.

### Input Arguments

#### **obj** — Object to test for being a DocBlock

string | character array | handle

Object to test for being a DocBlock, specified as a string or character array of the block path or handle.

### Output Arguments

#### **tf** — Whether input is a DocBlock

true | false

Whether input is a DocBlock, returned as 1 (true) if the input is a DocBlock. Otherwise, it returns 0 (false).

### See Also

DocBlock

**Introduced in R2019a**

## slreportgen.utils.isLookupTable

**Package:** slreportgen.utils

Check if lookup table block

### Syntax

```
tf = slreportgen.utils.isLookupTable(obj)
```

### Description

`tf = slreportgen.utils.isLookupTable(obj)` tests whether the `obj` is a lookup table block.

These lookup table blocks are supported:

- 1-D Lookup Table
- 2-D Lookup Table
- n-D Lookup Table
- Interpolation Using Prelookup
- Direct Lookup Table (n-D)
- Lookup Table Dynamic

### Examples

#### Test Whether Block Is a Lookup Table Block

Use `isLookupTable` to test whether a block is a type of lookup table block.

Find blocks in a container, such as a model, and obtain its results.

```
blkfinder = slreportgen.finder.BlockFinder(model_name);  
results = find(blkfinder);
```

Then, loop through the results and test whether each block is a lookup table block. For each result that is a lookup table block, create a `LookupTable` reporter, and add the reporter to the report.

```
for i=1:length(results)  
    block = results(i).Object;  
    if slreportgen.utils.isLookupTable(block)  
        rptr = LookupTable(block);  
        add(rpt,rptr);  
    end  
end
```

### Input Arguments

**obj** — Simulink block to query

string | character array



Simulink block to query for whether it is a lookup table block, specified as a string or character array of its block path or handle.

## Output Arguments

**tf** — Whether input is a lookup table block

true | false

Whether input is a lookup table block, returned as 1 (true) if the input is a lookup table block. Otherwise, it returns 0 (false).

## See Also

[slreportgen.report.LookupTable](#) | [slreportgen.utils.isTruthTable](#)

**Introduced in R2018a**

## slreportgen.utils.isMaskedSystem

**Package:** slreportgen.utils

Check if system is masked subsystem block

### Syntax

```
tf = isMaskedSystem(system)
```

### Description

`tf = isMaskedSystem(system)` tests whether the input system is masked.

### Examples

#### Check If System Is Masked

```
load_system('sf_car')
tf = slreportgen.utils.isMaskedSystem('sf_car/Vehicle')
```

```
tf =
```

```
    logical
```

```
     1
```

In this case, the `sf_car` model is masked.

### Input Arguments

**system** — System to check for masking

string | character array

System to check for masking, specified as a string or character array of the Simulink path or handle.

### Output Arguments

**tf** — Whether input is a masked system

true | false

Whether input is a masked system, returned as 1 (true) if the input system is masked. Otherwise, it returns 0 (false).

### See Also

#### Topics

“Create Block Masks”

**Introduced in R2018b**

## slreportgen.utils.isMATLABFunction

**Package:** slreportgen.utils

Check if MATLAB function block or object

### Syntax

```
tf = slreportgen.utils.isMATLABFunction(obj)
```

### Description

`tf = slreportgen.utils.isMATLABFunction(obj)` tests whether the input `obj` is a Simulink MATLAB Function block or a Stateflow MATLAB function object.

### Examples

#### Test Whether Block Is a MATLAB Function Block

Use `isMATLABFunction` to test whether a block is a MATLAB Function block.

Find blocks in a container, such as a model, and obtain its results.

```
blkfinder = slreportgen.finder.BlockFinder(model_name);  
results = find(blkfinder);
```

Then, loop through the results and test whether each block is a MATLAB Function block. For each result that is a MATLAB Function block, create a MATLABFunction reporter, and add the reporter to the report.

```
for i=1:length(results)  
    block = results(i).Object;  
    if slreportgen.utils.isMATLABFunction(block)  
        rptr = MATLABFunction(block);  
        add(myReport,rptr);  
    end  
end
```

### Input Arguments

**obj** — Object to check for being a MATLAB Function block

string | character array

Simulink element or Stateflow object to check for being a MATLAB Function block, specified as a string or character array of the path or handle.

### Output Arguments

**tf** — Whether input is a MATLAB Function block or object

true | false

Whether input is a MATLAB Function block or object, returned as 1 (true) if the input is a MATLAB Function or object. Otherwise, it returns 0 (false).

**See Also**

`slreportgen.report.MATLABFunction`

**Introduced in R2018a**

## slreportgen.utils.isModel

**Package:** slreportgen.utils

Check if object is model

### Syntax

```
tf = slreportgen.utils.isModel(obj)
```

### Description

`tf = slreportgen.utils.isModel(obj)` returns 1 (true) if the input object is a Simulink model and 0 (false) if the object is not a model.

### Examples

#### Check Whether Object is a Model

```
load_system('vdp');  
tf = slreportgen.utils.isModel('vdp')  
  
tf =  
  
    logical  
  
     1
```

### Input Arguments

#### obj — Object to check

handle | string scalar | character vector | handle

Object to check, specified as a handle to the object or as a character vector or string scalar that contains the Simulink path of the object.

### Output Arguments

#### tf — Whether object is a model

1 (true) | 0 (false)

Whether the input object is a model, returned as 1 (true) if the input object is a model or 0 (false) if the input object is not a model.

### See Also

slreportgen.utils.getModelHandle | slreportgen.utils.isModelCompiled |  
slreportgen.utils.isModelLoaded

**Introduced in R2020b**

## slreportgen.utils.isModelCompiled

**Package:** slreportgen.utils

Check if model is compiled

### Syntax

```
tf = slreportgen.utils.isModelCompiled(model)
```

### Description

`tf = slreportgen.utils.isModelCompiled(model)` tests whether the input Simulink model is compiled.

### Examples

#### Check If Model Is Compiled

```
load_system('sf_car')
tf = slreportgen.utils.isModelCompiled('sf_car')

tf =

    logical

     0
```

In this case, the `sf_car` model is not compiled.

### Input Arguments

#### **model** — Model to check for being compiled

string | character array

Model to check for being compiled, specified as a string or character array of the model path or handle.

### Output Arguments

#### **tf** — Whether model is compiled

true | false

Whether model is compiled, returned as 1 (true) if the model is compiled. Otherwise, it returns 0 (false).

### See Also

`slreportgen.utils.compileModel` | `slreportgen.utils.isModelLoaded` | `slreportgen.utils.uncompileModel`



**Introduced in R2018b**

## slreportgen.utils.isModelLoaded

**Package:** slreportgen.utils

Check if model is loaded

### Syntax

```
tf = slreportgen.utils.isModelLoaded(model)
```

### Description

`tf = slreportgen.utils.isModelLoaded(model)` tests whether the input Simulink model is loaded into memory.

### Examples

#### Check If Model Is Loaded

```
load_system('sf_car')
tf = slreportgen.utils.isModelLoaded('sf_car')

tf =

    logical

     0
```

In this case, the `sf_car` model is not loaded.

### Input Arguments

**model** — Model to check for being loaded

string | character array

Model to check for being loaded, specified as a string or character array of the model path or handle.

### Output Arguments

**tf** — Whether input model is loaded

true | false

Whether input model is loaded into memory, returned as 1 (true) if the input model is loaded. Otherwise, it returns 0 (false).

