

**Simulink® Test™**

Getting Started Guide



**MATLAB® & SIMULINK®**

R2018a



# How to Contact MathWorks



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



Phone: 508-647-7000



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

## *Simulink® Test™ Getting Started Guide*

© COPYRIGHT 2015–2018 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](http://www.mathworks.com/trademarks) for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

### **Patents**

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

### **Revision History**

March 2015	Online Only	New for Version 1.0 (Release 2015a)
September 2015	Online Only	Revised for Version 1.1 (Release 2015b)
October 2015	Online only	Rereleased for Version 1.0.1 (Release 2015aSP1)
March 2016	Online Only	Revised for Version 2.0 (Release 2016a)
September 2016	Online Only	Revised for Version 2.1 (Release 2016b)
March 2017	Online Only	Revised for Version 2.2 (Release 2017a)
September 2017	Online Only	Revised for Version 2.3 (Release 2017b)
March 2018	Online Only	Revised for Version 2.4 (Release 2018a)

## Product Overview

### 1

<b>Simulink Test Product Description</b> .....	<b>1-2</b>
Key Features .....	<b>1-2</b>

## Introduction

### 2

<b>Refine, Test, and Debug a Subsystem</b> .....	<b>2-2</b>
Model and Requirements .....	<b>2-2</b>
Create a Harness for the Controller .....	<b>2-4</b>
Inspect and Refine the Controller .....	<b>2-6</b>
Add Test Inputs and Test the Controller .....	<b>2-6</b>
Debug the Controller .....	<b>2-7</b>
<b>Test Model Output Against a Baseline</b> .....	<b>2-10</b>
Create the Test Case .....	<b>2-10</b>
Run the Test Case and View Results .....	<b>2-11</b>
<b>Introduction to Test Manager</b> .....	<b>2-14</b>
Start Test Manager .....	<b>2-14</b>
Create Tests and Understand the Test Hierarchy .....	<b>2-14</b>
View Test Results .....	<b>2-16</b>
Share Results .....	<b>2-16</b>
Compare Test Files .....	<b>2-16</b>



# Product Overview

---

## Simulink Test Product Description

### **Develop, manage, and execute simulation-based tests**

Simulink Test provides tools for authoring, managing, and executing systematic, simulation-based tests of models, generated code, and simulated or physical hardware. It includes a Test Sequence block that lets you construct complex test sequences and assessments, and a Test Manager for managing and executing tests. Simulink Test enables functional, baseline, equivalence, and back-to-back testing, including software-in-the-loop (SIL), processor-in-the-loop (PIL), and real-time hardware-in-the-loop (HIL). You can apply pass and fail criteria that include absolute and relative tolerances, limits, logical checks, and temporal conditions. Setup and cleanup scripts help you automate or customize test execution.

You can create nonintrusive test harnesses to test components in the system model or in a separate test model. You can store test cases and their results, creating a repository for reviewing and investigating failures. You can generate reports, archive and review test results, rerun failed tests, and debug the component or system under test.

With Simulink Test and Simulink Requirements™, you can link test cases to requirements captured in Microsoft® Word, IBM® Rational® DOORS®, and other documents.

Support for industry standards is available through IEC Certification Kit (for IEC 61508 and ISO 26262) and DO Qualification Kit (for DO-178).

### **Key Features**

- Test harness for subsystem or model testing
- Test sequence block for running tests and assessments
- Pass-fail criteria, including tolerances, limits, and temporal conditions
- Baseline, equivalence, back-to-back, and real-time testing
- Setup and cleanup scripts for customizing test execution
- Test Manager for authoring, executing, and organizing test cases and their results
- Customizable report generation for documenting test outcomes

# Introduction

---

- “Refine, Test, and Debug a Subsystem” on page 2-2
- “Test Model Output Against a Baseline” on page 2-10
- “Introduction to Test Manager” on page 2-14

## Refine, Test, and Debug a Subsystem

In this section...
“Model and Requirements” on page 2-2
“Create a Harness for the Controller” on page 2-4
“Inspect and Refine the Controller” on page 2-6
“Add Test Inputs and Test the Controller” on page 2-6
“Debug the Controller” on page 2-7

Test harnesses provide a development and testing environment that leaves the main model design intact. You can test a functional unit of your model in isolation without altering the main model. This example demonstrates refining and testing a controller subsystem using a test harness. The main model is a controller-plant model of an air conditioning/heat pump unit. The controller must operate according to several simple requirements.

