

Simulink® Test™
Getting Started Guide



MATLAB® & SIMULINK®

R2020b



How to Contact MathWorks



Latest news: www.mathworks.com
Sales and services: www.mathworks.com/sales_and_services
User community: www.mathworks.com/matlabcentral
Technical support: www.mathworks.com/support/contact_us



Phone: 508-647-7000



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

Simulink® Test™ Getting Started Guide

© COPYRIGHT 2015–2020 by The MathWorks, Inc.

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

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

Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See www.mathworks.com/trademarks 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

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)
September 2018	Online Only	Revised for Version 2.5 (Release 2018b)
March 2019	Online Only	Revised for Version 3.0 (Release 2019a)
September 2019	Online Only	Revised for Version 3.1 (Release 2019b)
March 2020	Online Only	Revised for Version 3.2 (Release 2020a)
September 2020	Online Only	Revised for Version 3.3 (Release 2020b)

Product Overview

1

Simulink Test Product Description	1-2
Key Features	1-2

Introduction

2

Functional Testing for Verification	2-2
Test Authoring	2-2
Test Generation	2-3
Test Execution	2-3
Reporting	2-3

Plan Your Test

3

Test Case Planning	3-2
System to Test	3-2
Testing Goals	3-3
Controlling Parameters and Configuration Settings	3-4
Coverage	3-4
Iterations	3-5
Create and Run a Baseline Test	3-6
Start Test Manager	3-6
Verify Model Behavior Against Requirements	3-6
Create and Run a Baseline Test	3-10
Create a Test Harness	3-12
Create the Harness	3-12
Simulate the Test Harness	3-13
Test Using the Test Manager	3-13

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 simulation, baseline, and equivalence test templates that let you perform functional, unit, regression, and back-to-back testing using software-in-the-loop (SIL), processor-in-the-loop (PIL), and real-time hardware-in-the-loop (HIL) modes.

With Simulink Test you can create nonintrusive test harnesses to isolate the component under test. You can define requirements-based assessments using a text-based language, and specify test input, expected outputs, and tolerances in a variety of formats, including Microsoft® Excel®. Simulink Test includes a Test Sequence block that lets you construct complex test sequences and assessments, and a test manager for managing and executing tests. Observer blocks let you access any signal in the design without changing the model or the model interface. Large sets of tests can be organized and executed in parallel or on continuous integration systems.

You can trace tests to requirements (with Simulink Requirements™) and generate reports that include test coverage information from Simulink Coverage™.

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

Functional Testing for Verification

In this section...

“Test Authoring” on page 2-2

“Test Generation” on page 2-3

“Test Execution” on page 2-3

“Reporting” on page 2-3

You can use Simulink Test to author, manage, and execute tests for Simulink models and generated code. The Test Manager provides an interactive way to author tests from scratch, import existing test data and harness models, and organize your tests. You can run test cases individually, in batch, or as a filtered subset of the test file, and you can control parameters and iterate over parameter values. The modes in which you can run tests are in-model, software-in-the-loop (SIL), processor-in-the-loop (PIL), and hardware-in-the-loop (HIL). To run HIL tests, the target computer must have Simulink Real-Time™ installed. You can also run the same tests back-to-back in multiple releases of MATLAB®.

Results include a concise pass/fail summary for elements in your test hierarchy, including iterations, test cases, test suites, and the test file. Visualization tools help you drill down into individual data sets to determine, for example, the time and cause of a particular failure. Coverage results from Simulink Coverage help quantify the extent to which your model or code is tested.

For example, you can:

- Compare results between your model and generated code by running back-to-back equivalence tests between different environments, such as model simulation, SIL, PIL, and HIL execution.
- Optimize your model or code by iterating over parametric values or configuration parameters.
- Start testing on a unit level by using test harnesses, and reuse those tests as you scale up to the integration and system level.
- Run models that contain test vectors and assessments inside the Simulink block diagram.

Simulink Test includes a comprehensive programmatic interface for writing test scripts, and Simulink tests can be integrated with MATLAB tests using MATLAB Unit Test.

Test Authoring

When you author a test, you define test inputs, signals of interest, signal pass/fail tolerances, iterations over parametric values, and assessments for simulation behavior. You can author test input vectors in several ways:

- Graphically, such as with the Signal Editor
- From datasets, such as using Excel or MAT files
- As a sequence of test steps that progresses according to time or logical conditions

You can define assessments to indicate when functional requirements are not met. These assessments follow from your design requirements or your test plan. You can define assessments in several ways:

- With a structured assessment language. The structured language helps you assess complex timing behavior, such as two events that must happen within a certain time frame. It also helps you identify conflicts between requirements.

- With `verify` statements in a Test Assessment or Test Sequence block. For information on how to set up the blocks in your model, see “Assess Model Simulation Using `verify` Statements”.
- With blocks in the Model Verification block library.
- With tolerances you set on the simulation data output. Tolerances define the acceptable delta from baseline data or another simulation.
- With a custom criteria script that you author using MATLAB.