### See Also

`slreportgen.utils.getModelHandle`

**Introduced in R2018b**

# slreportgen.utils.isModelReferenceBlock

**Package:** slreportgen.utils

Check if object is Model block

## Syntax

```
tf = slreportgen.utils.isModelReferenceBlock(obj)
```

## Description

`tf = slreportgen.utils.isModelReferenceBlock(obj)` tests whether the input `obj` is a Model block.

## Examples

### Check If Object Is a Model Block

```
load_system('sf_car')
tf = slreportgen.utils.isModelReferenceBlock('sf_car/Engine')

tf =
    logical
     0
```

In this case, the Engine subsystem in the `sf_car` model is not a Model block.

## Input Arguments

**obj** — Object to check for being a Model block

string | character array

Object to check for being a Model block, specified as a string or character array of the Simulink object path or handle.

## Output Arguments

**tf** — Whether input is a Model block

true | false

Whether input is a Model block, returned as 1 (true) if the input is a Model block. Otherwise, it returns 0 (false).

## See Also

`Simulink.SubSystem.convertToModelReference`

**Introduced in R2018b**

# slreportgen.utils.isSID

**Package:** slreportgen.utils

Check if name is Simulink Identifier (SID) string

## Syntax

```
tf = slreportgen.utils.isSID(name)
```

## Description

`tf = slreportgen.utils.isSID(name)` tests whether the input name is a syntactically correct Simulink Identifier (SID).

## Examples

### Check If Input Uses Valid SID Syntax

```
load_system('f14')

sid_1 = slreportgen.utils.isSID('f14')
sid_2 = slreportgen.utils.isSID('f14/Controller')
sid_3 = slreportgen.utils.isSID('valid_syntax_not_valid_sid')
getsid_4 = Simulink.ID.getSID('f14/Controller');
sid_4 = slreportgen.utils.isSID(getsid_4)

sid_1 =
    logical
     1
sid_2 =
    logical
     0
sid_3 =
    logical
     1
sid_4 =
    logical
     1
```

## Input Arguments

**name** — Name to check for being an SID

string | character array

Name to check for being an SID, specified as a string or character array. This utility checks only that the syntax of the input is valid. It does not check whether the input is a valid SID.

## Output Arguments

**tf** — Whether input is an SID

true | false

Whether input is a syntactically correct SID, returned as 1 (true) if the input is an SID. Otherwise, it returns 0 (false).

## See Also

**Introduced in R2018b**

# slreportgen.utils.isStateTransitionTableBlock

**Package:** slreportgen.utils

Check if object is Transition Table block

## Syntax

```
tf = slreportgen.utils.isStateTransitionTableBlock(obj)
```

## Description

`tf = slreportgen.utils.isStateTransitionTableBlock(obj)` tests whether the input `obj` is a Stateflow State Transition Table block.

## Examples

### Check If Block Is a State Transition Table Block

```
load_system('sf_car')
tf = slreportgen.utils.isStateTransitionTableBlock...
    ('sf_car/shift_logic/downshifting')

tf =

    logical

     0
```

In this case, the downshifting object in the Stateflow `shift_logic` chart of the `sf_car` model is not a State Transition Table block.

## Input Arguments

**obj** — Object to check for being a Stateflow Transition Table block

string | character array

Object to check for being a Stateflow Transition Table block, specified as a string or character array of the object path or handle.

## Output Arguments

**tf** — Whether input is a State Transition Table block

true | false

Whether input is a State Transition Table block, returned as 1 (true) if the input is a State Transition Table block. Otherwise, it returns 0 (false).

## **See Also**

**Introduced in R2018b**



# slreportgen.utils.isTestSequence

**Package:** slreportgen.utils

Check if Test Sequence block or object

## Syntax

```
tf = slreportgen.utils.isTestSequence(obj)
```

## Description

`tf = slreportgen.utils.isTestSequence(obj)` tests if the input `obj` is a Simulink Test Sequence block or a Stateflow Test Sequence object.

## Examples

### Test If Object Is a Test Sequence

```
load_system('sltestTestSequenceDebouncerExample')
tf = slreportgen.utils.isTestSequence...
    ('sltestTestSequenceDebouncerExample/Debouncer_Test')

tf =

    logical

     1
```

In this case, the `Debouncer_Test` block in the `sltestTestSequenceDebouncerExample` model is a Test Sequence block.

## Input Arguments

**obj** — Object to check for being a Test Sequence block or object

string | character array | handle

Object to check for being a Test Sequence block or object, specified as a string or character array of the object path or handle.

## Output Arguments

**tf** — Whether input is a Test Sequence block or object

true | false

Whether input is a Test Sequence block or object, returned as 1 (true) if the input is a Test Sequence. Otherwise, it returns 0 (false).

## See Also

Test Sequence

**Introduced in R2019a**

# slreportgen.utils.isTruthTable

**Package:** slreportgen.utils

Check if object is Truth Table

## Syntax

```
tf = slreportgen.utils.isTruthTable(obj)
```

## Description

`tf = slreportgen.utils.isTruthTable(obj)` tests if the input `obj` is a Simulink Truth Table block or a Stateflow Truth Table object.

## Examples

### Check If Object Is a Truth Table

```
load_system('sf_climate_control')
tf = slreportgen.utils.isTruthTable...
    ('sf_climate_control/ClimateController')

tf =

    logical

     1
```

In this case, the `ClimateController` block in the `sf_climate_control` model is a Truth Table block.

## Input Arguments

**obj** — Object to check for being a Truth Table block or object

string | character array

Object to check for being a Truth Table block or object, specified as a string or character array of the object path or handle.

## Output Arguments

**tf** — Whether input is a Truth Table block or object

true | false

Whether input is a Truth Table block or object, returned as 1 (true) if the input system is a truth table. Otherwise, it returns 0 (false).

## See Also

slreportgen.report.TruthTable | slreportgen.utils.isLookupTable

**Introduced in R2018b**

# slreportgen.utils.isValidSlSystem

**Package:** slreportgen.utils

Check if system is valid Simulink system

## Syntax

```
tf = slreportgen.utils.isValidSlSystem(system)
```

## Description

`tf = slreportgen.utils.isValidSlSystem(system)` tests if the input `system` is a valid Simulink block diagram or subsystem block. Use this utility to determine whether a system is in memory.

## Input Arguments

**system** — System to check for validity

string | character array

System to check for validity, specified as a string or character array of the path or handle of a Simulink block diagram or subsystem block.

## Output Arguments

**tf** — Whether system is valid

true | false

Whether system is valid, returned as 1 (true) if the input system is valid. Otherwise, it returns 0 (false).

## See Also

**Introduced in R2018b**

## slreportgen.utils.loadAllSystems

**Package:** slreportgen.utils

Load all systems

### Syntax

```
slreportgen.utils.loadAllSystems(name)
```

### Description

`slreportgen.utils.loadAllSystems(name)` loads all systems, including masking subsystems and libraries, into memory for the specified Simulink model name. If the input name is not a model and is, for example, a block, this utility obtains the containing model and then, loads all of its systems.

### Input Arguments

**name** — Name or handle of Simulink model or Stateflow object

string | character array

Name or handle of Simulink model or Stateflow object, specified as a string or character array, for which to load all of its systems.

### See Also

`slreportgen.utils.isValidSlSystem`

**Introduced in R2018b**

# slreportgen.utils.pathJoin

**Package:** slreportgen.utils

Combine two diagram path parts

## Syntax

```
diagrampath = slreportgen.utils.pathJoin(parent,name)
```

## Description

`diagrampath = slreportgen.utils.pathJoin(parent,name)` combines the parent and name paths to create a full Simulink diagram path. Any newlines are converted to spaces.