### Model and Requirements

- 1 Access the model. Enter

```
cd(fullfile(docroot, 'toolbox', 'sltest', 'examples'))
```

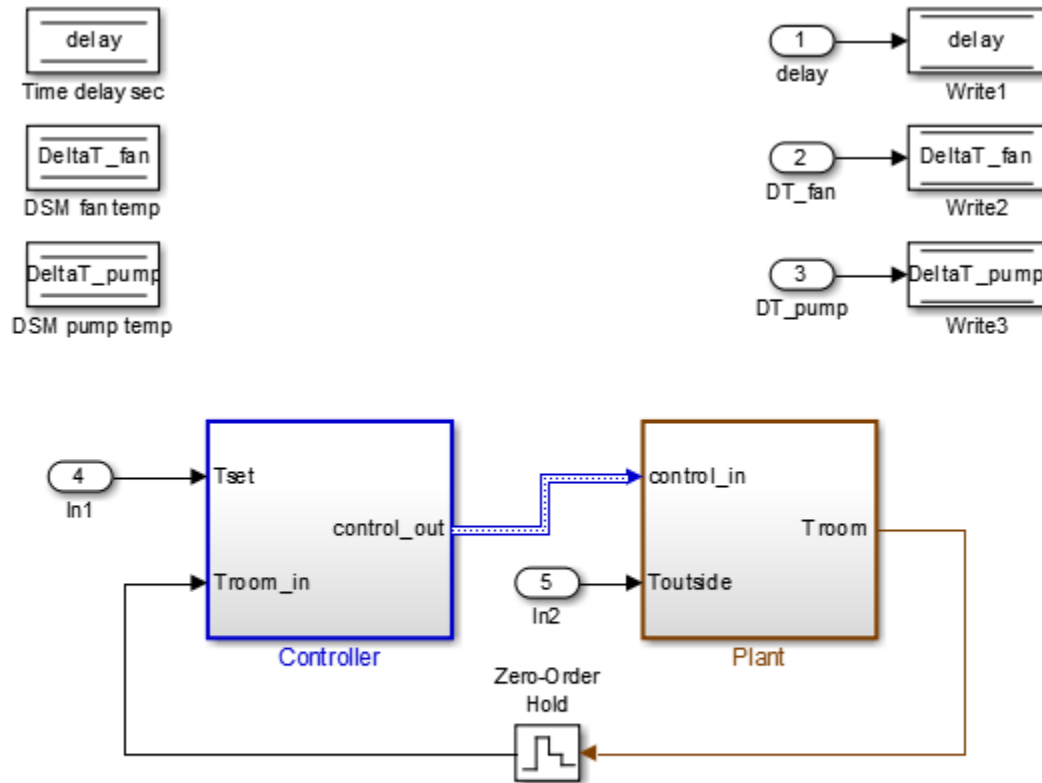
- 2 Copy this model file and supporting files to a writable location on the MATLAB® path:

```
sltestHeatpumpExample.slx  
sltestHeatpumpBusPostLoadFcn.mat  
PumpDirection.m
```

- 3 Open the model.

```
open_system('sltestHeatpumpExample')
```





Copyright 1990-2014 The MathWorks, Inc.

In the example model:

- The controller accepts the room temperature and the set temperature inputs.
- The controller output is a bus with signals controlling the fan, heat pump, and the direction of the heat pump (heat or cool).
- The plant accepts the control bus. The heat pump and the fan signals are Boolean, and the heat pump direction is specified by +1 for cooling and -1 for heating.

The test covers four temperature conditions. Each condition corresponds to one operating state with fan, pump, and pump direction signal outputs.

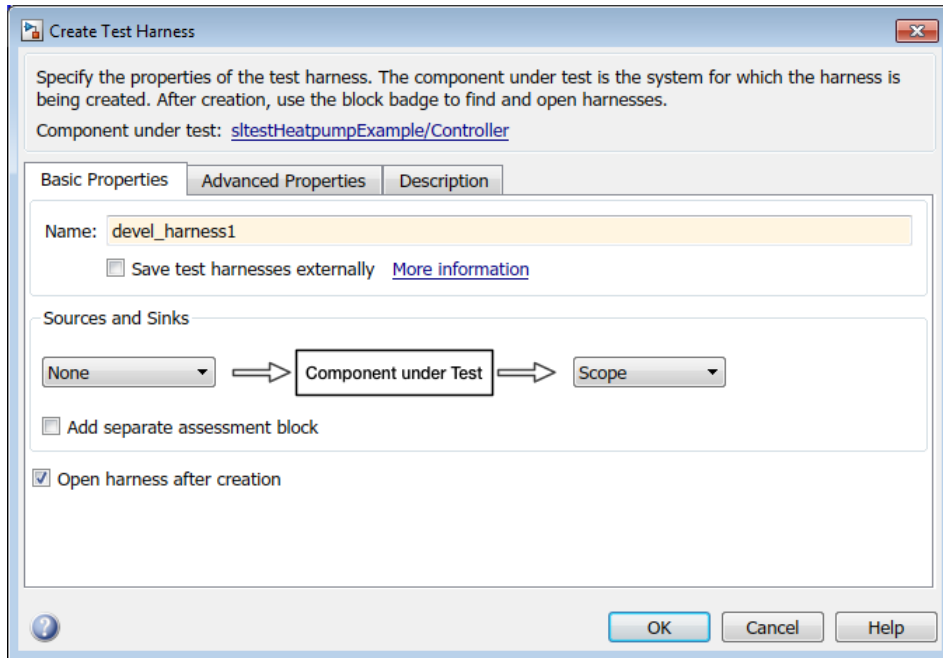
Temperature condition	System state	Fan command	Pump command	Pump direction
$ T_{room} - T_{set}  < \Delta T_{fan}$	idle	0	0	0
$\Delta T_{fan} \leq  T_{room} - T_{set}  < \Delta T_{pump}$	fan only	1	0	0
$ T_{room} - T_{set}  \geq \Delta T_{pump}$ and $T_{set} < T_{room}$	cooling	1	1	-1
$ T_{room} - T_{set}  \geq \Delta T_{pump}$ and $T_{set} > T_{room}$	heating	1	1	1

