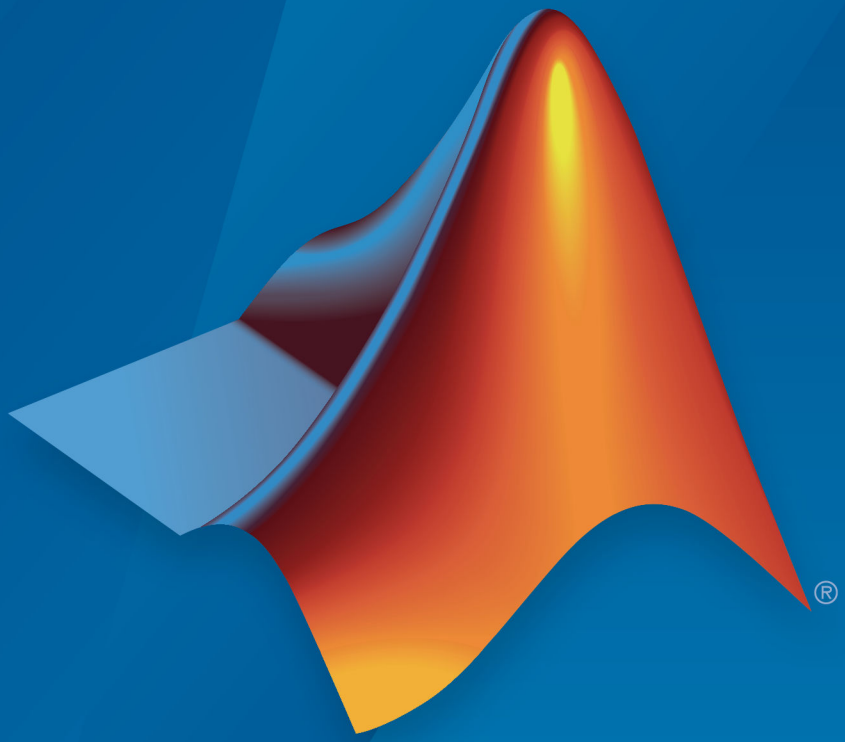
Simulink® Report Generator™

User's Guide



MATLAB® & SIMULINK®

R2018a

MathWorks®

How to Contact MathWorks

Latest news: `www.mathworks.com`

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

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

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

Phone: 508-647-7000

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

Simulink® Report Generator™ User's Guide

© COPYRIGHT 1999–2018 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` 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` 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)

Contents

1 Getting Started

Simulink Report Generator Product Description **1-2**
Key Features **1-2**

Relationship Between Simulink Report Generator and MATLAB Report Generator **1-3**
Finders and Reporters **1-3**
Web Views and Embedded Web Views **1-5**

System Design Documentation and Results Reporting **1-8**
Types of Reports **1-8**
System Design Documentation **1-8**
Results Reporting **1-9**

Report Generation for Simulink and Stateflow Elements . . . **1-10**
Simulink Report API Classes **1-12**
Find and Report on Blocks in a Model **1-15**
Use Specific Finders and Reporters for Different Block Types **1-17**
Find and Report on Stateflow Elements **1-23**

Generate Reports Without Customizing **1-29**
Predefined Standard Reports **1-29**
Report API **1-29**
Web View **1-30**

Report Creation Workflow **1-31**

Report Components **1-32**
About Report Components **1-32**
Report Structure Components **1-32**
System-Based Components **1-33**
User-Supplied Information Components **1-34**

Dynamic Reporting Components 1-34
Format Control at the Component Level 1-35

Working with the Report Explorer 1-36
About the Report Explorer 1-36

2 Generate System Design Description Reports

System Design Description 2-2
Predefined Standard Reports 2-2
What Is the System Design Description? 2-2
What You Can Do with the Report 2-3
Report Contents 2-3

Generate a System Design Description Report 2-7

Customize the System Design Description 2-8
Using the Report Explorer to Customize the Report 2-8
Building a Dialog Box for a Custom Report Setup File 2-9

3 System Design Description

System Design Description Dialog Box 3-2
System Design Description Overview 3-2
Title 3-3
Subtitle 3-3
Authors 3-4
Image 3-4
Legal Notice 3-5
Design details 3-5
Model references 3-6
Subsystems from custom libraries 3-7
Requirements traceability 3-7
Glossary and report explanation 3-7
File format 3-8

Stylesheet or Template 3-9
File name 3-12
Folder 3-12
If report exists, increment name to prevent overwriting 3-12
Package type 3-13

4 Creating Simulink Reports

Create a Simulink Report Generator Report 4-2

Report on MATLAB Function 4-16
Find and Report on MATLAB Function Blocks 4-16
Find and Report on Stateflow MATLAB Functions 4-19
Customize MATLAB Function Reporter Output 4-20

Create a Simulink Report Generator Report Interactively . . 4-24
Specify Report Options in the Setup File 4-24
Add Report Content with Components 4-26
Generate the Report 4-67

Generate a Report Associated with a Model 4-70

Logical and Looping Components 4-71

Filter with Loop Context Functions 4-72
Create and Save the Setup File 4-72
Add Components 4-72
Run the Report 4-73

Loop Context Functions 4-75
For Simulink Modeling Elements 4-75
For Stateflow Modeling Elements 4-75

5 Export Simulink Models to Web Views

Web Views **5-2**
What Is a Web View? **5-2**
System Requirements **5-2**
Web View Files **5-2**

Export Models to Web View Files **5-4**
Open the Web View Dialog Box **5-4**
Export a Model to a Web View **5-4**

Display and Navigate a Web View **5-6**
Display a Web View When You Export It **5-6**
Open a Web View File in a Web Browser **5-6**
View Contents of a System **5-7**
View Block Parameters and Signal Properties **5-8**
Access Optional Web View Information **5-9**

Search a Web View **5-11**
Perform a Search **5-11**
Sort Search Results **5-13**
Navigate Between Search Results and Model Elements **5-13**

Create and Use a Web View **5-14**
About This Tutorial **5-14**
Export Specific Systems **5-14**
Navigate the Web View **5-16**
Display Parameters and Properties of Blocks and Signals . . . **5-17**
Open the Web View **5-19**

Optional Web Views **5-21**

Capture and View Optional Web View Information **5-22**
Capture Optional Web View Information for a Model **5-22**
View Optional Web View Information **5-22**

Embedded Web View Reports **5-24**
What Is Embedded Web View? **5-24**
Navigating an Embedded Web View Report **5-25**
Embedded Web View Packaging **5-28**
View Embedded Web View Reports **5-28**

Create an Embedded Web View Report Generator **5-30**
Create an Embedded Web View Report Generator Class **5-30**

Specify Export Options for Embedded Web View Report **5-32**

Specify Document Content for Embedded Web View Report **5-33**

Generate Table of Contents for Embedded Web View Report **5-34**

Get Model Objects for Embedded Web View Report **5-35**

Create Hyperlinks for Embedded Web View Report **5-37**

Suppress Link Warning Messages for Embedded Web View Report **5-39**

Generate an Embedded Web View Report **5-40**
Class Definition File for an Embedded Web View **5-41**

Web View **5-46**
Web View Export Dialog Box Overview **5-46**
Systems to Export **5-46**
Referenced Models **5-47**
Library Links **5-47**
MathWorks Library Links **5-47**
Masked Subsystems **5-48**
Package name **5-48**
Folder **5-48**
If package exists, increment name to prevent overwriting . . . **5-48**
Package Type **5-49**
Include Model Coverage view **5-49**
Include Embedded Coder view **5-49**
Include Requirements view **5-49**
Include Coverage view **5-50**

Components — Alphabetical List
6

Classes — Alphabetical List
7

Functions - Alphabetical List
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 1-8
- “Report Generation for Simulink and Stateflow Elements” on page 1-10
- “Generate Reports Without Customizing” on page 1-29
- “Report Creation Workflow” on page 1-31
- “Report Components” on page 1-32
- “Working with the Report Explorer” on page 1-36

Simulink Report Generator Product Description

Design and automatically generate reports from Simulink models and simulations

Simulink Report Generator automatically generates richly formatted reports from Simulink models and simulations. You can design and generate reports in PDF, Microsoft® Word, Microsoft PowerPoint®, and HTML. You can insert design artifacts, including system design descriptions, generated code, requirements traceability, and testing reports, into the generated report. Artifacts for DO-178, ISO 26262, IEC 61508, and related industry standards can also be included. With Simulink Report Generator you can create a web view that lets you view and navigate a model from within a web browser.

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 `mlreporgen.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 1-10 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. Fcn

Table 1.1. vdp/Fcn Properties

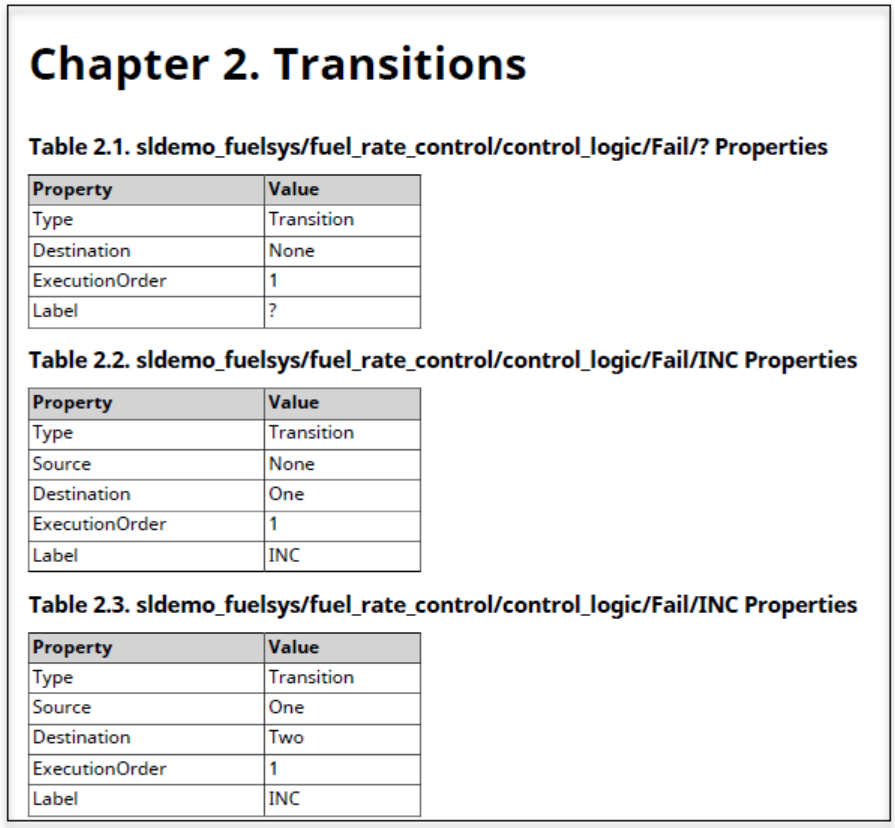
Property	Value
Type	Block
Block Type	Fcn
Expression	1 - u*u
Sample time (-1 for inherited)	-1

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

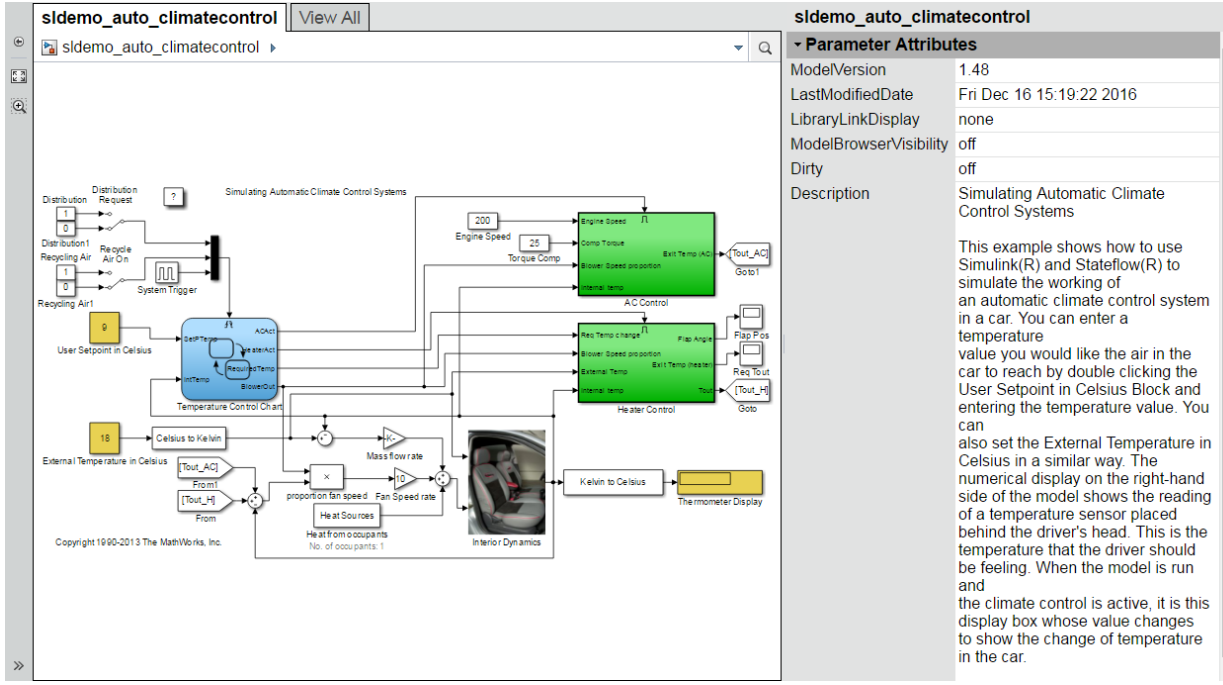
Table 2.3. sldemo_fuelsys/fuel_rate_control/control_logic/Fail/INC Properties

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

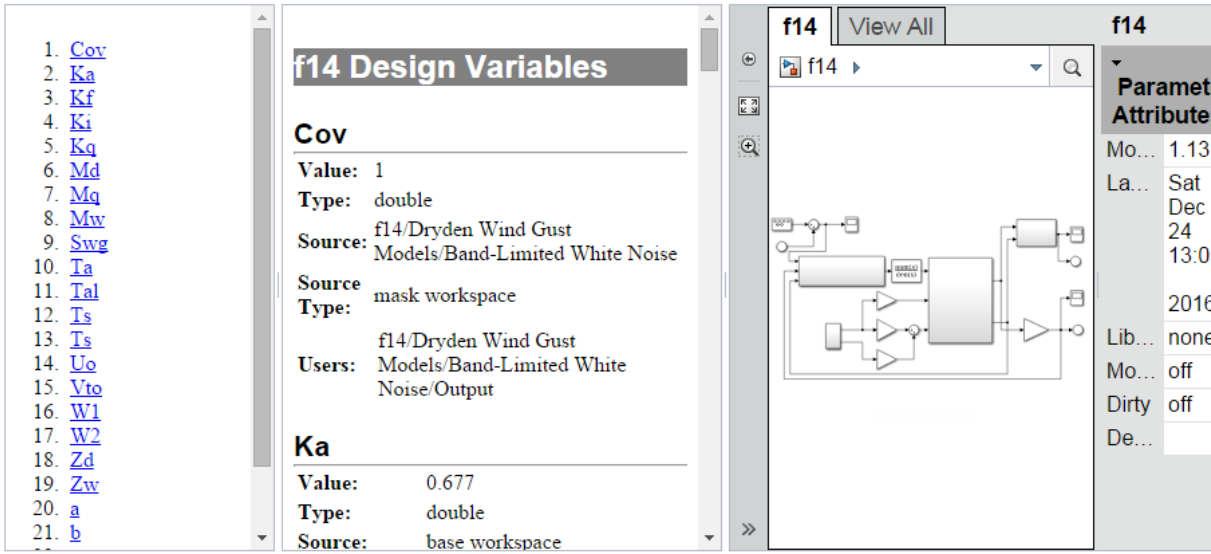
Web Views and Embedded Web Views

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 1-10
- “Report Generation for Common Report Elements” (MATLAB Report Generator)
- “Create a Report Generator” (MATLAB Report Generator)
- “Create a Simulink Report Generator Report” on page 4-2
- “Create and Use a Web View” on page 5-14
- “Create an Embedded Web View Report Generator” on page 5-30

System Design Documentation and Results Reporting

Types of Reports

Two common goals for creating reports are:

- `1 on page 1-8 — Capture information about the design decisions, structure, implementation, and operational details of a system.
- Results reporting on page 1-9 — 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`

System Information	Examples of Components to Use
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.

Results Information	Examples of Components to Use
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`

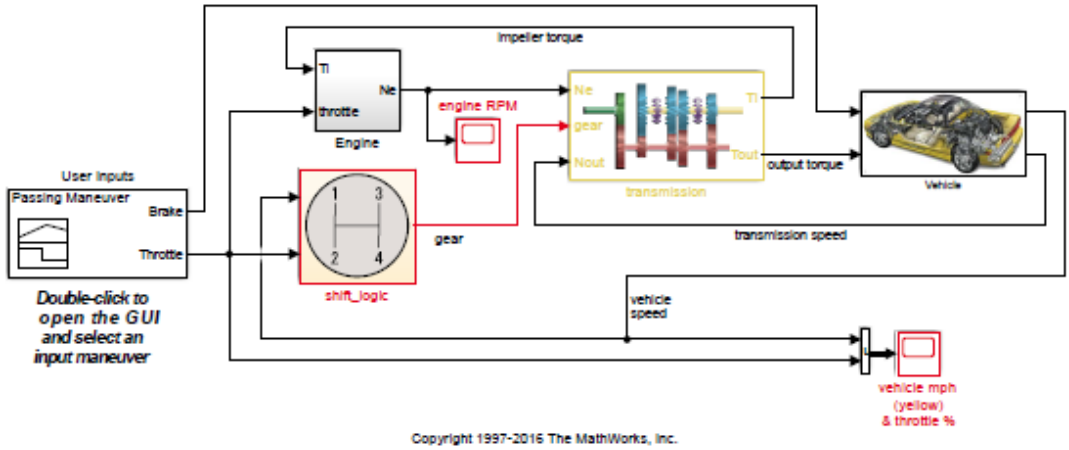
You can use components such as the `Model Simulation` component to control how the model simulates. Other components, such as the `Scope Snapshot`, show the results of the simulation.

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



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 “Report Generation for Common Report Elements” (MATLAB Report Generator) 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.pdftx
```

For a detailed example of editing a template, see the "Customize a Report API Template" section of “Report Generation for Common Report Elements” (MATLAB Report Generator)

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.

Finders and Results Classes

Report API Class	Description	Returns
`slreportgen.finder.AnnotationFinder`	Finds Simulink block diagram annotations	Annotation finder. Its associated `find` method returns a list of `DiagramElementResult` objects.
`slreportgen.finder.BlockFinder`	Finds blocks in a Simulink block diagram	Block finder. Its associated `find` method returns a list of `BlockResult` objects.
`slreportgen.finder.BlockResult`	Create finder result object for block	Result object for block
`slreportgen.finder.ChartDiagramFinder`	Finds Stateflow charts in a model	Stateflow chart diagram finder. Its associated `find` method returns a list of `DiagramResult` objects.
`slreportgen.finder.DiagramElementFinder`	Finds elements of a Simulink block diagram or Stateflow chart	Diagram element finder. Its associated `find` method returns a list of `DiagramElementResult` objects.
`slreportgen.finder.DiagramElementResult`	Container for a diagram element found by a `DiagramElementFinder`	Result object for diagram element
`slreportgen.finder.DiagramFinder`	Finds block diagrams and charts in a Simulink model	Diagram finder Its associated `find` method returns a list of `DiagramResult` objects.
`slreportgen.finder.DiagramResult`	Container for a diagram element found by the `DiagramElementFinder`	Result object for diagram
`slreportgen.finder.StateFinder`	Finds states in a Stateflow chart	State finder. Its associated `find` method returns a list of `DiagramElementResult` objects.

Report API Class	Description	Returns
`slreportgen.finder.StateflowDiagramElementFinder`	Finds elements of a Stateflow chart	Stateflow chart diagram finder. Its associated `find` method returns a list of `DiagramElementResult` objects.
`slreportgen.finder.SystemDiagramFinder`	Finds system block diagrams in a Simulink model	Simulink block diagram finder. Its associated `find` method returns a list of `DiagramResult` objects.

Reporters Classes

Report API Class	Description	Returns
`slreportgen.report.Diagram`	Creates snapshot of a Simulink block diagram or a Stateflow chart	Returns an image and caption of a Simulink block diagram or a Stateflow chart
`slreportgen.report.LookupTable`	Reports breakpoints and output points of Simulink lookup table block	Returns a table and plot of Lookup table block output values vs. breakpoints, and a data types table
`slreportgen.report.MATLABFunction`	Reports script used by `MATLABFunction` block or Stateflow MATLAB function	Returns a table of MATLAB Function block properties, function input and output argument summary, and the MATLAB code used by the block.
`slreportgen.report.SimulinkObjectProperties`	Creates table of the properties of a Simulink object	Returns a table of property value of the specified Simulink object
`slreportgen.report.StateflowObjectProperties`	Creates table of the properties of a Stateflow object	Returns a table of property value of the specified Stateflow 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.

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

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);
   ```

```
        add(sect,result)
        add(ch,sect)
    end
    add (report,ch);
```

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

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

The full program is

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

model_name = 'vdp';
load_system(model_name);

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

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

close(report)
close_system(model_name)
rptview(report)
```

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. Fcn

Table 1.1. vdp/Fcn Properties

Property	Value
Type	Block
Block Type	Fcn
Expression	1 - u*u
Sample time (-1 for inherited)	-1

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);
     add(sec,rptr)
     add(ch,sec)

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

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

The full program is

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

model_name = 'sldemo_radar_eml';
load_system(model_name)

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

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);
      add(sec,rptr)
      add(ch,sec)
   else
      sec = Section(blks(i).Name);
      add(sec,blks(i))
      add(ch,sec)
   end
end
add(report,ch)

close(report)
```

```
close_system(model_name)
rptview(report)
```

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	
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
deltat	Parameter	NaN	double	1
xhatOut	Output	2	double	4

MATLAB Function Function Script

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

1

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

1.3. More Info

Table 1.4. sldemo_radar_eml/More Info Properties

Property	Value
Type	System

1.4. Radar Measurement Noise

Table 1.5. sldemo_radar_eml/Radar Measurement Noise Properties

Property	Value
Type	Block
Block Type	SubSystem

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
   add(chapter,state)
end
add(rpt,chapter)
```

Chapter 1. States

Table 1.1. sldemo_fuelsys/fuel_rate_control/control_logic/O2 Properties

Property	Value
Type	AND State
Description	This state determines the validity of the exhaust gas oxygen sensor (EGO) data.
Label	O2

Table 1.2. sldemo_fuelsys/fuel_rate_control/control_logic/Pressure Properties

Property	Value
Type	AND State
Description	This state assesses the validity of the manifold absolute pressure (MAP) sensor.
Label	Pressure

Table 1.3. sldemo_fuelsys/fuel_rate_control/control_logic/Throttle Properties

Property	Value
Type	AND State
Description	This state determines the validity of the throttle sensor signal.
Label	Throttle

Table 1.4. sldemo_fuelsys/fuel_rate_control/control_logic/Speed Properties

Property	Value
Type	AND State
Description	This state infers the validity of the speed sensor data. A failure is indicated by the pre sence 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';
   add(chapter,rptr)
end
add(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');
add(rpt,tp)
add(rpt,TableOfContents);

chapter = Chapter('Title','States');
stFinder = StateFinder(subsys);
states = find(stFinder);
for state = states
   add(chapter,state)
end
add(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';
   add(chapter,rptr)
end
add(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
- “Report Generation for Common Report Elements” (MATLAB Report Generator)
- “Templates” (MATLAB Report Generator)

Generate Reports Without Customizing

In this section...
“Predefined Standard Reports” on page 1-29
“Report API” on page 1-29
“Web View” on page 1-30

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
- Design Requirements (requires Simulink Requirements)

You generate these reports using the **File > Reports** menu in the Simulink Editor. 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 2-7.

The Design Requirements report includes information about all the requirements associated with the model and its objects. You must have Simulink Requirements installed to use the Design Requirements 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 Editor, select **Tools** > **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” (MATLAB Report Generator) and “Report Creation”.

See Also

Related Examples

- “Generate a Report Using a Template” (MATLAB Report Generator)
- “Add Report Content with Components” on page 4-26

More About

- “Report Setup” (MATLAB Report Generator)
- “Layout Style Sheets” (MATLAB Report Generator)

Report Components

In this section...
“About Report Components” on page 1-32
“Report Structure Components” on page 1-32
“System-Based Components” on page 1-33
“User-Supplied Information Components” on page 1-34
“Dynamic Reporting Components” on page 1-34
“Format Control at the Component Level” on page 1-35

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 1-36 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 1-32	Include a title page, sections, and other components to organize the content of a report.
“System-Based Components” on page 1-33	Include components that obtain information directly from a model to include in a report.
“User-Supplied Information Components” on page 1-34	Include text and graphics that you supply, independent of a model.
“Dynamic Reporting Components” on page 1-34	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 4-32.

To organize a report into sections, use `Chapter/Subsection` components. For an example, see “Create a Section for Each Iteration” on page 4-48.

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” (MATLAB Report Generator)
- “Summary Table Components” (MATLAB Report Generator)
- “Create the Body of the Report” on page 4-40

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” (MATLAB Report Generator).

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” (MATLAB Report Generator)
- “Add Logical Then and Logical Else Components” on page 4-36
- “Create the Body of the Report” on page 4-40
- “Filter with Loop Context Functions” on page 4-72

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” (MATLAB Report Generator).

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

Outline Pane Context	Properties Pane Contents
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.

2

Generate System Design Description Reports

- “System Design Description” on page 2-2
- “Generate a System Design Description Report” on page 2-7
- “Customize the System Design Description” 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 2-3
“Report Contents” on page 2-3

Predefined Standard Reports

From the Simulink Editor, you can access 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 2-7.

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.

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

Note To view step-by-step tutorials for creating and generating a report, see the Introduction to System Design Description Reports example.

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 2-7.

Summary Version

Section	Information
Report Overview	Model version

Section	Information
Root System	• Block diagram representing the algorithms that compute root system outputs • Description (if available from model) • Interface: name, data type, and other properties of the system input and output signals • Subsystems: the path and a block diagram for each subsystem • State charts • Requirements (optional)
Subsystems	• Path • Block diagram
System Design Variables	• Design variables • Functions in design variable expressions

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	• Block diagram representing the algorithms that compute root system outputs • Description (if available from model) • Interface: name, data type, and other properties of the root system input and output signals • Block parameters • Includes detailed information about MATLAB Function blocks • Block execution order for root system and atomic subsystems • Look-up tables • Simulink workspace variables • Model configuration sets • State charts • Requirements (optional)
Subsystems	The same type of information as the information for the root system, as well as: • Path of the subsystem in the model • (For atomic subsystems) Checksum that indicates whether the version of an atomic subsystem that generates the report differs from other versions of the subsystem • Referenced models (optional) • Subsystems from custom libraries (optional)

Section	Information
State Charts	• State chart • States • Transitions between the states • Junctions • Events that trigger state transitions • Data types • Targets • Truth tables

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. From the **File** menu, select **Reports > System Design Description**.
3. In the System 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 2-8.

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 2-8 “Building a Dialog Box for a Custom Report Setup File” on page 2-9

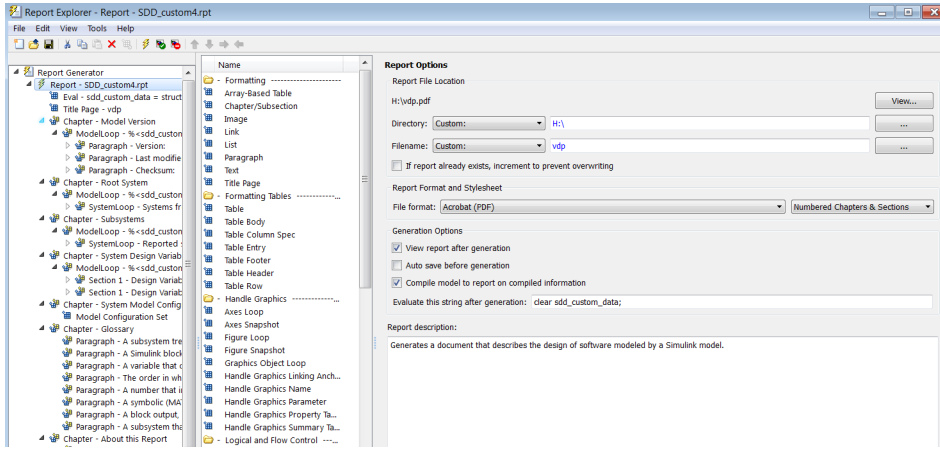
You can create customized versions of the System Design Description report by using the Report Explorer and, optionally the MATLAB tools for building graphical user interfaces.

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 4-26 and “Information Components” (MATLAB Report Generator).

   - 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” (MATLAB Report Generator)).
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.

3

System Design Description

System Design Description Dialog Box

In this section...
“System Design Description Overview” on page 3-2 “Title” on page 3-3 “Subtitle” on page 3-3 “Authors” on page 3-4 “Image” on page 3-4 “Legal Notice” on page 3-5 “Design details” on page 3-5 “Model references” on page 3-6 “Subsystems from custom libraries” on page 3-7 “Requirements traceability” on page 3-7 “Glossary and report explanation” on page 3-7 “File format” on page 3-8 “Stylesheet or Template” on page 3-9 “File name” on page 3-12 “Folder” on page 3-12 “If report exists, increment name to prevent overwriting” on page 3-12 “Package type” on page 3-13

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

"Use Predefined 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

"Use Predefined 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

“Use Predefined 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, click the menu **File > Model Properties**, and then click **History** tab.

See Also

“Use Predefined 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

“Use Predefined 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

“Use Predefined 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

“Use Predefined Reports”

Model references

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

Settings

Default: Enabled

See Also

- “Use Predefined Reports”
- “Model Referencing” (Simulink)

Subsystems from custom libraries

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

Settings

Default: Enabled

See Also

- “Use Predefined 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

- “Use Predefined 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

“Use Predefined 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

“Use Predefined 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

“Use Predefined 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

“Use Predefined 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

“Use Predefined 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

“Use Predefined 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

“Use Predefined Reports”

Creating Simulink Reports

- “Create a Simulink Report Generator Report” on page 4-2
- “Report on MATLAB Function” on page 4-16
- “Create a Simulink Report Generator Report Interactively” on page 4-24
- “Generate a Report Associated with a Model” on page 4-70
- “Logical and Looping Components” on page 4-71
- “Filter with Loop Context Functions” on page 4-72
- “Loop Context Functions” on page 4-75

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 SimulinkSimulink 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');
```