You can use existing test data and test models with Simulink Test. For example, if you have data from field testing, you can test your model or code by mapping the data to your test case. If you have existing test models that use Model Verification blocks, you can organize those tests and manage results in the Test Manager.

Test Generation

Using Simulink Design Verifier™, you can generate test cases that achieve test objectives or increase model or code coverage. You can generate test cases from the Test Manager, or from the Simulink Design Verifier interface. Either way, you can include the generated test cases with your original tests to create a test file that achieves complete coverage. You can also link the new test cases to additional requirements.

Test Execution

You can control test execution modes from the Test Manager. For example, you can:

- Run tests in multiple releases of MATLAB. Multiple release testing allows you to leverage recent test data while executing your model in its production version.
- Run back-to-back tests to verify generated code. You can run the same test in model, SIL, and PIL mode and compare numeric results to demonstrate code-model equivalence.
- Run HIL tests to verify systems running on real-time hardware using Simulink Real-Time, including `verify` statements in your model that help you determine whether functional requirements are met.
- Decrease test time by running tests in parallel using the Parallel Computing Toolbox™ or MATLAB Parallel Server™, or running a filtered subset of your entire test file.

Reporting

When reporting your test results, you can set report properties that match your development environments. For example, reporting can depend on whether tests passed or failed, and reports can include data plots, coverage results, and requirements linked to your test cases. You can create and store custom MATLAB figures that render with a report. Reporting options persist with your test file, so they run every time you execute a test.

A MATLAB Report Generator™ license adds additional customization options, including:

- Creating reports from a Microsoft Word or PDF template
- Assembling reports using custom objects that aggregate individual results

See Also

Related Examples

- “Create and Run a Baseline Test” on page 3-6
- “Inputs”
- “Test Execution”

Plan Your Test

- “Test Case Planning” on page 3-2
- “Create and Run a Baseline Test” on page 3-6
- “Create a Test Harness” on page 3-12

Test Case Planning

In this section...

“System to Test” on page 3-2

“Testing Goals” on page 3-3

“Controlling Parameters and Configuration Settings” on page 3-4

“Coverage” on page 3-4

“Iterations” on page 3-5

You can use Simulink Test to functionally test models and code. Before you create a test, consider:

- What model or model component are you testing?
- Does the component integrate code, such as using a C Caller block?
- Do you need to run your test in multiple environments, such as using generated code on an external target?
- What is the test objective? For example, do you need requirements verification, data comparison, or a quick test of your design?
- Does your test use multiple parametric values?
- Do you require coverage results?

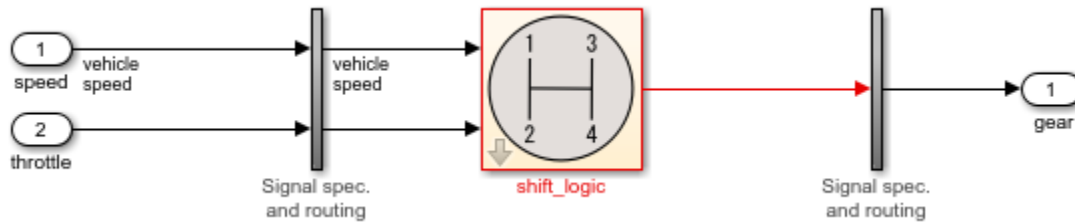
System to Test

You can test a whole model, or you can focus on a model component. You can create a test case for the entire model, or use a test harness.

- A test harness lets you isolate a whole model for testing. You can add verification blocks and logic to the test harness to keep your main model clear. You can also add other models that simulate the environment.
- A test harness lets you isolate a model component from the main model for unit testing. For information, see “Create a Test Harness” on page 3-12.

A test harness is associated with a block or an entire model. A test harness contains a copy of the block or a reference to the model, and inputs and outputs placed for testing purposes. You can add other blocks to the test harness. You can save the harness with your model file, or you can save it in a file separate from the model. The test harness works the same whether it is internal or external to the model. You can also specify whether to synchronize changes between the model and the test harness.

The figure shows an example of a test harness. The component under test is the `shift_logic` block, which is copied from the main model during harness creation. The copy of the `shift_logic` block is linked to the main model. The inputs are Inport blocks and the output is an Outport block. The vertical subsystems contain signal specification blocks and routing that connects the component interface to the inputs and outputs.



Before you create a test harness, decide whether to save the harness with the model and how to synchronize the harness with the model. For more information, see “Test Harness and Model Relationship” and “Synchronize Changes Between Test Harness and Model”.

Testing Goals

Before you author your test, understand your goals.

Requirements Verification

You can assess whether a model behaves according to requirements. For example, suppose your requirements state that a transmission must shift gears at certain speeds. Create a test case for the transmission controller model, or create a test harness for the controller component. Verify whether the model meets requirements by:

- Authoring verify statements in the model or test harness.
- Including “Model Verification Blocks” blocks in the model or test harness.
- Capturing simulation output in the test case, and comparing simulation output to baseline data.