## Create a Harness for the Controller

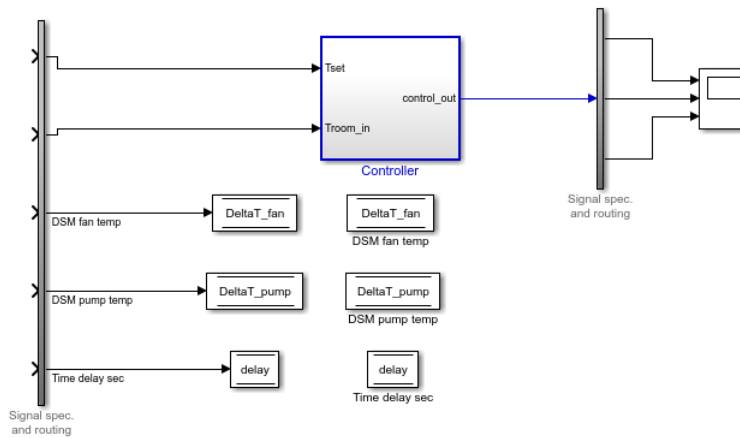
- 1 Right-click the Controller subsystem and select **Test Harness > Create for 'Controller'**.
- 2 Set the harness properties:

In the **Basic Properties** tab:

- **Name:** devel\_harness\_1
- Clear **Save test harness externally**
- **Sources and Sinks:** None and Scope
- Clear **Add separate assessment block**
- Select **Open harness after creation**

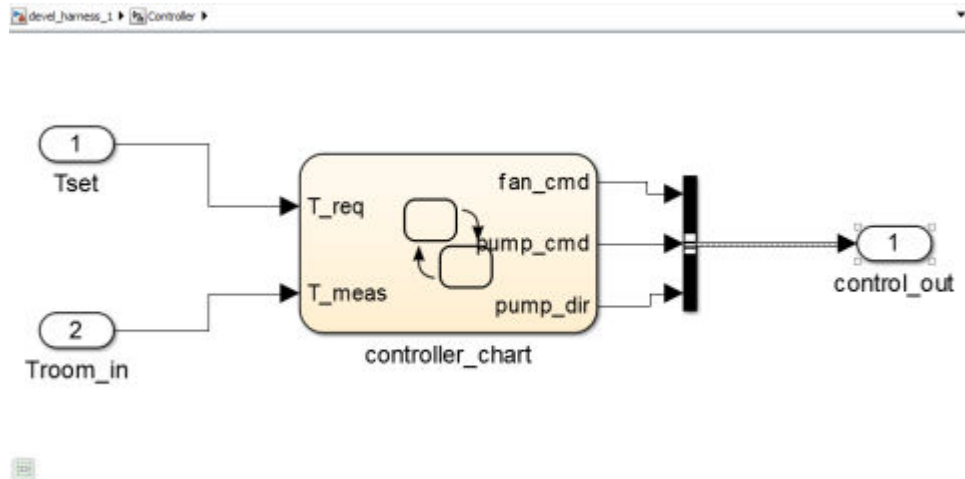


- 3 Click **OK** to create the test harness.



### Inspect and Refine the Controller

- 1 In the test harness, double-click Controller to open the subsystem.
- 2 Connect the chart to the Inport blocks.

