

**Simulink® Test™**  
Getting Started Guide



**MATLAB® & SIMULINK®**

R2023a



## How to Contact MathWorks



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



Phone: 508-647-7000



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

### *Simulink® Test™ Getting Started Guide*

© COPYRIGHT 2015–2023 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)
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)
March 2021	Online Only	Revised for Version 3.4 (Release 2021a)
September 2021	Online Only	Revised for Version 3.5 (Release 2021b)
March 2022	Online Only	Revised for Version 3.6 (Release 2022a)
September 2022	Online Only	Revised for Version 3.7 (Release 2022b)
March 2023	Online Only	Revised for Version 3.8 (Release 2023a)

## Product Overview

### 1

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

## Introduction

### 2

<b>Functional Testing for Verification</b> .....	<b>2-2</b>
Test Authoring .....	<b>2-2</b>
Test Generation .....	<b>2-3</b>
Test Execution .....	<b>2-3</b>
Reporting .....	<b>2-3</b>

## Plan Your Test

### 3

<b>Test Planning and Strategies</b> .....	<b>3-2</b>
Identify the Test Goals .....	<b>3-2</b>
Test a Whole Model or Specific Components .....	<b>3-3</b>
Use a Test Harness .....	<b>3-4</b>
Determine the Inputs and Outputs .....	<b>3-4</b>
Optimize Test Execution Time .....	<b>3-4</b>
Use the Programmatic or GUI Interface .....	<b>3-5</b>
<b>Create a Simple Baseline Test</b> .....	<b>3-6</b>
<b>Create a Test Harness</b> .....	<b>3-12</b>
Create the Harness .....	<b>3-12</b>
Simulate the Test Harness .....	<b>3-13</b>
Test Using the Test Manager .....	<b>3-13</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 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 Requirements Toolbox™) 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. Coverage is not supported for SIL or PIL blocks.
- 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**

- “Test Planning and Strategies” on page 3-2
- “Create a Simple Baseline Test” on page 3-6
- “Inputs”
- “Test Execution”

# Plan Your Test

---

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

## Test Planning and Strategies

### In this section...

- “Identify the Test Goals” on page 3-2
- “Test a Whole Model or Specific Components” on page 3-3
- “Use a Test Harness” on page 3-4
- “Determine the Inputs and Outputs” on page 3-4
- “Optimize Test Execution Time” on page 3-4
- “Use the Programmatic or GUI Interface” on page 3-5

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

- Your test goals on page 3-2
- Whether to test the whole model or individual components on page 3-3
- Whether to use a test harness on page 3-4
- What your inputs and outputs should be on page 3-4
- How to optimize your test simulation time on page 3-4
- Whether to use the interactive GUI or the programmatic interface on page 3-5

### Identify the Test Goals

Before you author your test, understand your goals. You might have one or more of these goals:

#### Simulation Testing

In cases where your test only requires a model to simulate without errors or to perform regression testing, you can run a simulation test. Simulation tests are 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. For an example, see “Test a Simulation for Run-Time Errors”.

#### Requirements Verification

If you have Requirements Toolbox installed, you can assess whether a model behaves according to requirements by linking one or more test cases to requirements authored in the Requirements Toolbox. You can verify whether the model meets requirements by:

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

For an example the uses `verify` statements, see “Test Downshift Points of a Transmission Controller”.

#### Data Comparison

You can compare simulation results to baseline data or to another simulation. You can also compare results between different MATLAB releases.

In a baseline test, you first 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. For more information, see “Compare Model Output to Baseline Data”

In an equivalence test, you compare two simulations to see whether they are equivalent. For example, you can compare results from two solvers, or compare results from normal mode simulation to generated code in software-in-the-loop (SIL), processor-in-the-loop (PIL), or real time hardware-in-the-loop (HIL) mode. You can explore the impact of different parameter values or calibration data sets by running equivalence tests. For an equivalence testing example, see “Test Two Simulations for Equivalence”.

In a multiple release test you run tests in more than one installed MATLAB version. Use multiple release testing to verify that tests pass and produce the same results in different versions.

### **SIL and PIL Testing**

You can verify the output of generated code by running back-to-back simulations in normal and SIL or PIL modes. Running back-to-back tests in normal and SIL or PIL mode is a form of equivalence testing. 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 value and timing technically acceptable differences between the model and generated code. Tolerances can also apply to code running on hardware in real time. You cannot use SIL or PIL mode in Subsystem models.