Run the test case and capture results in the Test Manager. You can link the test case to requirements authored in Simulink Requirements, collect coverage with Simulink Coverage, and add test cases for more scenarios. For an example, see “Test Downshift Points of a Transmission Controller”.

Data Comparison

Using the Test Manager, you can compare simulation results to baseline data, or to another simulation. In either case, you must specify the signals to capture.

In a baseline test, establish the baseline data, which are the expected outputs. You can define baseline data manually, import baseline data from an Excel or MAT file, or capture baseline data from a simulation.

In equivalence testing, you compare two simulations to see whether they are equivalent. For example, you can compare results from two solvers, or compare results from simulations in normal and software-in-the-loop (SIL) mode. Explore the impact of different parameter values or calibration data sets by running back to back tests. For an example, see “Test Two Simulations for Equivalence”.

For comparison tests, you can accept results that fall within a technically acceptable difference by specifying value or time tolerances. You can specify this tolerance before you run the test, or view the results and adjust the tolerance afterward. For more information, see “Set Signal Tolerances”.

Simulation Testing

In cases where your test only requires a model to simulate without errors, you can run a simulation test. A simulation tests is useful if your model is still in development, or if you have an existing test

model that contains inputs and assessments and logs relevant data. Using the Test Manager to run simulation tests allows you to manage numerous tests and to capture and manage results systematically.

By default, running a test in the Test Manager disables breakpoints for Simulink, Stateflow®, and Test Sequence blocks. Use **Run with Stepper** in the Test Manager to run a test in the simulation stepper and use the breakpoints.

Note that Simulink and Stateflow breakpoints are not supported when running a test from the Test Manager. You can select the **Run with Stepper** button in the Test Manager, run the test, and use the breakpoints with the simulation stepper.

Multiple Release Testing

You can set up your test to run other releases of MATLAB that you have installed, starting with R2011b. This capability lets you run tests in releases that do not have Simulink Test. You can run the same test in multiple releases to verify that it passes in each of them. For more information, see “Run Tests in Multiple Releases of MATLAB”.

SIL and PIL Testing

You can verify the output of generated code by running back-to-back simulations in model and SIL (software-in-the-loop) or PIL (processor-in-the-loop) mode. The same back-to-back test can run multiple test scenarios by iterating over different test vectors, defined in a MAT or Excel file. You can apply tolerances to your results, to allow for technically acceptable differences between model and generated code. Tolerances can include acceptable differences for values and timing, which can apply with hardware running in real time. You cannot use SIL or PIL mode in; Subsystem models.

Real-Time Testing

With Simulink Real-Time, you can include effects of physical plants, signals, and embedded hardware by executing tests in HIL (hardware-in-the-loop) mode on a real-time target computer. By running a baseline test in real-time, you can compare results against known good data. You can also run a back-to-back test between model, SIL, or PIL, and a real-time simulation. In either case, the Test Manager allows selection of the target computer configured with Simulink Real-Time. You can only execute real-time HIL tests on target computers using Simulink Real-Time.

Controlling Parameters and Configuration Settings

You can control parameters and configuration settings from the Test Manager, for example, to explore design options or iterate over calibration sets, solvers, and data type settings. You can create sets of parameters of interest and override values when each iteration runs. You can also perform parameter sweeps by writing more complex scripts directly in the Test Manager.

Coverage

With Simulink Coverage, you can collect coverage data to help quantify the extent to which your model or code is tested. When you set up coverage collection for your test file, the test results include coverage for the system under test and, optionally, referenced models. You can specify the coverage metrics to return.

If your results show incomplete coverage, you can increase coverage by:

- Adding test cases manually to the test file.
- Generating test cases to increase coverage, with Simulink Design Verifier. You can generate test cases from the Test Manager results.

In either case, you can link new test cases to requirements. This is required for certain certifications.

Iterations

An iteration is a variation of a test case that uses a particular set of data. For example, suppose that you have multiple sets of input data. You can set up one iteration of your test case for each external input file, where otherwise the tests are the same. You can also specify parameter sets and override values in iterations. Another way to override parameters is using scripted parameter sweeps, which enable you to iterate through many values.

When you design your test, you decide whether you want multiple tests or iterations of the same test. One advantage of using iterations is that you can run your tests using fast restart. Iterations make sense if you are changing only parameters, inputs, or configuration settings but otherwise the tests are the same. Use separate test cases if you need independent configuration control or each test relates to a different requirement.

For more information on iterations, see “Test Iterations”.

See Also

Test Manager

Related Examples

- “Functional Testing for Verification” on page 2-2
- “Create and Run a Baseline Test” on page 3-6

Create and Run a Baseline Test

In this section...

“Start Test Manager” on page 3-6

“Verify Model Behavior Against Requirements” on page 3-6

“Create and Run a Baseline Test” on page 3-10

In this tutorial, you set up and run two tests:

- The first test verifies whether a model meets a requirement.
- The second test compares a simulation result to baseline data.