## Examples

### Combine Parent Diagram Path and Diagram Name

```
slreportgen.utils.pathJoin("sf_car/transmission",...  
    "transmission ratio")
```

```
diagpath =
```

```
    "sf_car/transmission/transmission ratio"
```

## Input Arguments

### **parent** — Parent diagram path

string | character array

Parent diagram path, specified as a string or character array.

### **name** — Diagram name

string | character array

Diagram name, specified as a string or character array.

## Output Arguments

### **diagrampath** — Full diagram path

string

Full diagram path, returned as a string.

## See Also

slreportgen.utils.pathParts | slreportgen.utils.pathSplit

**Introduced in R2018b**

## slreportgen.utils.pathParts

**Package:** slreportgen.utils

Split diagram path into parent and diagram parts

### Syntax

```
[parent,name] = slreportgen.utils.pathParts(diagramPath)
```

### Description

[parent,name] = slreportgen.utils.pathParts(diagramPath) splits a Simulink diagram path into its parent diagram path and its diagram name.

### Examples

#### Split Full Diagram Path into Parent and Diagram Parts

```
[parent,name] = slreportgen.utils.pathParts...  
('sf_car/transmission/transmission ratio')
```

```
parent =
```

```
    'sf_car/transmission'
```

```
name =
```

```
    'transmission ratio'
```

### Input Arguments

#### diagramPath — Full diagram path

string | character array

Full diagram path, specified as a string or character array.

### Output Arguments

#### parent — Parent diagram path

string

Parent diagram path, returned as a string.

#### name — Diagram name

string

Diagram name, returned as a string.

### See Also

slreportgen.utils.pathJoin | slreportgen.utils.pathSplit



**Introduced in R2018b**

## slreportgen.utils.pathSplit

**Package:** slreportgen.utils

Split diagram path into array of diagram parts

### Syntax

```
parts = slreportgen.utils.pathSplit(diagrampath)
```

### Description

`parts = slreportgen.utils.pathSplit(diagrampath)` splits the input Simulink diagram path into a string array of diagram names.

### Examples

#### Split Diagram Path Into Separate Parts

```
parts = slreportgen.utils.pathSplit...  
    ("sf_car/transmission/transmission ratio")
```

```
parts =
```

```
    1×3 string array
```

```
    "sf_car"    "transmission"    "transmission ratio"
```

### Input Arguments

#### **diagrampath** — Full diagram path

string | character array

Full diagram path, specified as a string or character array.

### Output Arguments

#### **parts** — Diagram path parts

string array

Diagram path parts, returned as a string array.

### See Also

`slreportgen.utils.pathJoin` | `slreportgen.utils.pathParts`

**Introduced in R2018b**

# slreportgen.utils.traceSignal

**Package:** slreportgen.utils

Trace input signal to its source

## Syntax

```
[srcblock,srcport,srcportnum] = slreportgen.utils.traceSignal(inport)
```

## Description

[srcblock,srcport,srcportnum] = slreportgen.utils.traceSignal(inport) traces the signal entering the port of the inPort element to its source block, source port, and source port number. This utility traces a signal to its nonvirtual source, which means that it traces a signal originating in a subsystem to its source in that subsystem. It ignores the output port blocks of the subsystem.

## Examples

### Trace Block Input Signal

```
model= 'f14';
load_system(model)
srcBlock = 'f14/Aircraft Dynamics Model/Transfer Fcn.1';
ports = get_param(srcBlock, 'PortHandles');
[sb,sp,spn] = slreportgen.utils.traceSignal(ports.Inport);
fprintf('Block whose input to trace: %s\n',srcBlock)
fprintf('Source block: %s\n',sb)
fprintf('Source port number: %d\n',spn)
bdclose(model)
```

```
Block whose input to trace: f14/Aircraft Dynamics Model/Transfer Fcn.1
Source block: f14/Aircraft
Dynamics
Model/Sum2
Source port number: 1
```

## Input Arguments

**inport** — Port handles

string | character array

Port handles, specified as a string or character array of Simulink handles.

## Output Arguments

**srcblock** — Signal source block

character array

Signal source block, returned as a character array. If the signal cannot be traced, this utility returns -1 as the value of `srcblock`.

**srcport – Signal source port**

Simulink handle

Signal source port value, returned as a Simulink handle.

**srcportnum – Signal source port number**

integer

Signal source port number, returned as an integer.

**See Also**

**Introduced in R2018b**

# slreportgen.utils.uncompileModel

**Package:** slreportgen.utils

Uncompile model

## Syntax

```
slreportgen.utils.uncompileModel(modelname)
```

## Description

`slreportgen.utils.uncompileModel(modelname)` uncompiles the model specified in the `modelname` input. If `modelname` is a Simulink block or Stateflow object, the model that contains that block or object is uncompiled.

## Input Arguments

**modelname — Model to uncompile**

character array | string

Model to uncompile, specified as a character array or string of the model handle, block handle, or Stateflow objects. If `modelname` is an element in the model, the containing model is uncompiled.

## See Also

`slreportgen.utils.compileModel` | `slreportgen.utils.isModelCompiled`

**Introduced in R2018b**

## createDiagramLink

**Class:** `slreportgen.webview.EmbeddedWebViewDocument`

**Package:** `slreportgen.webview`

Link to embedded Web view report

### Syntax

```
diaglink = createDiagramLink(wvdoc,dhandle,domlabel)
```

### Description

`diaglink = createDiagramLink(wvdoc,dhandle,domlabel)` updates a DOM object in an embedded Web view Document panel so that it links to a diagram anchor handle in the Simulink Web view. The `diaglink` DOM object is of the same type as `domlabel` or if `domlabel` is a string, an `mlreportgen.DOM.Text` object is created.

### Input Arguments

**wvdoc — Web view document**

`slreportgen.webview.WebViewDocument` object

Web view document, specified as an `slreportgen.webview.WebViewDocument` object.

**dhandle — Handle of Web view diagram anchor**

character vector | object handle

Handle of Web view diagram anchor, specified as a character vector of the path or as an object handle. You can use the `getExportDiagrams` method to obtain the diagram paths and handles.

Example: Character vector: `'vdp'`. Object handle: `get_param('vdp','handle')`

**domlabel — DOM object from which to link**

DOM object | character vector

DOM object from which to link, specified as a valid DOM object or as a character vector. If you enter a character vector, an `mlreportgen.DOM.Text` object is created.

### Output Arguments

**diaglink — Diagram link**

DOM object

Diagram link to the specified Simulink Web view diagram, returned as a DOM object. The DOM object has an attribute that marks it as a link.

### Examples

## Create a Link to a Web View

Use `createDiagramLink` to create links from level-two headings in the document pane to the associated diagrams in the embedded web view. This example also uses `createElementLink` to create links from block names in the document pane to blocks in the embedded web view.