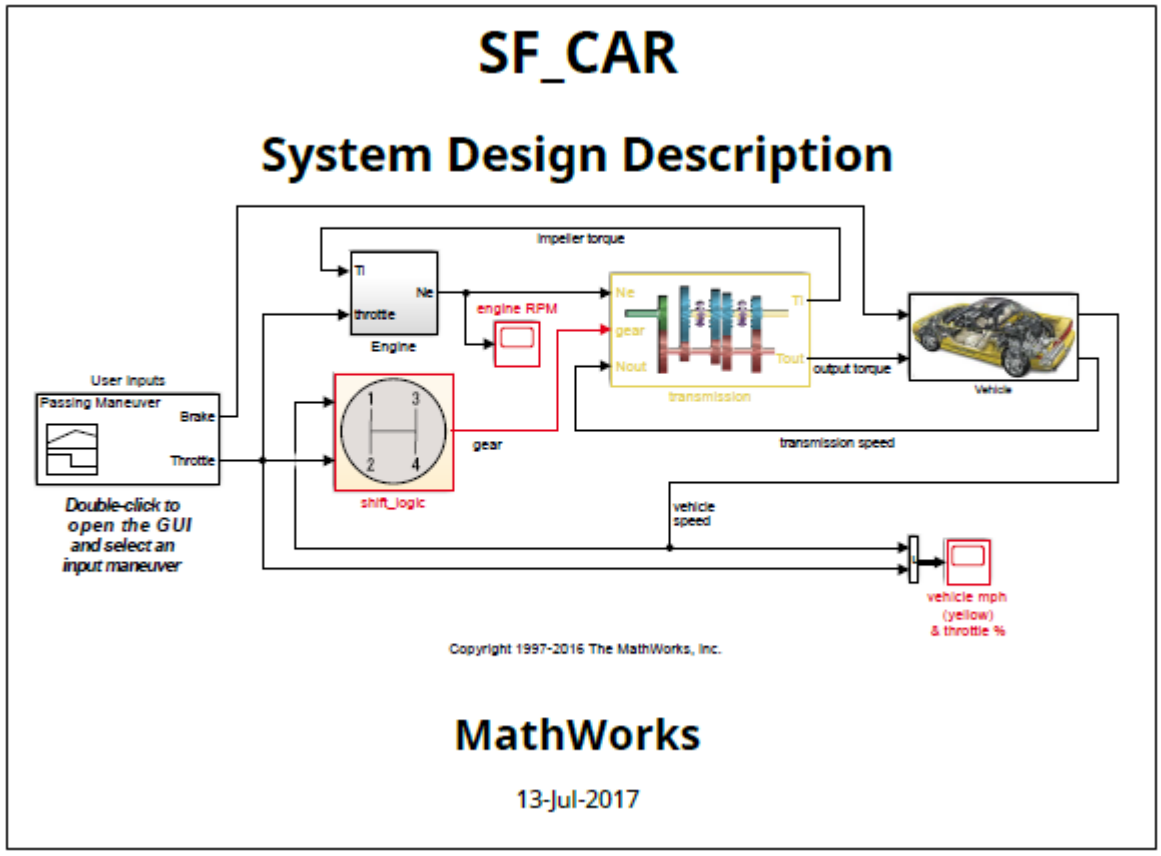
To customize properties that apply to the whole report, see `slreportgen.report.Report`.

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);
add(rpt,tp);
```

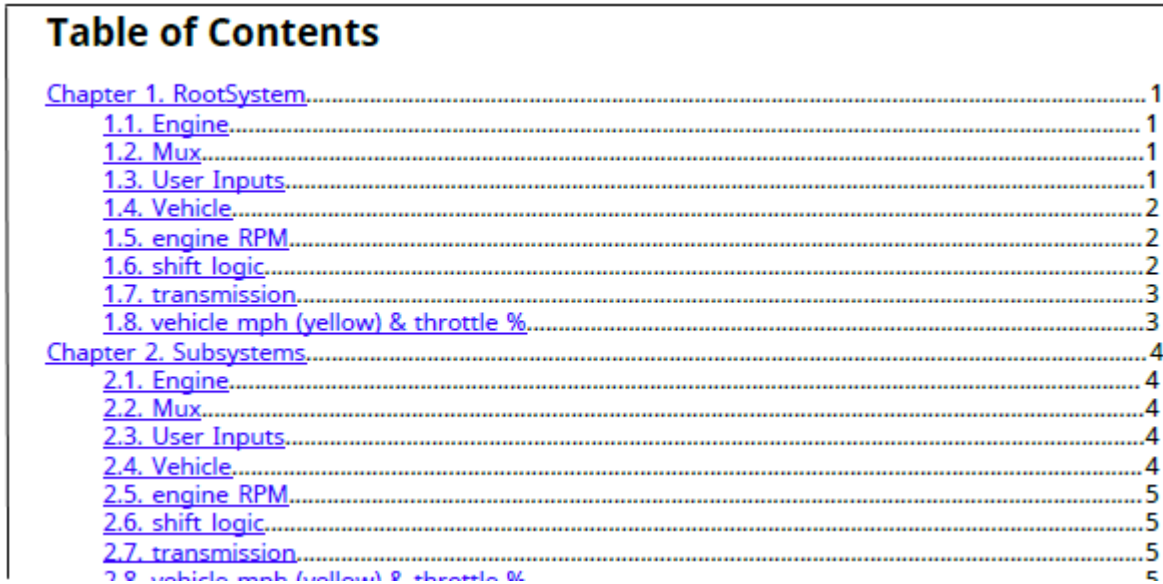


To customize additional title page properties, see `mlreportgen.report.TitlePage`.

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;
add(rpt,toc);
```



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

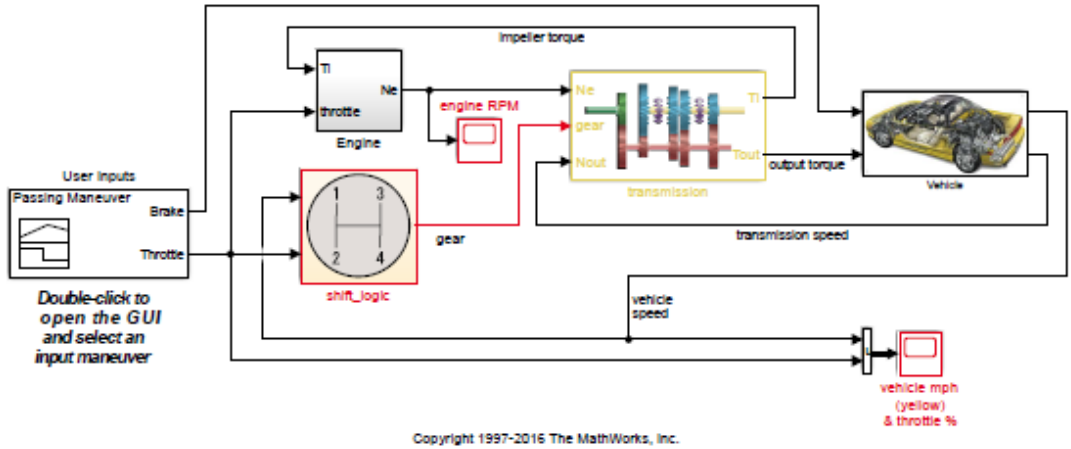
6 Add a chapter for the root system.

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");
add(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.

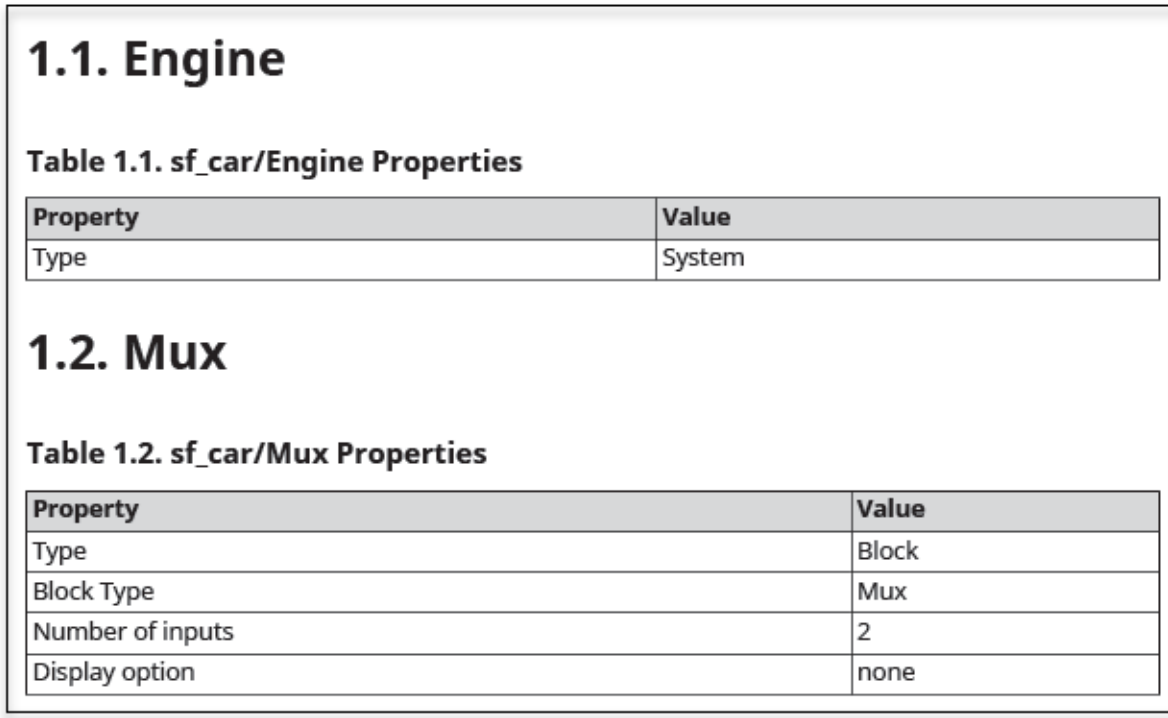
Use the block finder constructor (`slreportgen.report.BlockFinder`) to create a block finder for the root diagram. Then, use the `find` function of the block finder. The `find` function returns an array of block result objects (`slreportgen.report.BlockResult`), each of which contains a block.

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,' '));
    add(section,block);
    add(ch,section);
end
add(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

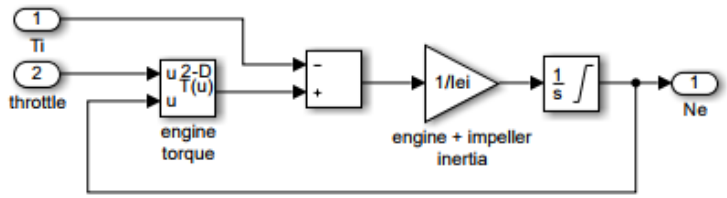


Figure 2.1. Engine

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

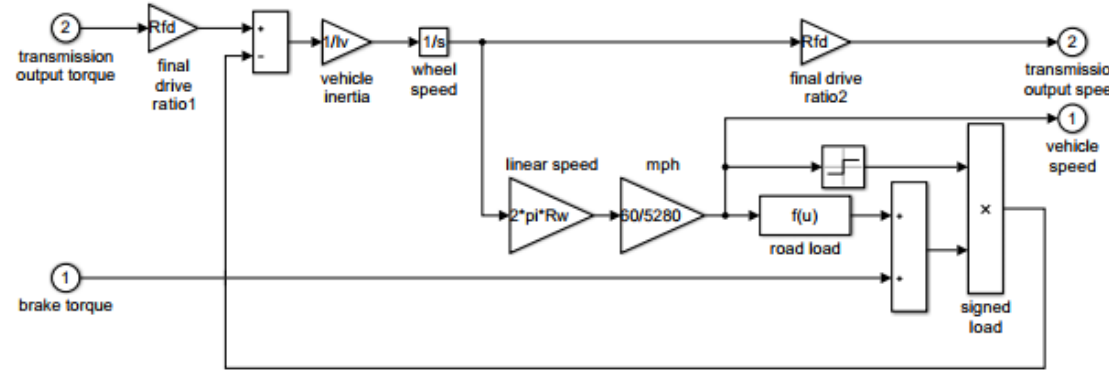
For more information, see `slreportgen.finder.SystemDiagramFinder` and `slreportgen.finder.DiagramResult`

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);
    add(section1,system);

    blkFinder1 = BlockFinder(system);
    elems = find(blkFinder1);
    for elem = elems
       section2 = Section("Title",...
           strrep(elem.Name,newline,' '));
       add(section2,elem);
       add(section1,section2);
    end
    add(ch,section1);
end
add(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);
   add(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);
       add(objSection,sfObj);
       add(section,objSection);
   end
   add(ch,section);
end
add(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

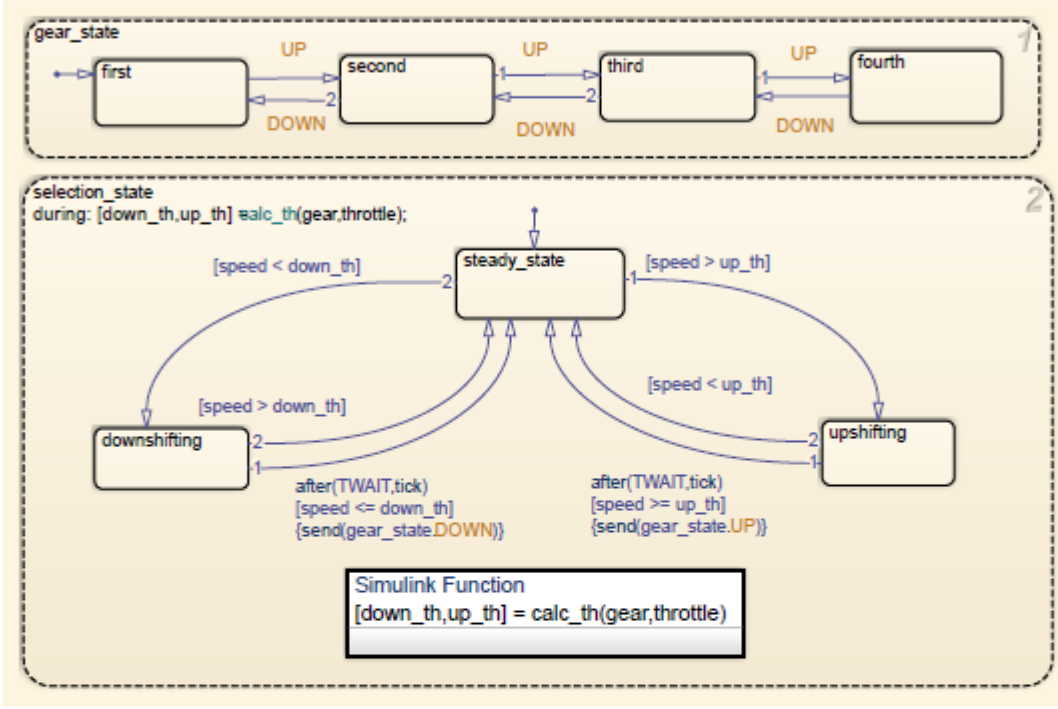


Figure 3.1. shift_logic

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

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

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

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

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

ch = Chapter("Title", "Stateflow Charts");
chdiagFinder = ChartDiagramFinder(model);
while hasNext(chdiagFinder)
   chart = next(chdiagFinder);
   section = Section("Title",chart.Name);
   add(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);
       add(objSection,sfObj);
       add(section,objSection);
   end
   add(ch,section);
end
add(rpt,ch);

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

See Also

`mlreportgen.report.Report.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 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 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);
```

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 fima th	Other:UserSpecified

Input fi math	`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', 'rectangl
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;
```

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.htmtx` 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.htmtx');
     ```

  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.htmtx`.

```
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” (MATLAB Report Generator) for additional information.

See Also

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

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 4-24
- "Add Report Content with Components" on page 4-26

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 `Dynamic Simulink Report`. The report is for the `vdp` model.

Note For another set of step-by-step examples for creating and generating a report, see the Introduction to System Design Description Reports example.

Specify Report Options in the Setup File

To create and configure the report setup file:

1. Start a Simulink software session.
2. Open the Report Explorer. From the MATLAB Toolstrip, in the **Apps** tab, in the **Database Connectivity 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)`.

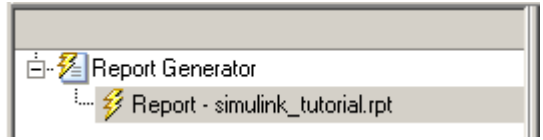
c Enter a description for the report. In the **Report description** text box, replace the existing contents with the following text.

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 4-26.

Add Report Content with Components

- “Report Components” on page 4-26
- “Add MATLAB Code” on page 4-27
- “Add a Title Page” on page 4-32
- “Open the Simulink Model” on page 4-34
- “Add Logical Then and Logical Else Components” on page 4-36
- “Error If Model Cannot Be Opened” on page 4-37
- “Create the Body of the Report” on page 4-40
- “Process with a Model Loop Component” on page 4-41
- “Add a Paragraph for Each Model” on page 4-43
- “Insert a Snapshot of the Model” on page 4-44
- “Add a Loop for Processing the Model” on page 4-45
- “Block Parameter Value from a MATLAB Expression” on page 4-47
- “Create a Section for Each Iteration” on page 4-48
- “Insert the Block Value” on page 4-49
- “Set a Parameter Value” on page 4-50
- “Check Value Using a Logical If Component” on page 4-52
- “Simulate the Model Using a Model Simulation Component” on page 4-54
- “Create a Post-Test Analysis Section” on page 4-60

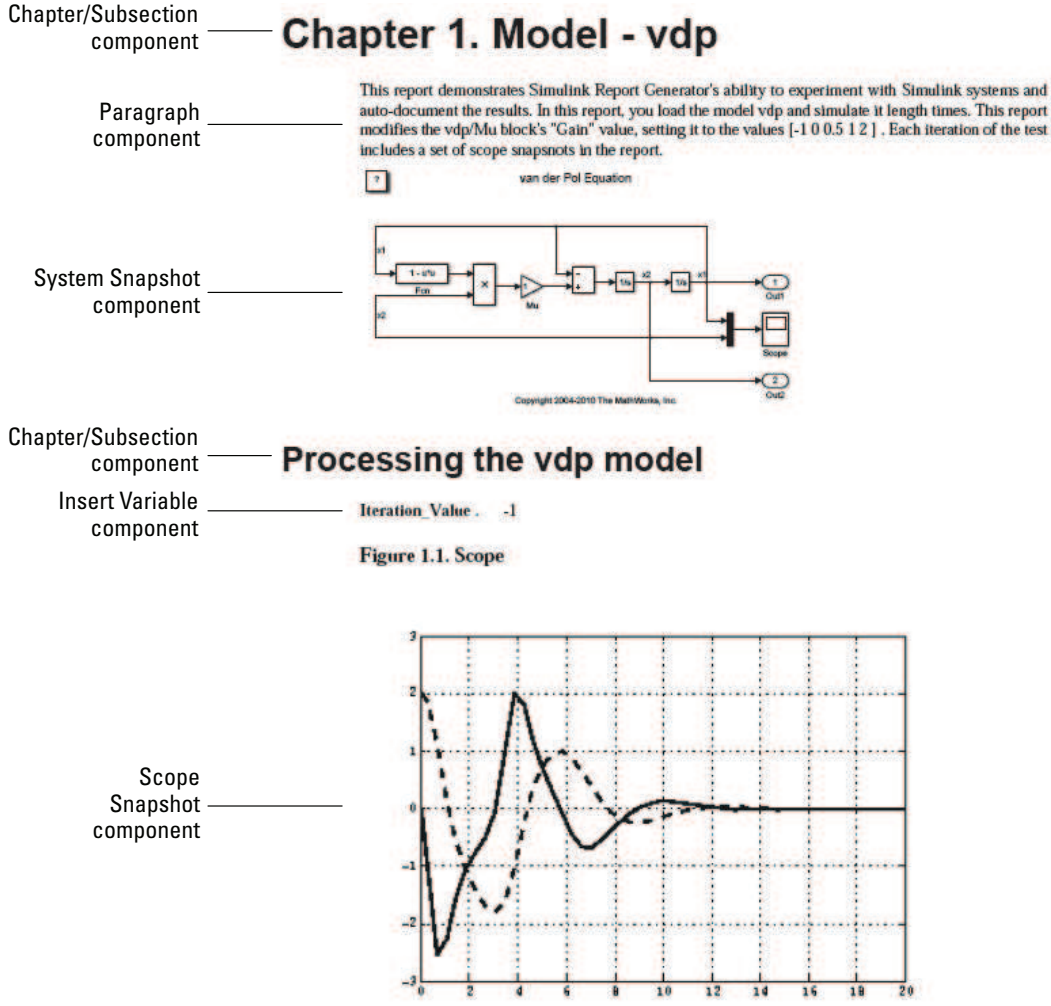
Report Components

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 1-32.

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.



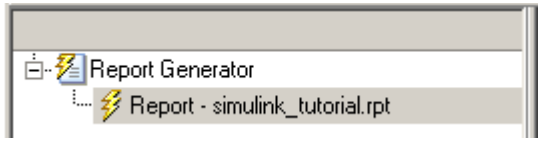
Add MATLAB Code

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 4-24.

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.

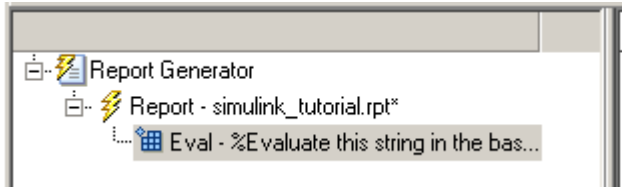
1. In the Outline pane on the left, select `simulink_tutorial.rpt`.



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.

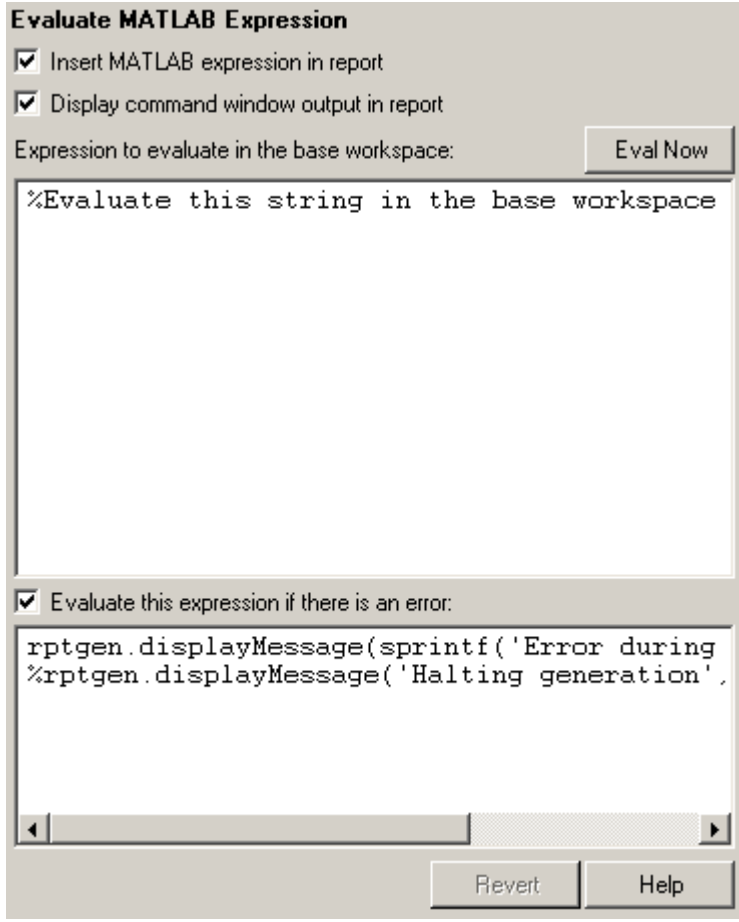
Note You cannot edit the component information in the Properties pane on the right until you add the component to the report.

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

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

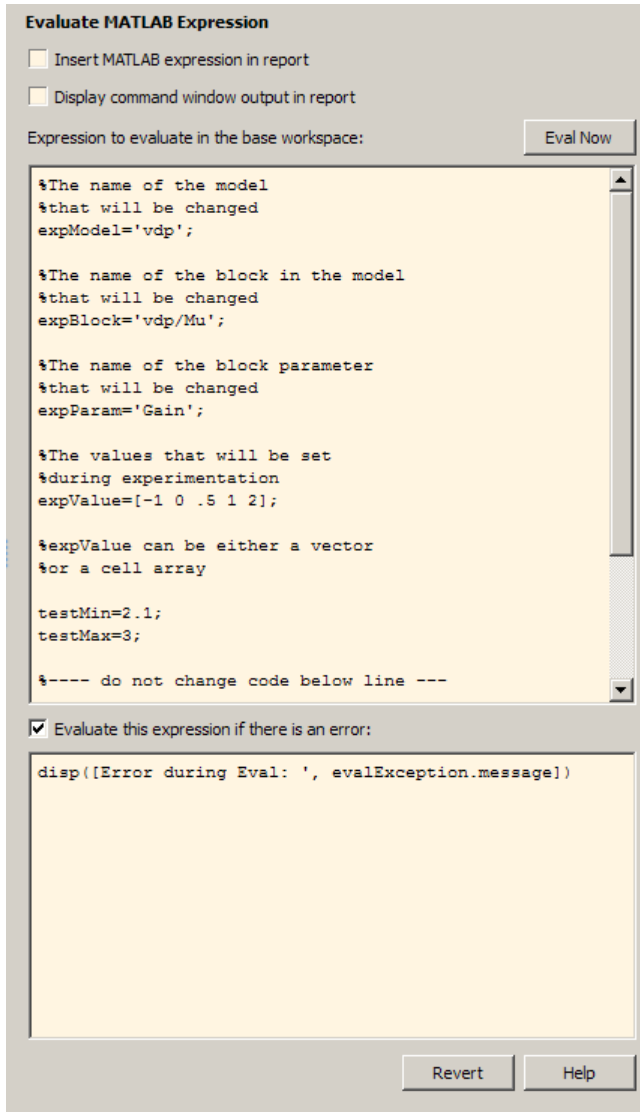
Note When you change a field in the Properties pane on the right, the field background changes color (the default is a cream color), indicating that there are unapplied changes to that field. As soon as you perform operations on another component, the Simulink Report Generator software applies the changes,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

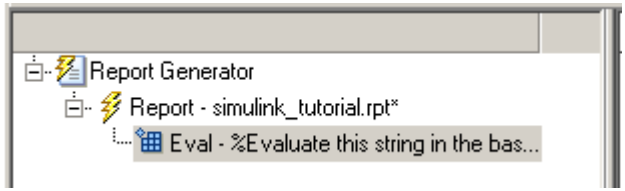
Add a Title Page

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 4-24.

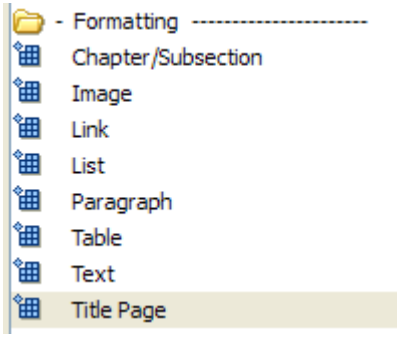
To see the completed report setup file, open `Simulink Dynamic Report`. The report is for the `vdp` model.

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.

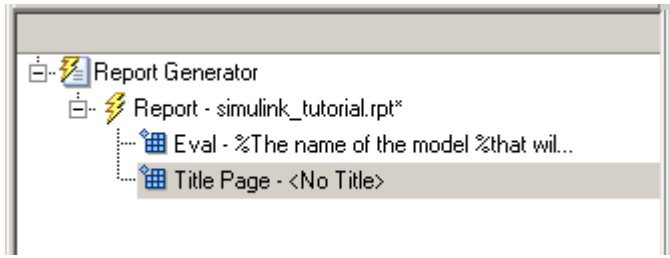


2 In the Library pane in the middle, under the **Formatting** category, click **Title Page**.



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:

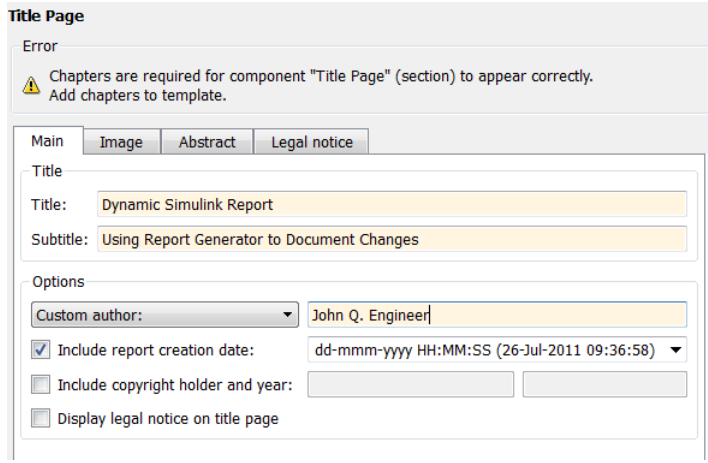
```
Using Simulink Report Generator to Document Changes
```

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.



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 4-24.

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 4-27 tries to open the `vdp` model:

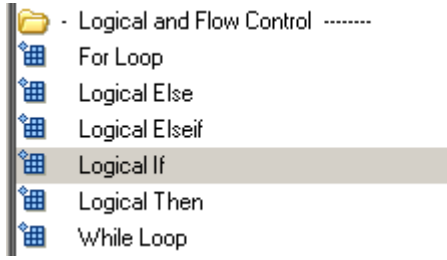
```
try
   open_system(expModel);
end
```

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

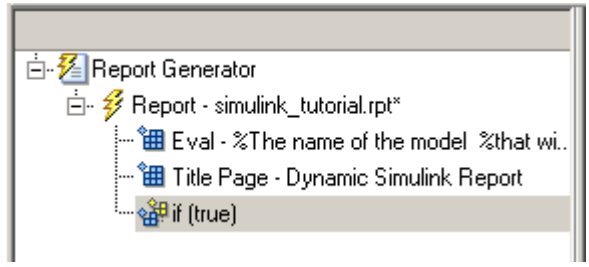
To see if the `vdp` model was successfully opened, test the result of the `open_system` command using a **Logical If** component.

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 4-36 explains how to add two child components to the **if** component.

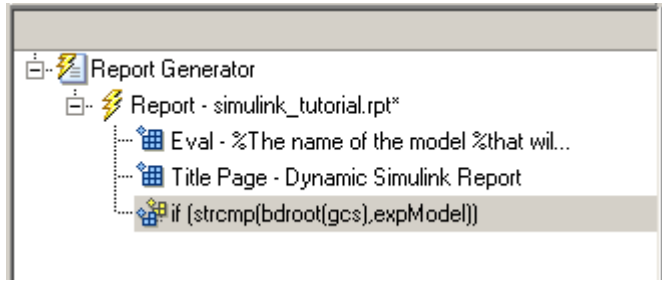
4 In the Properties pane on the right, in the **Test expression** text box, replace the default text, `true`, with the following text:

```
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.



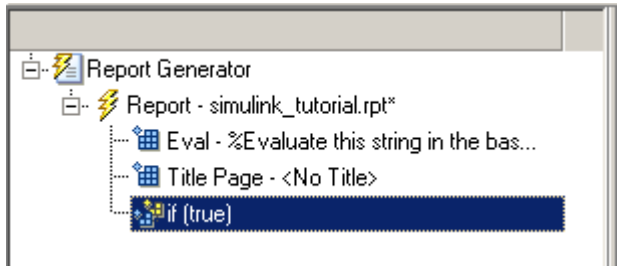
Add Logical Then and Logical Else Components

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 4-24.

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

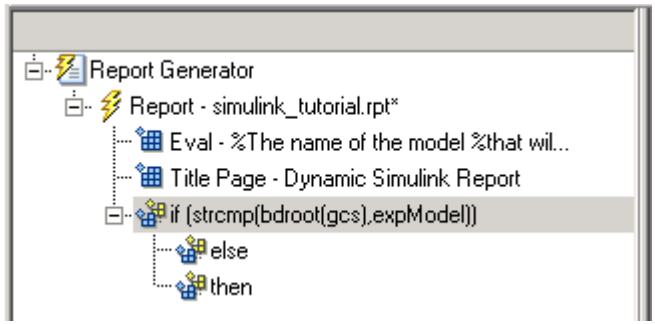
The `if strcmp(bdroot(gcs), expModel)` component has two possible results. Add two child components to the report setup file to process these cases.

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

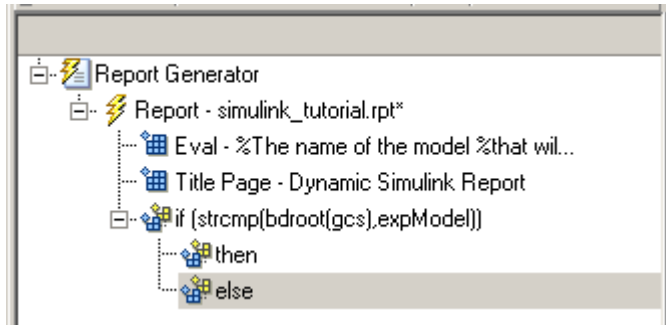


2. In the Library pane in the middle, under the **Logical and Flow Control** category, double-click **Logical Then**.
3. In the Outline pane on the left, select the **if** component again.
4. In the Library pane in the middle, under the **Logical and Flow Control** category, double-click **Logical Else**.

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.

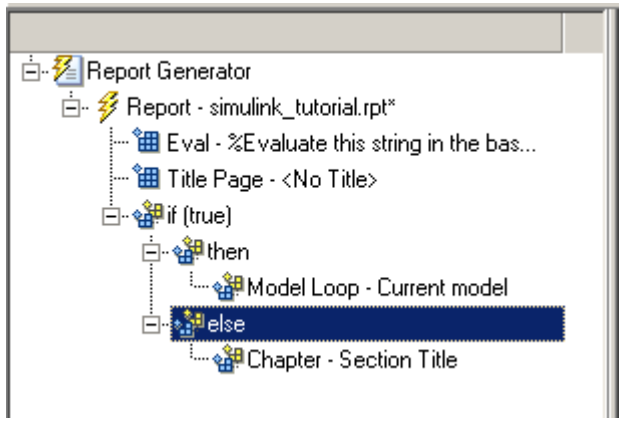
Error If Model Cannot Be Opened

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 4-24.

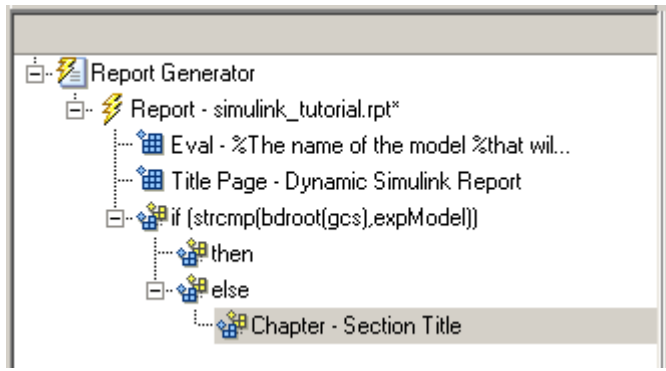
To see the completed report setup file, open `Simulink Dynamic Report`. The report is for the `vdp` model.

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.



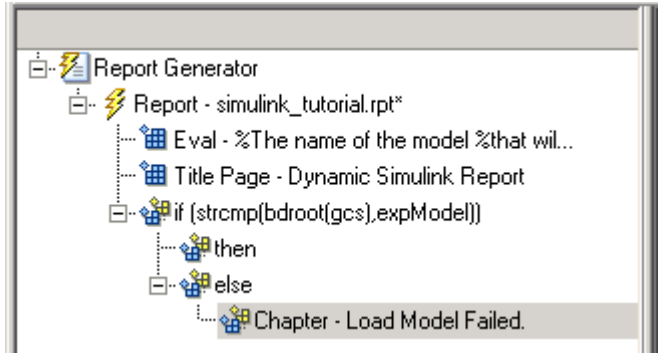
Note When you add a component to a report, it is added by default as a child component unless the selected component cannot have child components.

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.
```

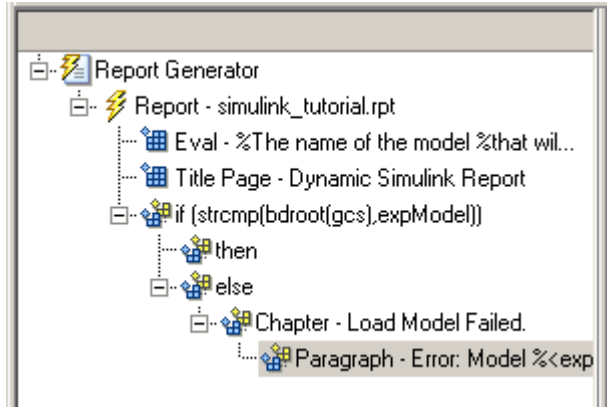
The expression `%<expModel>` indicates that the value of the workspace variable `expModel` is inserted into the text, as in the following example.

```
Error: Model vdp could not be opened.
```

7 In the Outline pane on the left, select the **Chapter/Section** component.

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 4-24.

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 4-41
- “Add a Paragraph for Each Model” on page 4-43
- “Insert a Snapshot of the Model” on page 4-44
- “Add a Loop for Processing the Model” on page 4-45
- “Block Parameter Value from a MATLAB Expression” on page 4-47
- “Create a Section for Each Iteration” on page 4-48
- “Insert the Block Value” on page 4-49
- “Set a Parameter Value” on page 4-50
- “Check Value Using a Logical If Component” on page 4-52
- “Simulate the Model Using a Model Simulation Component” on page 4-54
- “Create a Post-Test Analysis Section” on page 4-60

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 4-36.

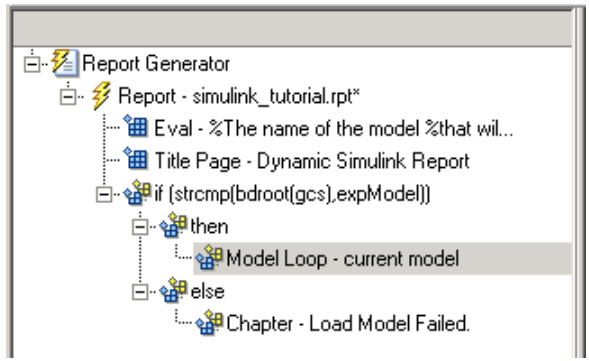
Process with a Model Loop Component

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 4-24.

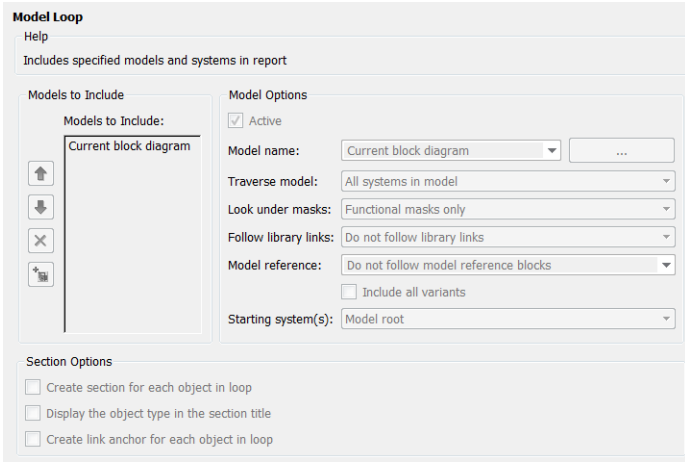
To see the completed report setup file, open `Simulink Dynamic Report`.

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.



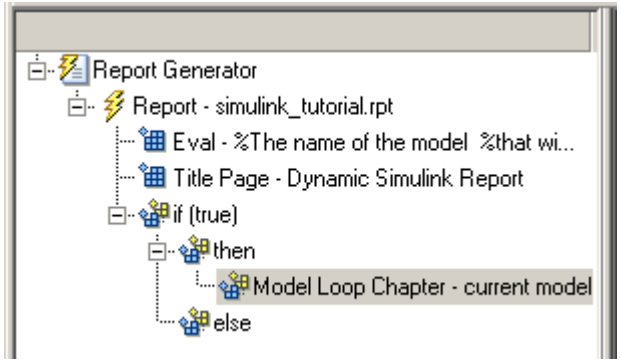
3 In the Properties pane on the right:

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

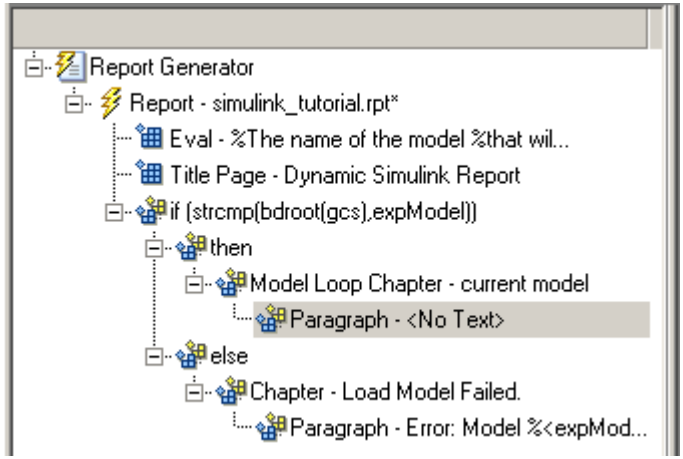
Note This section builds on the previous tasks described in the step-by-step example summarized in “Create a Simulink Report Generator Report Interactively” on page 4-24.

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 snapsnots 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 4-24.

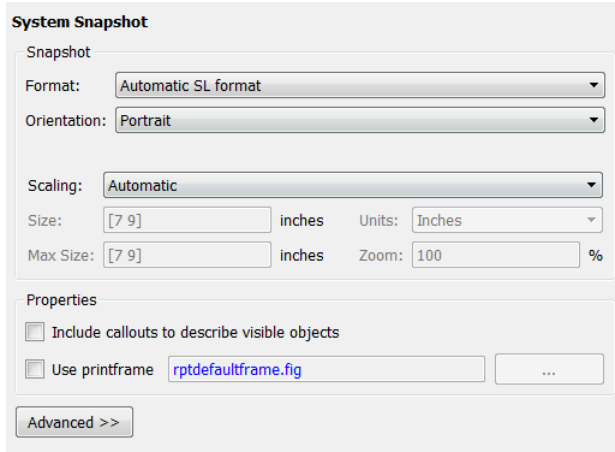
To see the completed report setup file, open `Simulink Dynamic Report`. The report is for the `vdp` model.

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.



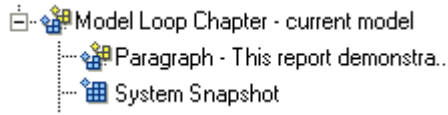
3 In the Properties pane on the right:

   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.

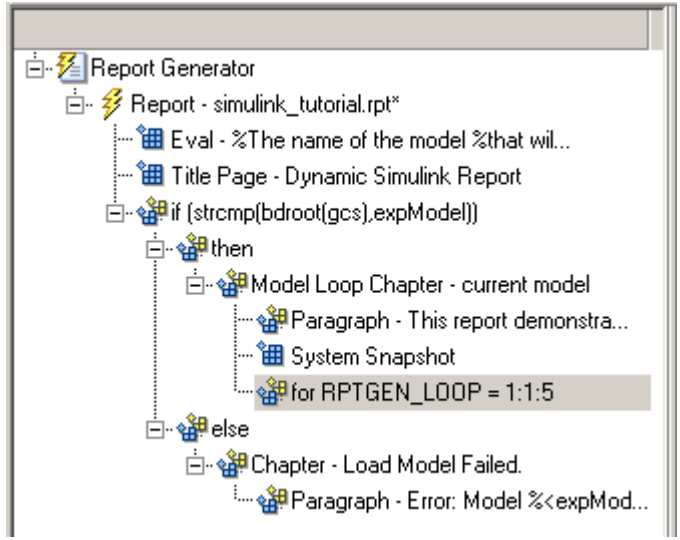
Add a Loop for Processing the Model

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 4-24.

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.

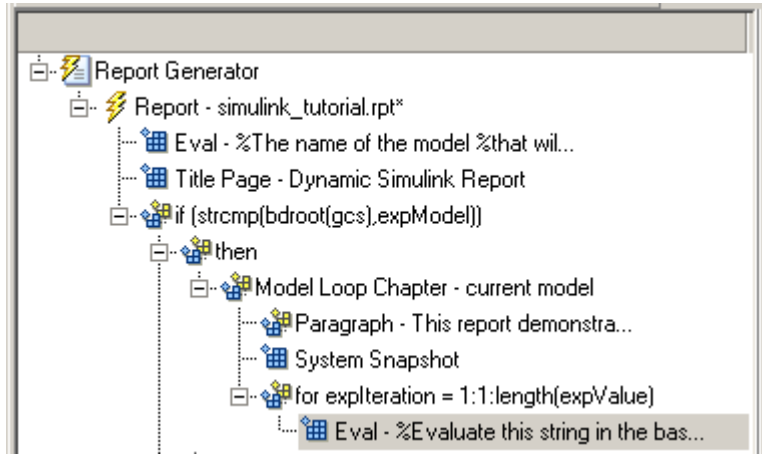
Block Parameter Value from a MATLAB Expression

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 4-24.

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

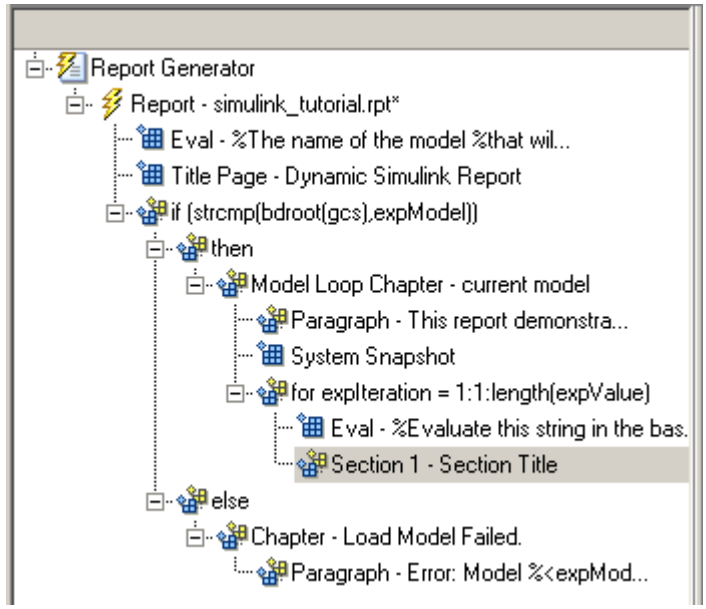
Note This section builds on the previous tasks described in the step-by-step example summarized in “Create a Simulink Report Generator Report Interactively” on page 4-24.

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

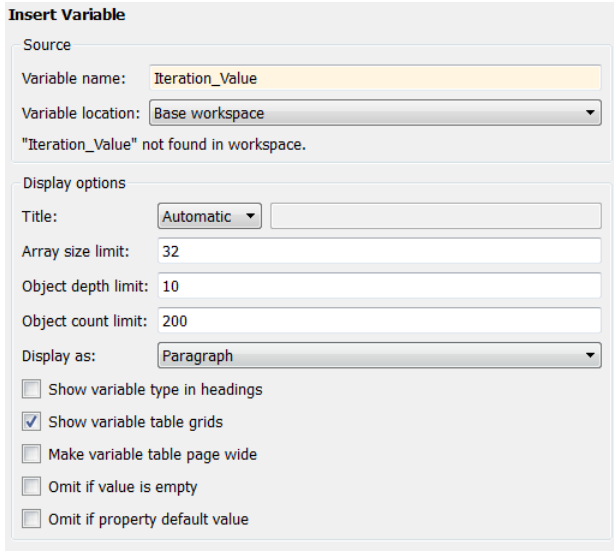
Note This section builds on the previous tasks described in the step-by-step example summarized in “Create a Simulink Report Generator Report Interactively” on page 4-24.

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.

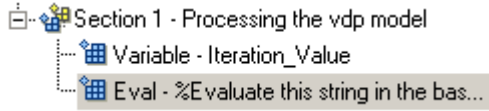
Set a Parameter Value

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 4-24.

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.

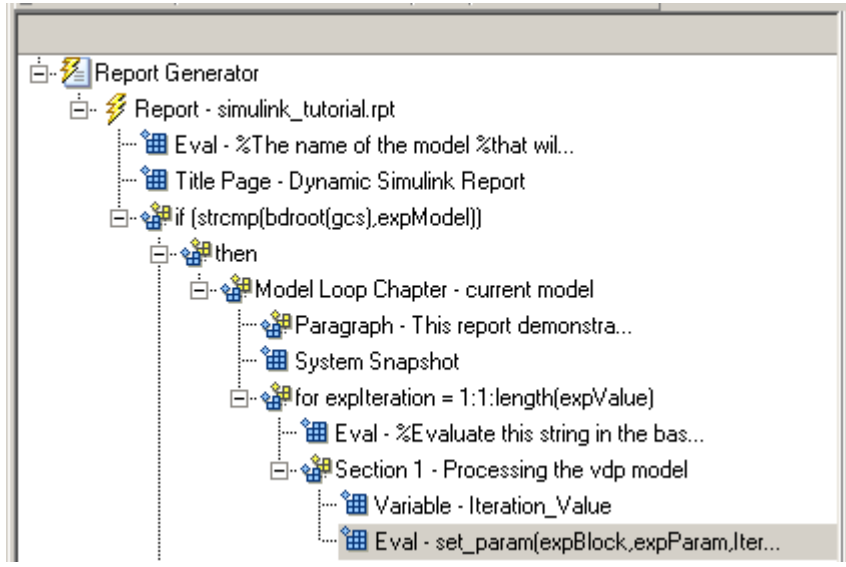
   ```
   set_param(expBlock,expParam,Iteration_Value);
   okSetValue=(1);
   ```

   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.



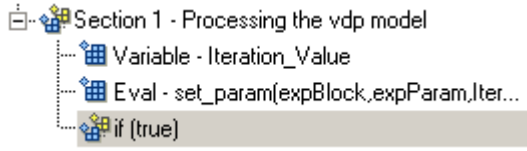
Check Value Using a Logical If Component

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 4-24.

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:

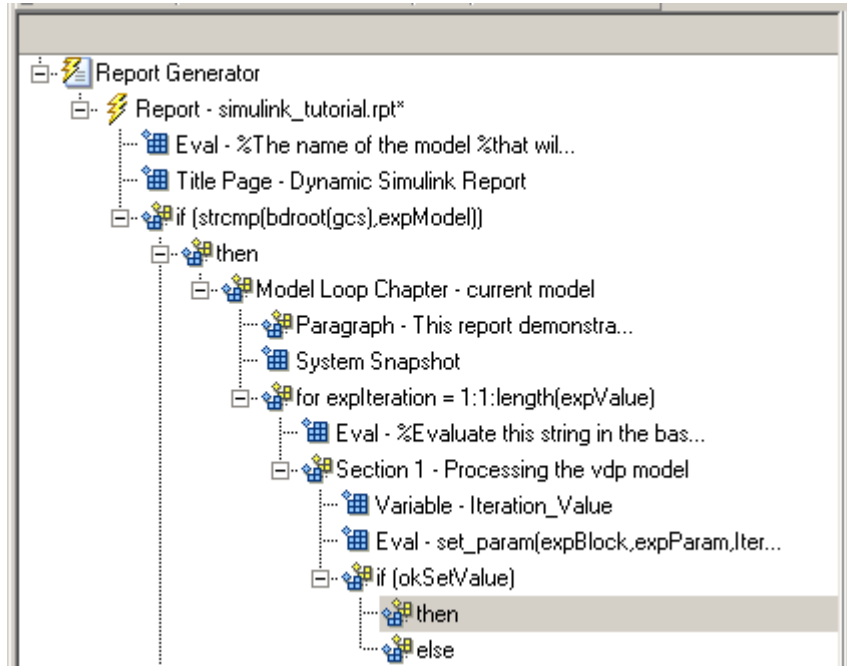
1 In the Outline pane on the left, select the **if (okSetValue)** component.

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.

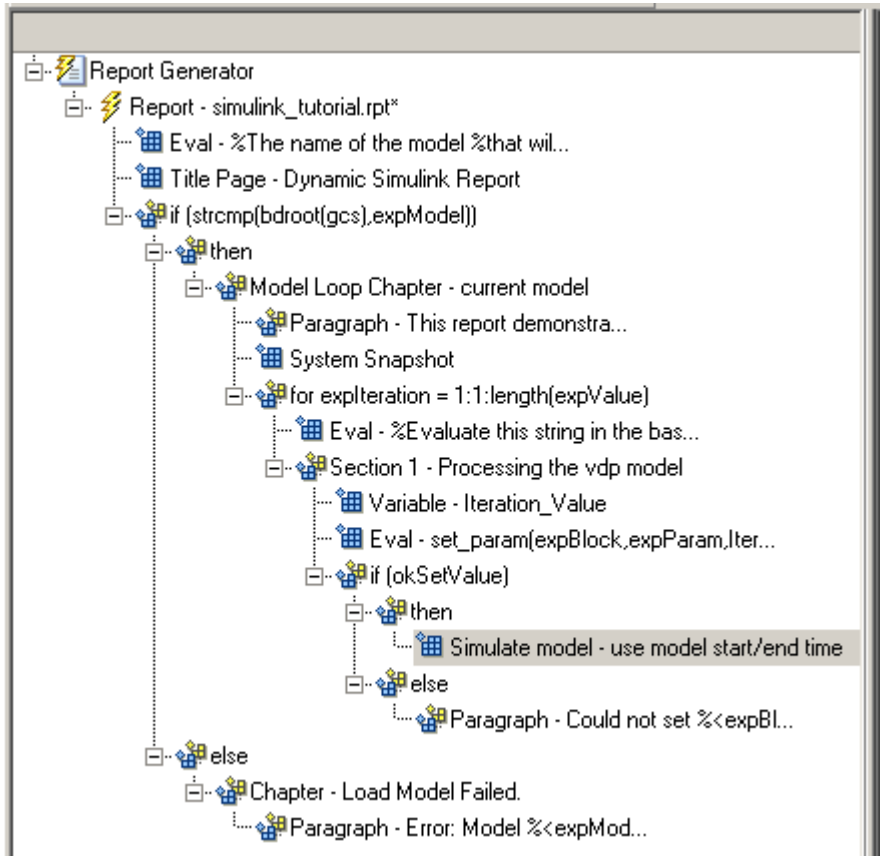
Simulate the Model Using a Model Simulation Component

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 4-24.

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:

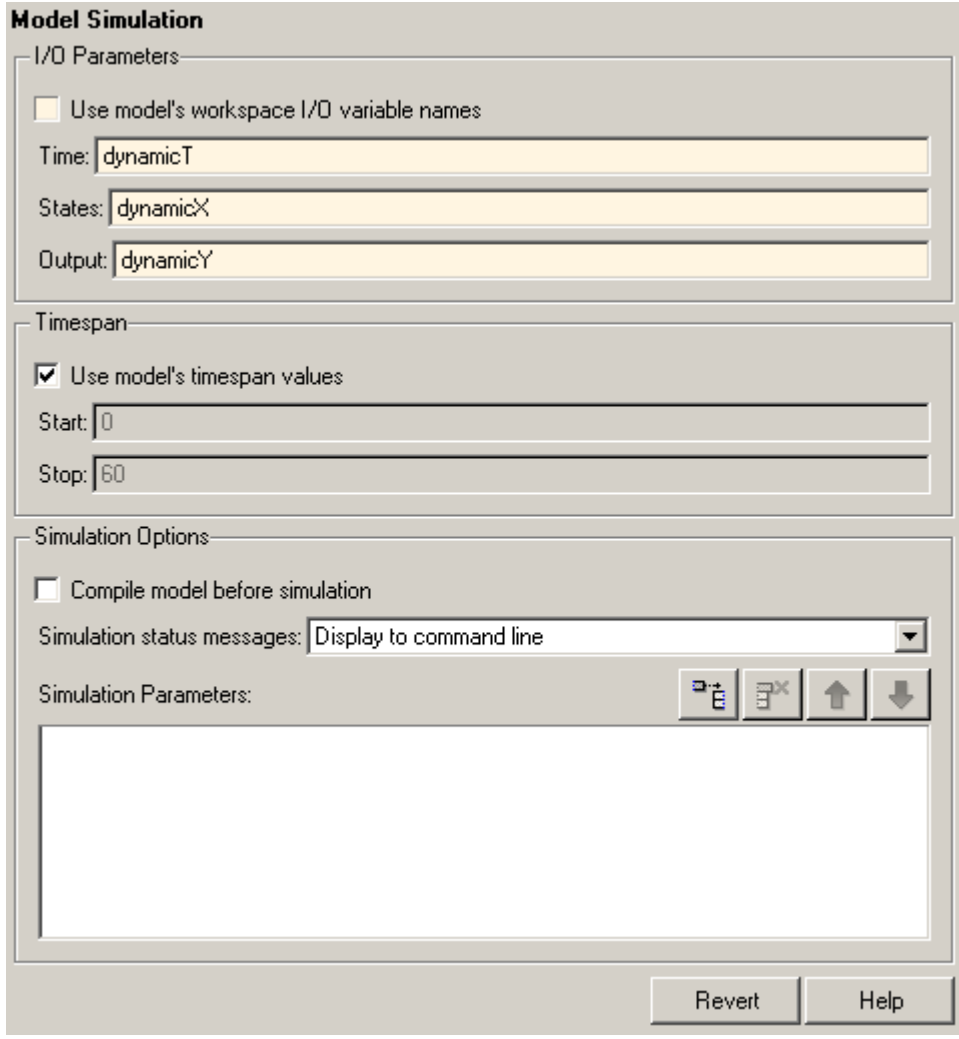
   a Clear the **Use model's workspace I/O variable names** check box.

   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.



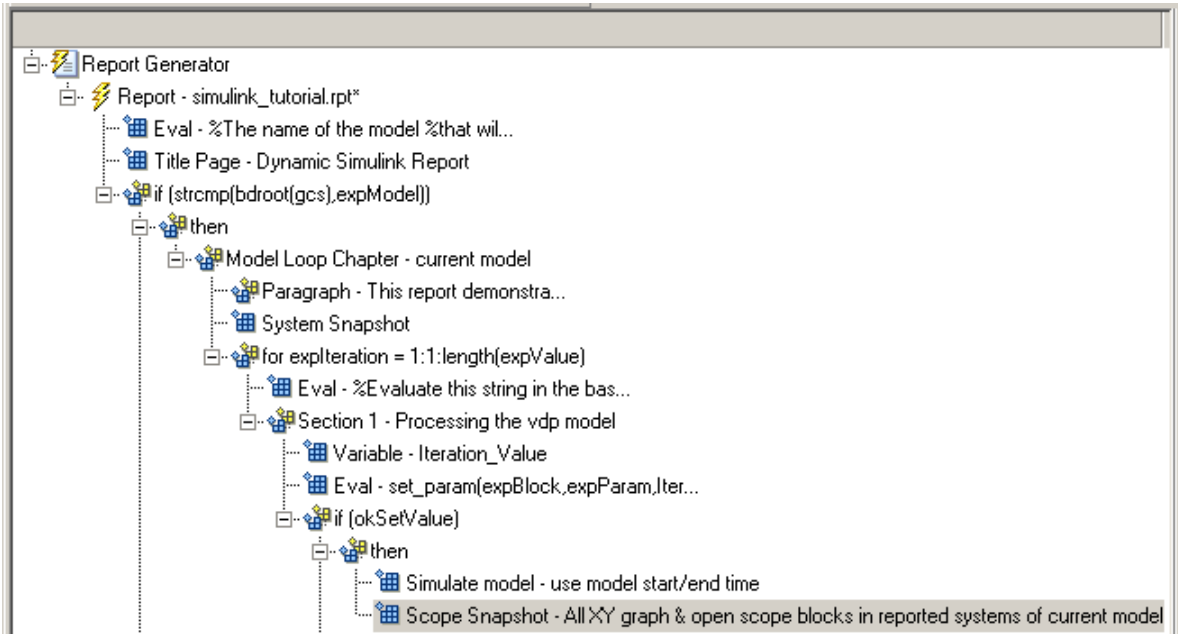
4 In the Outline pane on the left, select the **Simulate model** component.

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.

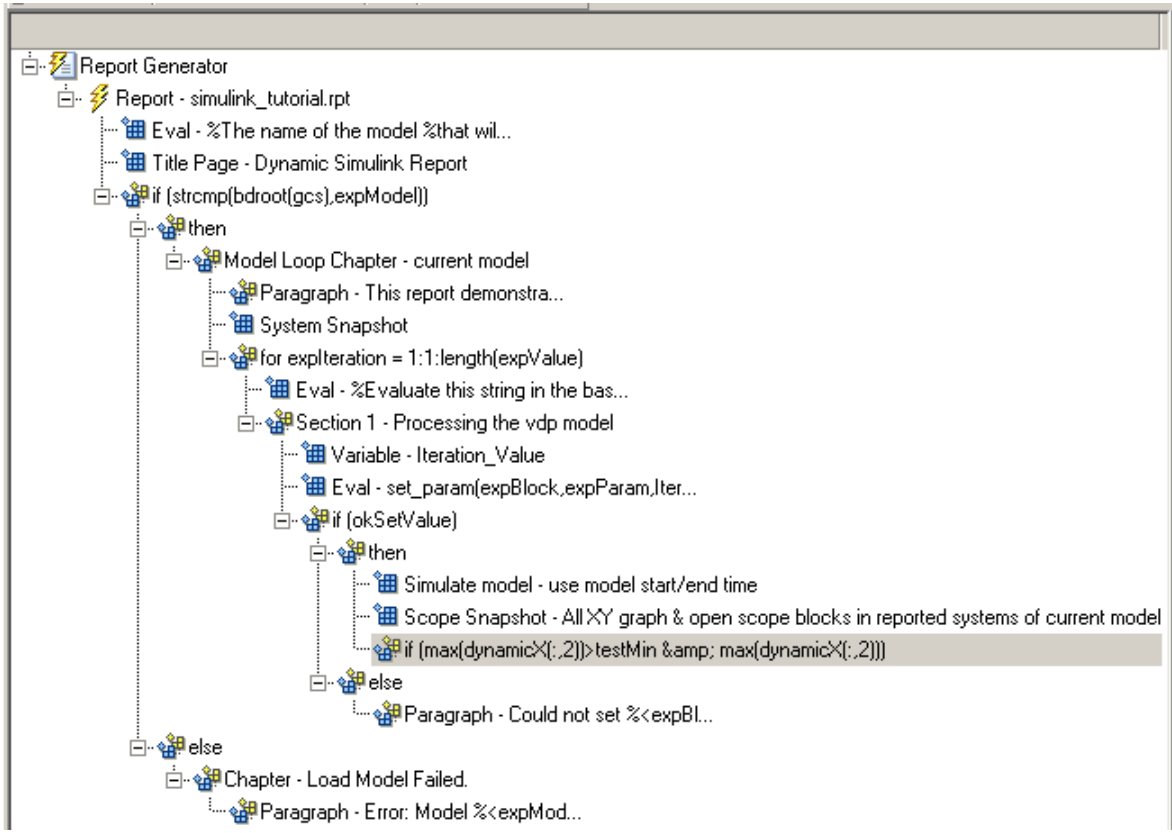
   c Double-click the **Logical If** component.

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))
```