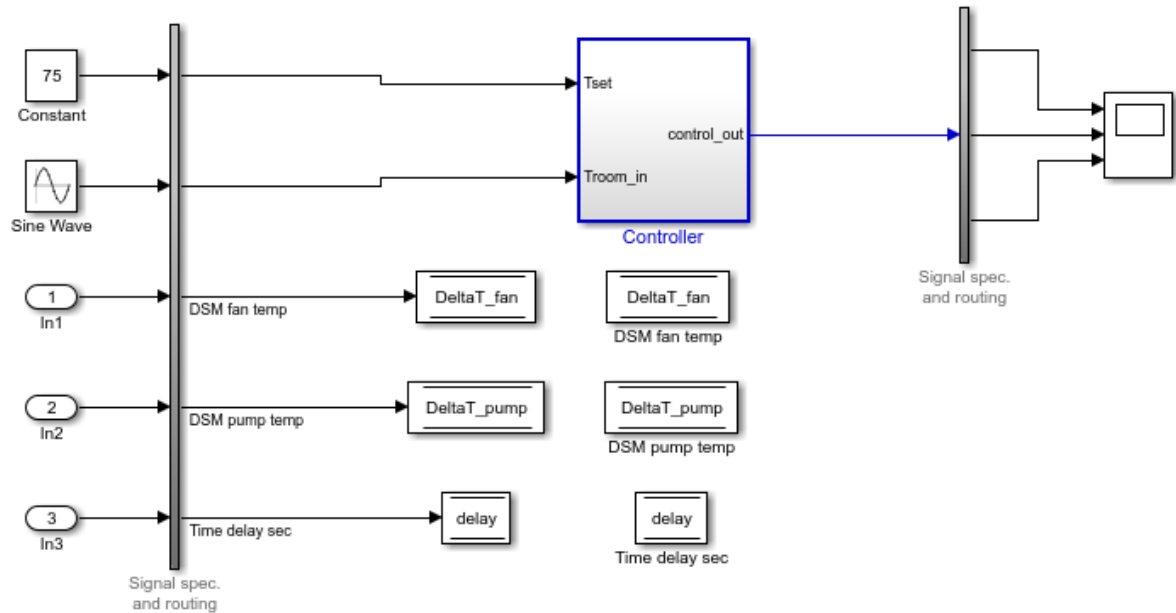


- 3 In the test harness, click the Save button to save the test harness and model.

### Add Test Inputs and Test the Controller

- 1 Navigate to the top level of `devel_harness_1`.
- 2 Create a test input for the harness with a constant `Tset` and a time-varying `Troom`. Connect a Constant block to the `Tset` input and set the value to 75.
- 3 Add a Sine Wave block to the harness model to simulate a temperature signal. Connect the Sine Wave block to the conversion subsystem input `Troom_in`.
- 4 Double-click the Sine Wave block and set the parameters:
  - **Amplitude:** 15
  - **Bias:** 75
  - **Frequency:**  $2 \cdot \pi / 3600$
  - **Phase (rad):** 0
  - **Sample time:** 1

- Select **Interpret vector parameters as 1-D**.
- 5 Connect Inport blocks to the Data Store Write inputs.

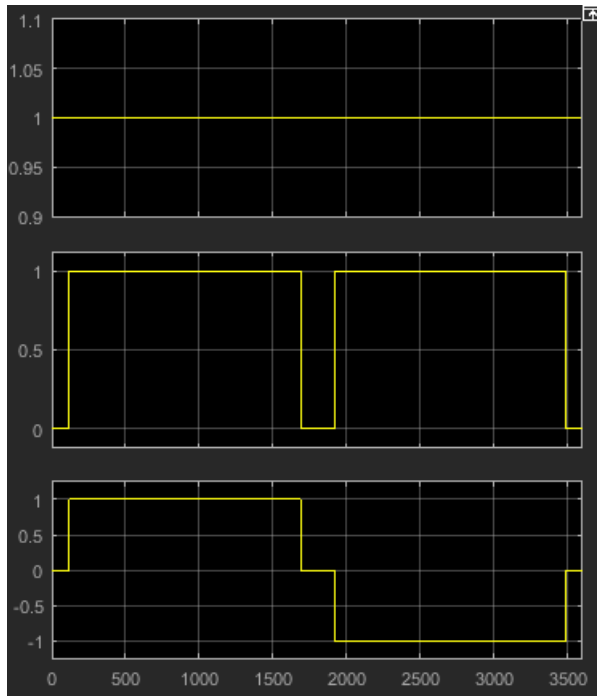


- 6 In the Configuration Parameters dialog box, in the **Data Import/Export** pane, select **Input** and enter `u`. `u` is an existing structure in the MATLAB base workspace.
- 7 In the **Solver** pane, set **Stop time** to 3600.
- 8 Open the scope in the test harness and change the layout to show three plots.
- 9 Click **Run** to simulate.

## Debug the Controller

- 1 Observe the controller output. `fan_cmd` is 1 during the IDLE condition where  $|T_{room} - T_{set}| < \Delta T_{fan}$ .