Start Test Manager

- 1 Open the `sltestCruiseControlDefective` model in the `matlab/examples/simulinktest/main` folder.
- 2 To start the Test Manager, on the **Apps** tab, under Model Verification, Validation, and Test, click **Simulink Test**. On the **Tests** tab, click **Simulink Test Manager**.

Verify Model Behavior Against Requirements

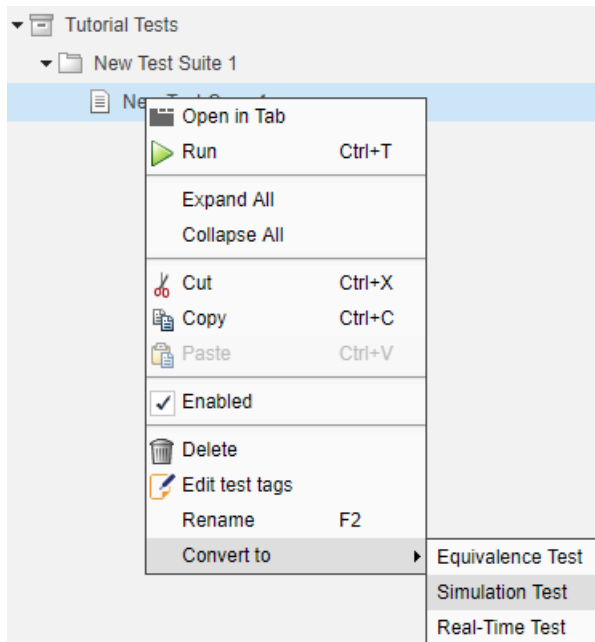
In this example, you create a test file structure, add a test case to it, and run the test.

Create the Test File Structure

- 1 Set the current folder to a writable folder.
- 2 Create a test file. From the Test Manager toolstrip, select **New > Test File**.
- 3 Name the file `Tutorial Tests` and save it.

A test file contains one or more test suites, and test suites contain one or more test cases. Use this structure to organize related tests. This structure also helps you run tests individually or to run all tests in the file or suite.

- 4 The default test case is a baseline test. Convert it to a Simulation test. Select the test case and, from the context menu, select **Convert to > Simulation Test** and click **Yes** to confirm the conversion.



- 5 Rename the test `My Verification Test`. You can use the context menu in the test browser and select **Rename**. Or, you can click the name and type a new one in the test case.

Assign a Model to the Test Case

The model uses a Verification Subsystem block from the Simulink Design Verifier library. The subsystem contains an Assertion block that checks whether the system disengages if the brake has been applied for three time steps. Signal logging is enabled for the Signal Builder block outputs.

Set the test case to use the `sltestCruiseControlDefective` model. In Test Manager, under

System Under Test, click the **Use current model** button .

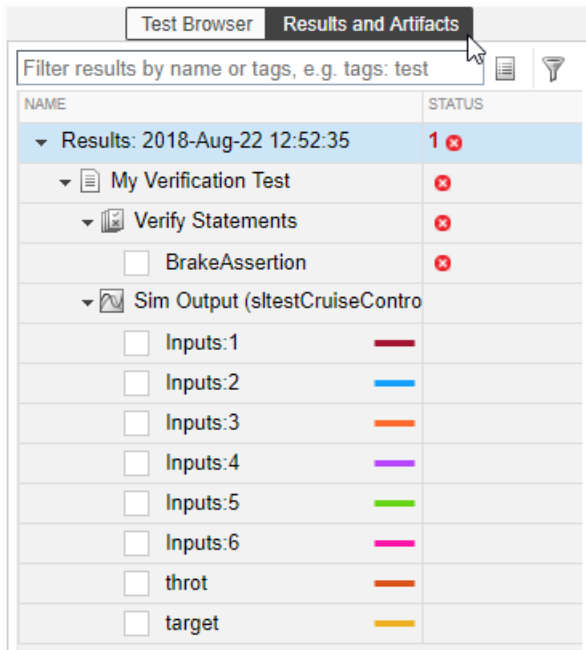
You can run this test case without specifying other assessments, because the model contains an Assertion block. Using the Test Manager, as opposed to simulating the model itself, allows you to view, manage, and export results. Also, it sets up a test framework for you to add more tests, capture baseline data, and reuse tests for other verification activities such as code generation verification.