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.

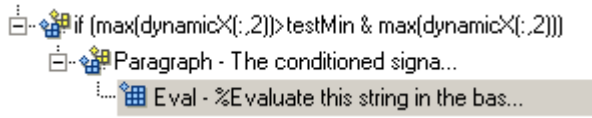
```
The conditioned signal has a maximum value
of %<max(dynamicX(:,2))>, which lies in the
desired range of greater than %<testMin> and
less than %<testMax>.
```

12 To save the success values to insert into a table at the end of the iterations, use an **Evaluate MATLAB Expression** component.

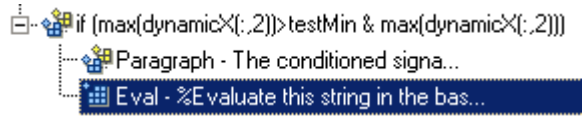
a In the Outline pane on the left, select the **Paragraph** component.

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 `expOkValues` array:

```
expOkValues=[expOkValues;...
     {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.

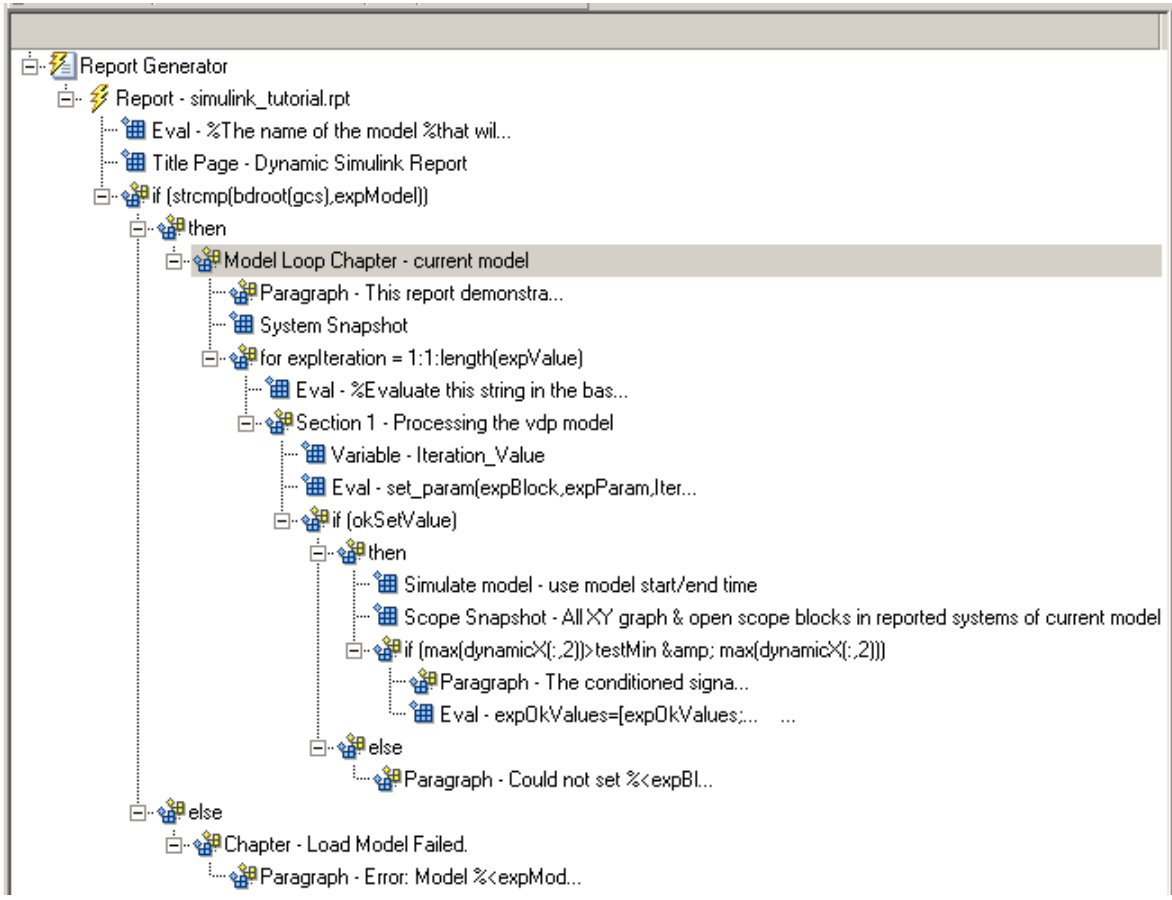
Create a Post-Test Analysis Section

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 4-24.

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

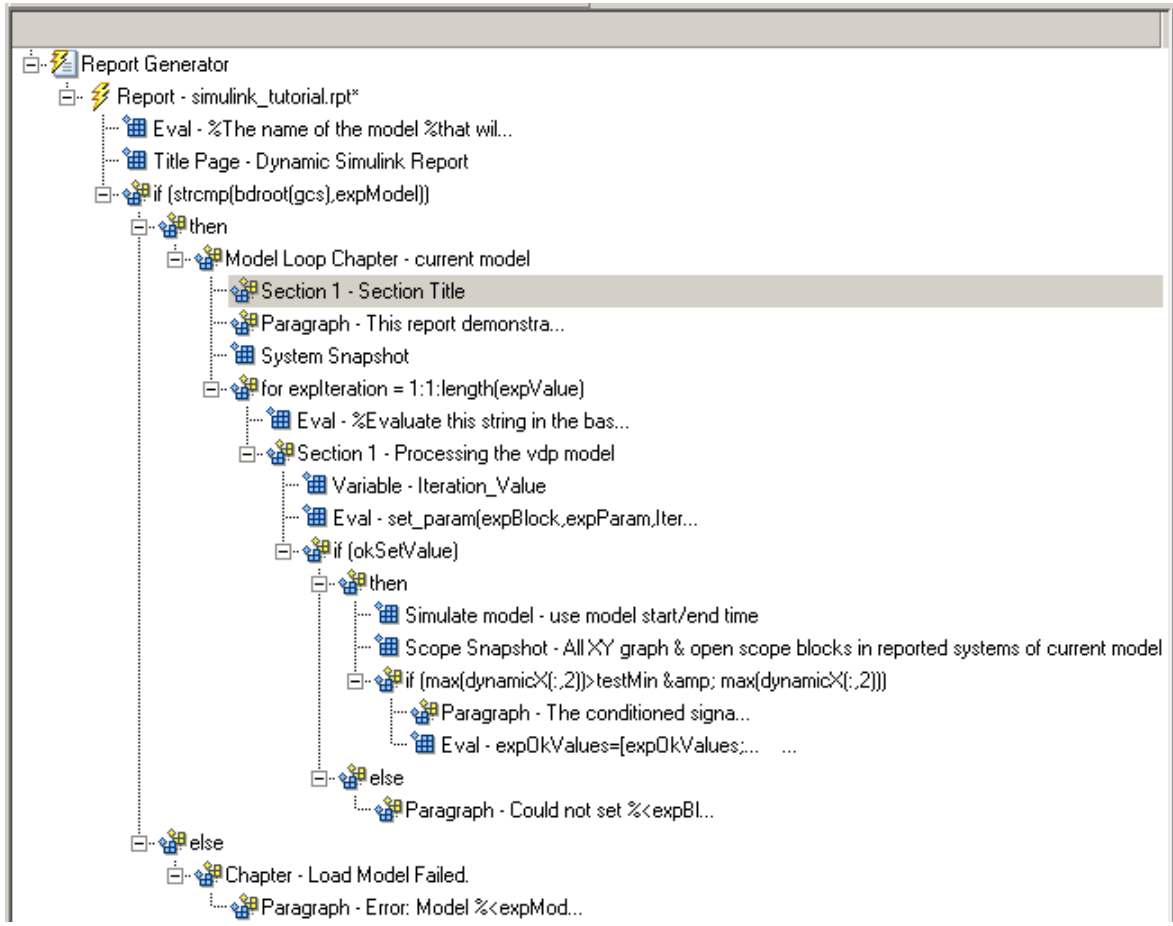
Now that you have collected all the desired values, create the post-test analysis section by creating a table and inserting it into your report at the end of this chapter.

1 In the Outline pane on the left, select the **Model Loop Chapter** component.

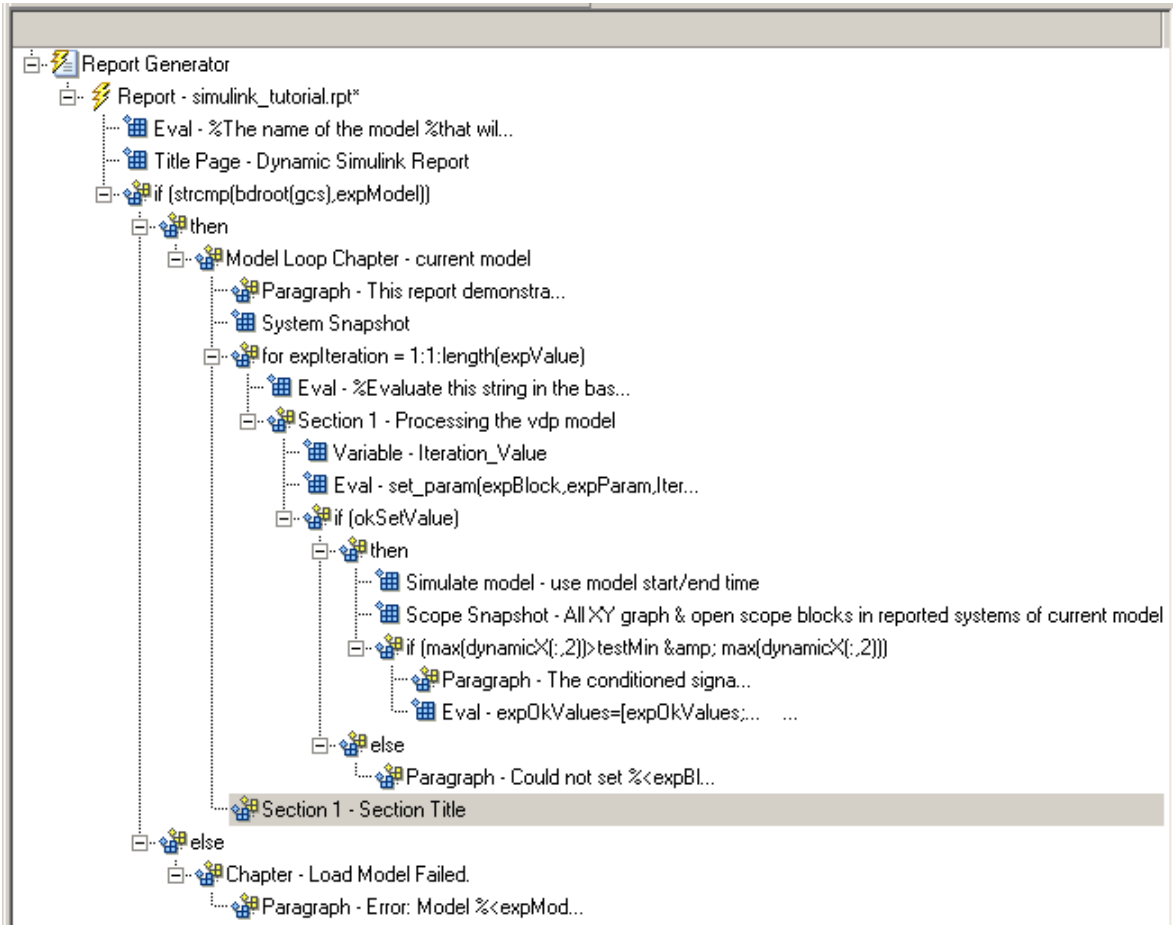


2 In the Library pane in the middle, under the **Formatting** category, double-click **Chapter/Subsection**.

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:

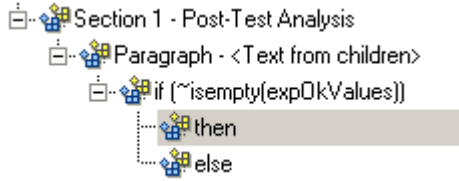
```
~isempty(expOkValues)
```

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:

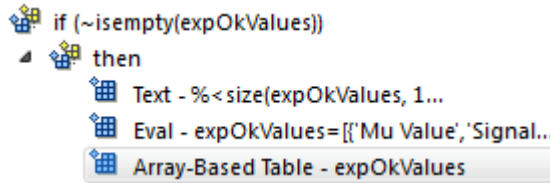
```
expOkValues=[{'Mu Value','Signal Maximum'} expOkValues];
```

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 **Table** component so it becomes a sibling of the **Text** and **Eval** components.



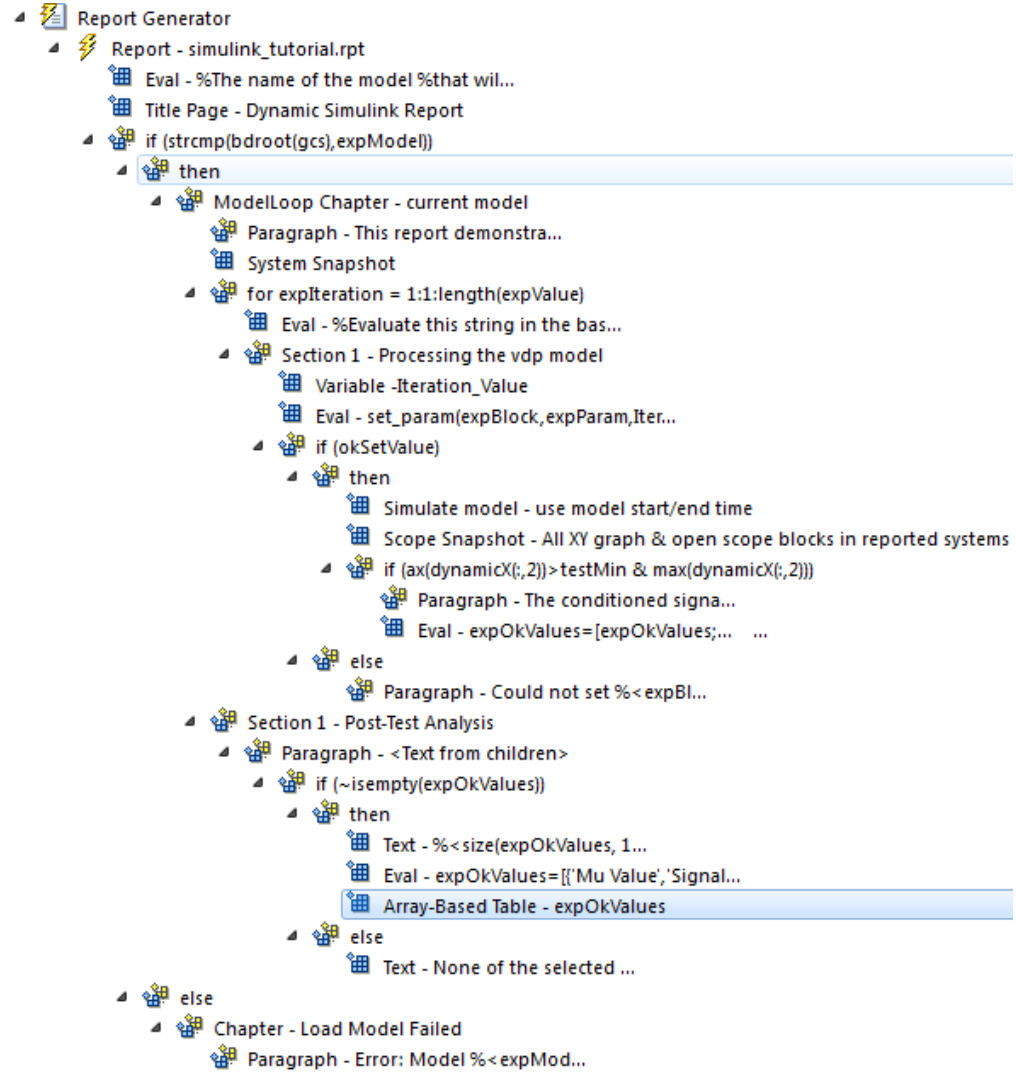
13 In the Properties pane on the right:

a In the **Workspace variable name** text box, enter `expOkValues`. The Simulink Report Generator software uses the contents of `expOkValues` 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.

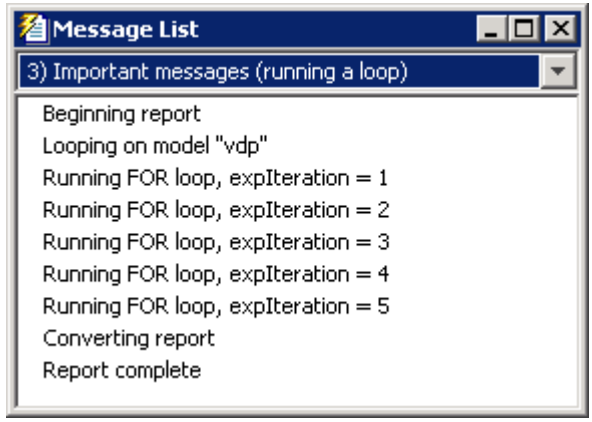


Generate the Report

To generate the report, click the Report icon on the toolbar. The following occurs:

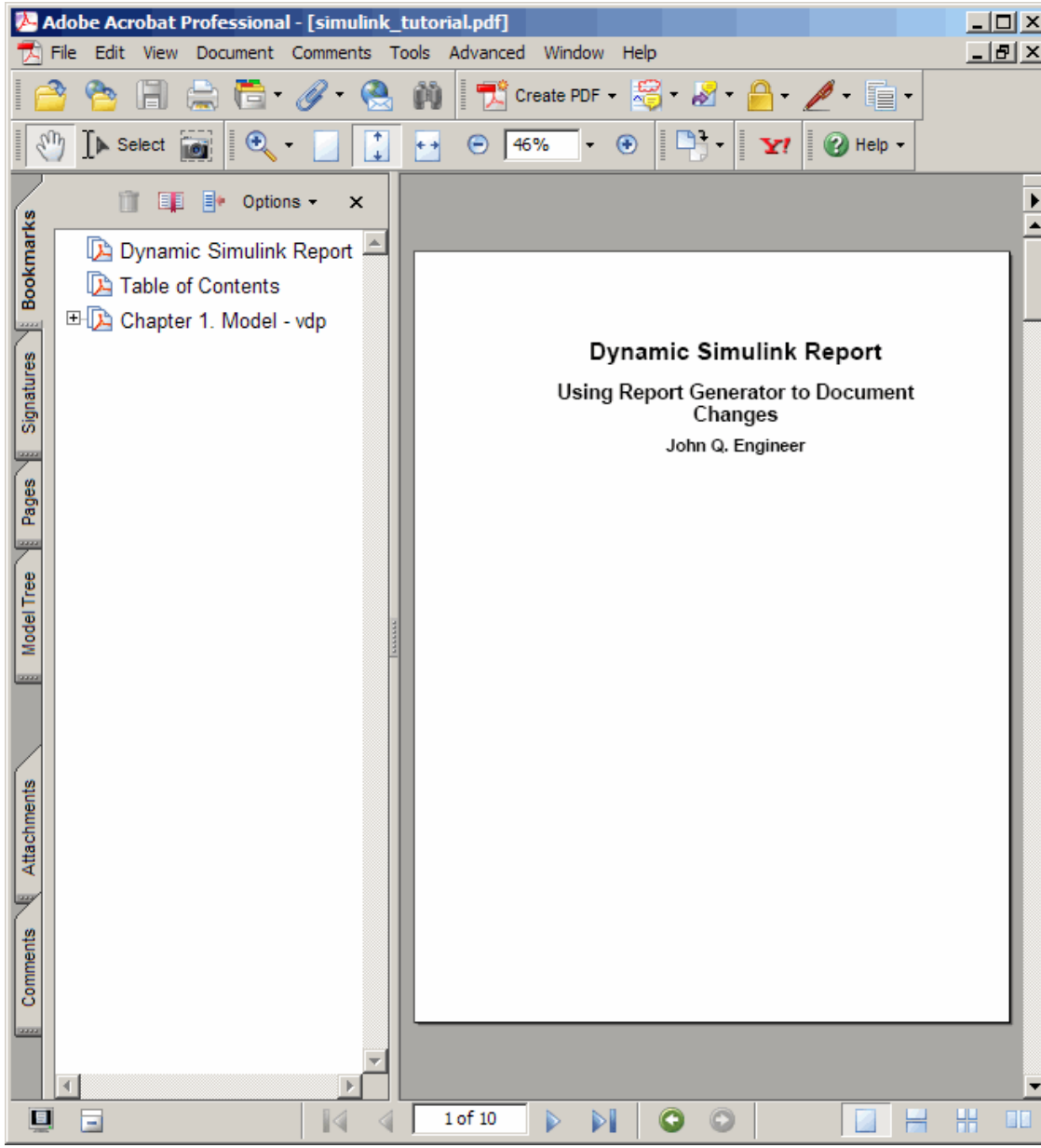
1. A Message List window appears, displaying informational and error messages as the report is processed. Specify the level of detail you would like the Message List window to display while the report is being generated. Options range from `0` (least detail) to `6` (most detail). Click the selection 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 is complete, Adobe® Acrobat® Reader opens your report in PDF format.

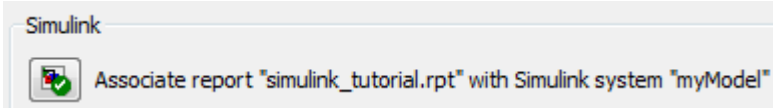

Adobe Acrobat Professional - [simulink_tutorial.pdf]
File Edit View Document Comments Tools Advanced Window Help
Create PDF
Select
46%
Help
Options
Bookmarks
Signatures
Pages
Model Tree
Attachments
Comments
Dynamic Simulink Report
Table of Contents
Chapter 1. Model - vdp
Dynamic Simulink Report
Using Report Generator to Document Changes
John Q. Engineer
1 of 10

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” (MATLAB Report Generator)

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” (MATLAB Report Generator).

You can use loop context functions with loop components. For details, see:

- “Filter with Loop Context Functions” on page 4-72
- “Loop Context Functions” on page 4-75

Filter with Loop Context Functions

In this section...
“Create and Save the Setup File” on page 4-72 “Add Components” on page 4-72 “Run the Report” on page 4-73

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 4-75.

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_outport.rpt`.

Add Components

Add these components to the report, in order.

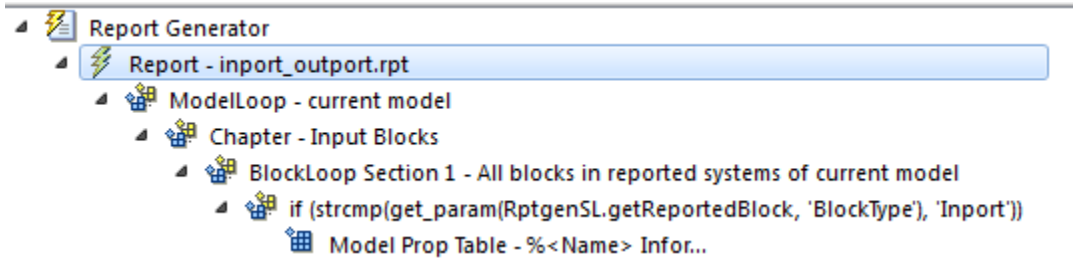
From this Library Folder	Add this Component	Set this Property
Simulink	**Model Loop**	N/A
Formatting	**Chapter**	**Title** to `Inport Blocks`
Simulink	**Block Loop**	N/A

From this Library Folder	Add this Component	Set this Property

Logical and Flow Control	**Logical If**	**Test Expression** to
		`strcmp(get_param...`
		`(RptgenSL.getReportedBlock,'BlockType'),...`
		`'Inport')`

`Simulink`	Simulink Property Table	N/A

The report setup file looks like this:

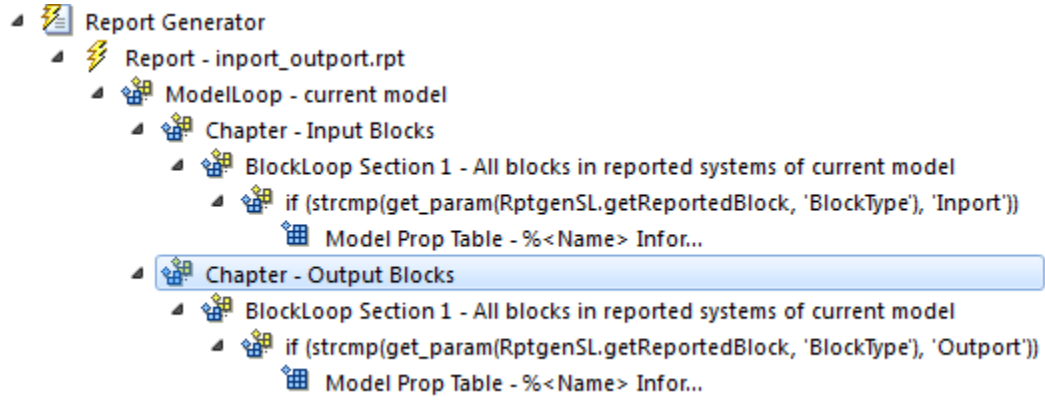


Run the Report

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 Outport blocks only, as shown below.


Report Generator
Report - inport_outport.rpt
ModelLoop - current model
Chapter - Input Blocks
BlockLoop Section 1 - All blocks in reported systems of current model
if (strcmp(get_param(RptgenSL.getReportedBlock, 'BlockType'), 'Inport'))
Model Prop Table - %<Name> Infor...
Chapter - Output Blocks
BlockLoop Section 1 - All blocks in reported systems of current model
if (strcmp(get_param(RptgenSL.getReportedBlock, 'BlockType'), 'Outport'))
Model Prop Table - %<Name> Infor...

Loop Context Functions

In this section...
“For Simulink Modeling Elements” on page 4-75 “For Stateflow Modeling Elements” on page 4-75

You can use these loop context functions in similar ways as shown in “Filter with Loop Context Functions” on page 4-72.

For Simulink Modeling Elements

Modeling Element	Looping Component	Function
Simulink modeling elements		
Block	**Block Loop**	`RptgenSL.getReportedBlock`
Signal	**Signal Loop**	`RptgenSL.getReportedSignal`
System	**System Loop**	`RptgenSL.getReportedSystem`
Model	**Model Loop**	`RptgenSL.getReportedModel`

For Stateflow Modeling Elements

Modeling Element	Looping Component	Function
Object	**Object Loop**	`RptgenSF.getReportedObject`
State	**State Loop**	`RptgenSF.getReportedState`
Chart	**Chart Loop**	`RptgenSF.getReportedChart`

Export Simulink Models to Web Views

- “Web Views” on page 5-2
- “Export Models to Web View Files” on page 5-4
- “Display and Navigate a Web View” on page 5-6
- “Search a Web View” on page 5-11
- “Create and Use a Web View” on page 5-14
- “Optional Web Views” on page 5-21
- “Capture and View Optional Web View Information” on page 5-22
- “Embedded Web View Reports” on page 5-24
- “Create an Embedded Web View Report Generator” on page 5-30
- “Specify Export Options for Embedded Web View Report” on page 5-32
- “Specify Document Content for Embedded Web View Report” on page 5-33
- “Generate Table of Contents for Embedded Web View Report” on page 5-34
- “Get Model Objects for Embedded Web View Report” on page 5-35
- “Create Hyperlinks for Embedded Web View Report” on page 5-37
- “Suppress Link Warning Messages for Embedded Web View Report” on page 5-39
- “Generate an Embedded Web View Report” on page 5-40
- “Web View” on page 5-46

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 5-6
- “Create and Use a Web View” on page 5-14

More About

- “Optional Web Views” on page 5-21

Export Models to Web View Files

In this section...
“Open the Web View Dialog Box” on page 5-4 “Export a Model to a Web View” on page 5-4

Open the Web View Dialog Box

To export a Web view, use the Web View dialog box. The way you access the dialog box differs, depending on whether you are using the Simulink Editor or the Report Explorer. You can also open the dialog box from the command line.

Interface	What You Do
Simulink Editor	Select **File** > **Export Model to** > **Web**.
Report Explorer	Select **Tools** > **Export Simulink to Web**.
Command line	Use the `slwebview` function without arguments.

Export a Model to a Web View

1. Open the model to export.
2. In the Simulink Editor, select **File** > **Export Model to** > **Web**.
3. In **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, in **Include Options**, select any kinds of systems you want the Web viewer 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 left **Referenced Models** unchecked 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. In the **Systems to Exclude** list of the systems you have selected to export, 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 Click **Export**.

Note If you use the Web View export option in the Report Explorer Properties pane, then click **Export model**. If you change the visible system or chart while the Report Explorer **Web View** pane is visible, the pane does not automatically change to show information about the newly visible system. To update the **Web View** pane, click **Refresh**.

---

9 If you have Simulink Coverage™ installed, you can use the **Optional Views** tab to select **Model Coverage** to export the associated kind of information.

If you have Simulink Requirements installed, you can use the **Optional Views** tab to select **Requirements** to export the associated kind of information.

See Also

Functions
`slwebview`

Related Examples

- “Display and Navigate a Web View” on page 5-6
- “Create and Use a Web View” on page 5-14

More About

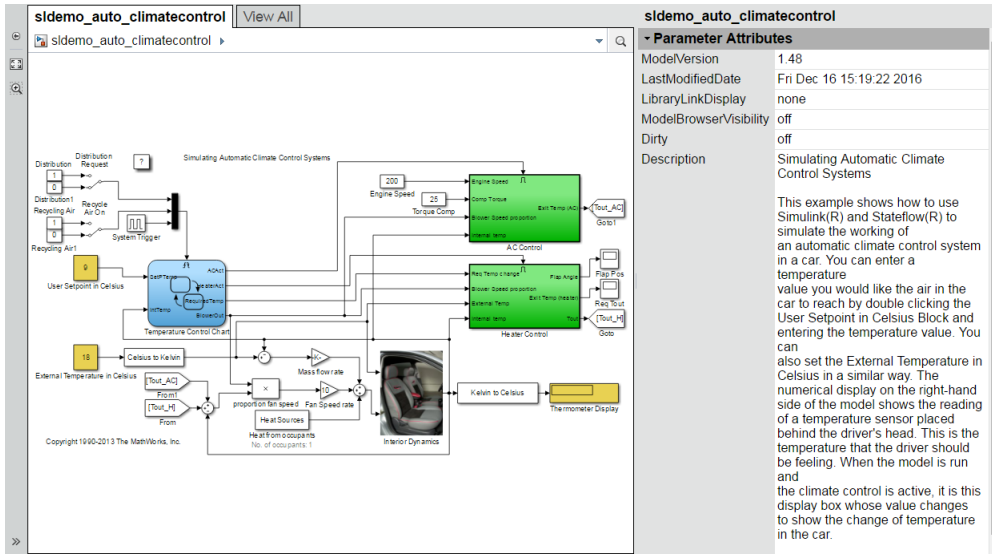
- “Web Views” on page 5-2
- “Web View Files” on page 5-2
- “Optional Web Views” on page 5-21

Display and Navigate a Web View

In this section...
“Display a Web View When You Export It” on page 5-6
“Open a Web View File in a Web Browser” on page 5-6
“View Contents of a System” on page 5-7
“View Block Parameters and Signal Properties” on page 5-8
“Access Optional Web View Information” on page 5-9

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, from the folder that contains the Web view files, select the HTML file.

Open the `webview.html` file to display the Web view. For details about file packaging and location, see “Web View Files” on page 5-2.

To use a Google® Chrome Browser, you need to do some setup.

Open a Web View in a Google Chrome Browser on a Windows Platform

1. If you do not already have a shortcut set up for Google Chrome, click the Windows **Start** button and search for `Chrome`.
2. Right-click and drag the Google Chrome icon to an open area on your desktop.
3. Right-click the icon and select **Create shortcut**.
4. Right-click the shortcut and select **Properties**.
5. In the **Target** edit box, append the following text: `--allow-file-access-from-files`. Be sure to use two hyphens at the beginning. Click **OK**.
6. Close all open Google Chrome browsers.
7. Use the shortcut to open a Google Chrome browser.
8. Open the Web view file.

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
```