Write a class, `ExampleWebView`, that is a subclass of `slreportgen.webview.EmbeddedWebViewDocument`. Use `createDiagramLink` and `createElementLink` in the `fillContent` method.

```
classdef ExampleWebView < slreportgen.webview.EmbeddedWebViewDocument

    methods
        function wvdoc = ExampleWebView(reportPath,modelName)
            % Invoke the EmbeddedWebViewDocument constructor, which
            % saves the report path and model name for use by the
            % report's fill methods.
            wvdoc@slreportgen.webview.EmbeddedWebViewDocument(reportPath,modelName);
        end

        function fillContent(wvdoc)
            % Fill the Content hole in the report template with design
            % variable information. You can use DOM or Report API methods
            % to create, format, add, and append content to this report.

            [~, handles] = getExportDiagrams(wvdoc);

            n = numel(handles);
            for i = 1:n
                diagHandle = handles{i};
                diagHeading = createDiagramLink(wvdoc,diagHandle, ...
                    mlreportgen.dom.Heading(2,get_param(diagHandle,'Name')));
                append(wvdoc,diagHeading);

                blockFinder = slreportgen.finder.BlockFinder(diagHandle);

                while hasNext(blockFinder)
                    r = next(blockFinder);
                    elemHandle = r.Object;
                    elemHeading = createElementLink(wvdoc,elemHandle, ...
                        mlreportgen.dom.Heading(3,get_param(elemHandle,'Name')));

                    append(wvdoc,elemHeading);
                end
            end
        end
    end
end
```

Create an object of the `ExampleWebView` class and use its methods to generate the embedded web view report.

```
model = 'vdp';
open_system(model);
wvdoc = ExampleWebView('myReport',model);
open(wvdoc);
fill(wvdoc);
```

```
close(wvdoc);  
rptview(wvdoc);
```

## **More About**

### **Diagram**

Diagram refers to a Simulink model, subsystem, or Stateflow chart.

### **Element**

Element refers to an individual item within a diagram, such as a block, annotation, state, or transition.

## **See Also**

[createDiagramTwoWayLink](#) | [createElementLink](#) | [createElementTwoWayLink](#) | [getExportDiagrams](#) | [slreportgen.webview.EmbeddedWebViewDocument](#) | [slreportgen.webview.WebViewDocument](#)

### **Topics**

“Create Hyperlinks for Embedded Web View Report” on page 5-28

“Embedded Web View Reports” on page 5-19

“Create an Embedded Web View Report Generator” on page 5-23

### **Introduced in R2017a**



# createDiagramTwoWayLink

**Class:** slreportgen.webview.EmbeddedWebViewDocument

**Package:** slreportgen.webview

Link and anchor in embedded Web view report

## Syntax

```
diag2link = createDiagramTwoWayLink(wvdoc,dhandle,domlabel)
```

## Description

`diag2link = createDiagramTwoWayLink(wvdoc,dhandle,domlabel)` creates a two-way connection between a Simulink Web view diagram and a DOM object in an embedded Web view Document panel. The `diag2link` DOM object is updated to include attributes that identifies it as a link. The `diag2link` DOM object is of the same type as `domlabel` or if `domlabel` is a string, an `mlreportgen.DOM.Text` object is created.

## Input Arguments

### **wvdoc** — Web view document

slreportgen.webview.WebViewDocument object

Web view document, specified as an `slreportgen.webview.WebViewDocument` object.

### **dhandle** — Handle of Web view diagram anchor

character vector | object handle

Handle of Web view diagram anchor, specified as a character vector of the path or as an object handle. You can use the `getExportDiagrams` method to obtain the diagram paths and handles.

Example: Character vector: 'vdp'. Object handle: `get_param('vdp','handle')`

### **domlabel** — DOM object from which to link

DOM object | character vector

DOM object from which to link, specified as a valid DOM object or as a character vector. If you enter a character vector, an `mlreportgen.DOM.Text` object is created.

## Output Arguments

### **diag2link** — Diagram link and Web view anchor

DOM object

Diagram link and Web view anchor, returned as a DOM object. The `diag2link` object connects a DOM object in the embedded Web view Document panel to the specified Simulink web view diagram. The DOM object is updated with an attribute that indicates it is a link to the anchor.

## Examples

### Create Two-Way Diagram Link

Use `createDiagramTwoWayLink` to create two-way links between level-two headings in the document pane and the associated diagrams in the embedded web view. This example also uses `createElementTwoWayLink` to create links between block names in the document pane and blocks in the embedded web view.

Write a class, `ExampleWebView`, that is a subclass of `slreportgen.webview.EmbeddedWebViewDocument`. Use `createDiagramTwoWayLink` and `createElementTwoWayLink` in the `fillContent` method.

```
classdef ExampleWebView< slreportgen.webview.EmbeddedWebViewDocument

    methods
        function wvdoc = ExampleWebView(reportPath,modelName)
            % Invoke the EmbeddedWebViewDocument constructor, which
            % saves the report path and model name for use by the
            % report's fill methods.
            wvdoc@slreportgen.webview.EmbeddedWebViewDocument(reportPath,modelName);
        end

        function fillContent(wvdoc)
            % Fill the Content hole in the report template with design
            % variable information. You can use DOM or Report API methods
            % to create, format, add, and append content to this report.

            [~, handles] = getExportDiagrams(wvdoc);

            n = numel(handles);
            for i = 1:n
                diagHandle = handles{i};
                diagHeading = createDiagramTwoWayLink(wvdoc,diagHandle, ...
                    mlreportgen.dom.Heading(2,get_param(diagHandle,'Name')));
                append(wvdoc,diagHeading);

                blockFinder = slreportgen.finder.BlockFinder(diagHandle);

                while hasNext(blockFinder)
                    r = next(blockFinder);
                    elemHandle = r.Object;
                    elemHeading = createElementTwoWayLink(wvdoc,elemHandle, ...
                        mlreportgen.dom.Heading(3,get_param(elemHandle,'Name')));

                    append(wvdoc,elemHeading);
                end
            end
        end
    end
end
```

Create an object of the `ExampleWebView` class and use its methods to generate the embedded web view report.

```
model = 'vdp';  
open_system(model);  
wvdoc = ExampleWebView('myReport',model);  
open(wvdoc);  
fill(wvdoc);  
close(wvdoc);  
rptview(wvdoc);
```

## More About

### Diagram

Diagram refers to a Simulink model, subsystem, or Stateflow chart.

### Element

Element refers to an individual item within a diagram, such as a block, annotation, state, or transition.

## See Also

[createDiagramLink](#) | [createElementLink](#) | [createElementTwoWayLink](#) | [getExportDiagrams](#) | [slreportgen.webview.EmbeddedWebViewDocument](#) | [slreportgen.webview.WebViewDocument](#)

### Topics

“Create Hyperlinks for Embedded Web View Report” on page 5-28

“Embedded Web View Reports” on page 5-19

“Create an Embedded Web View Report Generator” on page 5-23

### Introduced in R2017a

## createElementLink

**Class:** `slreportgen.webview.EmbeddedWebViewDocument`

**Package:** `slreportgen.webview`

Element link in embedded Web view report

### Syntax

```
elemLink = createElementLink(wvdoc,ehandle,domlabel)
```

### Description

`elemLink = createElementLink(wvdoc,ehandle,domlabel)` updates a DOM object in an embedded Web view Document panel so that it links to an element anchor in the Simulink Web view. The DOM object is of the same type as `domlabel` or, if `domlabel` is a string, an `mlreportgen.DOM.Text` object is created.

### Input Arguments

**wvdoc — Web view document**

`slreportgen.webview.WebViewDocument` object

Web view document, specified as an `slreportgen.webview.WebViewDocument` object.

**ehandle — Handle of Web view element anchor**

character vector | object handle

Handle of Web view element anchor, specified as a character vector of the path or as an object handle. You can use the `getExportDiagrams` method to obtain the element paths and handles.

Example: Character vector: `'vdp/mu'`. Object handle: `get_param('vdp/mu','handle')`

**domlabel — DOM object from which to link**

DOM object | character vector

DOM object from which to link, specified as a valid DOM object or as a character vector. If you enter a character vector, an `mlreportgen.DOM.Text` object is created.

### Output Arguments

**elemLink — Element link**

array of character vectors

Element link to the specified Simulink Web view element, returned as a DOM object. The DOM object has an attribute that marks it as a link.