Run the Test and Examine Results

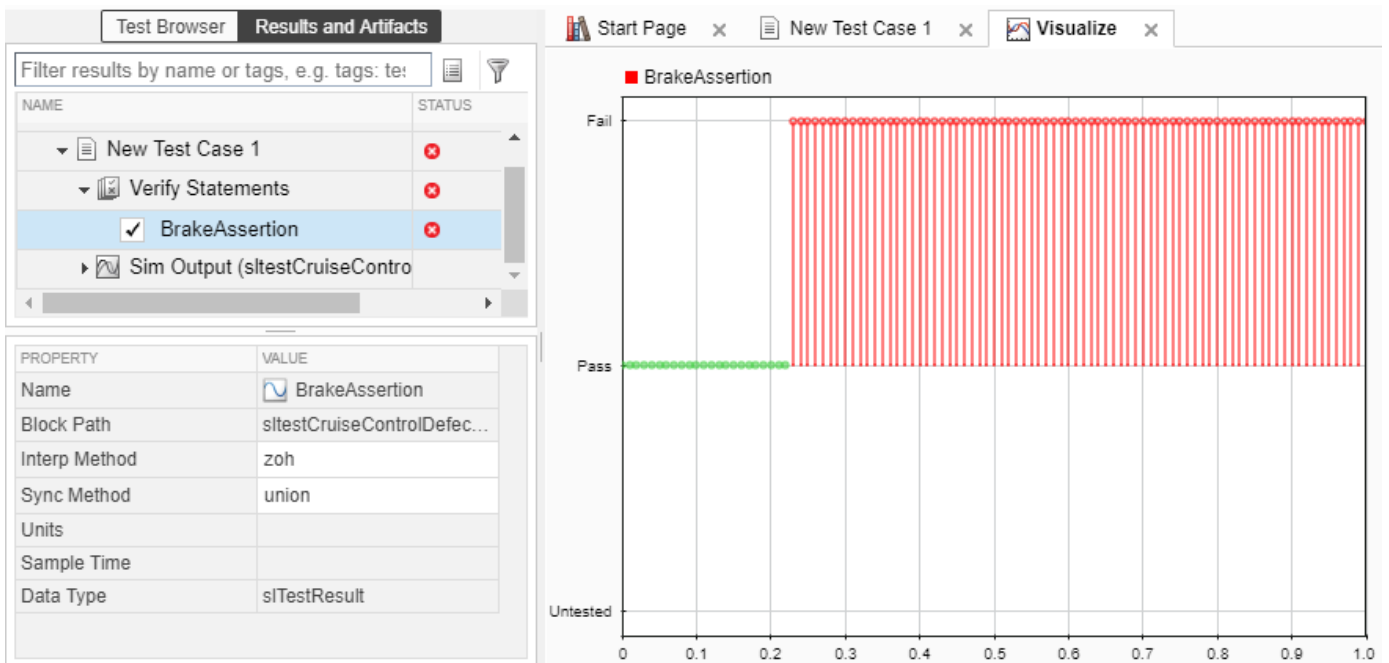
- 1 Click **Run** in the Test Manager Toolstrip.

When you click **Run**, the left navigation changes from the **Test Browser** to the **Results and Artifacts** pane.

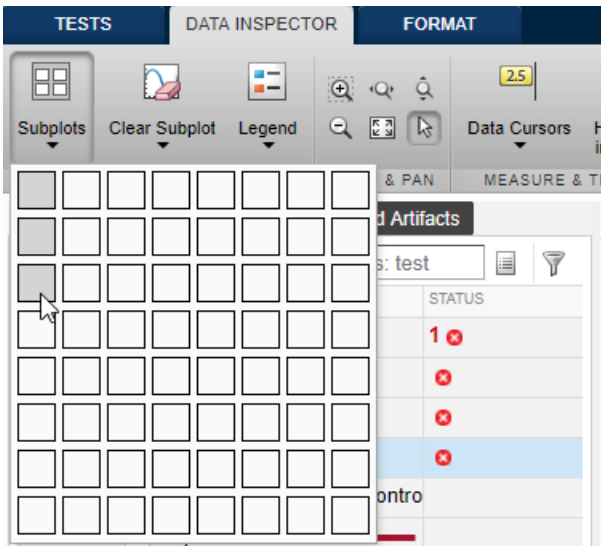
- 2 Examine the results. The test failed. Right-click the newest result and select **Expand All Under** to see all the results. The test failed because `BrakeAssertion` failed.