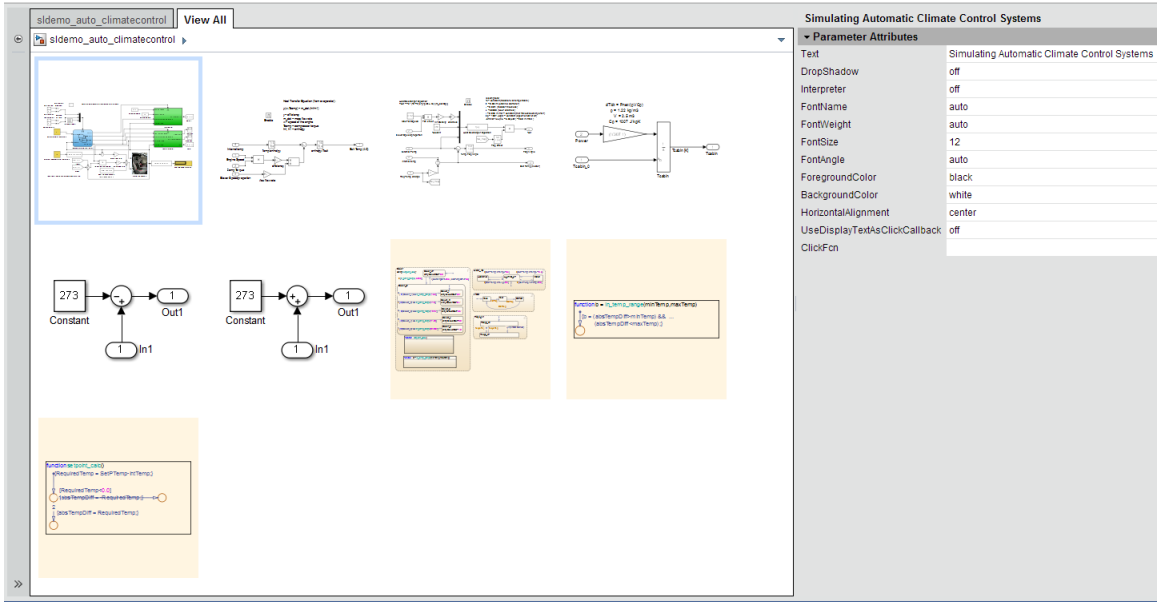
Open a Web View in a Google Chrome Browser on a Linux Platform

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.



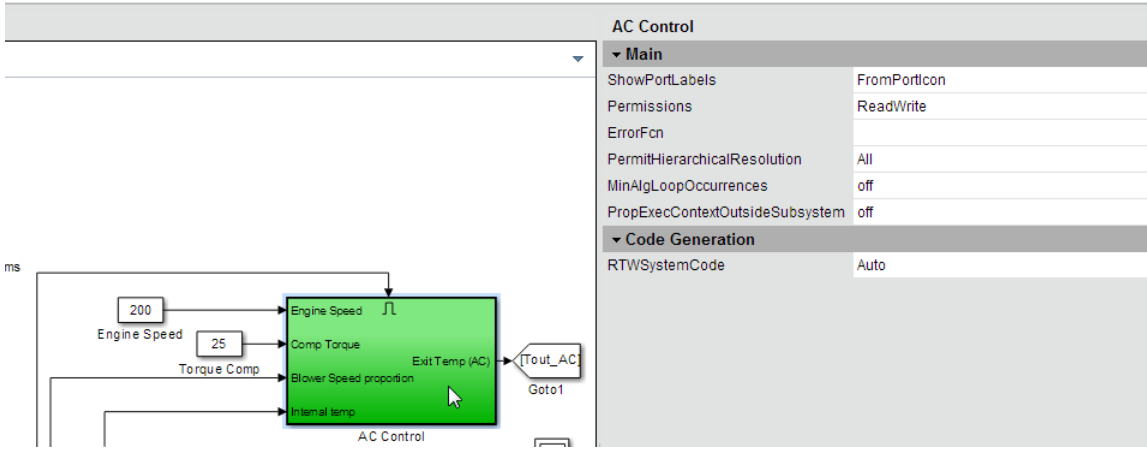
To view the contents of a specific system, use one of these approaches:

- 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 **Informer** pane below the model.

See Also

Functions

`slwebview`

Related Examples

- “Create and Use a Web View” on page 5-14

More About

- “Web Views” on page 5-2
- “Web View Files” on page 5-2

- “Optional Web Views” on page 5-21

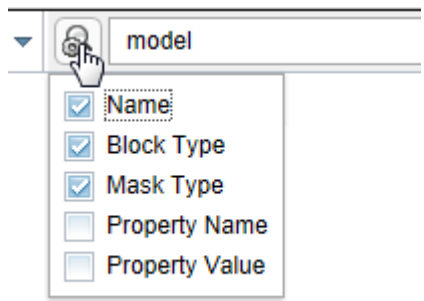
Search a Web View

In this section...
“Perform a Search” on page 5-11 “Sort Search Results” on page 5-13 “Navigate Between Search Results and Model Elements” on page 5-13

Perform a Search

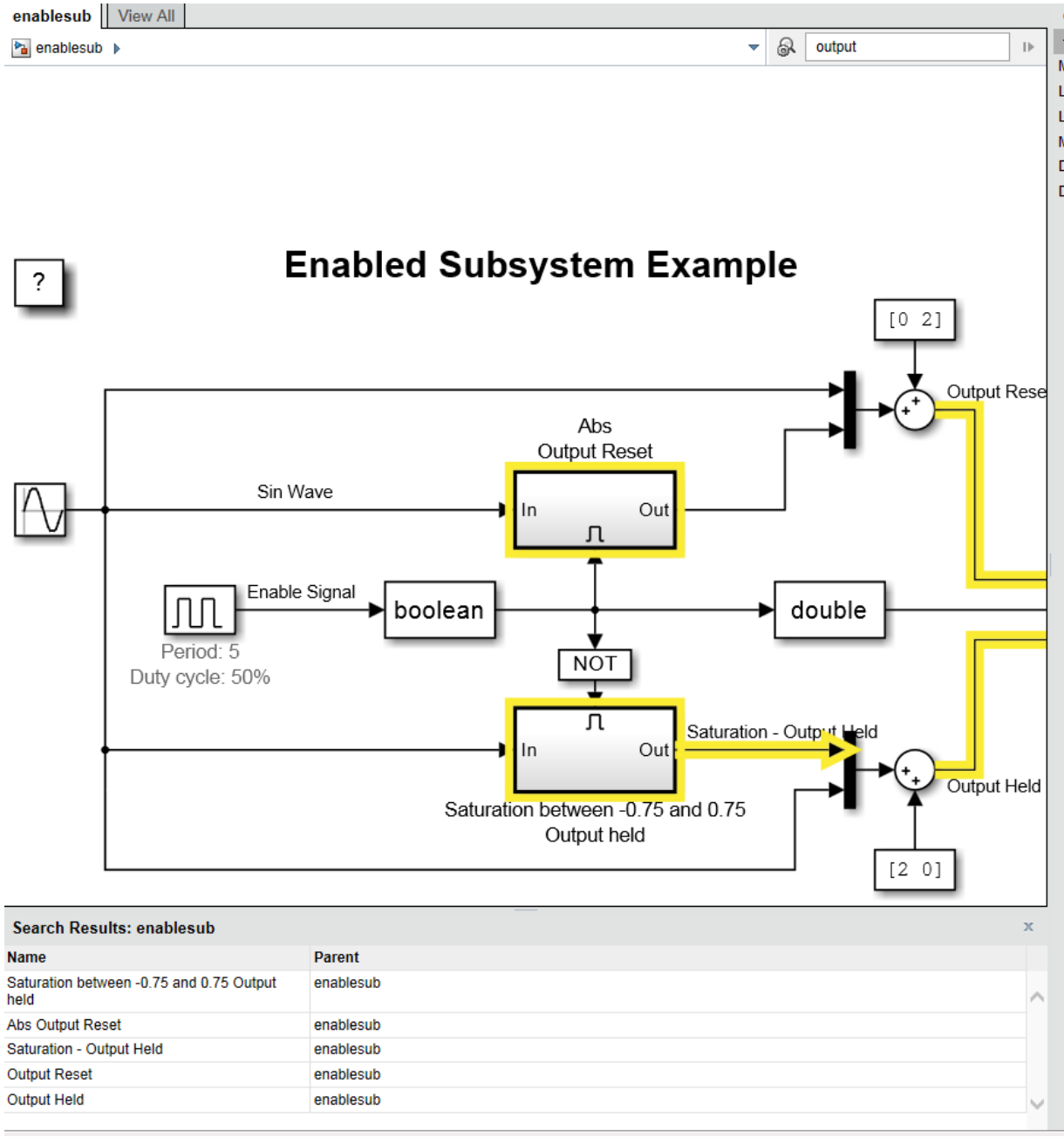
1. In a Web View, at the top of the displayed tab, click the search button .
2. In the search box, enter the search term.

   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
?
[0 2]
Output Rese
Abs
Output Reset
Sin Wave
In
Out
Enable Signal
boolean
double
Period: 5
Duty cycle: 50%
NOT
In
Out
Saturation - Output Held
Output Held
Saturation between -0.75 and 0.75
Output held
[2 0]
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 5-14
“Export Specific Systems” on page 5-14
“Navigate the Web View” on page 5-16
“Display Parameters and Properties of Blocks and Signals” on page 5-17
“Open the Web View” on page 5-19

About This Tutorial

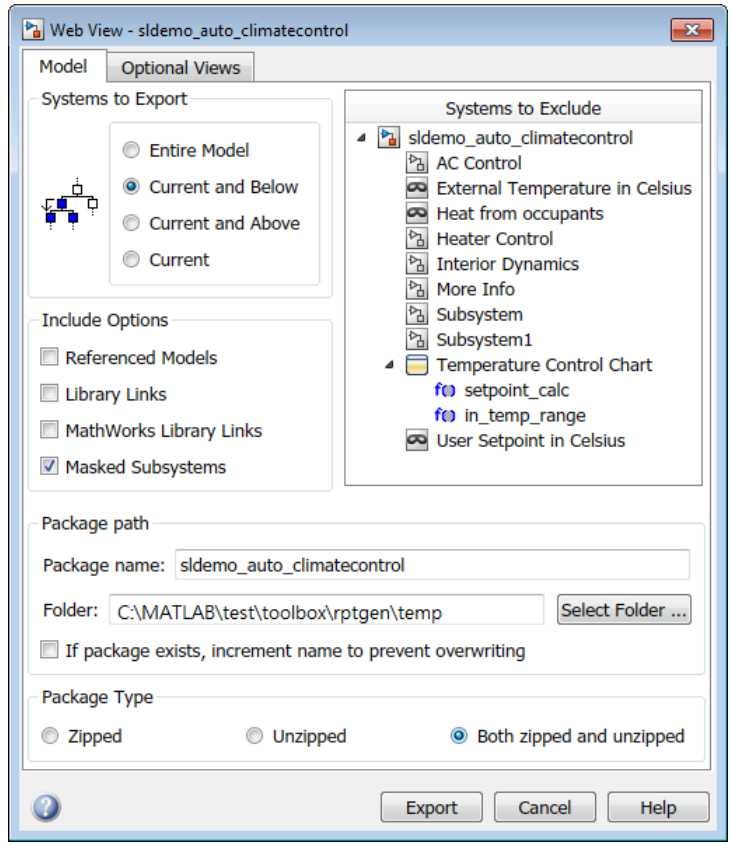
This tutorial takes you through the steps to export a Simulink model to a Web view.

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.

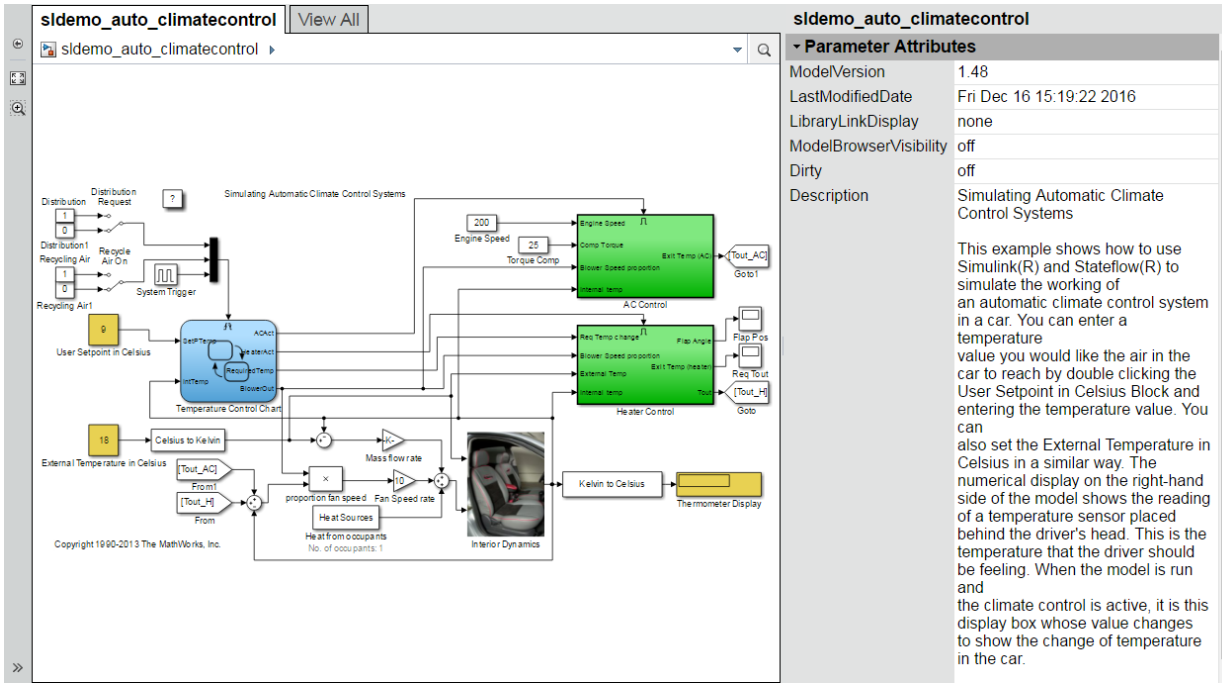
1. At the MATLAB command prompt, enter `sldemo_auto_climatecontrol` to open the Simulink model.
2. In the Simulink Editor, select **File > Export Model to > Web**.



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.

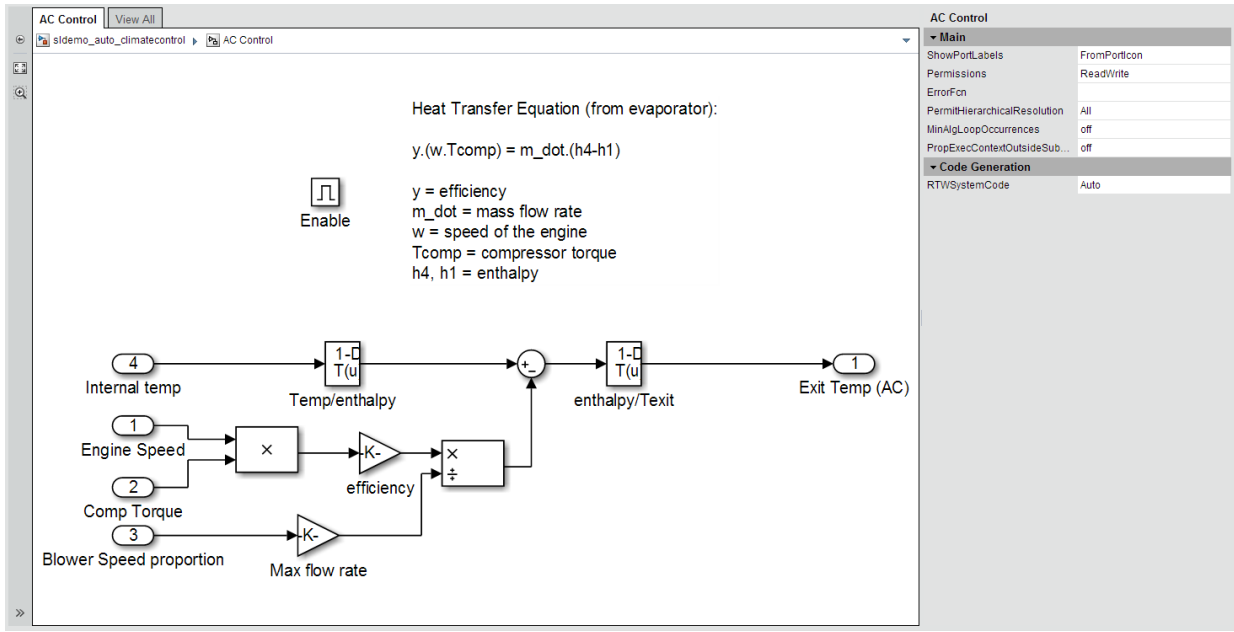


The Temperature Control Chart appears in the top level of the model, but you cannot open that chart in the Web view to see its contents.