### **Real-Time Testing**

With Simulink Real-Time, you can include the 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 a model, SIL, or PIL, and a real-time simulation.

### **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. Coverage is supported for Model Reference blocks, atomic Subsystem blocks, and top-level models if they are configured for Software-in-the-Loop (SIL) or Processor-in-the-Loop (PIL). Coverage is not supported for SIL or PIL S-Function blocks.

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.

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

## **Test a Whole Model or Specific Components**

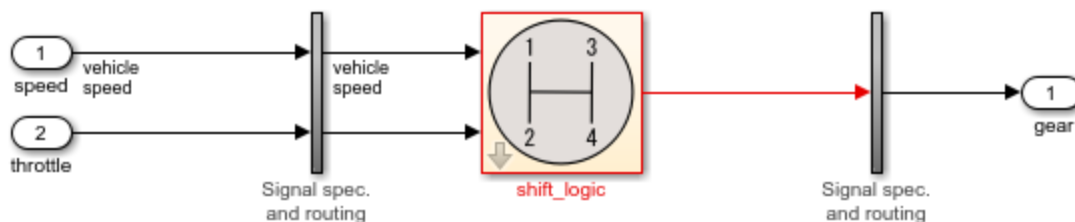
You can test a whole model or a model component. One way to perform testing is to first unit test a component and then increase the number of components being tested to perform integration and system-level testing. By using this testing scheme, you can more easily identify and debug test case and model design issues. If you want to obtain aggregated coverage, test your whole model.

## Use a Test Harness

A test harness contains a copy of the component being tested or, for testing a whole model, a reference to the model. Harnesses also contain the inputs and outputs used for testing. Using a test harness isolates the model or component being tested from the main model, which keeps the main model clean and uncluttered.

You can save the harness internally with your model file, or in an external file separate from the model. Saving the harness internally simplifies sharing the model and tests because there are fewer artifacts, and allows you to more easily debug testing issues. Saving the harness externally lets you reuse and share just the harness, and clearly separate your design from your testing artifacts. The test harness works the same whether it is internal or external to the model. For more information about test harnesses, see “Test Harness and Model Relationship” and “Create a Test Harness” on page 3-12.

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.

## Determine the Inputs and Outputs

Consider what input signals and signal tolerances to use to test your model and where to obtain them from. You can add a Signal Editor block to your test harness to define the input signals. Alternatively, you can define and use inputs from a Microsoft Excel or MAT file, or a Test Sequence block. If you have Simulink Design Verifier installed, you can generate test inputs automatically.

For the output from your test, consider whether you want the test details and test results data saved to a report for archiving or distribution. You can also use outputs as inputs to Test Assessment blocks to assess and verify the test results. For baseline tests, use outputs to compare the actual results to the expected results.

## Optimize Test Execution Time

Ways that you can optimize testing time include using iterations, parallel testing, and running a subset of test cases.

When you design your test, decide whether you want multiple tests or iterations of the same test. An iteration is a variation of a test case. Use iterations for different sets of input data, parameter sets, configuration sets, test scenarios, baseline data, and simulation modes in iterations. One advantage of

using iterations is that you can run your tests using fast restart, which avoids having to recompile the model for each iteration and thus, reduces simulation time.

For example, if you have multiple sets of input data, you can set up one test case iteration for each external input file. Another way to override parameters is using scripted parameter sweeps, which enable you to iterate through many values. 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”.

If you have Parallel Computing Toolbox, you can run tests in parallel on your local machine or cluster. If you have MATLAB Parallel Server, you can run tests in parallel on a remote cluster. Testing in parallel is useful when you have a large number of test cases or test cases that take a long time to run. See “Run Tests Using Parallel Execution”.

Another way to optimize testing time is to run a subset of test cases by tagging the tests to run.

For more information on improving and optimizing model simulation, see the Optimize Performance section under Simulation in the Simulink documentation.

## **Use the Programmatic or GUI Interface**

To create and run tests Simulink Test provides the Test Manager, which is an interactive tool, and API commands and functions you can use in test scripts and at the command-line. With the Test Manager you can manage numerous tests and capture and manage results systematically. If you need to define new test assessments, use the Test Manager because this functionality is not available in the API. If you are working in a Continuous Integration environment, use the API. Otherwise, you can use whichever option you prefer.