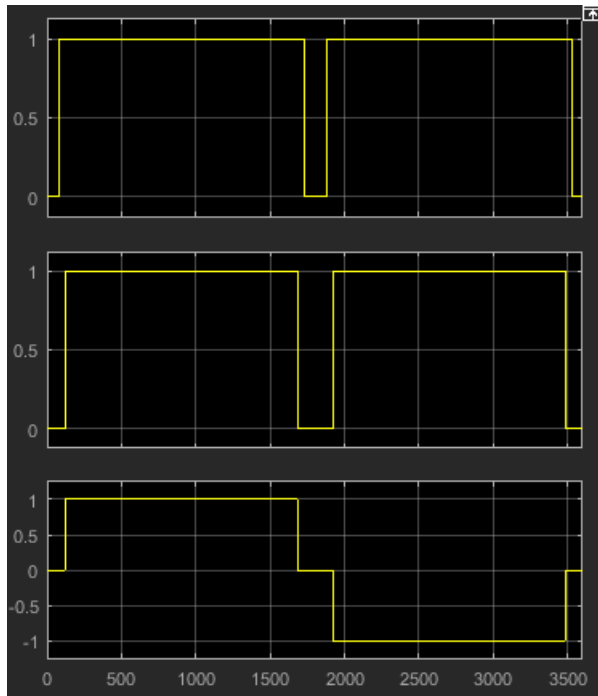
This is a bug. `fan_cmd` should equal 0 at IDLE. The `fan_cmd` control output must be changed for IDLE.



- 2 In the harness model, open the Controller subsystem.
- 3 Open controller\_chart.
- 4 In the IDLE state, fan\_cmd is set to return 1. Change fan\_cmd to return 0. IDLE is now:

```
IDLE
entry:
fan_cmd = 0;
pump_cmd = 0;
pump_dir = 0;
```

- 5 Simulate the harness model again and observe the outputs.



6 `fan_cmd` now meets the requirement to equal 0 at IDLE.

## See Also

### Related Examples

- “Test a Model Component Using Signal Functions”
- “Test Downshift Points of a Transmission Controller”


## Test Model Output Against a Baseline

To test the simulation output of a model against a defined baseline, use a baseline test case. In this example, use the `sldemo_absbrake` model to compare the simulation output to a baseline captured from an earlier state of the model.

### Create the Test Case


- 1 Open the `sldemo_absbrake` model.
- 2 To open the Test Manager from the model, select **Analysis > Test Manager**.
- 3 From the Test Manager toolstrip, click **New** to create a test file. Name and save the test file.

The test file consists of a test suite that contains one baseline test case. They appear in the **Test Browser** pane.

- 4 Right-click the baseline test case in the **Test Browser** pane, and select **Rename**. Rename the test case to `Slip Baseline Test`.
- 5 Under **System Under Test** in the test case, click the **Use current model** button  to load the `sldemo_absbrake` model into the test case.
- 6 To record a baseline from the system under test, under **Baseline Criteria**, click **Capture**.
- 7 In the Capture Baseline dialog box, for the file format, select `Excel`. Specify a location to save the baseline to and click **Create**.
- 8 The baseline criteria file and the logged signals appear in the table. Set the **Absolute Tolerance** of the `Ww` signal to 15.

SIGNAL NAME	ABS TOL	REL TOL	LEADING TOL	LAGGING TOL
▼ <input checked="" type="checkbox"/> My_mat_base.mat	0	0.00%	0	0
<input checked="" type="checkbox"/> Ww	15	0.00%	0	0
<input checked="" type="checkbox"/> Vs	0	0.00%	0	0
<input checked="" type="checkbox"/> Sd	0	0.00%	0	0
<input checked="" type="checkbox"/> slp	0	0.00%	0	0

---

**Tip** To add or remove columns in the baseline criteria table, click the column selector button .

---



For more information about tolerances and criteria, see “Apply Tolerances to Test Criteria”.

## Run the Test Case and View Results

- 1 In the `sldemo_absbrake` model, set the **Desired relative slip** constant block to `0.22`.
- 2 In the Test Manager, select the Slip Baseline Test case in the **Test Browser** pane.
- 3 On the Test Manager toolbar, click **Run**.

In the **Results and Artifacts** pane, the new test result appears at the top of the table.