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.

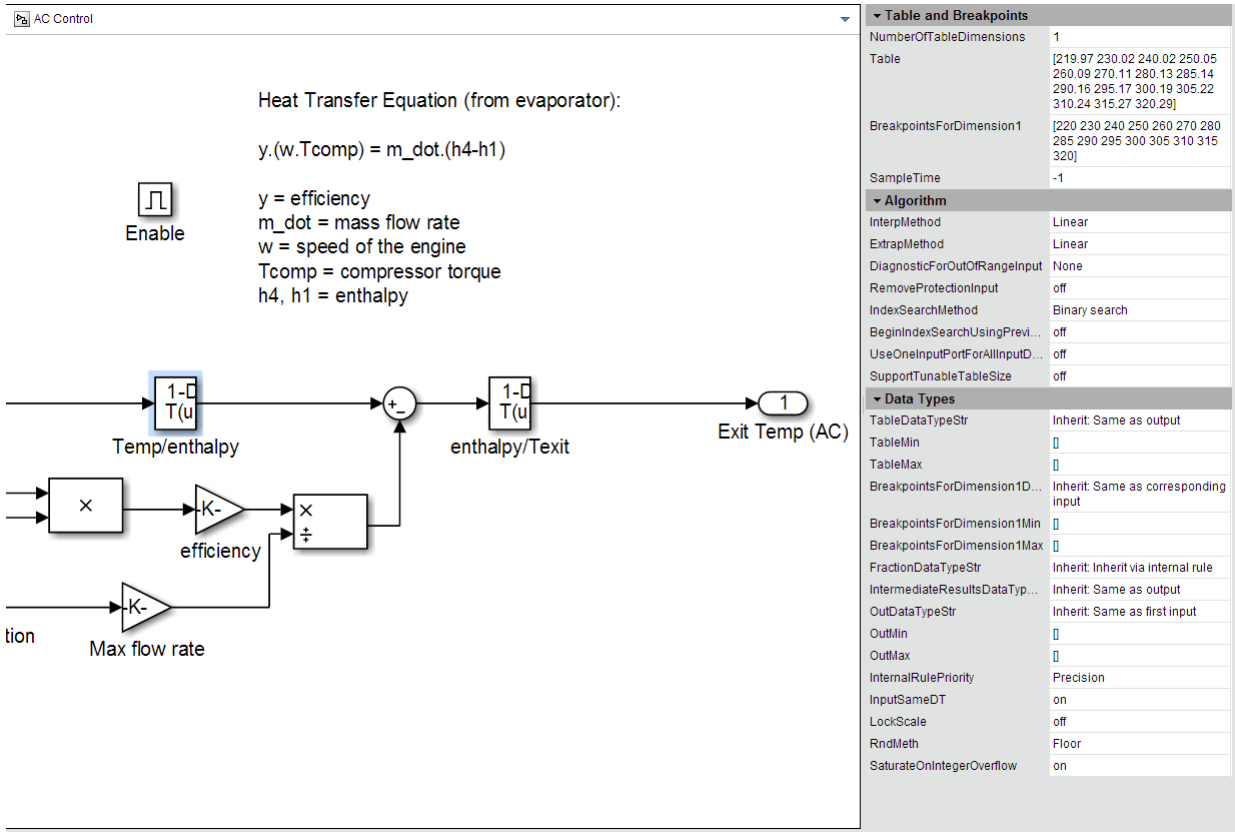


The AC Control subsystem appears in the model viewer. The tab label reflects the name of the currently displayed subsystem.

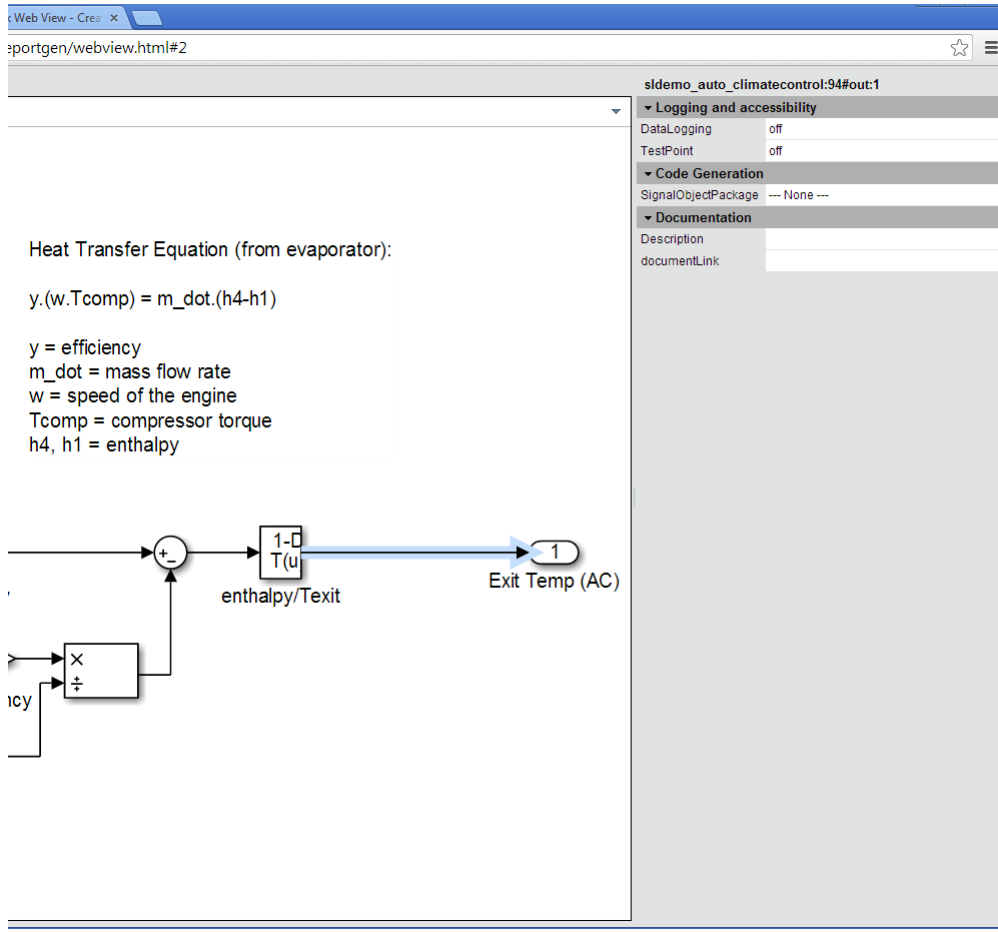
2. Open the model browser (hidden by default). Click **Hide/Show Model Browser** ».
3. Open another system, in a separate tab. In the model browser, **CTRL+click** select the Heater Control system.
4. Drag the AC Control system to the top of model viewer. Place the cursor in the display area, hold down 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.



4 Click the input signal for the Exit Temp (AC) block to display the signal properties.



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 5-14), 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 5-6

More About

- “Web Views” on page 5-2

Optional Web Views

Optional Web views provide information about a model in addition to the standard Web view information about blocks and signals in a model.

The Simulink Report Generator includes the following optional Web views that you can capture and view if you have required software installed:

- Requirements (requires Simulink Requirements)
- Model coverage (requires Simulink Coverage)

These optional views display requirements or model coverage information associated with the current Web view.

To include model coverage information in a Web view, set up a coverage report for the model and simulate the model before selecting the model coverage optional view option.

See Also

Related Examples

- “Capture and View Optional Web View Information” on page 5-22
- “Export Models to Web View Files” on page 5-4

More About

- “Optional Web Views” on page 5-21

Capture and View Optional Web View Information

In this section...
“Capture Optional Web View Information for a Model” on page 5-22 “View Optional Web View Information” on page 5-22

Capture Optional Web View Information for a Model

When you create a Web view, you can include optional Web view information.

To add the model coverage optional view information to a Web view for a model, you must have Simulink Coverage installed. To add the requirements optional view information to a Web view for a model, you must have Simulink Requirements installed.

1. In the Web View dialog box, open the **Optional Views** tab.
2. Select each optional view (for example, **Model Coverage** or **Requirements**) that you want to capture the associated information for.

Tip The **Optional Views** tab appears only if you can access an optional Web view.

Alternatively, in the Simulink Editor, click **Analysis > Requirements Traceability > Generate Web View**.

View 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.

In a Web view, click a highlighted block (for example, blocks with an orange border have requirements information). The information for that block appears in the **Informer** pane below the model.

See Also

Related Examples

- “Export Models to Web View Files” on page 5-4

More About

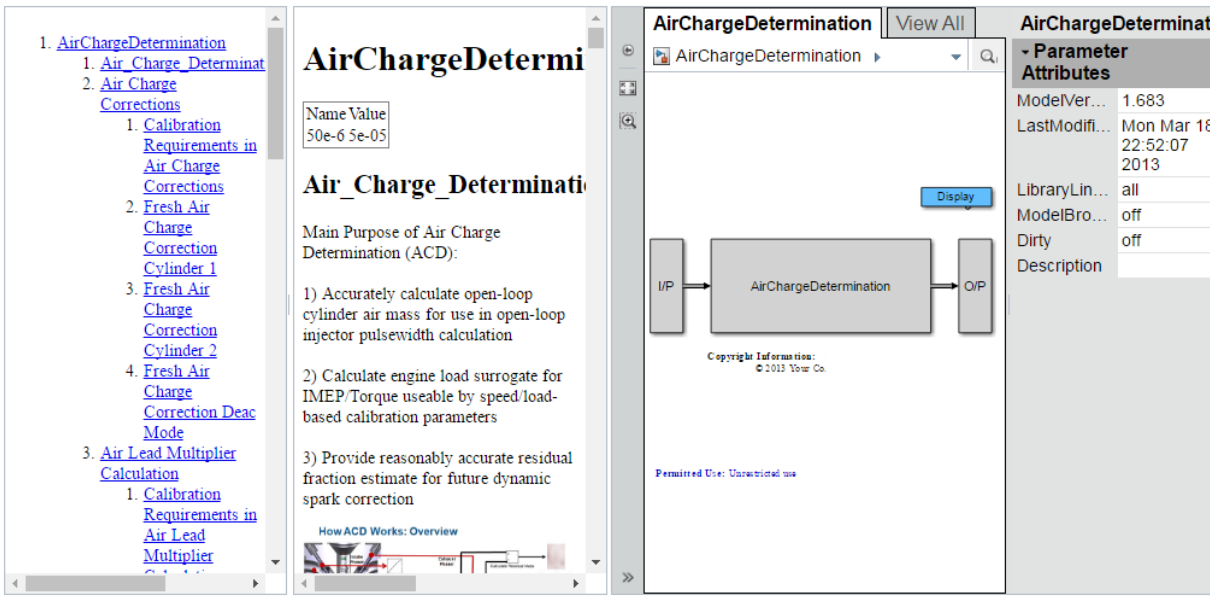
- “Optional Web Views” on page 5-21

Embedded Web View Reports

In this section...
“What Is Embedded Web View?” on page 5-24 “Navigating an Embedded Web View Report” on page 5-25 “Embedded Web View Packaging” on page 5-28 “View Embedded Web View Reports” on page 5-28

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.



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

This example uses the MATLAB Report Generator DOM API and the Simulink Report Generator Embedded Web View API. You use the DOM API to generate the calibration guide content (see “MATLAB Report Generator APIs” (MATLAB Report Generator)). You 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.

- `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 Block 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.

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 System 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:
  - `System input signals`
  - `System output signals`
  - `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:
  - `Block input signals`
  - `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:
  - `Block input signals`
  - `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 4-75.

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 outports. 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(gcb,'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.

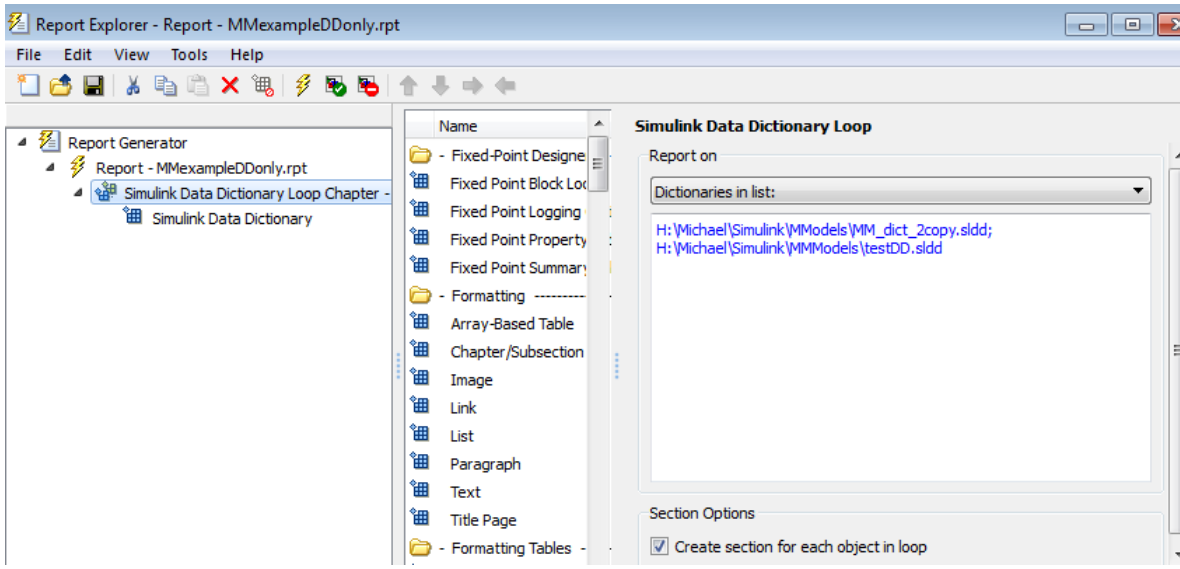
Fields to Report

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:

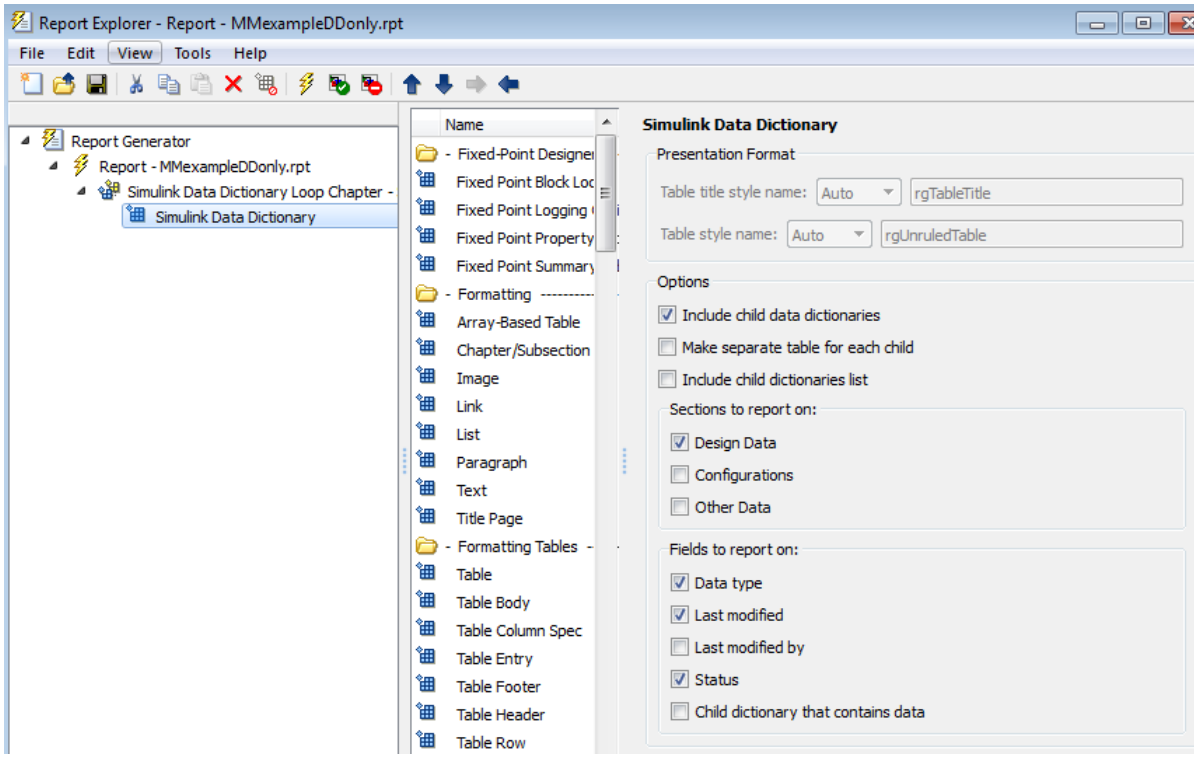
- **Data type**
- **Last modified**
- **Last modified by**
- **Status**
- **Referenced dictionary that contains data**

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.

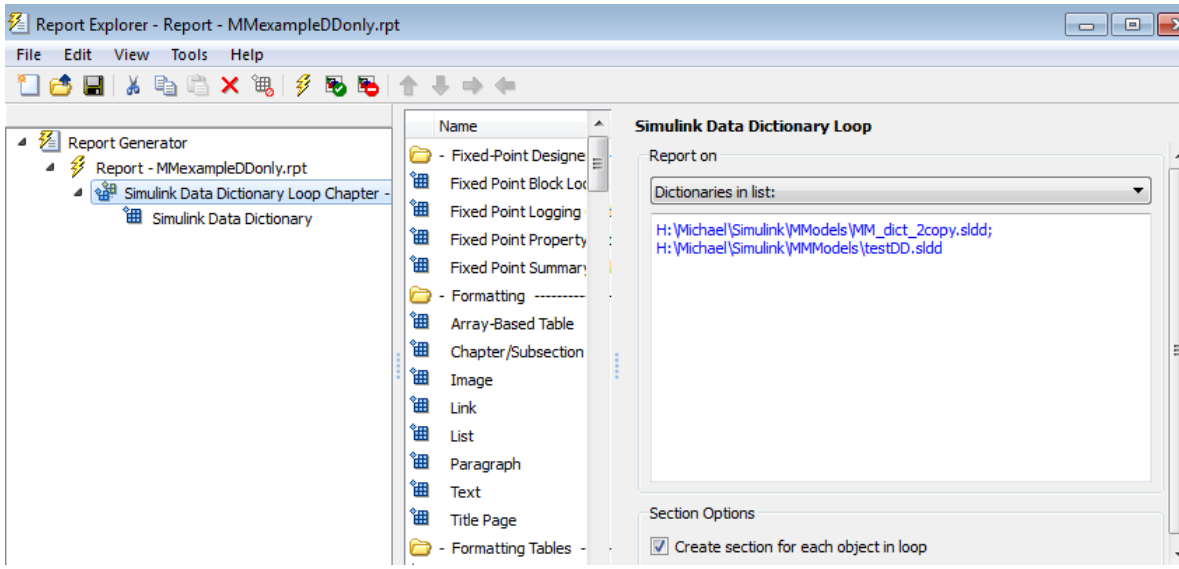
Use a Summary Table component to show annotation objects in reports. Each Summary Table component creates a single table with each reported annotation on a single row of the table.

Section Options

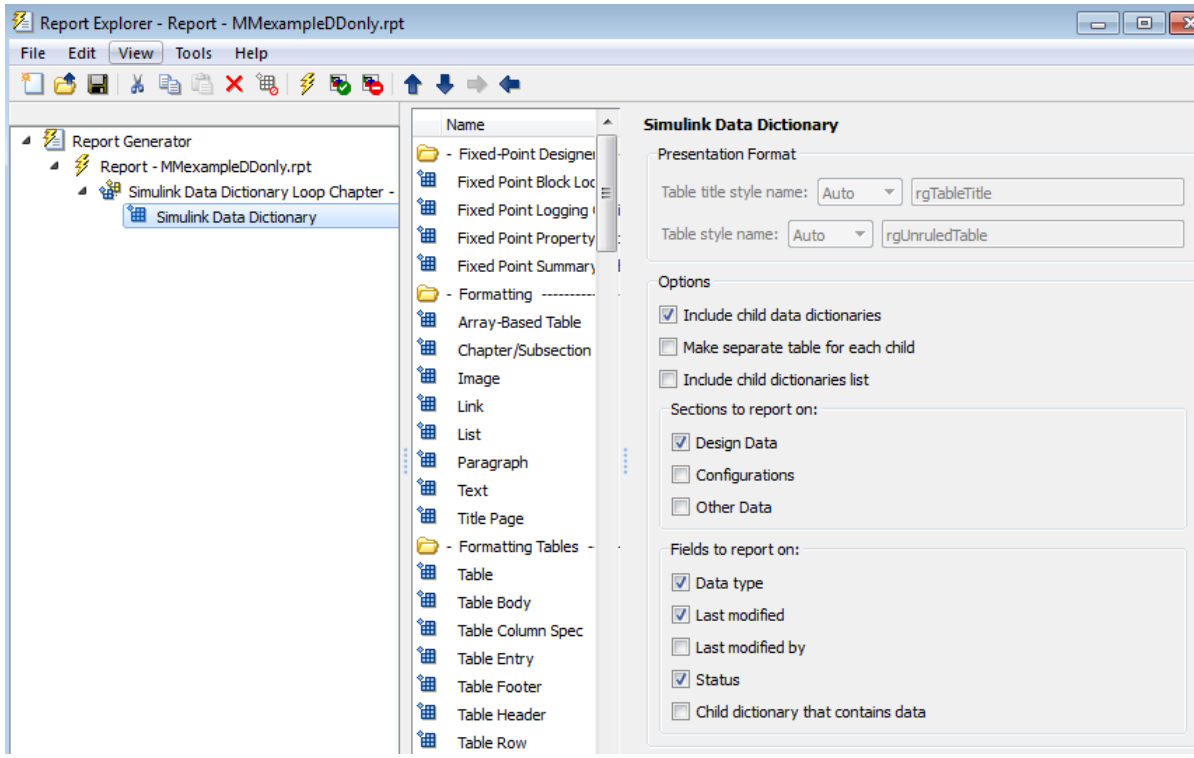
Create section for each object in loop: Create a separate chapter for each data dictionary.

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

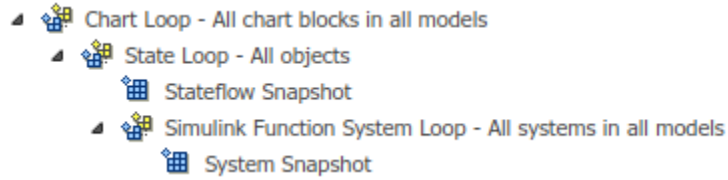
`Block Loop`, `System Loop`

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 descendent 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” (Simulink), 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`
`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
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

Block	Library	Reference Block	Status
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 Cx Cz Kg Ki	Simulink	Interpolation (n-D) using PreLookup	resolved
Alpha Index Mach Index Mach Index |Alpha| Index	Simulink	PreLookup Index Search	resolved

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” (MATLAB Report Generator).

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” (MATLAB Report Generator).

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.

Solver ode45	*ZeroCross* on	*StartTime* 0.0 *StopTime* 60
RelTol 1e-4	*AbsTol* 1e-6	*Refine* 1
InitialStep auto	*FixedStep* auto	*MaxStep* auto
LimitMaxRows off	*MaxRows* 1000	*Decimation* 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**.

`%<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

Name	Blocks
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 LoopSimulink 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

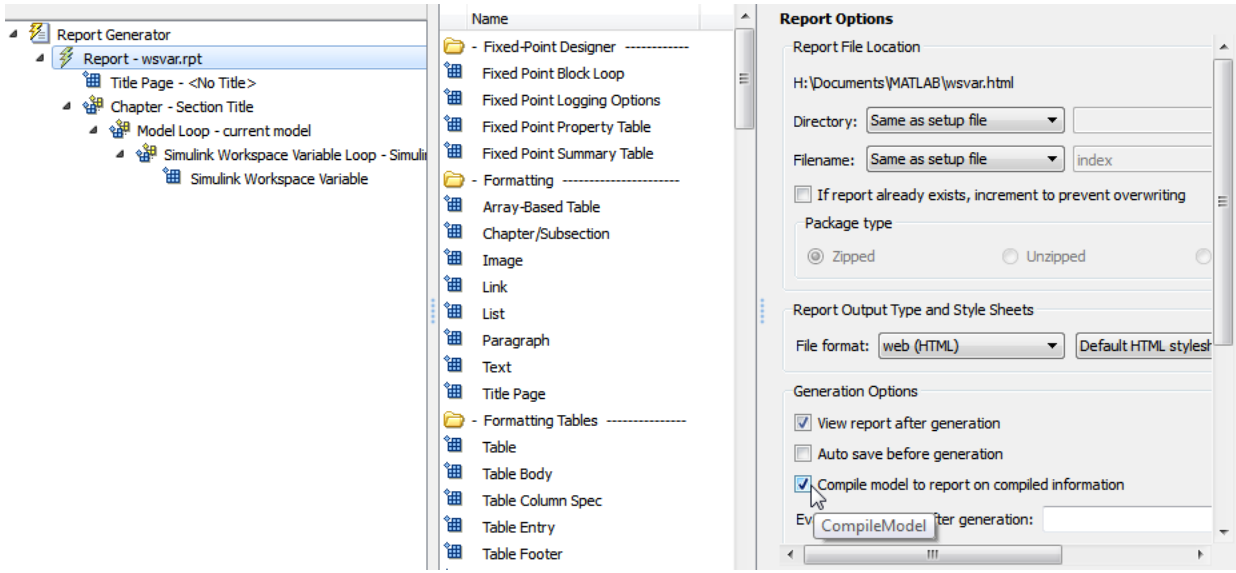
This component provides information about those workspace variables that the Simulink model uses, in a loop generated by a `Simulink Workspace Variable Loop` component. Your report setup must include `Simulink Workspace Variable` component as a child of a `Simulink Workspace Variable Loop` component.

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`

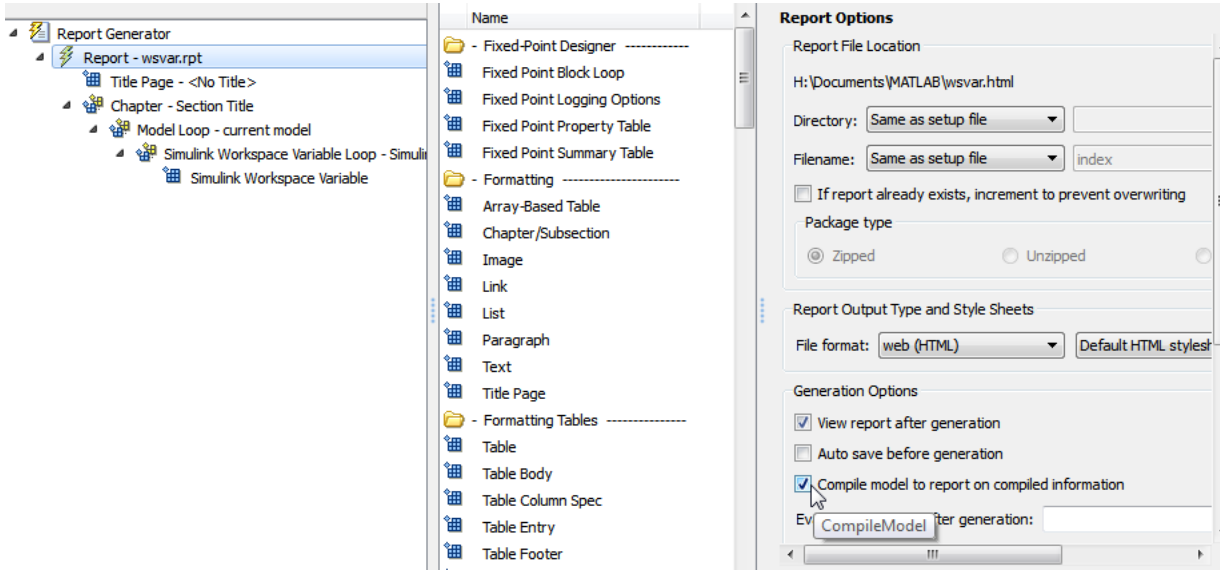
Simulink Workspace Variable Loop

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 4-75.

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`, `State 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” (MATLAB Report Generator).

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” (MATLAB Report Generator).

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`, and `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 descendent 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 4-75.

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” (Simulink), 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 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.

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.