### **See Also** **Test Manager**

### **Related Examples**

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

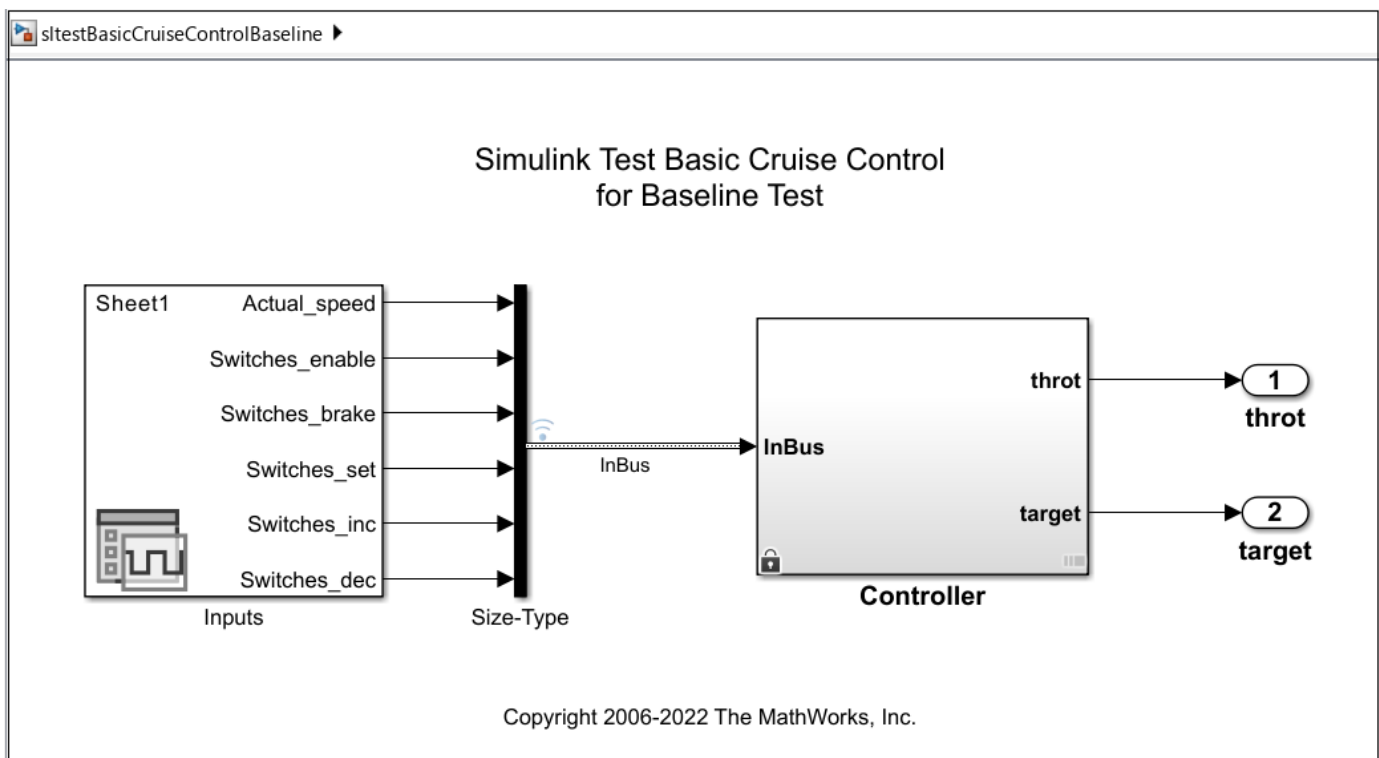
## Create a Simple Baseline Test

This example shows how to create a simple test that compares test results to baseline data. Baseline tests are useful for regression testing when you change a model and want to ensure the output matches the baseline data.

The model used in this example is `sltestBasicCruiseControlBaseline`, which has a subsystem named `Controller` with a bus of six input signals and throttle and target speed output signals.