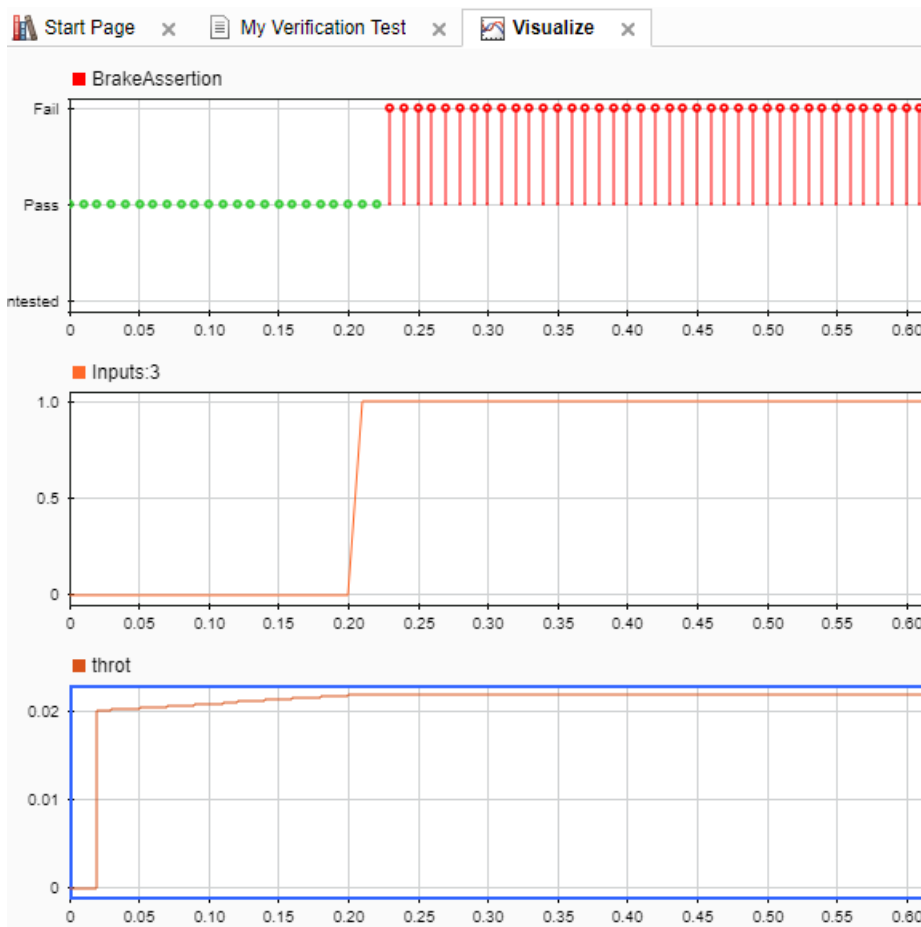
- 3 Click **BrakeAssertion** and select the check box to visualize the results. You can see that the test failed at .23 seconds.



- 4 Plot signals from the output. From the Data Inspector Toolstrip, click **Subplots** and select three plots.



- 5 After you create the plots, the BrakeAssertion signal stays in the first plot. Click a plot where you want a signal to appear and then click the signal. Using this technique, add Input 3 (brake input) in plot 2 and throt in plot 3.



These plots show you that when the brake goes on (Input 3) the throttle stays on. The throttle should go to 0.

Locate the Source of the Failure

To locate the source of failure in the model, in the test results, right-click the **BrakeAssertion** failure and select **Highlight in Model**. The model highlights the BrakeAssertion block.


Create and Run a Baseline Test

Baseline tests compare the outputs from a simulation against the expected outputs. In a corrected version of the model, the throttle goes to 0 if the brake is applied for three time steps. Because the model is correct, you want to commit the correct results to a test case to test against in the future.

Create Test Case and Assign System Under Test

- 1 Open the model `sltestBasicCruiseControlHarnessModel`.
- 2 In the test browser, select the test file `Tutorial Tests`. From the Test Manager Toolstrip, select **New > Baseline Test**.

The test file contains a new test suite that contains a new baseline test case.

- 3 Rename the test case `My Baseline Test`.
- 4 In the test case, under **System Under Test**, click **Use current model** .

Capture the Baseline

You can capture the baseline test outputs in Test Manager as an Excel file or a MAT-file. Alternatively, you can add an external file to the test case as the baseline.

- 1 In the test case, under **Baseline Criteria**, click **Capture**.
- 2 Set **File format** to `Excel`, enter `baseline1` for the file name, and click **Capture**.

The test case adds the baseline data—the output from the logged signals and the two Output blocks.

Open the Excel file and observe the data. Select the baseline and click **Edit**. Time-series values appear for each of the logged signals and outputs in the model. Close the Excel file.

▼ BASELINE CRITERIA*

Include baseline data in test result

SIGNAL NAME	SHEETS	ABS TOL	REL TOL	LEADING TOL	LAGGING TOL	+
▶ <input checked="" type="checkbox"/> baseline1	baseline1	0	0.00%	0	0	

▶ ITERATIONS

Run a Baseline Test

One reason to run a baseline test is for regression testing, to see whether the model output still matches the baseline. In this example, because you just created the baseline, you expect the test to pass.

Select the test case and click **Run**. The results match the baseline and the test passes. When you expand the results, you can see that the verify statements now pass.