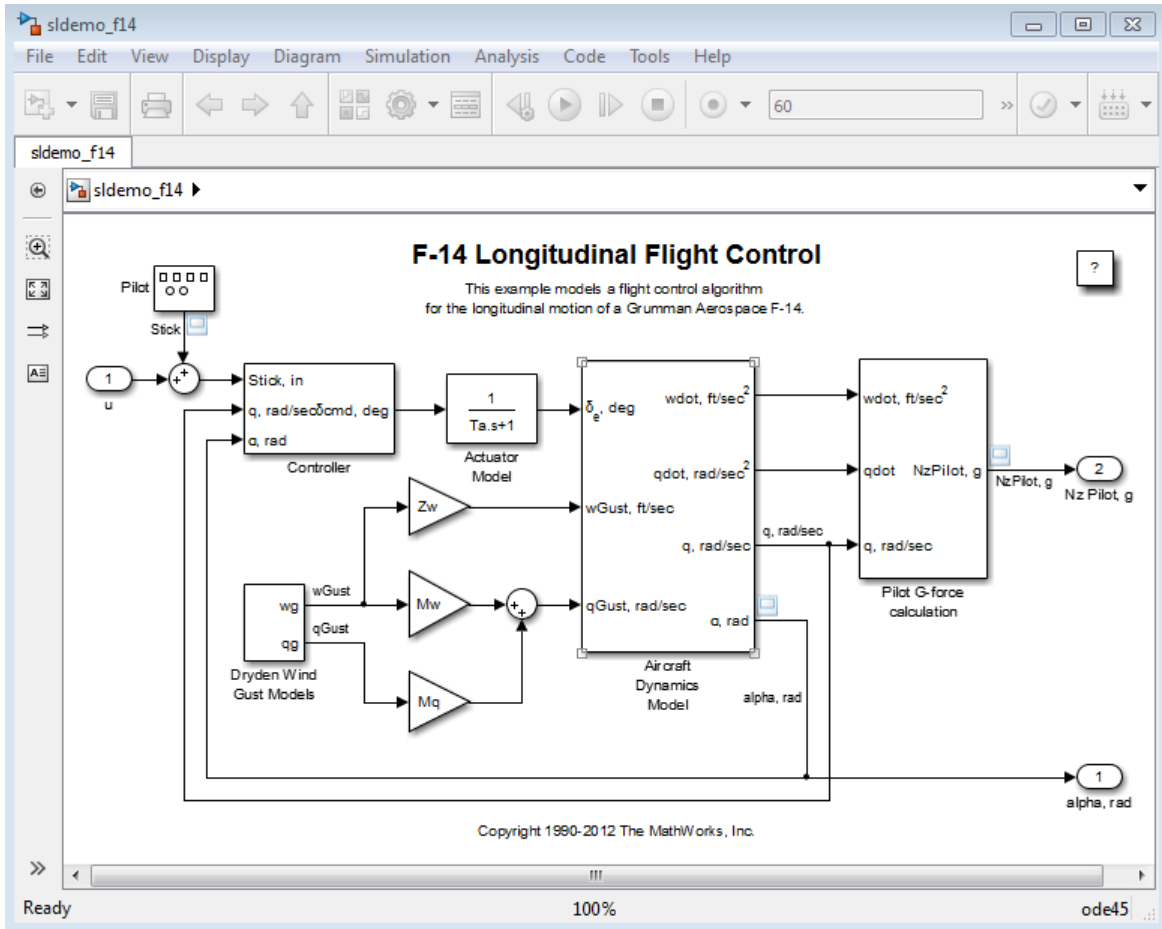
    Note Selecting **Use printframe paper settings** deactivates the `Custom` and `Zoom` options and automatically scales the image to the print frame size.

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 — Alphabetical List

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 Simulinkmodels, with two-way hyperlinks between the document and the web view.

This class provides the following facilities for generating embedded web view reports:

- A DOM document to generate the report
- A DOM 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.Embedded WebViewDocument(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.Embedded WebViewDocument(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 `.htmx`. If you do not specify an extension, the report generator appends `.htmx`.

`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 `.htmtx` 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

Method	Purpose
`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
`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

See Also

`slreportgen.webview.EmbeddedWebViewDocument` |
`slreportgen.webview.ExportOptions`

Topics

“Create an Embedded Web View Report Generator” on page 5-30
“Embedded Web View Reports” on page 5-24

Introduced in R2017a

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 | character vector | ...

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
- 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.

`SnapshotFormat` — Snapshot image format

`auto` (default) | ...

Snapshot image format, specified as a string or character array. Supported formats are:

- `auto` — Uses `.svg` format for HTML or PDF output. Uses `.emf` format for Word output on Windows. Uses `.png` for Word output on Unix or Mac.
- `.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

`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.

`Height` — Height of snapshot image

string

Height of snapshot image for custom scaling, specified as a string. This property applies only if `Scaling` 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`

Units	Units Abbreviation
millimeters	`mm`
picas	`pc`
points	`pt`

`Width` — Width of snapshot image

string

Width of snapshot image for custom scaling, specified as a string. This property applies only if `Scaling` 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 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
string | character array | reporter or report | DOM document or document part

Source of template for this reporter, specified as one of these values:

- String or character array specifying the path of the file containing the template for this reporter
- Reporter or report whose template is used as the template for this reporter or whose template library contains the template for this reporter
- DOM document or document part whose template is used as the template for this reporter or whose template library contains the template for this reporter

The specified template must be of the same type as the report to which this reporter is appended. For example, for a 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
string | character array

Name of template for this reporter, specified as a string or character array. The template for this reporter is in the template library of the template source (`TemplateSrc`) for this reporter.

`LinkTarget` — Hyperlink target for this reporter
string | character array | `mlreportgen.dom.LinkTarget` object

Hyperlink target for this reporter, specified as a string, 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.

Example: `firstchapter.LinkTarget = 'ch1_link';`

Methods

createTemplate	Create diagram template
customizeReporter	Create custom diagram reporter class
getClassFolder	Diagram class definition file location

Inherited Methods

`getImpl`	Get implementation of reporter

Copy Semantics

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

Examples

Add Top Level of Model Diagram

Add a snapshot of the top level of the `vdp` model to a report.

```
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.SnapshotFormat = '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);
```

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.Report` | `slreportgen.report.SimulinkObjectProperties` | `slreportgen.report.StateflowObjectProperties`

Topics

“Report Generation for Simulink and Stateflow Elements” on page 1-10

“Report Generation for Common Report Elements” (MATLAB Report Generator)

Introduced in R2017b

slreportgen.report.LookupTable class

Package: slreportgen.report

Lookup table blocks 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

`reporter = 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.

`reporter = 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 versusvs breakpoints, and a data types table.

`reporter = 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 `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.

`PlotReporter` — Lookup table data plot reporter
`Figure` reporter (default) | custom reporter

Lookup table data plot reporter, specified as `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
string | character array | reporter or report | DOM document or document part

Source of template for this reporter, specified as one of these values:

- String or character array specifying the path of the file containing the template for this reporter
- Reporter or report whose template is used as the template for this reporter or whose template library contains the template for this reporter
- DOM document or document part whose template is used as the template for this reporter or whose template library contains the template for this reporter

The specified template must be of the same type as the report to which this reporter is appended. For example, for a 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
string | character array

Name of template for this reporter, specified as a string or character array. The template for this reporter is in the template library of the template source (`TemplateSrc`) for this reporter.

`LinkTarget` — Hyperlink target for this reporter
string | character array | `mlreportgen.dom.LinkTarget` object

Hyperlink target for this reporter, specified as a string, 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.

Example: `firstchapter.LinkTarget = 'ch1_link';`

Methods

createTemplate	Create Simulink lookup table block reporter template
customizeReporter	Create custom LookupTable reporter class
getClassFolder	Lookup Table reporter class definition file location

Inherited Methods

`getImpl`	Get implementation of reporter

Copy Semantics

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

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
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

1

Chapter 1. sf_car/Engine/engine torque

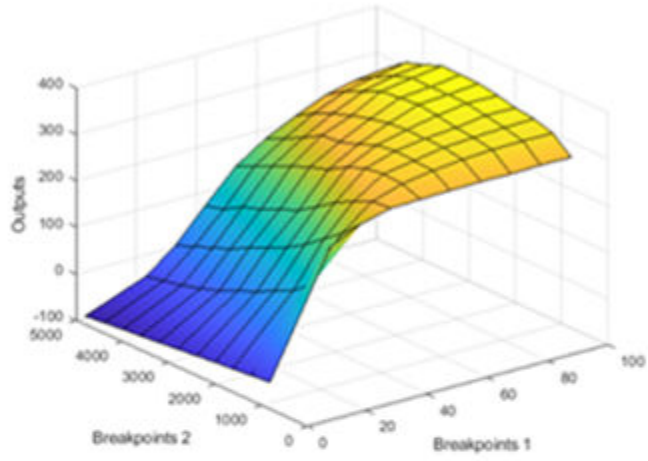


Figure 1.1. engine torque (:,:)

Table 1.2. engine torque Data Types

Data Type	Value
Table data type	double
Intermediate results data type	double
Output data type	double
Fraction data type	double
Breakpoints data type for dimension 1	double
Breakpoints data type for dimension 2	double

Note:

- The reported lookup table values result from evaluating the expression 'emap' during report generation. The values during simulation may differ from the reported values.
- The reported 'BreakPoint1' result from evaluating the expression 'thvec' during report generation. The values during simulation may differ from the reported values.
- The reported 'BreakPoint2' result from evaluating the expression 'nevec' during report generation. The values during simulation may differ from the reported values.

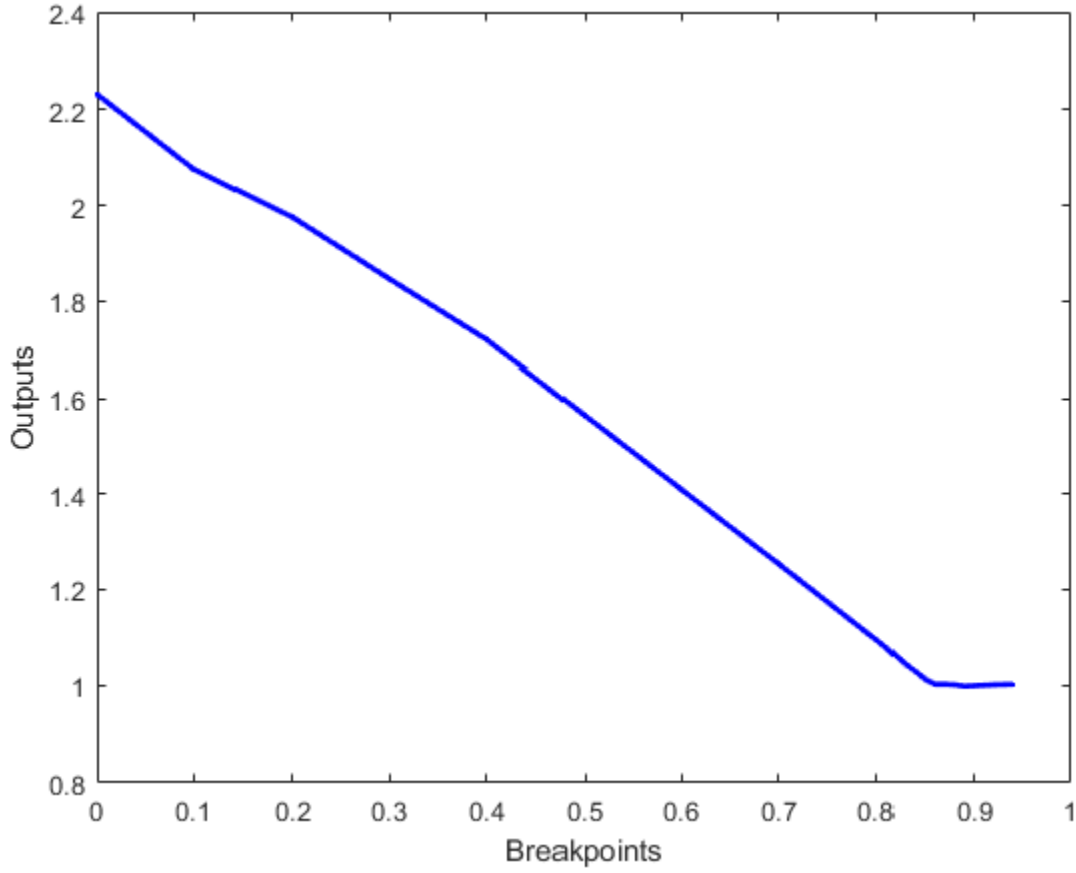
2

Change Lookup Table Plot Height and Width

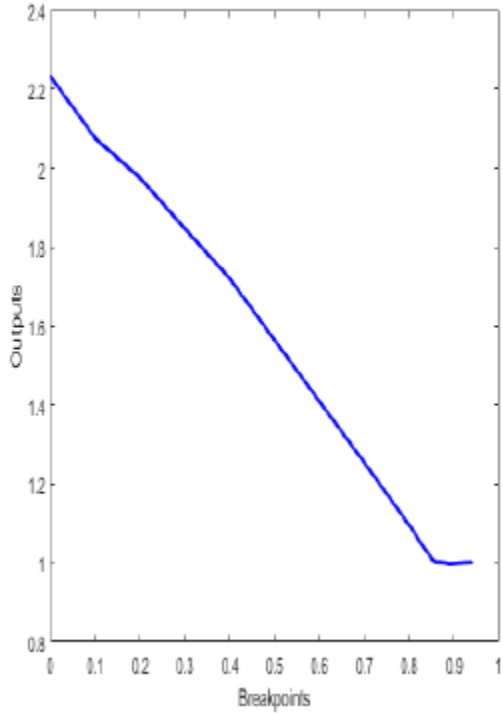
Create a PDF report generator that specifies the plot height and width returned by the `LookupTable` reporter. This example uses the `sf_car` model and reports on its `Torque ratio` Lookup Table (n-D) block. This block is a 1-D lookup table and is in the `transmission/Torque Converter` subsystem of the `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.



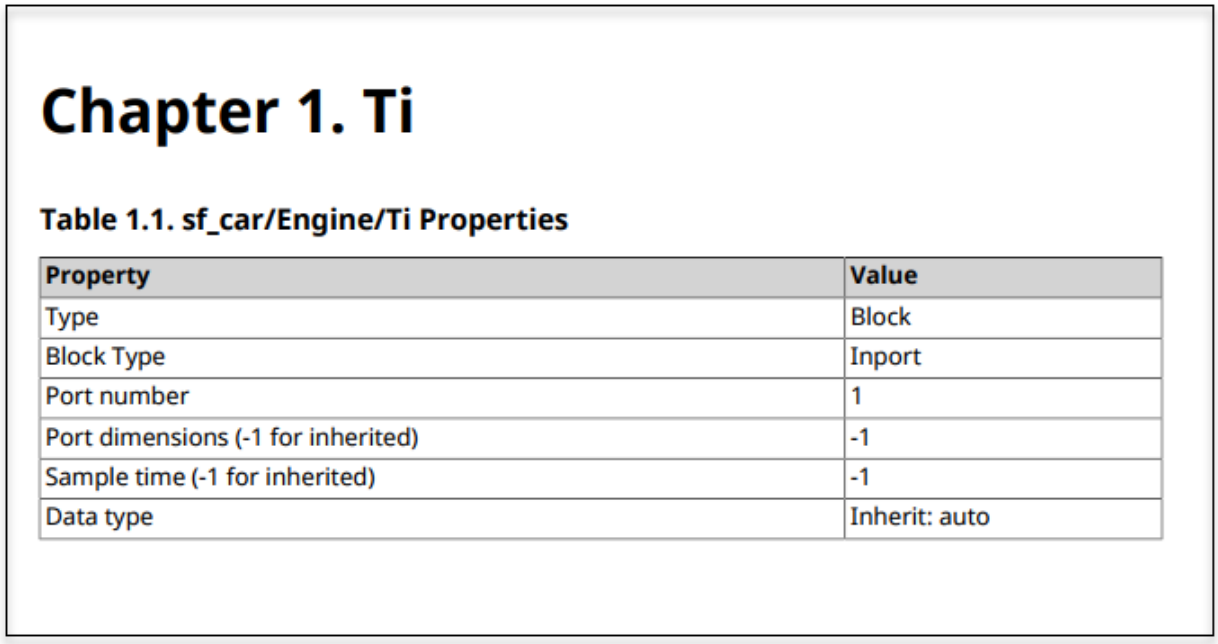
Find and Report on Lookup Tables and Other Blocks

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

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

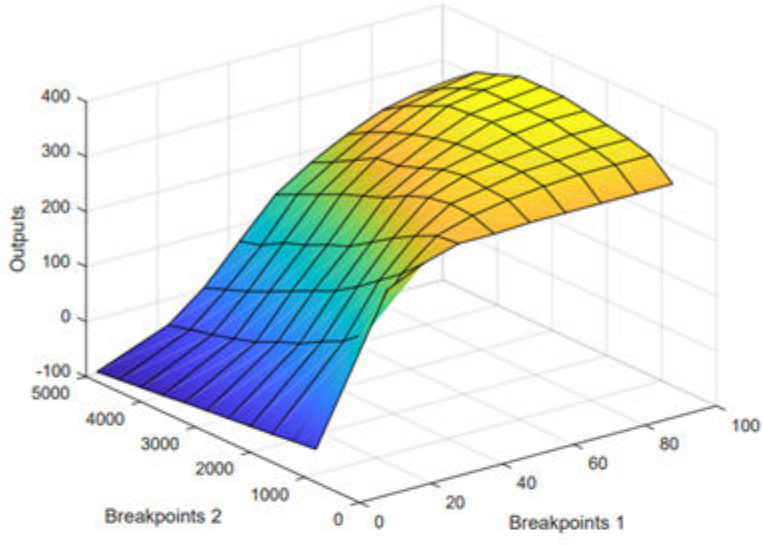


Figure 5.1. engine torque (:,:)

Table 5.2. engine torque Data Types

Data Type	Value
Table data type	double
Intermediate results data type	double
Output data type	double
Fraction data type	double
Breakpoints data type for dimension 1	double
Breakpoints data type for dimension 2	double

Note:

- The reported lookup table values result from evaluating the expression 'emap' during report generation. The values during simulation may differ from the reported values.
- The reported 'BreakPoint1' result from evaluating the expression 'thvec' during report generation. The values during simulation may differ from the reported values.
- The reported 'BreakPoint2' result from evaluating the expression 'nevec' during report generation. The values during simulation may differ from the reported values.

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.htmtx` 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`

Topics

“Report Generation for Simulink and Stateflow Elements” on page 1-10
“Report Generation for Common Report Elements” (MATLAB Report Generator)
“Templates” (MATLAB Report Generator)

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 `Compiiled 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 Properties	Description
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
string | character array | reporter or report | DOM document or document part

Source of template for this reporter, specified as one of these values:

- String or character array specifying the path of the file containing the template for this reporter
- Reporter or report whose template is used as the template for this reporter or whose template library contains the template for this reporter
- DOM document or document part whose template is used as the template for this reporter or whose template library contains the template for this reporter

The specified template must be of the same type as the report to which this reporter is appended. For example, for a 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
string | character array

Name of template for this reporter, specified as a string or character array. The template for this reporter is in the template library of the template source (`TemplateSrc`) for this reporter.

`LinkTarget` — Hyperlink target for this reporter
string | character array | `mlreportgen.dom.LinkTarget` object

Hyperlink target for this reporter, specified as a string, 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.

Example: `firstchapter.LinkTarget = 'ch1_link';`

Methods

createTemplate	Create MATLAB Function reporter template
customizeReporter	Create custom MATLAB Function reporter class
getClassFolder	MATLAB Function reporter class definition file location

Inherited Methods

`getImpl`	Get implementation of reporter

Copy Semantics

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

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] = 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).
%
```

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`

Topics

“Report on MATLAB Function” on page 4-16
“Report Generation for Simulink and Stateflow Elements” on page 1-10
“Report Generation for Common Report Elements” (MATLAB Report Generator)

Introduced in R2018a

slreportgen.report.Report class

Package: slreportgen.report

Superclass for report creation

Description

`slreportgen.report.Report` is a container for a report based on Simulinkreporters 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 `mlreporgen.report.Report`, to create Simulink reports, which are reports that use Simulink reporters to generate content. You can also use MATLABreporters 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.

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 `mlreporten.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 (MATLAB).

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 1-10

“Report Generation for Common Report Elements” (MATLAB Report Generator)

Introduced in R2017b

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 a `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
string | character array | reporter or report | DOM document or document part

Source of template for this reporter, specified as one of these values:

- String or character array specifying the path of the file containing the template for this reporter
- Reporter or report whose template is used as the template for this reporter or whose template library contains the template for this reporter
- DOM document or document part whose template is used as the template for this reporter or whose template library contains the template for this reporter

The specified template must be of the same type as the report to which this reporter is appended. For example, for a 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
string | character array

Name of template for this reporter, specified as a string or character array. The template for this reporter is in the template library of the template source (`TemplateSrc`) for this reporter.

`LinkTarget` — Hyperlink target for this reporter
string | character array | `mlreportgen.dom.LinkTarget` object

Hyperlink target for this reporter, specified as a string, 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.

Example: `firstchapter.LinkTarget = 'ch1_link';`

Methods

createTemplate	Create Simulink object properties reporter template
customizeReporter	Create custom Simulink object properties class
getClassFolder	Simulink object properties class definition file location

Inherited Methods

`getImpl`	Get implementation of reporter

Copy Semantics

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

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 1-10
"Report Generation for Common Report Elements" (MATLAB Report Generator)

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 a `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

string | character array | reporter or report | DOM document or document part

Source of template for this reporter, specified as one of these values:

- String or character array specifying the path of the file containing the template for this reporter
- Reporter or report whose template is used as the template for this reporter or whose template library contains the template for this reporter
- DOM document or document part whose template is used as the template for this reporter or whose template library contains the template for this reporter

The specified template must be of the same type as the report to which this reporter is appended. For example, for a 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

string | character array

Name of template for this reporter, specified as a string or character array. The template for this reporter is in the template library of the template source (`TemplateSrc`) for this reporter.

`LinkTarget` — Hyperlink target for this reporter

string | character array | `mlreportgen.dom.LinkTarget` object

Hyperlink target for this reporter, specified as a string, 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.

Example: `firstchapter.LinkTarget = 'ch1_link';`

Methods

createTemplate	Create StateflowStateflow object properties reporter template
customizeReporter	Create custom Stateflow object properties class
getClassFolder	Stateflow object properties class definition file location

Inherited Methods

`getImpl`	Get implementation of reporter

Copy Semantics

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

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 1-10
“Report Generation for Common Report Elements” (MATLAB Report Generator)

Introduced in R2017b

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 has two ways to find objects — find mode and iterator mode. In find mode, use its `find` method to return the results of a search as an array of results. Then, use a for loop to process the results. In iterator mode, use its `hasNext` and `next` methods to return the results of a search one at a time. Using these iterator methods, use a while loop to process the results. Neither of these methods has a performance advantage over the other, so use either mode depending on whether you prefer while or for loops.

`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 (MATLAB).

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 1-10
“Report Generation for Common Report Elements” (MATLAB Report Generator)

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 has two ways to find objects — find mode and iterator mode. In find mode, use its `find` method to return the results of a search as an array of results. Then, use a for loop to process the results. In iterator mode, use its `hasNext` and `next` methods to return the results of a search one at a time. Using these iterator methods, use a while loop to process the results. Neither of these methods has a performance advantage over the other, so use either mode depending on whether you prefer while or for loops.

`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 block 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.

`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 (MATLAB).

Examples

Find Inport and Outport 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", "OutportBlocks");
    ioFinder = BlockFinder(diagram.Object);
    ioFinder.BlockTypes = "Outport";
    outblocks = find(ioFinder);
    for block = blocks
        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 1-10
“Report Generation for Common Report Elements” (MATLAB Report Generator)

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. Elements include blocks, lines, states, and other diagram elements found by `slreportgen.finder.BlockFinder`.

Construction

`result = BlockResult(elem)` 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

getReporter	Get block reporter

Copy Semantics

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

See Also

`slreportgen.finder.BlockFinder` | `slreportgen.report.Report`

Topics

“Report Generation for Simulink and Stateflow Elements” on page 1-10
“Report Generation for Common Report Elements” (MATLAB Report Generator)

Introduced in R2017b

slreportgen.finder.ChartDiagramFinder class

Package: slreportgen.finder

Create Stateflowchart 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`.

`IncludeMaskedSubsytems` — 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 (MATLAB).

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 1-10
“Report Generation for Common Report Elements” (MATLAB Report Generator)

Introduced in R2017b

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 has two ways to find objects — find mode and iterator mode. In find mode, use its `find` method to return the results of a search as an array of results. Then, use a for loop to process the results. In iterator mode, use its `hasNext` and `next` methods to return the results of a search one at a time. Using these iterator methods, use a while loop to process the results. Neither of these methods has a performance advantage over the other, so use either mode depending on whether you prefer while or for loops.

`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 Simulinkor Stateflowdiagram 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`

Fully Qualified Name	Short Name
`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 (MATLAB).

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 1-10
“Report Generation for Common Report Elements” (MATLAB Report Generator)

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`.

`IncludeMaskedSubsytems` — 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 (MATLAB).

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

Open the `f14` model and find all Gain blocks with a value of `Zw`.

```
model = 'f14';
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 1-10
“Report Generation for Common Report Elements” (MATLAB Report Generator)

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 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 diagrram 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

getReporter	Get diagram element reporter

Copy Semantics

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

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 1-10
"Report Generation for Common Report Elements" (MATLAB Report Generator)

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.

Example: `"sf_car/Engine"`

`Tag` — Additional information
string | character vector | object | ...

Additional information to add to this result. You can set it to any type of value.

Methods

getReporter	Get diagram reporter

Copy Semantics

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

See Also

`slreportgen.finder.ChartDiagramFinder` | `slreportgen.finder.DiagramFinder` | `slreportgen.report.Diagram` | `slreportgen.report.Report`

Topics

"Report Generation for Simulink and Stateflow Elements" on page 1-10
"Report Generation for Common Report Elements" (MATLAB Report Generator)

Introduced in R2017b

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 has two ways to find objects — find mode and iterator mode. In find mode, use its `find` method to return the results of a search as an array of results. Then, use a for loop to process the results. In iterator mode, use its `hasNext` and `next` methods to return the results of a search one at a time. Using these iterator methods, use a while loop to process the results. Neither of these methods has a performance advantage over the other, so use either mode depending on whether you prefer while or for loops.

`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 (MATLAB).

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 1-10
“Report Generation for Common Report Elements” (MATLAB Report Generator)

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 has two ways to find objects — find mode and iterator mode. In find mode, use its `find` method to return the results of a search as an array of results. Then, use a for loop to process the results. In iterator mode, use its `hasNext` and `next` methods to return the results of a search one at a time. Using these iterator methods, use a while loop to process the results. Neither of these methods has a performance advantage over the other, so use either mode depending on whether you prefer while or for loops.

`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 Stateflowdiagram 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`

Fully Qualified Name	Short Name
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 (MATLAB).

Examples

Find Stateflow States and Transitions

Create a report that finds Stateflow states and transitions in the `fuelsys` model

```
import slreportgen.report.*
import slreportgen.finder.*
model_name = 'fuelsys';
load_system(model_name);

rpt = slreportgen.report.Report('output','pdf');
add(rpt,mlreportgen.report.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("Container",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.finder.StateflowDiagramElementFinder` | `slreportgen.report.Diagram` | `slreportgen.report.Report` | `slreportgen.report.StateflowObjectProperties`

Topics

“Report Generation for Simulink and Stateflow Elements” on page 1-10
“Report Generation for Common Report Elements” (MATLAB Report Generator)

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`.

`IncludeMaskedSubsytems` — 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 `slreportgen.finder.SystemDiagramFinder` object. `results` is an array of `slreportgen.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 `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 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 (MATLAB).

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 1-10
“Report Generation for Common Report Elements” (MATLAB Report Generator)

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 = wvdoc.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 7-120 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 dependants
function name or handle

Function to determine whether to export a system and its descendents, specified as a function name or function handle. The function must return `True` to include the system or `False` to exclude the system.

Definitions

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 `.htmx`. If you do not specify an extension, the report generator appends `.htmx`.

`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 `.htmtx` 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.*
load_system('f14')
d = WebViewDocument('f14WebView', 'f14');
d(fill);
rptview(d);

close_system('f14');
```

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.*
load_system('f14');
load_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)

close_system('f14');
close_system('sf_car');
```

Export Selected Systems to a Web View

This example exports `f14`'s root system and `Aircraft Dynamics Model` subsystem.

```
import slreportgen.webview.*
load_system('f14');
wvdoc = WebViewDocument('myWebView', ...
    {'f14', 'f14/Aircraft Dynamics Model'});
wvdoc.ExportOptions.SearchScope = 'Current';
fill(wvdoc)
rptview(wvdoc)
close_system('f14');
```

Export Subsystem's Ancestors to a Web View

This example exports `f14`'s root system and `Controller` subsystem.