### Examples

## Create Link from Element to Diagram

Use `CreateElementLink` to create links from block names in the document pane to blocks in the diagram in the embedded web view. This example also uses `createDiagramLink` to create links from level two headings in the document pane to diagrams in the embedded web view.

Write a class, `ExampleWebView`, that is a subclass of `slreportgen.webview.EmbeddedWebViewDocument`. Use `CreateElementLink` and `createDiagramLink` in the `fillContent` method.

```
classdef ExampleWebView < slreportgen.webview.EmbeddedWebViewDocument

    methods
        function wvdoc = ExampleWebView(reportPath,modelName)
            % Invoke the EmbeddedWebViewDocument constructor, which
            % saves the report path and model name for use by the
            % report's fill methods.
            wvdoc@slreportgen.webview.EmbeddedWebViewDocument(reportPath,modelName);
        end

        function fillContent(wvdoc)
            % Fill the Content hole in the report template with design
            % variable information. You can use DOM or Report API methods
            % to create, format, add, and append content to this report.

            [~, handles] = getExportDiagrams(wvdoc);

            n = numel(handles);
            for i = 1:n
                diagHandle = handles{i};
                diagHeading = createDiagramLink(wvdoc,diagHandle, ...
                    mlreportgen.dom.Heading(2,get_param(diagHandle,'Name')));
                append(wvdoc,diagHeading);

                blockFinder = slreportgen.finder.BlockFinder(diagHandle);

                while hasNext(blockFinder)
                    r = next(blockFinder);
                    elemHandle = r.Object;
                    elemHeading = createElementLink(wvdoc,elemHandle, ...
                        mlreportgen.dom.Heading(3,get_param(elemHandle,'Name')));
                    append(wvdoc,elemHeading);
                end
            end
        end
    end
end
```

Create an object of the `ExampleWebView` class and use its methods to generate the embedded web view report.

```
model = 'vdp';
open_system(model);
wvdoc = ExampleWebView('myReport',model);
open(wvdoc);
fill(wvdoc);
```

```
close(wvdoc);  
rptview(wvdoc);
```

## **More About**

### **Diagram**

Diagram refers to a Simulink model, subsystem, or Stateflow chart.

### **Element**

Element refers to an individual item within a diagram, such as a block, annotation, state, or transition.

## **See Also**

[createDiagramLink](#) | [createDiagramTwoWayLink](#) | [createElementTwoWayLink](#) | [getExportDiagrams](#) | [slreportgen.webview.EmbeddedWebViewDocument](#) | [slreportgen.webview.WebViewDocument](#)

### **Topics**

“Create Hyperlinks for Embedded Web View Report” on page 5-28

“Embedded Web View Reports” on page 5-19

“Create an Embedded Web View Report Generator” on page 5-23

### **Introduced in R2017a**

# createElementTwoWayLink

**Class:** slreportgen.webview.EmbeddedWebViewDocument

**Package:** slreportgen.webview

Element link and anchor in embedded Web view report

## Syntax

```
elem2link = createElementTwoWayLink(wvdoc,ehandle,domlabel)
```

## Description

`elem2link = createElementTwoWayLink(wvdoc,ehandle,domlabel)` creates a two-way connection between a Simulink Web view element and a DOM object in an embedded Web view Document panel. The `elem2link` DOM object is updated to include attributes that identifies it as a link. The `elem2link` DOM object is of the same type as `domlabel` or if `domlabel` is a string, an `mlreportgen.DOM.Text` object is created.

## Input Arguments

### **wvdoc — Web view document**

slreportgen.webview.WebViewDocument object

Web view document, specified as an `slreportgen.webview.WebViewDocument` object.

### **ehandle — Handle of Web view element anchor**

character vector | object handle

Handle of Web view element anchor, specified as a character vector of the path or as an object handle. You can use the `getExportDiagrams` method to obtain the element paths and handles.

Example: Character vector: 'vdp/mu'. Object handle: `get_param('vdp/mu','handle')`

### **domlabel — DOM object from which to link**

DOM object | character vector

DOM object from which to link, specified as a valid DOM object or as a character vector. If you enter a character vector, an `mlreportgen.DOM.Text` object is created.

## Output Arguments

### **elem2link — Element link and Web view anchor**

DOM object

Element link and Web view anchor, returned as a DOM object. The `elem2link` object connects a DOM object in the embedded Web view Document panel to the specified Simulink Web view element. The DOM object is updated with an attribute that indicates it is a link to the anchor.

## Examples

### Create Two-Way Element Link in Embedded Web View Report

Use `CreateElementTwoWayLink` to create two-way links between block names in the document pane and blocks in the diagram in the embedded web view. This example also uses `createDiagramTwoWayLink` to create links between level two headings in the document pane and diagrams in the embedded web view.

Write a class, `ExampleWebView`, that is a subclass of `slreportgen.webview.EmbeddedWebViewDocument`. Use `CreateElementTwoWayLink` and `createDiagramTwoWayLink` in the `fillContent` method.

```
classdef ExampleWebView< slreportgen.webview.EmbeddedWebViewDocument

    methods
        function wvdoc = ExampleWebView(reportPath,modelName)
            % Invoke the EmbeddedWebViewDocument constructor, which
            % saves the report path and model name for use by the
            % report's fill methods.
            wvdoc@slreportgen.webview.EmbeddedWebViewDocument(reportPath,modelName);
        end

        function fillContent(wvdoc)
            % Fill the Content hole in the report template with design
            % variable information. You can use DOM or Report API methods
            % to create, format, add, and append content to this report.

            [~, handles] = getExportDiagrams(wvdoc);

            n = numel(handles);
            for i = 1:n
                diagHandle = handles{i};
                diagHeading = createDiagramTwoWayLink(wvdoc,diagHandle, ...
                    mlreportgen.dom.Heading(2,get_param(diagHandle,'Name')));
                append(wvdoc,diagHeading);

                blockFinder = slreportgen.finder.BlockFinder(diagHandle);

                while hasNext(blockFinder)
                    r = next(blockFinder);
                    elemHandle = r.Object;
                    elemHeading = createElementTwoWayLink(wvdoc,elemHandle, ...
                        mlreportgen.dom.Heading(3,get_param(elemHandle,'Name')));

                    append(wvdoc,elemHeading);
                end
            end
        end
    end
end
```

Create an object of the `ExampleWebView` class and use its methods to generate the embedded web view report.



```
model = 'vdp';  
open_system(model);  
wvdoc = ExampleWebView('myReport',model);  
open(wvdoc);  
fill(wvdoc);  
close(wvdoc);  
rptview(wvdoc);
```

## More About

### Diagram

Diagram refers to a Simulink model, subsystem, or Stateflow chart.

### Element

Element refers to an individual item within a diagram, such as a block, annotation, state, or transition.

## See Also

[createDiagramLink](#) | [createDiagramTwoWayLink](#) | [createElementLink](#) | [getExportDiagrams](#) | [slreportgen.webview.EmbeddedWebViewDocument](#) | [slreportgen.webview.WebViewDocument](#)

### Topics

“Create Hyperlinks for Embedded Web View Report” on page 5-28

“Embedded Web View Reports” on page 5-19

“Create an Embedded Web View Report Generator” on page 5-23

### Introduced in R2017a

## getReportObject

**Class:** `slreportgen.webview.EmbeddedWebViewDocument`

**Package:** `slreportgen.webview`

Returns the report object for an embedded web view report

### Syntax

```
rptobj = getReportObject(rpt)
```

### Description

`rptobj = getReportObject(rpt)` returns the `slreportgen.report.Report` object associated with an embedded web view report. You can use the report object to get the DOM object that implements a reporter in your report. Examining the DOM implementation can help you to debug report generation issues.