Test Browser Results and Artifacts

Filter results by name or tags, e.g. tags: test

NAME	STATUS
▶ Results: 2018-Aug-22 12:52:35	1 ❌
▼ Results: 2018-Aug-22 16:53:33	1 ✅
▼ My Baseline Test	✅
▶ Baseline Criteria Result	✅
▼ Verify Statements	✅
<input type="checkbox"/> BrakeAssertion	✅
▶ Sim Output (sltestBasicCruiseC	

See Also

Test Manager

Related Examples

- “Test Case Planning” on page 3-2
- “Examine Test Failures and Modify Baselines”
- “Create a Test Harness” on page 3-12

Create a Test Harness

A test harness is a model that isolates the component under test, with inputs, outputs, and verification blocks configured for testing scenarios. You can create a test harness for a model component or for a full model. A test harness gives you a separate testing environment for a model or a model component. For example:

- You can unit-test a subsystem by isolating it from the rest of the model.
- You can create a closed-loop testing scenario for a controller by adding a plant model to the test harness.
- You can keep your main model clear of unneeded verification blocks by placing Model Verification and Test Assessment blocks in the test harness.

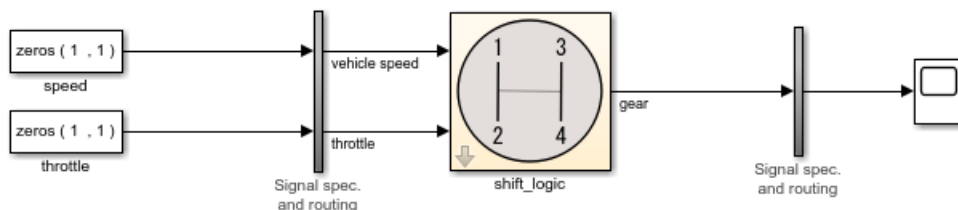
To assign a test harness to a test case, select **Test Harness** in the **System Under Test** section of the Test Manager.

You can save the harness with your model, or you can save it in an external file. If your model is under change management, consider saving the test harness in an external file. The harness works the same whether it is internal or external to the model. For more information, see “Manage Test Harnesses” and “Synchronize Changes Between Test Harness and Model”.

Create the Harness

In this example, you create a harness to test the `shift_logic` subsystem of the model `sltestCarRootInport`.

- 1 Open the model `sltestCarRootInport` from the folder `matlab/examples/simulinktest/main`.
- 2 Right-click the `shift_logic` subsystem. From the context menu, select **TestHarness > Create for 'shift_logic'**.
- 3 In the Create Test Harness dialog box, specify the inputs, outputs, and other options:
 - a Use Constant blocks to provide input signals. Under **Sources and Sinks**, set the source to Constant and the sink to Scope.
 - b Leave the other options with their default selections. By default:
 - The harness saves with the model file.
 - The harness synchronizes with the model on open, which means that changes to the model update the harness.
- 4 Click **OK** to create the test harness.



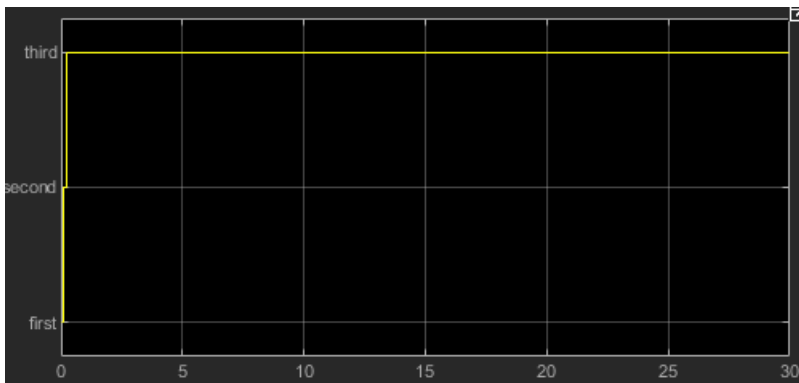
At the center of the harness is a copy of the `shift_logic` subsystem. The `shift_logic` subsystem is the component under test.

The two vertical subsystems contain signal specification and routing. For more information on test harness architecture, see “Test Harness Construction for Specific Model Elements”.

Simulate the Test Harness

Assign values to the Constant blocks to test the component:

- 1 Change the value of the speed block to 50.
- 2 Change the value of the throttle block to 30.
- 3 Click Run in the Simulation tab to simulate the harness.
- 4 Open the scope and look at the result. The shift controller selects third gear.




Test Using the Test Manager

In the previous case, you supplied test inputs with Constant blocks. You can also use test inputs from external data files.