```
wvdoc = slreportgen.webview.WebViewDocument(...
    'mydoc', 'f14/Controller');
wvdoc.ExportOptions.SearchScope = 'CurrentAndAbove';

fill(wvdoc)
rptview('myWebView')
```

See Also

`slreportgen.webview.EmbeddedWebViewDocument` | `slreportgen.webview.ExportOptions`

Introduced in R2017a

Functions – Alphabetical List

slreportgen.report.Diagram.createTemplate

Class: slreportgen.report.Diagram
Package: slreportgen.report

Create diagram template

Syntax

```
template = createTemplate(templatePath,type)
```

Description

`template = 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` — Location of reporter template
character vector | character array | template source object

Location of the reporter template, specified as a character vector, character array, or template source object.

`type` — Output format
`'html'` | `'html-file'` | `'docx'` | `'pdf'`

Format for the output, specified as `'html'`, `'html-file''docx'`, or `'pdf'`. If you specify a template using the `templatePath` property, the value for `type` must match the template type.

Output Arguments

`template` — Template name
character vector | character array

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 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.htmtx`. 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 = customizeReporter(classpath)
```

Description

`reporter = 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

getClassFolder

Class: slreportgen.report.Diagram
Package: slreportgen.report

Diagram class definition file location

Syntax

```
path = getClassFolder()
```

Description

`path = 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

slreportgen.report.LookupTable.createTemplate

Class: slreportgen.report.LookupTable
Package: slreportgen.report

Create Simulink lookup table block reporter template

Syntax

```
template = createTemplate(templatePath,type)
```

Description

`template = 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` — Location of reporter template

character vector | character array | template source object

Location of the reporter template, specified as a character vector, character array, or template source object.

`type` — Output format

`'html'` | `'html-file'` | `'docx'` | `'pdf'`

Format for the output, specified as `'html'`, `'html-file''docx'`, or `'pdf'`. If you specify a template using the `templatePath` property, the value for `type` must match the template type.

Output Arguments

`template` — Template name
character vector | character array

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 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.htmtx. 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” (MATLAB Report Generator)
“Modify Styles in HTML Templates” (MATLAB Report Generator)
“Modify Styles in PDF Templates” (MATLAB Report Generator)

Introduced in R2018a

slreportgen.report.LookupTable.customizeReporter

Class: slreportgen.report.LookupTable
Package: slreportgen.report

Create custom LookupTable reporter class

Syntax

```
reporter = customizeReporter(classpath)
```

Description

`reporter = 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 = getClassFolder()
```

Description

`path = 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.

Examples

See Also

`slreportgen.report.LookupTable` | `slreportgen.report.Report`

Introduced in R2018a

slreportgen.utils.isLookupTable

Determine if object is a lookup table block

Syntax

```
tf = slreportgen.utils.isLookupTable(obj)
```

Description

`tf = slreportgen.utils.isLookupTable(obj)` returns `1` if `obj` is a lookup table block. Otherwise, it returns `0`.

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

path | handle

Simulink block to query for whether it is a lookup table block, specified as a block path or handle.

Output Arguments

`tf` — True or false

`1` | `0`

True or false result, returned as logical, indicating whether the block is a lookup table block.

See Also

`slreportgen.report.MATLABFunction`

Introduced in R2018a

slreportgen.utils.isMATLABFunction

Determine if object is a MATLAB Function block or Stateflow object

Syntax

```
tf = slreportgen.utils.isMATLABFunction(obj)
```

Description

`tf = slreportgen.utils.isMATLABFunction(obj)` returns `1` if `obj` is a MATLAB Function block or a Stateflow MATLAB function object. Otherwise, it returns `0`.

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` — Simulink element or Stateflow object to query

path | handle

Simulink element or Stateflow object to query for whether it is a MATLAB Function block or object. Specify the input `obj` as its path or handle.

Output Arguments

`tf` — True or false

`1` | `0`

True or false result, returned as logical, indicating whether the element or object is a MATLAB Function.

See Also

`slreportgen.report.MATLABFunction`

Introduced in R2018a

slreportgen.report.MATLABFunction.createTemplate

Class: slreportgen.report.MATLABFunction
Package: slreportgen.report

Create MATLAB Function reporter template

Syntax

```
template = createTemplate(templatePath,type)
```

Description

`template = 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` — Location of reporter template

character vector | character array | template source object

Location of the reporter template, specified as a character vector, character array, or template source object.

`type` — Output format

`'html'` | `'html-file'` | `'docx'` | `'pdf'`

Format for the output, specified as `'html'`, `'html-file''docx'`, or `'pdf'`. If you specify a template using the `templatePath` property, the value for `type` must match the template type.

Output Arguments

`template` — Template name
character vector | character array

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 MATLAB Function Reporter Template

Before you run this example, create a copy of the default HTML MATLAB Function template in the `mytemplates` folder. Name the copied template `myMFunction.htmtx`. Edit the template as desired. To use the new template for the image, assign its path to the `TemplateSrc` property of `slreportgen.report.MATLABFunction`.

```
import mlreportgen.report.*
import slreportgen.report.*
rpt = Report('My Report','html');
mfunction = MATLABFunction();
template = MATLABFunction.createTemplate('mytemplates\myMFunction','html');
mfunction.TemplateSrc = template;
```

See Also

`slreportgen.report.MATLABFunction` | `slreportgen.report.Report`

Topics

“Modify Styles in a Microsoft Word Template” (MATLAB Report Generator)
“Modify Styles in HTML Templates” (MATLAB Report Generator)
“Modify Styles in PDF Templates” (MATLAB Report Generator)

Introduced in R2018a

slreportgen.report.MATLABFunction.customizeReporter

Class: slreportgen.report.MATLABFunction
Package: slreportgen.report

Create custom MATLAB Function reporter class

Syntax

```
reporter = customizeReporter(classpath)
```

Description

`reporter = 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 yourMATLAB 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 = getClassFolder()
```

Description

`path = 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.SimulinkObjectProperties.createTemplate

Class: slreportgen.report.SimulinkObjectProperties
Package: slreportgen.report

Create Simulink object properties reporter template

Syntax

```
template = createTemplate(templatePath,type)
output_args = createTemplate(input_args,Name,Value)
```

Description

`template = 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 t.

`output_args = createTemplate(input_args,Name,Value)` <verb phase> with additional options specified by one or more `Name,Value` pair arguments.

Input Arguments

`templatePath` — Location of reporter template

character vector | character array | template source object

Location of the reporter template, specified as a character vector, character array, or template source object.

`type` — Output format

`'html'` | `'html-file'` | `'docx'` | `'pdf'`

Format for the output, specified as `'html'`, `'html-file''docx'`, or `'pdf'`. If you specify a template using the `templatePath` property, the value for `type` must match the template type.

Output Arguments

`template` — Template name
character vector | character array

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 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.htmtx`. 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” (MATLAB Report Generator)
“Modify Styles in HTML Templates” (MATLAB Report Generator)
“Modify Styles in PDF Templates” (MATLAB Report Generator)

Introduced in R2017b

slreportgen.report.StateflowObjectProperties.customizeReporter

Class: slreportgen.report.StateflowObjectProperties
Package: slreportgen.report

Create custom Simulink object properties class

Syntax

```
reporter = customizeReporter(classpath)
```

Description

`reporter = 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/@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 Simulink object properties reporter.

```
objprop = MySFObjProps();
```

See Also

`slreportgen.report.Report` |
`slreportgen.report.SimulinkObjectProperties`

Introduced in R2017b

getClassFolder

Class: slreportgen.report.SimulinkObjectProperties
Package: slreportgen.report

Simulink object properties class definition file location

Syntax

```
path = getClassFolder()
```

Description

`path = 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 StateflowStateflow object properties reporter template

Syntax

```
template = createTemplate(templatePath,type)
```

Description

`template = 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` — Location of reporter template

character vector | character array | template source object

Location of the reporter template, specified as a character vector, character array, or template source object.

`type` — Output format

`'html'` | `'html-file'` | `'docx'` | `'pdf'`

Format for the output, specified as `'html'`, `'html-file''docx'`, or `'pdf'`. If you specify a template using the `templatePath` property, the value for `type` must match the template type.

Output Arguments

`template` — Template name
character vector | character array

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 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.htmtx`. 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\myObjsReporter','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 = customizeReporter(classpath)
```

Description

`reporter = 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

getClassFolder

Class: slreportgen.report.StateflowObjectProperties
Package: slreportgen.report

Stateflow object properties class definition file location

Syntax

```
path = getClassFolder()
```

Description

`path = 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

getReporter

Class: slreportgen.finder.BlockResult
Package: slreportgen.finder

Get block reporter

Syntax

```
result = getReporter(path_handle)
```

Description

`result = getReporter(path_handle)` returns a reporter that generates a properties table for the block returned in the result.

Input Arguments

`path_handle` — Block path or handle

path | handle

Path or handle of the block.

Output Arguments

`result` — Block reporter object

path

Block reporter object, returned as a `slreportgen.finder.SimulinkObjectProperties` object. This reporter generates a properties table for the block.

See Also

`slreportgen.finder.BlockFinder` |
`slreportgen.report.SimulinkObjectProperties`

Introduced in R2017b

getReporter

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

slreportgen.webview.EmbeddedWebViewDocument.createDiagramLink

Package: slreportgen.webview

Link to embedded Web view report

Syntax

```
diaglink = createDiagramLink(wvdoc,dhandle,domlabel)
```

Description

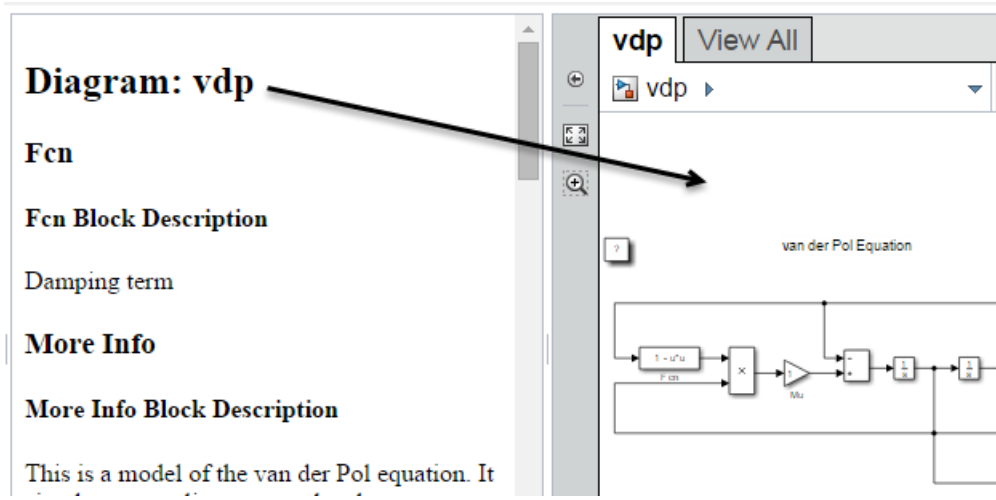
`diaglink = createDiagramLink(wvdoc,dhandle,domlabel)` updates a DOM object in an embedded Web view Document panel so that it links to a diagram anchor handle in the Simulink Web view. The `diaglink` DOM object is of the same type as `domlabel` or if `domlabel` is a string, an `mlreportgen.DOM.Text` object is created.

Examples

Create a Link to a Web View

Create a link from a level 2 heading in the center text pane to a Simulink model.

```
diagHeading = createDiagramLink(h,diagHandle,...
     mlreportgen.dom.Heading(2,get_param(diagHandle,'Name')));
```



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.

Definitions

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

`slreportgen.webview.EmbeddedWebViewDocument` |
`slreportgen.webview.EmbeddedWebViewDocument.createDiagramTwoWayLink` |
`slreportgen.webview.EmbeddedWebViewDocument.createElementLink` |
`slreportgen.webview.EmbeddedWebViewDocument.createElementTwoWayLink` |
`slreportgen.webview.WebViewDocument` |
`slreportgen.webview.WebViewDocument.getExportDiagrams`

Topics

“Create an Embedded Web View Report Generator” on page 5-30
“Embedded Web View Reports” on page 5-24

Introduced in R2017a

slreportgen.webview.EmbeddedWebViewDocument.createDiagramTwoWayLink

Package: slreportgen.webview

Link and anchor in embedded Web view report

Syntax

```
diag2link = createDiagramTwoWayLink(wvdoc,dhandle,domlabel)
```

Description

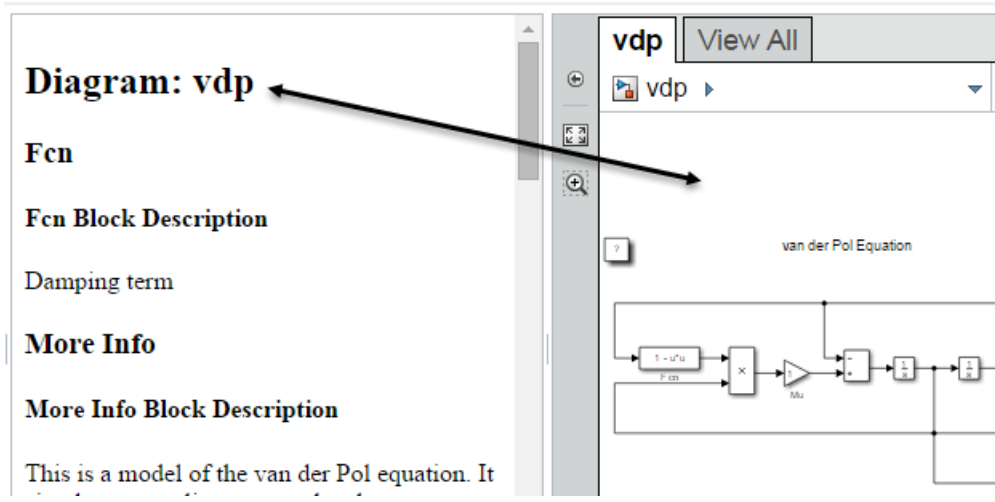
`diag2link = createDiagramTwoWayLink(wvdoc,dhandle,domlabel)` creates a two-way connection between a Simulink Web view diagram and a DOM object in an embedded Web view Document panel. The `diag2link` DOM object is updated to include attributes that identifies it as a link. The `diag2link` DOM object is of the same type as `domlabel` or if `domlabel` is a string, an `mlreportgen.DOM.Text` object is created.

Examples

Create Two-Way Diagram Link

Create a two-way connection between a level 2 heading in the center text pane and a diagram in the Web view. This creates both a link and an anchor.

```
diagramHeading = createDiagramTwoWayLink(h,diagramPath, ...
   mlreportgen.dom.Heading(2,['Diagram: ' diagramPath]));
```



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.

Definitions

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

`slreportgen.webview.EmbeddedWebViewDocument` |
`slreportgen.webview.EmbeddedWebViewDocument.createDiagramLink` |
`slreportgen.webview.EmbeddedWebViewDocument.createElementLink` |
`slreportgen.webview.EmbeddedWebViewDocument.createElementTwoWayLink` |
`slreportgen.webview.WebViewDocument` |
`slreportgen.webview.WebViewDocument.getExportDiagrams`

Topics

“Create an Embedded Web View Report Generator” on page 5-30
“Embedded Web View Reports” on page 5-24

Introduced in R2017a

slreportgen.webview.EmbeddedWebViewDocument.createElementLink

Package: slreportgen.webview

Element link in embedded Web view report

Syntax

```
elemLink = createElementLink(wvdoc,ehandle,domlabel)
```

Description

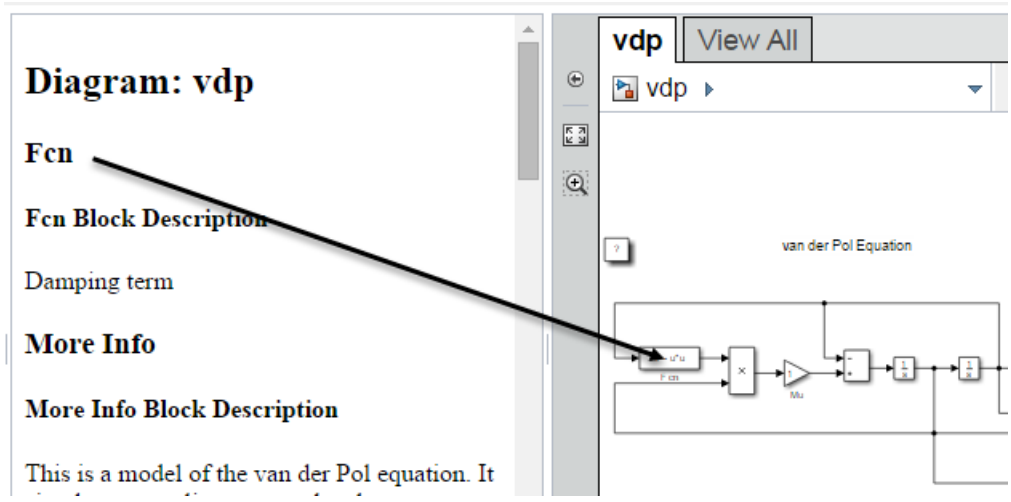
`elemLink = createElementLink(wvdoc,ehandle,domlabel)` updates a DOM object in an embedded Web view Document panel so that it links to an element anchor in the Simulink Web view. The DOM object is of the same type as `domlabel` or, if `domlabel` is a string, a `mlreportgen.DOM.Text` object is created.

Examples

Create Link to Element to Web View

Create a link between a level 2 heading in the center text pane and an element in the Web view.

```
blockHeading = createElementLink(h,blockHandle, ...
   mlreportgen.dom.Heading(2, get_param(blockHandle,'Name')));
```



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.

Definitions

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

`slreportgen.webview.EmbeddedWebViewDocument` |
`slreportgen.webview.EmbeddedWebViewDocument.createDiagramLink` |
`slreportgen.webview.EmbeddedWebViewDocument.createDiagramTwoWayLink` |
`slreportgen.webview.EmbeddedWebViewDocument.createElementTwoWayLink` |
`slreportgen.webview.WebViewDocument` |
`slreportgen.webview.WebViewDocument.getExportDiagrams`

Topics

“Create an Embedded Web View Report Generator” on page 5-30
“Embedded Web View Reports” on page 5-24

Introduced in R2017a

slreportgen.webview.EmbeddedWebViewDocument.createElementTwoWayLink

Package: slreportgen.webview

Element link and anchor in embedded Web view report

Syntax

```
elem2link = createDiagramTwoWayLink(wvdoc,ehandle,domlabel)
```

Description

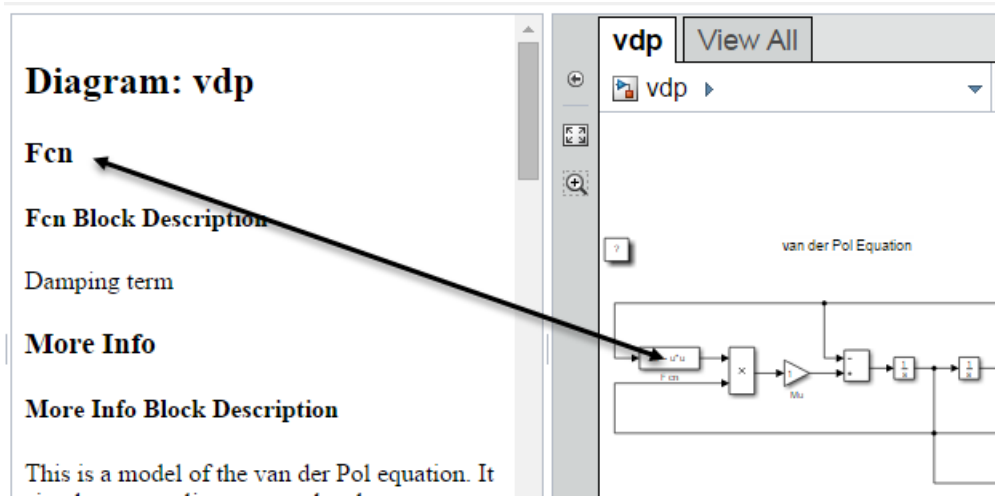
`elem2link = createDiagramTwoWayLink(wvdoc,ehandle,domlabel)` creates a two-way connection between a Simulink Web view element and a DOM object in an embedded Web view Document panel. The `elem2link` DOM object is updated to include attributes that identifies it as a link. The `elem2link` DOM object is of the same type as `domlabel` or if `domlabel` is a string, an `mlreportgen.DOM.Text` object is created.

Examples

Create Two-Way Element Link in Embedded Web View Report

Create a two-way connection between a level 2 heading in the center text pane and an element in the Web view. This creates both a link and an anchor.

```
blockHeading = createElementTwoWayLink(h,blockHandle, ...
   mlreportgen.dom.Heading(3, get_param(blockHandle,'Name')));
```



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.

Definitions

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

`slreportgen.webview.EmbeddedWebViewDocument` |
`slreportgen.webview.EmbeddedWebViewDocument.createDiagramLink` |
`slreportgen.webview.EmbeddedWebViewDocument.createDiagramTwoWayLink` |
`slreportgen.webview.EmbeddedWebViewDocument.createElementLink` |
`slreportgen.webview.WebViewDocument` |
`slreportgen.webview.WebViewDocument.getExportDiagrams`

Topics

“Create an Embedded Web View Report Generator” on page 5-30
“Embedded Web View Reports” on page 5-24

Introduced in R2017a

slreportgen.webview.WebViewDocument.fill

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 `fill`**`Id`**`(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 `fill`**`Id`** methods to fill the holes.

slreportgen.webview.WebViewDocument.append

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 `slreportgen.webview.WebViewDocument.fill`for more information.

slreportgen.webview.WebViewDocument.fillslwebview

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

`mlreportgen.dom.document.fill`

Introduced in R2017a

slreportgen.webview.WebViewDocument.getExportDiagrams

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.

Definitions

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

`slreportgen.webview.WebViewDocument.getExportModels` | `slreportgen.webview.WebViewDocument.getExportSimulinkSubSystems` | `slreportgen.webview.WebViewDocument.getExportStateflowCharts` | `slreportgen.webview.WebViewDocument.getExportStateflowDiagrams`

Introduced in R2017a

slreportgen.webview.WebViewDocument.-getExportModels

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.

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
```

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.

See Also

`slreportgen.webview.WebViewDocument.getExportDiagrams` |
`slreportgen.webview.WebViewDocument.getExportSimulinkSubSystems` |
`slreportgen.webview.WebViewDocument.getExportStateflowCharts` |
`slreportgen.webview.WebViewDocument.getExportStateflowDiagrams`

Introduced in R2017a

slreportgen.webview.WebViewDocument.-getExportSimulinkSubSystems

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.

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
```

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.

Definitions

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

`slreportgen.webview.WebViewDocument.getExportDiagrams` |
`slreportgen.webview.WebViewDocument.getExportModels` |
`slreportgen.webview.WebViewDocument.getExportStateflowCharts` |
`slreportgen.webview.WebViewDocument.getExportStateflowDiagrams`

Introduced in R2017a

slreportgen.webview.WebViewDocument.-getExportStateflowCharts

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.

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
```

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.

See Also

`slreportgen.webview.WebViewDocument.getExportDiagrams` |
`slreportgen.webview.WebViewDocument.getExportModels` |
`slreportgen.webview.WebViewDocument.getExportSimulinkSubSystems` |
`slreportgen.webview.WebViewDocument.getExportStateflowDiagrams`

Introduced in R2017a

slreportgen.webview.WebViewDocument.-getExportStateflowDiagrams

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.

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
```

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.

See Also

`slreportgen.webview.WebViewDocument.getExportDiagrams` |
`slreportgen.webview.WebViewDocument.getExportModels` |
`slreportgen.webview.WebViewDocument.getExportSimulinkSubSystems` |
`slreportgen.webview.WebViewDocument.getExportStateflowDiagrams`

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

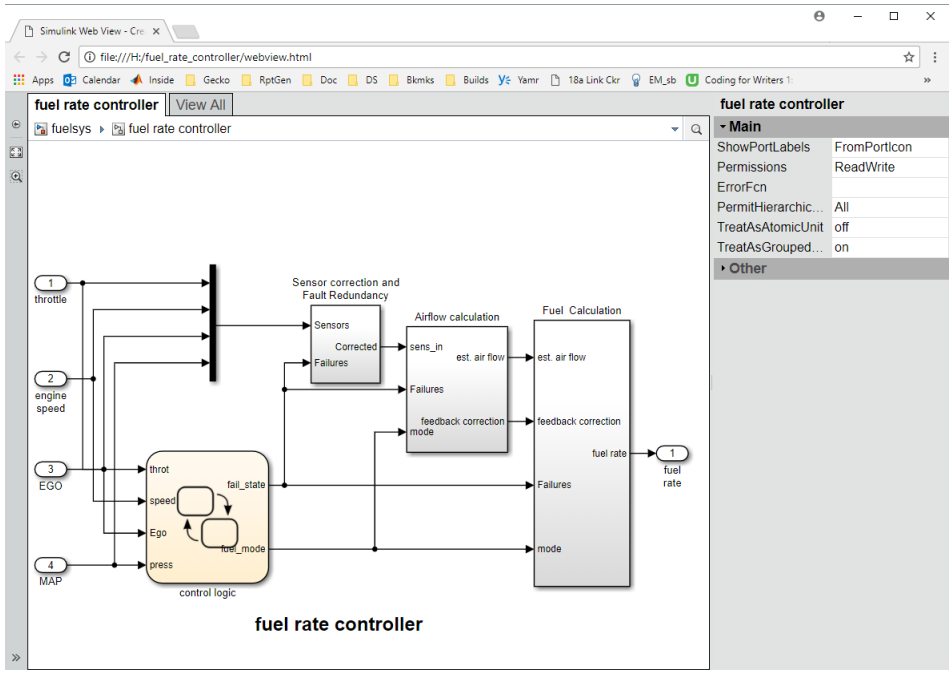
Open the fuel rate controller subsystem.

```
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\webview.html
```

The Web view displays in the system browser.



Export Web View with Access to Referenced Models

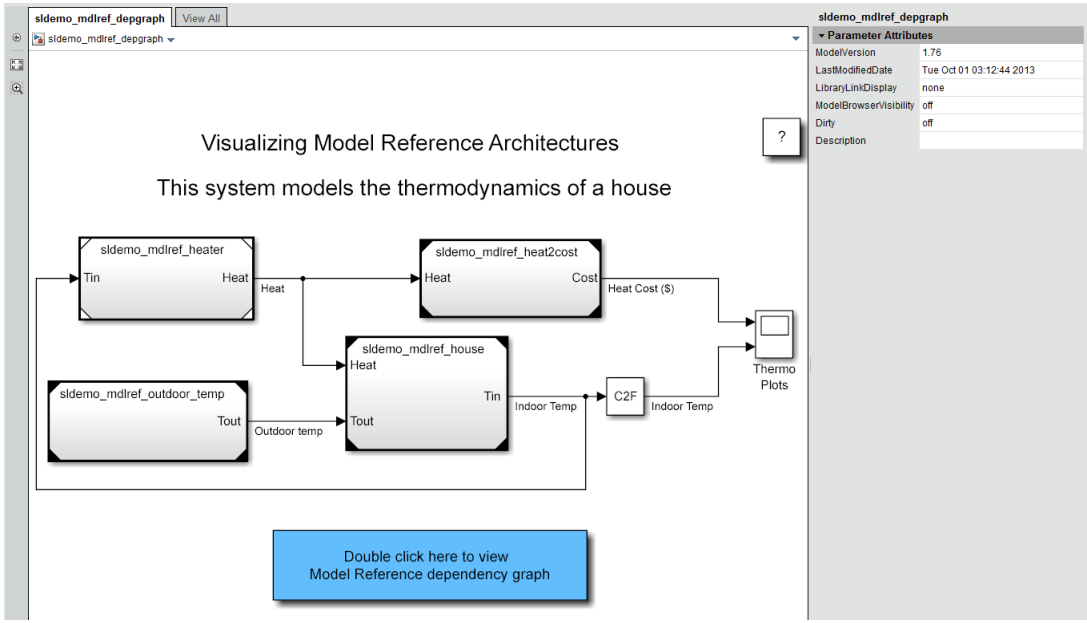
Open the `sldemo_mdlref_depgraph` model.

```
open_system('sldemo_mdlref_depgraph')
```

Export to a Web view the `sldemo_mdlref_depgraph` model and allow access to the models it references.

```
depgraph_web_view = slwebview...
('sldemo_mdlref_depgraph','FollowModelReference','on')

depgraph_web_view =

H:\sldemo_mdlref_depgraph\webview.html
```

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.

- “Create and Use a Web View” on page 5-14

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 single 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 5-14

Introduced in R2006a