### Open the Model

```
open_system('sltestBasicCruiseControlBaseline')
```

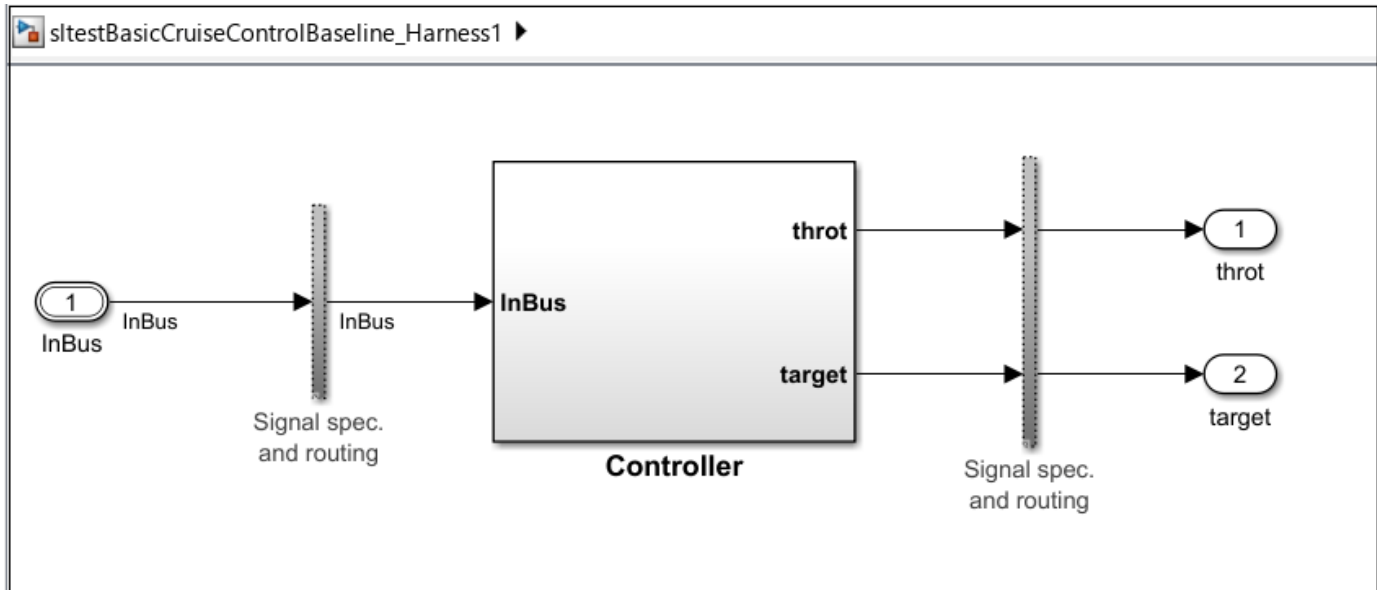


### Create the Test Harness

The test harness for this example is a separate model that you can use to test the `Controller` subsystem without changing the subsystem or main model.

1. In the main model, right-click the `Controller` block and select **Test Harness > Create for 'Controller'** from the menu.
2. In the Create Test Harness dialog box, click **OK** to create the test harness using the default values. The created test harness, `sltestBasicCruiseControlBaseline_Harness1`, opens.

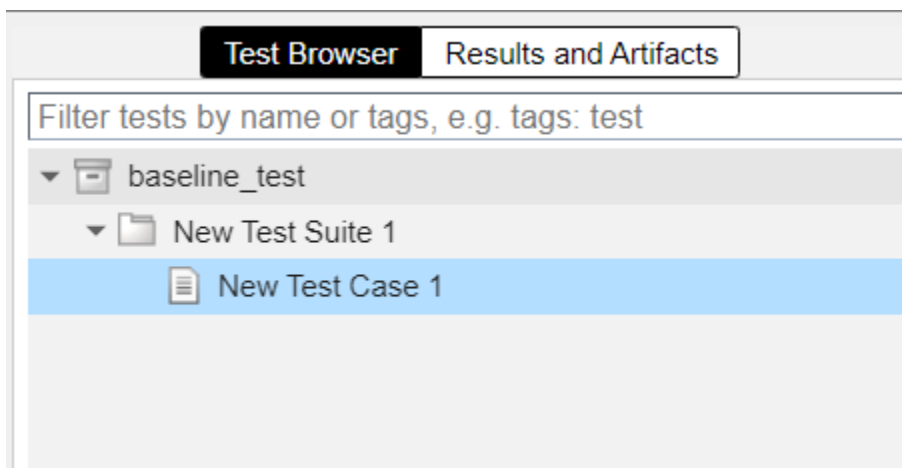




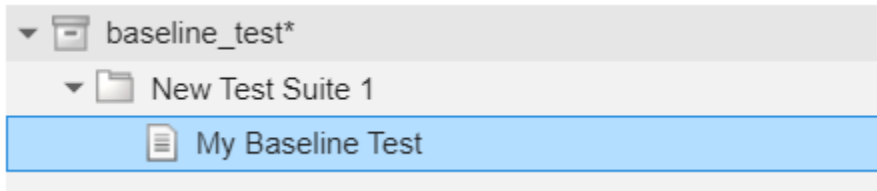
### Create the Test File and Test Structure

Open the Test Manager and create a test file and test file structure for your test. The default structure has one test suite, which has one test case. When you create the test file, it creates a new baseline test case by default. Test files can contain one or more test suites, which can contain one or more test cases.