### Input Arguments

**rpt — Embedded web view report**

object of subclass of `slreportgen.webview.EmbeddedWebViewDocument`

Embedded web view report, specified as an object of a subclass of `slreportgen.webview.EmbeddedWebViewDocument`.

### Output Arguments

**rptobj — Report object**

`slreportgen.report.Report`

Report object, returned as an `slreportgen.report.Report` object.

### Examples

#### Get DOM Implementation of a Reporter

Get the report object for an embedded web view report by calling the `getReportObject` method. Then, get the DOM object that implements the title page reporter in the report.

Create the embedded web view class used in “Generate an Embedded Web View Report” on page 5-34.

```
classdef SystemDesignVariables < slreportgen.webview.EmbeddedWebViewDocument
    methods
        function rpt = SystemDesignVariables(reportPath, modelName)
            rpt@slreportgen.webview.EmbeddedWebViewDocument(reportPath, ...
```

```

        modelName);

    rpt.ValidateLinksAndAnchors = false;

    rpt.ExportOptions.IncludeMaskedSubsystems = true;
    rpt.ExportOptions.IncludeSimulinkLibraryLinks = true;
    rpt.ExportOptions.IncludeReferencedModels = true;
end

function fillContent(rpt)
    import mlreportgen.dom.*
    import mlreportgen.report.*

    model = getExportModels(rpt);
    model= model{1};
    tp = TitlePage("Title",[model " Report"],"Author","");
    add(rpt,tp);
    finder = slreportgen.finder.ModelVariableFinder(model);
    ch = Chapter("Variables");
    while hasNext(finder)
        result = next(finder);
        s = Section(result.Name);
        reporter = getReporter(result);
        add(s,reporter);
        add(ch,s);
    end
    add(rpt,ch);
end
end
end
end
end

```

Using the MATLAB Editor, set a breakpoint at this line:

```
add(rpt,tp);
```

Run a script to generate the embedded web view report.

```

model = 'f14';
rptName = sprintf('%sVariables', model);
load_system(model);
rpt = SystemDesignVariables(rptName, model);
fill(rpt);
close(rpt);
close_system(model);
rptview(rptName);

```

MATLAB pauses at the breakpoint.

In the Editor, at the command prompt, run these commands:

```

rptObj = getReportObject(rpt);
impl = getImpl(tp,rptObj)

```

The DOM implementation for the title page reporter displays.

To end the debugging session, click **Quit Debugging**.

To clear the breakpoint, right-click the breakpoint icon and select **Clear Breakpoint** from the context menu.

### **See Also**

`slreportgen.report.Report`

### **Topics**

“Generate an Embedded Web View Report” on page 5-34

“Debug a MATLAB Program”

**Introduced in R2019b**

# fill

**Class:** `slreportgen.webview.WebViewDocument`

**Package:** `slreportgen.webview`

Fill report holes

## Syntax

```
fill(wvdoc)
```

## Description

`fill(wvdoc)` fills the holes in the report generated by `wvdoc`. This method loops through the holes in the report's template. For each hole, it determines if this object has a method named `fillId(wvdoc)`, where **Id** is the id of the hole. If the method exists, this method invokes the method to fill the hole.

## Input Arguments

**wvdoc — Web view document**

`slreportgen.webview.WebViewDocument` object

Web view document, specified as an `slreportgen.webview.WebViewDocument` object.

## Tips

- The default template specified by `wvdoc` contains a hole named `slwebview`. Thus invoking this method will invoke `slreportgen.webview.WebViewDocument.fillslwebview` to fill the hole with web views of the model(s) specified by `wvdoc`'s `ExportOptions` property.
- This method allows you to fill holes in a custom template by deriving a class from `slreportgen.webview.WebViewDocument` and providing the derived class with `fillId` methods to fill the holes.

## append

**Class:** `slreportgen.webview.WebViewDocument`

**Package:** `slreportgen.webview`

Append content to a web view document

### Syntax

```
append(wvdoc, content)
```

### Description

`append(wvdoc, content)` appends content to the web view document. You can append any type of content that can be appended to an `mlreportgen.dom.Document` object.

### Input Arguments

**wvdoc — Web view document**

`slreportgen.webview.WebViewDocument` object

Web view document, specified as an `slreportgen.webview.WebViewDocument` object.

**content — Objects to be appended to the web view document**

Any MATLAB or DOM objects that can be appended to an `mlreportgen.dom.Document` object.

The value of this object can be any type of MATLAB or DOM API object that can be appended to a DOM document.

### Tips

- Use this method to fill holes in a custom template that you create to be used with `slreportgen.webview.WebViewDocument` or `slreportgen.webview.EmbeddedWebViewDocument` objects.
- To fill the Content hole defined by the default template specified by a class that you derive from `slreportgen.webview.EmbeddedWebViewDocument` class, define a `fillContent` method in your derived class and use this method in the `fillContent` method to fill the hole with document content. See `fill` for more information.

## fillslwebview

**Class:** slreportgen.webview.WebViewDocument

**Package:** slreportgen.webview

Create and insert a web view in an HTML document

### Syntax

```
fillslwebview(wvdocgen)
```

### Description

`fillslwebview(wvdocgen)` creates the web view specified by the `ExportOptions` property of the specified web view document generator (i.e., `wvdocgen`) and inserts the web view in the generator's output document at the location specified by an `slwebview` hole in the generator's document template.

### Input Arguments

**wvdocgen — Web view document**

slreportgen.webview.WebViewDocument object

Web view document, specified as an `slreportgen.webview.WebViewDocument` object.

### See Also

`fill`

**Introduced in R2017a**

## getExportDiagrams

**Class:** slreportgen.webview.WebViewDocument

**Package:** slreportgen.webview

Get names of diagram paths and handles to export

### Syntax

```
[paths,handles] = getExportDiagrams(wvdoc)
```

### Description

[paths,handles] = getExportDiagrams(wvdoc) returns an array of diagram paths and handles to export.

### Examples

#### Get Diagram Paths and Handles

Get the paths and handles for a Web view of the f14 Simulink model and subsystems.

```
f14
wvdoc = slreportgen.webview.WebViewDocument('myWebview','f14');
[paths,handles] = getExportDiagrams(wvdoc)
```

paths =

5×1 cell array

```
'f14'
'f14/Aircraft Dynamics Model'
'f14/Controller'
'f14/Dryden Wind Gust Models'
'f14/Nz pilot calculation'
```

handles =

5×1 cell array

```
[ 2.0001]
[ 7.0001]
[39.0001]
[69.0004]
[84.0001]
```

### Input Arguments

**wvdoc** — Web view document

slreportgen.webview.WebViewDocument object



Web view document, specified as an `slreportgen.webview.WebViewDocument` object.

## Output Arguments

### **paths** — Diagram paths

cell array of character vectors

Diagram paths in the model, including the model name and its subsystem names, returned as a cell array of character vectors.

### **handles** — Diagram handles

array of character vectors

Diagram handles that correspond to the diagram paths. returned as an array of character vectors.

## More About

### **Diagram**

Diagram refers to a Simulink model, subsystem, or Stateflow chart.

### **Element**

Element refers to an individual item within a diagram, such as a block, annotation, state, or transition.