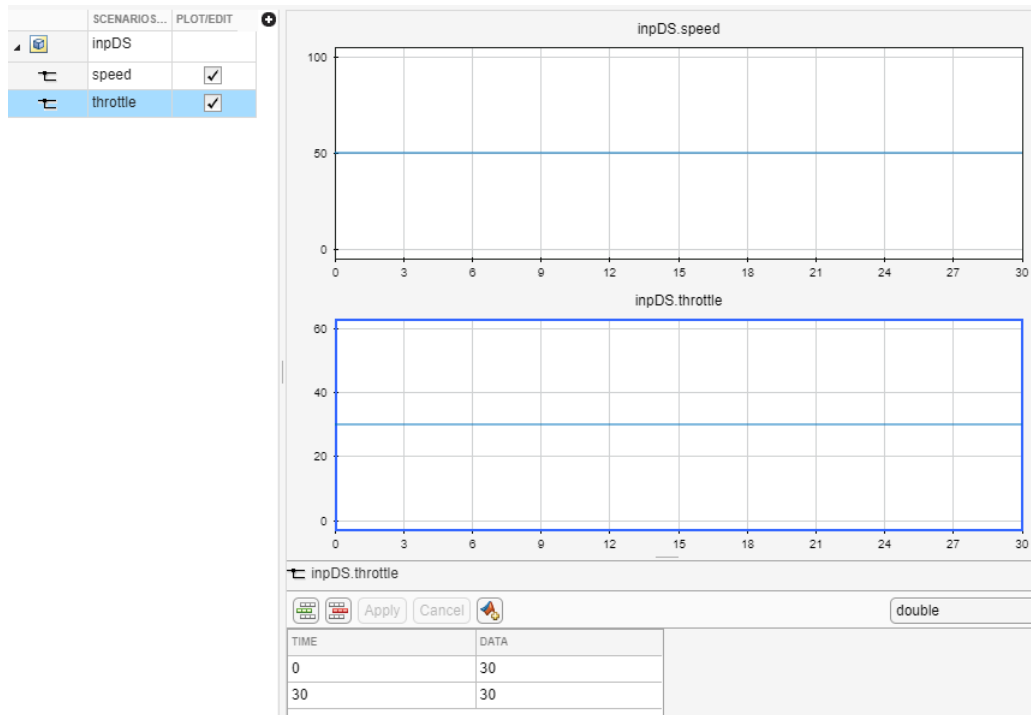
- 1 Create a test harness that uses Inport sources.
- 2 Create a test case that uses the test harness as the **System Under Test**.
- 3 Map external inputs to the test case.

Using a test case in the Test Manager allows you to iterate with different test vectors, add test cases, run batches of test cases, and organize your results. This example shows you how to use external data with a test harness, and simulate from the Test Manager.

- 1 To open the Test Manager, on the **Apps** tab, under Model Verification, Validation, and Test, click **Simulink Test**. Then, on the **Tests** tab, click **Simulink Test Manager**.
- 2 Select **New > Test File** from the Test Manager Toolstrip.
- 3 Name the file `ShiftLogicTest`.
- 4 Select **New Test Case 1**. In the **System Under Test** section, click **Use current model** .
- 5 For **Test Harness**, select `ShiftLogic_InportHarness` from the drop down list. The test harness already exists in the model.
- 6 In the **Inputs** section, click **Create**. Name the input data file `shift_logic_input` and select MAT file format.

7 In the Signal Editor, enter the values for the inputs:

- 1 Select the **speed** signal and enter 50 for times 0 and 30. Click **Apply** to update the plot.
- 2 Select the **throttle** signal and enter 30 for times 0 and 30. Click **Apply** to update the plot.



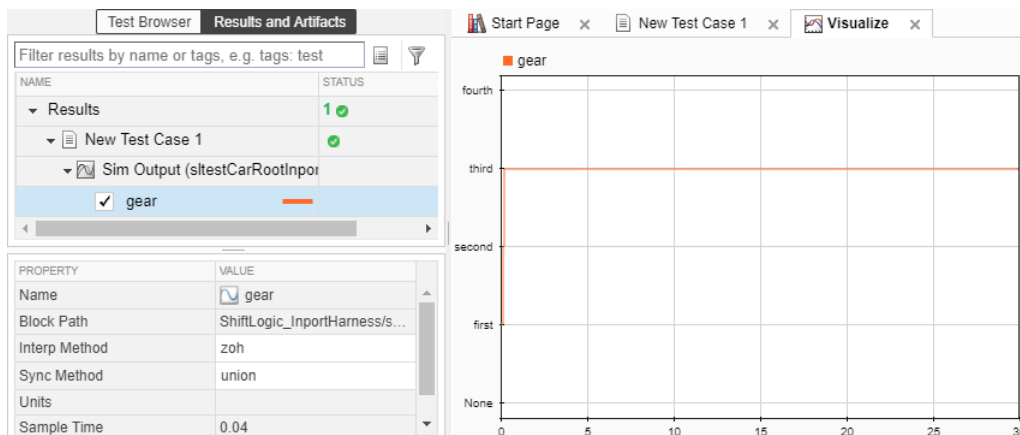
3 Click **Save** in the Signal Editor Toolstrip.

8 Select output data to capture.

- 1 In the **Simulation Outputs** section of the Test Manager, click **Add**.
- 2 In the test harness block diagram, select the gear signal line. Select the signal in the **Connect** dialog box.
- 3 Click **Done** to add the signal to the test case outputs.

9 Click **Run** in the Test Manager Toolstrip.

10 Expand the results and highlight the gear signal output. The plot shows the controller selects third gear.



See Also
Test Manager

Related Examples

- “Create and Run a Baseline Test” on page 3-6
- “Refine, Test, and Debug a Subsystem”