1. In the **Harness** tab, click **Simulink Test Manager**.
2. In the Test Manager, click **New > Test File** to open the Save File dialog box.
3. Enter `baseline_test` as the name of the test file and click **OK**. The test file structure opens in the **Test Browser** pane.



4. Right-click **New Test Case 1** and select **Rename** from the menu. Name the test case My Baseline Test.

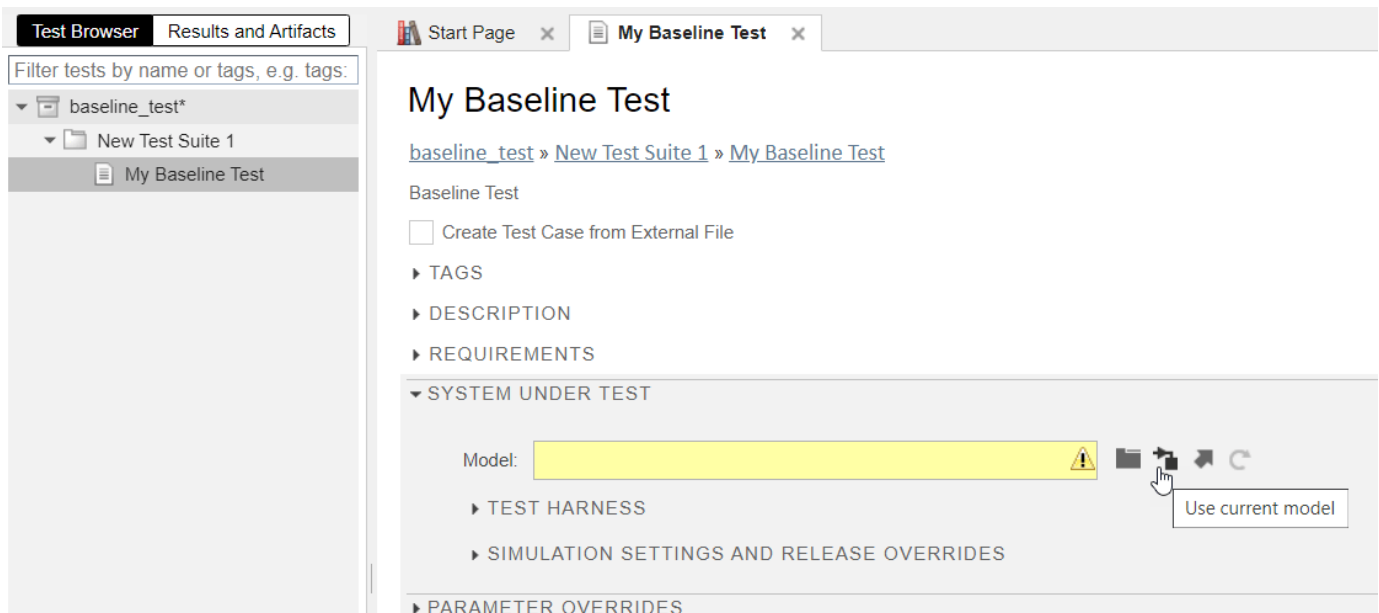


### Specify the Model and Test Harness

Set up the model and test harness to use for the test case.

1. Click **My Baseline Test** test case.

2. In the right pane, expand the **System Under Test** section. Click the Use current model icon next to the **Model** field to add `sltestBasicCruiseControlBaseline.slx` as the model to test.



3. Expand the **Test Harness** section and click the Update model harnesses icon next to the **Harness** drop-down menu to refresh the harness list.



4. Select `sltestBasicCruiseControlBaseline_Harness1` from the drop-down menu.

## Specify the Inputs

This example uses the inputs provided in the `inputs.mat` file.