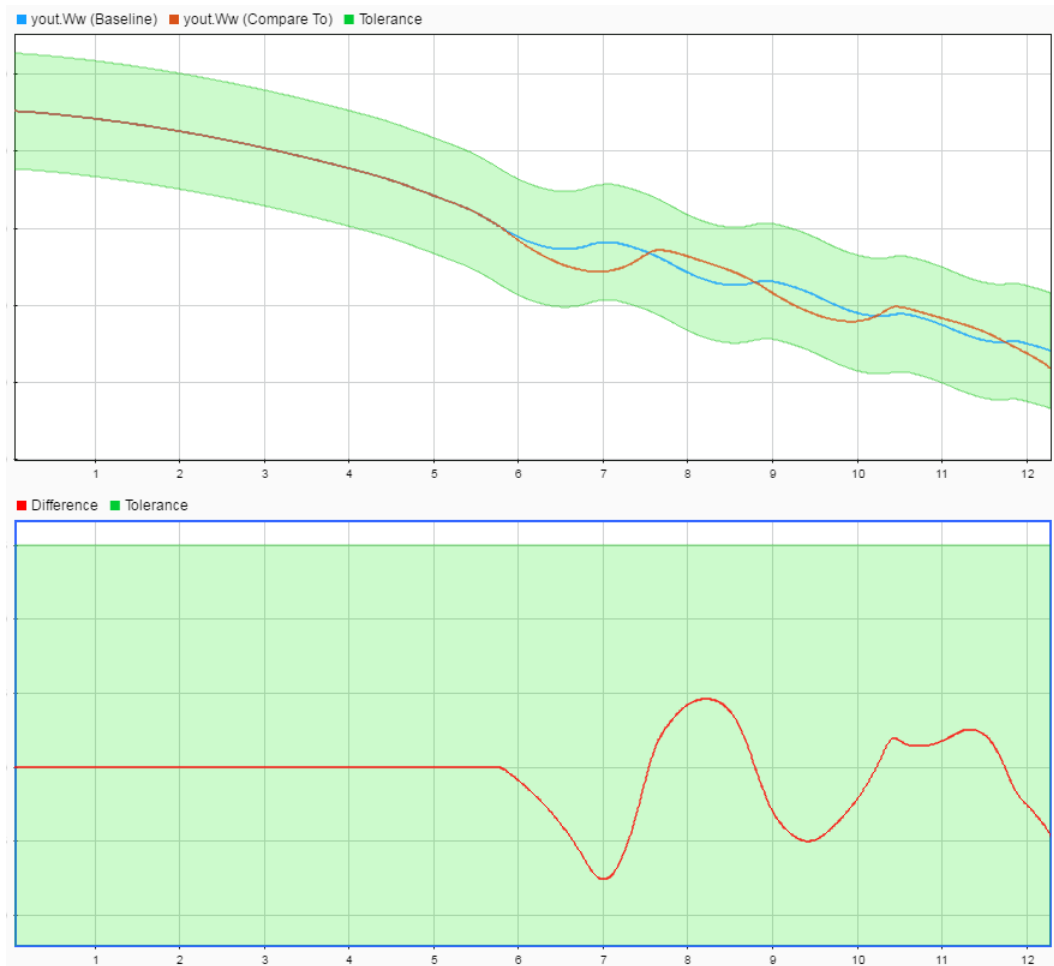
- 4 Expand the results until you see the baseline criteria result. Right-click the result and select **Expand All Under**.

The signal `yout.Ww` passes, but the overall baseline test fails because other signal comparisons specified in the **Baseline Criteria** section of the test case were not satisfied.

- 5 To view the `yout.Ww` signal comparison between the model and the baseline criteria, expand **Baseline Criteria Result** and click the option button next to the `yout.Ww` signal.

▼ Baseline Criteria Result	✘
<input type="radio"/> slp	✘
<input type="radio"/> yout.Sd	✘
<input type="radio"/> yout.Vs	✘
<input checked="" type="radio"/> yout.Ww	✔

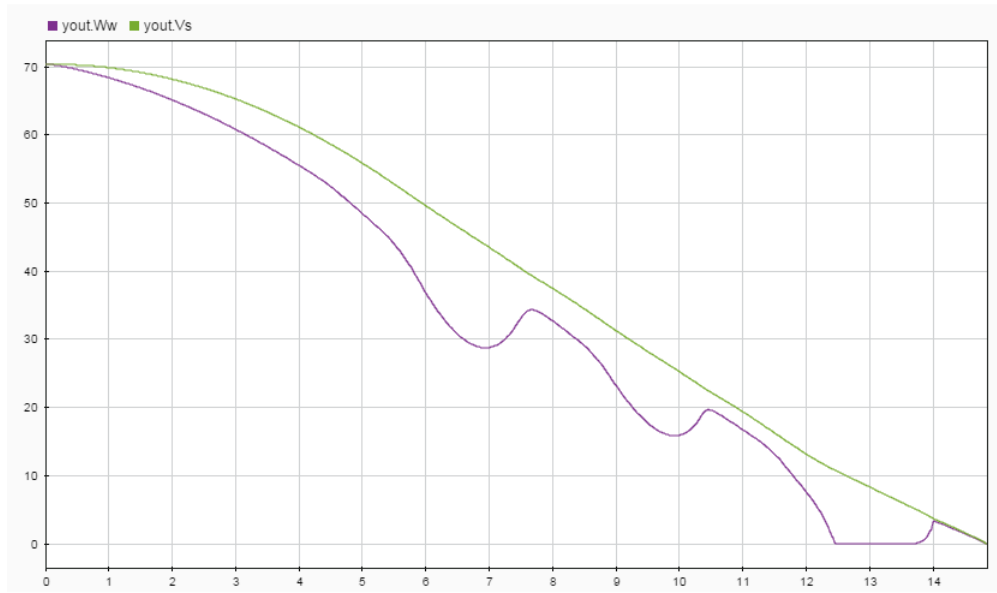
The **Comparison** tab opens and shows the criteria comparisons for the `yout.Ww` signal and the tolerance.