### **See Also**

[getExportModels](#) | [getExportSimulinkSubSystems](#) | [getExportStateflowCharts](#) | [getExportStateflowDiagrams](#)

**Introduced in R2017a**

## getExportModels

**Class:** slreportgen.webview.WebViewDocument

**Package:** slreportgen.webview

Model paths and handles to export

### Syntax

```
[paths,handles] = getExportModels(wvdoc)
```

### Description

[paths,handles] = getExportModels(wvdoc) returns an array of model paths and handles to export.

### Input Arguments

**wvdoc** — Web view document

slreportgen.webview.WebViewDocument object

Web view document, specified as an slreportgen.webview.WebViewDocument object.

### Output Arguments

**paths** — Model paths

array of character vectors

Model paths in the model, returned as an array of character vectors.

**handles** — Diagram handles

array of character vectors

Diagram handles that correspond to the diagram paths. returned as an array of character vectors.

### Examples

#### Get Model Paths and Handles

Get the path and handle for a Web view of the f14 Simulink model.

```
f14
wvdoc = slreportgen.webview.WebViewDocument('myWebview','f14');
[path,handle] = getExportModels(wvdoc)

path =
    cell
    'f14'
```

```
handle =  
    100.0002
```

**See Also**

[getExportDiagrams](#) | [getExportSimulinkSubSystems](#) | [getExportStateflowCharts](#) | [getExportStateflowDiagrams](#)

**Introduced in R2017a**

## getExportSimulinkSubSystems

**Class:** slreportgen.webview.WebViewDocument

**Package:** slreportgen.webview

Subsystem paths and handles to export

### Syntax

```
[paths,handles] = getExportDiagrams(wvdoc)
```

### Description

[paths,handles] = getExportDiagrams(wvdoc) returns a cell array of subsystem paths and handles to export.

### Input Arguments

**wvdoc** — Web view document

slreportgen.webview.WebViewDocument object

Web view document, specified as an slreportgen.webview.WebViewDocument object.

### Output Arguments

**paths** — Subsystem paths

cell array of character vectors

Subsystem paths in the model, returned as a cell array of character vectors.

**handles** — Subsystem handles

array of character vectors

Subsystem handles that correspond to the Subsystem paths, returned as an array of character vectors.

### Examples

#### Get Simulink Subsystems Paths and Handles

Get the paths and handles for a Web view of the f14 Simulink subsystems.

```
f14
wvdoc = slreportgen.webview.WebViewDocument('myWebview','f14');
[paths,handles] = getExportSimulinkSubSystems(wvdoc)
```

```
paths =
```

```
    4×1 cell array
```

```
'f14/Aircraft Dynamics Model'  
'f14/Controller'  
'f14/Dryden Wind Gust Models'  
'f14/Nz pilot calculation'
```

handles =

```
105.0001  
137.0001  
167.0004  
182.0001
```

## More About

### Diagram

Diagram refers to a Simulink model, subsystem, or Stateflow chart.

### Element

Element refers to an individual item within a diagram, such as a block, annotation, state, or transition.

## See Also

[getExportDiagrams](#) | [getExportModels](#) | [getExportStateflowCharts](#) | [getExportStateflowDiagrams](#)

### Introduced in R2017a

## getExportStateflowCharts

**Class:** slreportgen.webview.WebViewDocument

**Package:** slreportgen.webview

Stateflow chart paths and handles to export

### Syntax

```
[paths,handles] = getExportStateflowCharts(wvdoc)
```

### Description

[paths,handles] = getExportStateflowCharts(wvdoc) returns an array of the Stateflow chart paths and handles at the top level of the model to export.

### Input Arguments

**wvdoc — Web view document**

slreportgen.webview.WebViewDocument object

Web view document, specified as an slreportgen.webview.WebViewDocument object.

### Output Arguments

**paths — Stateflow chart paths**

cell array of character vectors

Stateflow chart paths in the model, returned as a cell array of character vectors.

**handles — Diagram handles**

array of character vectors

Stateflow chart handles that correspond to the paths. returned as an array of character vectors.

### Examples

#### Get Stateflow Charts Paths and Handles

Get the paths and handles for Stateflow charts in the sf\_cdplayer Simulink model.

```
sf_cdplayer
wvdoc = slreportgen.webview.WebViewDocument('myWebview','sf_cdplayer');
[paths,handles] = getExportStateflowCharts(wvdoc)
```

```
paths =
```

```
    3×1 cell array
```

```
    'sf_cdplayer/CdPlayerBehaviorModel'
```

```
'sf_cdplayer/CdPlayerModeManager'  
'sf_cdplayer/UserRequest'
```

```
handles =
```

```
Stateflow.Chart: 1-by-3
```

## **See Also**

[getExportDiagrams](#) | [getExportModels](#) | [getExportSimulinkSubSystems](#) | [getExportStateflowDiagrams](#)

**Introduced in R2017a**

## getExportStateflowDiagrams

**Class:** slreportgen.webview.WebViewDocument

**Package:** slreportgen.webview

Stateflow chart hierarchy paths and handles to export

### Syntax

```
[paths,handles] = getExportStateflowDiagrams(wvdoc)
```

### Description

[paths,handles] = getExportStateflowDiagrams(wvdoc) returns an array of Stateflow chart paths and handles to export. The paths and handles are returned for both charts at the top level of the model and all charts in the hierarchy.

### Input Arguments

**wvdoc** — Web view document

slreportgen.webview.WebViewDocument object

Web view document, specified as an slreportgen.webview.WebViewDocument object.

### Output Arguments

**paths** — Stateflow diagram paths

cell array of character vectors

Stateflow diagram paths in the model, returned as a cell array of character vectors.

**handles** — Stateflow diagram handles

array of character vectors

Stateflow diagram handles that correspond to the Stateflow diagram paths. returned as an array of character vectors.

### Examples

#### Get Stateflow Diagrams Paths and Handles

Get the paths and handles for Stateflow diagrams in the sf\_cdplayer Simulink model.

```
sf_cdplayer
wvdoc = slreportgen.webview.WebViewDocument('myWebview','sf_cdplayer');
[paths,handles] = getExportStateflowDiagrams(wvdoc)

paths =
```



7×1 cell array

```
'sf_cdplayer/CdPlayerBehaviorModel'  
'sf_cdplayer/CdPlayerModeManager'  
'sf_cdplayer/CdPlayerModeManager/AMMode'  
'sf_cdplayer/CdPlayerModeManager/CDMode'  
'sf_cdplayer/CdPlayerModeManager/CDMode/Play'  
'sf_cdplayer/CdPlayerModeManager/FMMode'  
'sf_cdplayer/UserRequest'
```

handles =

```
Stateflow.Object: 1-by-7
```

## See Also

[getExportDiagrams](#) | [getExportModels](#) | [getExportSimulinkSubSystems](#)

**Introduced in R2017a**

## slwebview

Export Simulink models to Web views

### Syntax

```
slwebview
filename = slwebview(system_name)
filename = slwebview(folder)
filename = slwebview(system_name,Name,Value)
```

### Description

slwebview starts the Web View dialog box in the Report Explorer.

filename = slwebview(system\_name) exports the subsystem system\_name and its child systems to the file filename.

filename = slwebview(folder) exports all models in a folder. See “RecurseFolder” on page 8-0 to include models in subfolders.

filename = slwebview(system\_name,Name,Value) provides additional options specified by one or more Name,Value pairs.