1. Expand the **Inputs** section.
2. Under **External Inputs**, click **Add**.
3. In the Add Input dialog box, specify or browse for the `inputs.mat` file and click **Add**.
4. The Add Input dialog box expands to show additional options. Under **Input Mapping**, leave the **Mapping Mode** property as `Block Name` and click **Map Inputs**. The Add Input dialog box updates to show that the inputs were successfully mapped.

**Add Input** ? X

**INPUT FILE SPECIFICATION**

File:

Add iterations to run this input

▼ **INPUT MAPPING**

Mapping Mode:  ▼

Compile the system under test

▼ **MAPPING STATUS**

*Successfully mapped inputs.*

PORT	BLOCK NAME	MAPPED SIGNAL	STATUS
1	sltestBasicCruiseControlBaseline...	InBus	✓

▶ **ADVANCED**

5. Click **Add** to close the dialog box and add the `inputs.mat` file to the **External Inputs** table.

### Capture the Baseline Data

Obtain and save the baseline data.

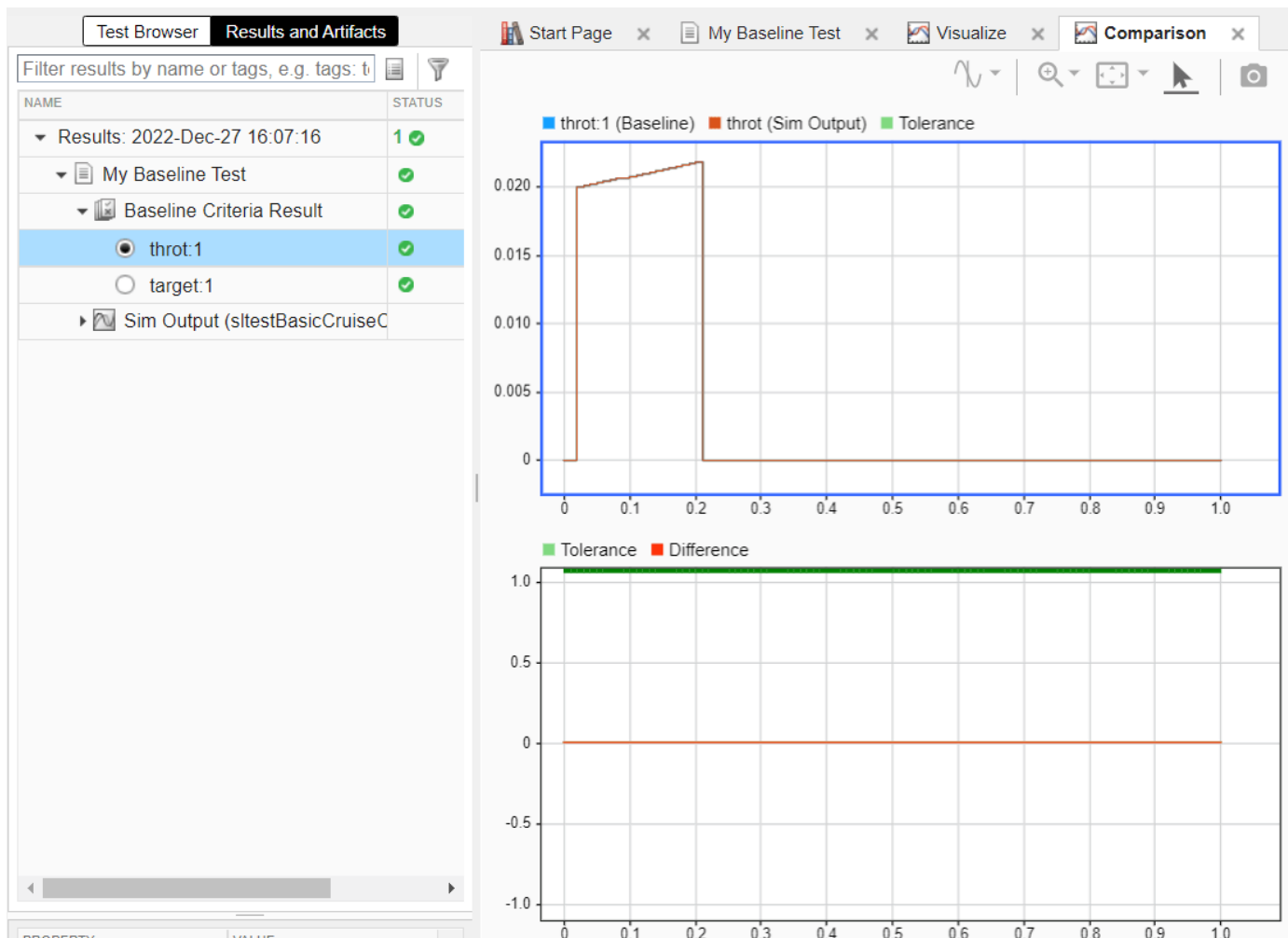
1. Expand the **Baseline Criteria** section.
2. Click **Capture** at the bottom of the section.
3. In the Capture Baseline dialog box, enter `baseline_data` as the file name. Then, click **Capture**. The model simulates and saves the baseline data.