- 6 You can also view signal data from the simulation. Expand **Sim Output** and select the signals you want to plot.

Sim Output (sldemo_absbrake)		
<input type="checkbox"/>	slp	—
<input type="checkbox"/>	yout.Sd	—
<input checked="" type="checkbox"/>	yout.Vs	—
<input checked="" type="checkbox"/>	yout.Ww	—

The **Visualize** tab opens and plots the simulation output.



For information on how to export results and generate reports from results, see “Export Test Results and Generate Reports”.

## See Also

### Related Examples

- “Apply Tolerances to Test Criteria”
- “Capture Baseline Criteria”
- “Run Tests in Multiple Releases”

## Introduction to Test Manager

In this section...
“Start Test Manager” on page 2-14
“Create Tests and Understand the Test Hierarchy” on page 2-14
“View Test Results” on page 2-16
“Share Results” on page 2-16
“Compare Test Files” on page 2-16

Test Manager in Simulink Test helps you to automate Simulink model testing and organize large sets of tests. You perform model tests in Test Manager using test cases in which you specify the criteria that determine a pass-fail outcome. After you run a test, you can view and share the results.

### Start Test Manager

You can start Test Manager from a model or from the MATLAB command prompt.

- To start Test Manager from a model, select **Analysis > Test Manager**.
- To start Test Manager from the command prompt, enter: `sltestmgr`.

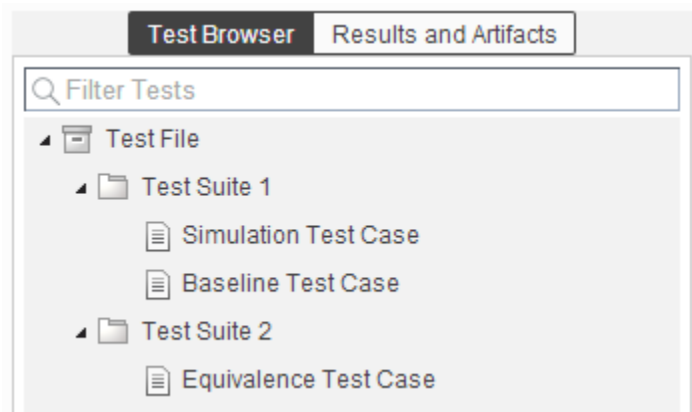
### Create Tests and Understand the Test Hierarchy

In Test Manager, you create test files, which contain one or more test suites that each contain one or more test cases.

To create a test file, select **New > Test File**. Name the file and click **Save**.

The test files and their contents appear in the **Test Browser** pane.

Each new test file contains a test suite, `New Test Suite 1`, which contains a test case, `New Test Case 1`. You can rename test suites and test cases in the browser. The figure shows a test file that contains two test suites that each contain test cases.



Add test suites and test cases to the test file hierarchy using the **New** menu. Use test suites to group related test cases. For each test case, specify details such as the model under test, the simulation outputs to capture, and parameter overrides to apply.

Run tests in the Test Manager, and view results in the **Results and Artifacts** pane. You can run a test file, test suite, or individual test cases.

For baseline and equivalence test cases, you can specify tolerances for the simulation outputs that determine pass or fail. For more information on setting tolerances, see “Apply Tolerances to Test Criteria”.

Using Test Manager, from the **New** menu, you can create these types of test cases:

- **Baseline** — A baseline test is a type comparison test. For a baseline test, you first generate a baseline set of simulation outputs as the basis for comparison. Running a baseline test compares the outputs of the comparison simulation to the baseline. With equivalence tests, you can specify tolerances that determine a range of values that allow the test to pass. That is, the results are equivalent even if not the same. You can set absolute, relative, leading, or lagging tolerances in the **Baseline Criteria** section of the test case. See “Test Model Output Against a Baseline”.
- **Equivalence** — An equivalence test in Test Manager compares signal outputs from two simulations. You can specify tolerances that help the test determine whether the results are equivalent. Set tolerances in the **Equivalence Criteria** section of the test case. See “Test Two Simulations for Equivalence”.
- **Simulation** — A simulation test checks that a simulation runs without errors, including model assertions. See “Test a Simulation for Run-Time Errors”.