### Examples

#### Export Web View for a Subsystem and Systems that Contain that Subsystem

Open the fuel rate controller subsystem.

```
open_system('fuelsys')
```

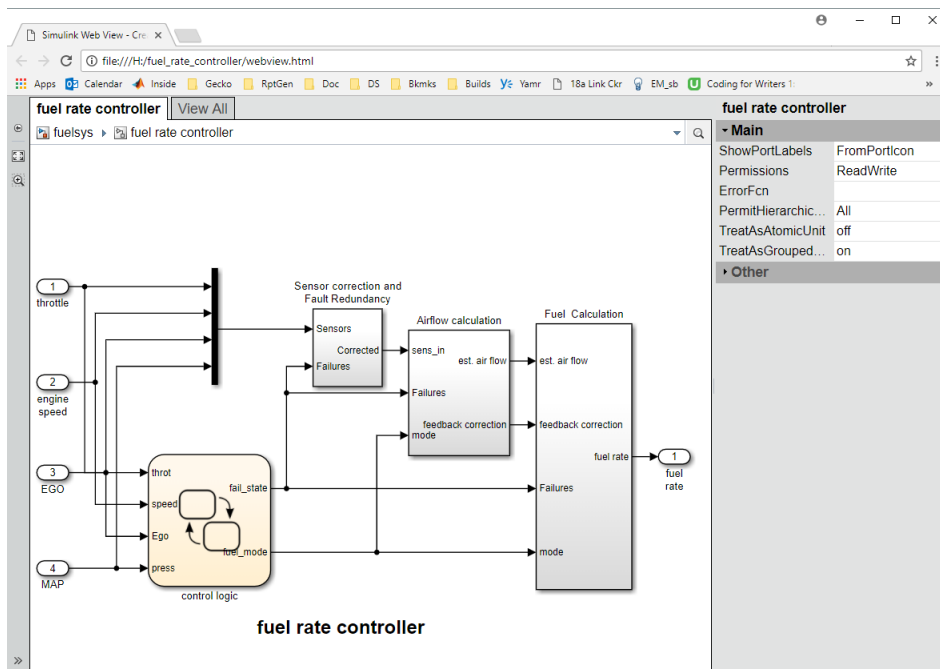
Export to a Web view the fuel rate controller subsystem and the system that contains it. Do not export the subsystems that it contains. This example assumes the current folder is the H: drive.

```
fuelsys_web_view = slwebview...
('fuelsys/fuel_rate_controller','SearchScope','CurrentAndAbove')
```

```
fuelsys_web_view =
```

```
H:\fuel_rate_controller.zip
```

The Web view displays in the system browser.



## Export Web View with Access to Referenced Models

Open the `sldemo_mdref_depgraph` model.

```
open_system('sldemo_mdref_depgraph')
```

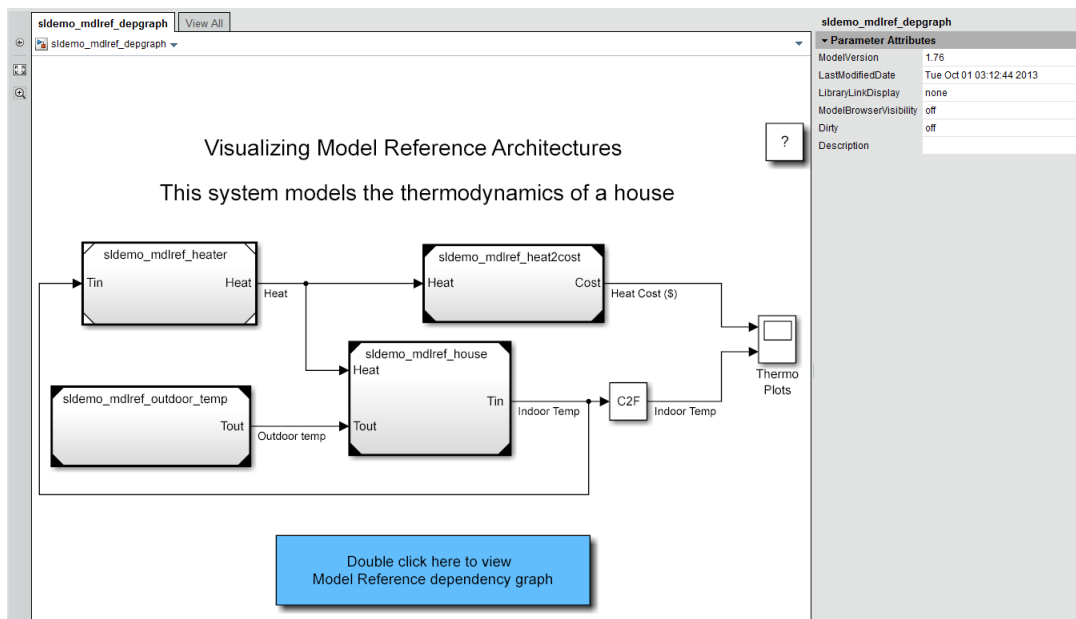
Export to a Web view the `sldemo_mdref_depgraph` model and allow access to the models it references.

```
depgraph_web_view = slwebview...
('sldemo_mdref_depgraph', 'FollowModelReference', 'on')
```

```
depgraph_web_view =
```

```
H:\sldemo_mdref_depgraph.zip
```

The Web view displays in the system browser. In the Web view, you can open the models referenced by the Model blocks.



Click a Model block to see its properties. Double-click a Model block to display the referenced model.

## Input Arguments

### **system\_name** — System to export to a Web view file

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. By default, child systems of the `system_name` system are also exported. Use the `SearchScope` name-value pair to export other systems, in relation to `system_name`.

### **folder** — Path to file folder of models to export

string containing the path to the file folder

Path to the file folder containing one or more models to export to a Web view file, specified as a string. All models in the folder are exported.

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

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

Example: `htmlFileName = slwebview(gcs, 'LookUnderMasks', 'all', ... 'FollowLinks', 'on')` Export to a Web view all layers of the model hierarchy to which the current system belongs, including the ability to interact with library links and masks.

### **SearchScope** — Systems to export, relative to the `system_name` system

'CurrentAndBelow' (default) | 'Current' | 'CurrentAndAbove' | 'All'

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

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

'CurrentAndAbove' exports the Simulink system or the Stateflow chart specified by the `system_name` 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 `system_name`.

Data Types: char

### **LookUnderMasks — 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 — 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 — 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

### **RecurseFolder — Whether to export models in subfolders**

false (default) | true

Whether to export models in subfolders to a Web View file, specified as a logical. If `false`, the Web view includes models only in the top-level folder and does not include models in subfolders. If `true`, models in subfolders are also included. This property applies only if you specify a `folder` as an input argument.

### **PackageName — Name of the web view output package**

model name (default)

Name of the web view output package, specified as a character vector. The web view output is a `.zip` file or folder of unzipped web view files, or both types of outputs.

Data Types: char

### **PackageFolder — Path of folder in which to place the packaged web view output**

current working directory (default)

Path in which to place the packaged web view, specified as a character vector. To save the packaged web view in the same folder as the model, use `$model` as the `PackageFolder`.

Data Types: `char`

### **PackagingType — Type of web view output package**

`'both'` (default) | `'zipped'` | `'unzipped'`

Type of web view output package, specified as a zipped file, a folder of unzipped files, or both a zipped file and folder of unzipped files.

Data Types: `char`

### **ViewFile — Whether to display the Web view in a Web browser when you export the Web view**

`true` (default) | `false`

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

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

Data Types: `logical`

### **ShowProgressBar — Whether to display the status bar when you export a Web view**

`true` (default) | `false`

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

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

Data Types: `logical`

## **Output Arguments**

### **filename — The name of the HTML file for displaying the Web view**

`string`

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

## **Tips**

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

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

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

## **See Also**

### **Topics**

“Create and Use a Web View” on page 5-11

**Introduced in R2006a**