### Run the Test and View the Results

Run the baseline test and display the throttle signal results.

1. In the Test Browser tab, ensure the **My Baseline Test** is selected. Click **Run** in the toolbar to run the baseline test case.
2. When the test finishes, the Test Manager opens the **Results and Artifacts** pane. In this example, the model did not change after the baseline was captured, so the baseline test passes.

In the Results and Artifacts pane, expand the results and select the **throt:1** signal.



The upper plot shows both the baseline data and the simulation output data, which match. The lower plot shows that the difference between the two data sets is zero for the entire run. The test results for the **target:1** signal also match and have zero difference.

### **Clean Up**

Clear the test results, close the Test Manager without saving, and close the model.

```
sltest.testmanager.clear  
sltest.testmanager.clearResults  
sltest.testmanager.close  
close_system('sltestBasicCruiseControlBaseline',0)
```

### **See Also**

#### **Related Examples**

- “Functional Testing for Verification” on page 2-2
- “Test Planning and Strategies” on page 3-2
- “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.

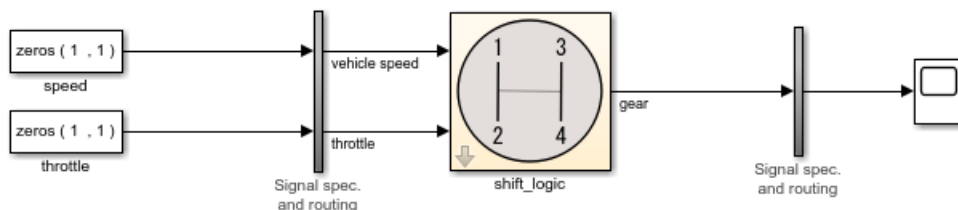
To assign a test harness to a test case, select **Test Harness** in the **System Under Test** section of the Test Manager.

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 directly from a model. The harness tests the `shift_logic` subsystem of the `sltestCarRootInport` model.

- 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.

The signal names used in the component under test propagate from the model to the test harness. For subsystem harnesses, some propagated signal names might be visible only after you compile the harness. For block diagram harnesses, signal names are propagated even if you do not select **Show propagated signals** in the Signal Properties dialog box.

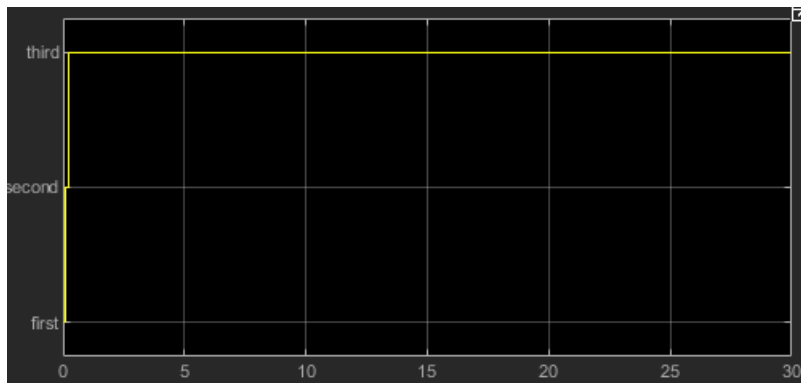
You can also create a harness, or multiple harnesses at the same time by using the `sltest.harness.create` function, the `sltest.testmanager.createTestForComponent` function, or the Create Test for Model Component wizard, which is available in the Test Manager. For information on the wizard, see “Generate Tests and Test Harnesses for a Model or Components”.

For information on test harness architecture, see “Test Harness Construction for Specific Model Elements”. For information on customizing the default harness settings when you create a new harness, see “Customize Test Harness Creation Default Property Values”.