- **Real-Time Test** — A baseline, equivalence, or simulation test that runs on the target hardware. See “Test Models in Real Time”.
- **Test Manager Generated Tests** — Tests that you do not need to configure:
  - **Test File from Model**, which generates a test file and uses signal builders and test harnesses in the model as the basis for generating test cases. See “Generate Tests from Model Elements”.
  - **Test for Subsystem**, which generates a test harness for the subsystem you select and generates a test case to run on the test harness. See “Generate Tests for a Subsystem”.

You can also have Test Manager generate test cases for you based on your model design or for a specific subsystem to test a subsystem in isolation. See “Generate Tests from Model Elements” and “Generate Tests for a Subsystem”.

### View Test Results

Tests can pass or fail. If all the criteria defined in a test case are satisfied, within the defined tolerances, then a test passes. If any of the criteria are not satisfied, then the test fails. After the test runs, you can see the results in the **Results and Artifacts** pane. Each test result has a summary page that highlights the outcome of the test: passed, failed, or incomplete. You can also see the simulation output in the results. You can further inspect the signal data from the simulation output using the data inspector view. To view a result in the data inspector view, select it.

### Share Results

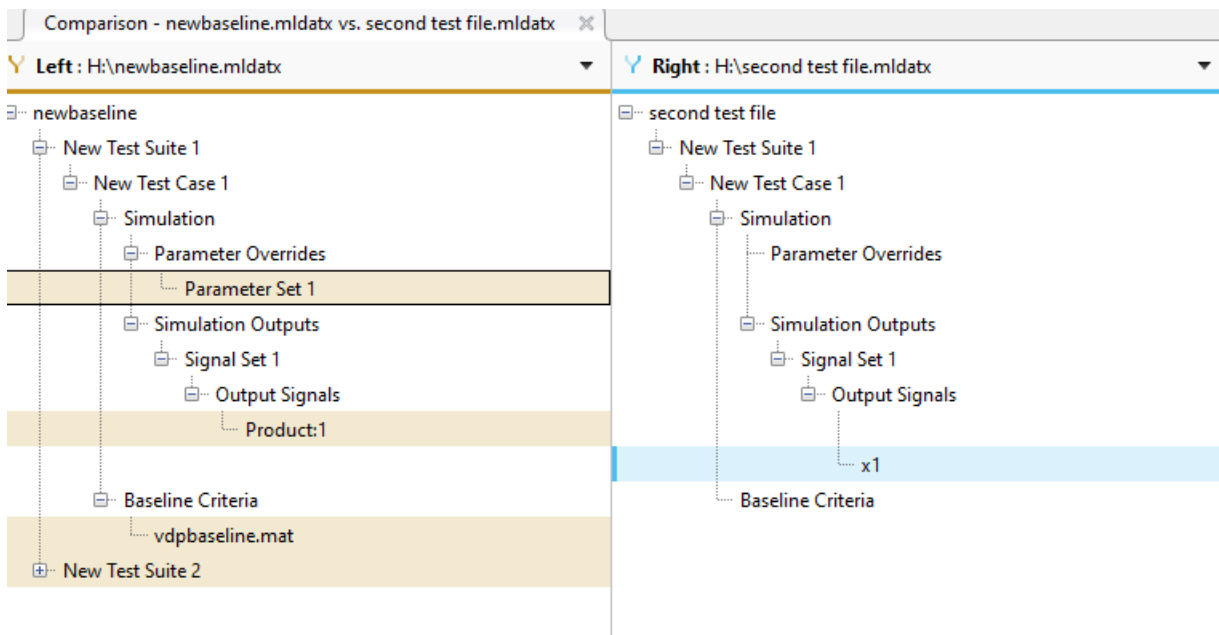
Once you have completed the test execution and analyzed the results, you can share the test results with others or archive them. If you want to share the results to view later in Test Manager, then you can export the results to a file. To archive the results in a document, generate a report, which can include the test outcome, test summary, and criteria used for test comparisons. See “Export Test Results and Generate Reports”.

### Compare Test Files

You can use the **Compare** command in the **File** section of the MATLAB toolstrip to compare two test files. Comparing test files is useful for determining the differences between two similar test files. For example, you can see whether they contain the same test cases and whether those test cases are configured identically.

- 1 From the **File** section of the MATLAB toolstrip, click **Compare**.
- 2 In the **First file or folder** box, enter the first test file that you want to compare. Test files are in the `.mldatx` format.
- 3 In the **Second file or folder** box, enter the second test file that you want to compare.
- 4 For **Comparison type**, select Simulink Test File Comparison. Then click **Compare**.

The figure shows an example of a comparison between two test files. The highlights indicate where one file specifies information that the comparison file does not. For example, `newbaseline.mldatx` includes a test suite that the other file does not contain.



## See Also

### Related Examples

- “Test Model Output Against a Baseline”

- “Test Two Simulations for Equivalence”
- “Code Generation Verification Workflow with Simulink Test”