## 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.




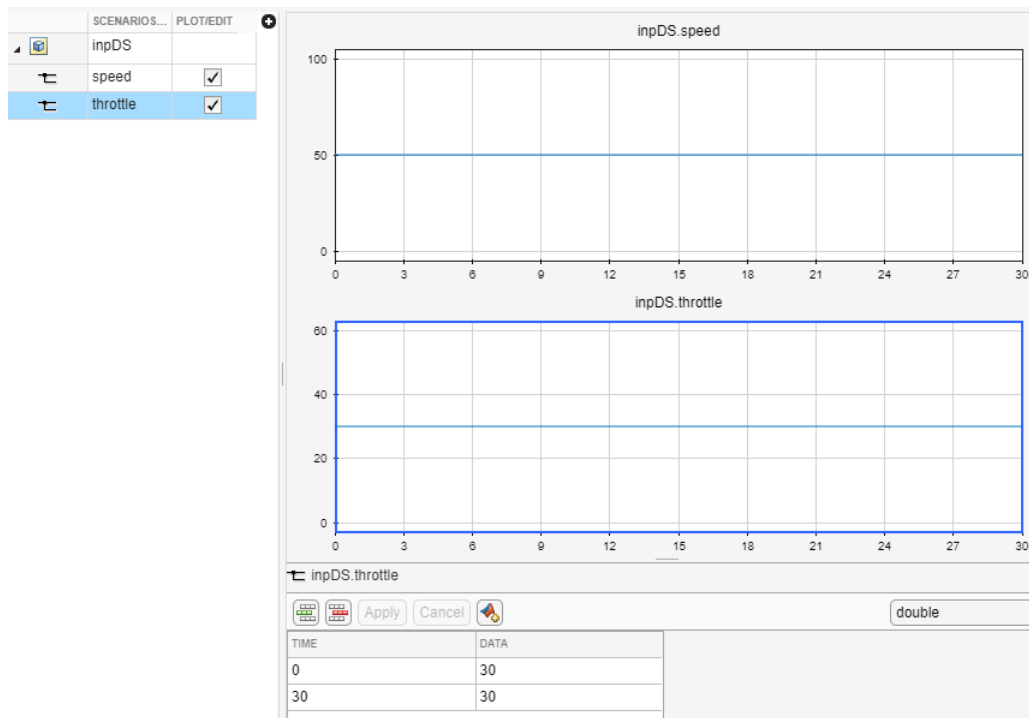
## Test Using the Test Manager

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.

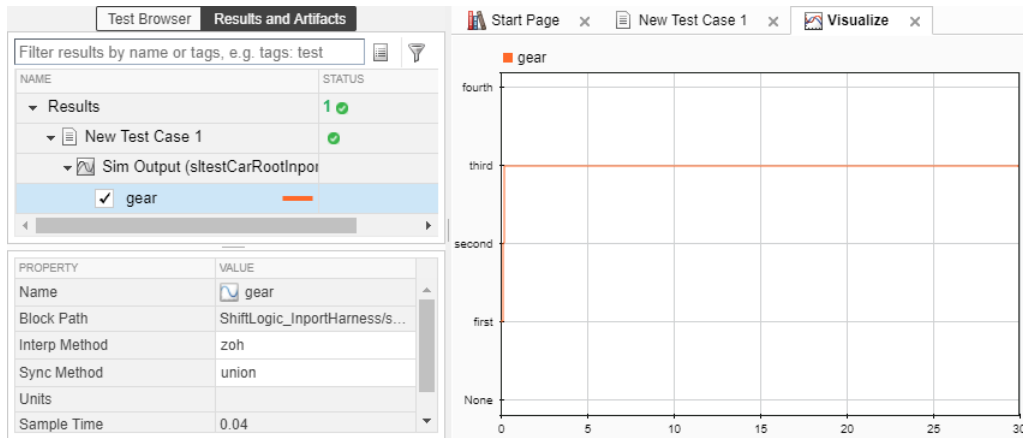
Using a test case in the Test Manager allows you to iterate with different test vectors, add test cases, run batches of test cases, and organize your results. This example shows you how to use external data with a test harness, and simulate from the Test Manager.

- 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. Press Enter to update the plot.
  - 2 Select the **throttle** signal and enter 30 for times 0 and 30. Press Enter 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

- "Functional Testing for Verification" on page 2-2
- "Test Harness and Model Relationship"
- "Create or Import Test Harnesses and Select Properties"
- "Create a Simple Baseline Test" on page 3-6
- "Refine, Test, and Debug a Subsystem"

