

SystemTest

For Use with MATLAB® and Simulink®

- Computation
- Visualization
- Modeling
- Simulation
- Testing

User's Guide

Version 1



How to Contact The MathWorks:



www.mathworks.com
comp.soft-sys.matlab
www.mathworks.com/contact_TS.html

Web
Newsgroup
Technical Support



suggest@mathworks.com
bugs@mathworks.com
doc@mathworks.com
service@mathworks.com
info@mathworks.com

Product enhancement suggestions
Bug reports
Documentation error reports
Order status, license renewals, passcodes
Sales, pricing, and general information



508-647-7000 (Phone)



508-647-7001 (Fax)



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

For contact information about worldwide offices, see the MathWorks Web site.

SystemTest User's Guide

© COPYRIGHT 2006 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, Simulink, Stateflow, Handle Graphics, Real-Time Workshop, and xPC TargetBox are registered trademarks of The MathWorks, Inc.

Other product or brand names are trademarks or registered trademarks of their respective holders.

Patents

The MathWorks products are protected by one or more U.S. patents. Please see www.mathworks.com/patents for more information.

Revision History

March 2006 Online Only New for Version 1.0 (Release 2006a+)

Getting Started With SystemTest

1

What Is SystemTest?	1-2
A Quick Tour of SystemTest	1-3
Getting Familiar with the Desktop	1-3
Setting SystemTest Preferences	1-5
Viewing Test Results	1-6
Building a Test: An Example	1-7
Planning Your Test	1-7
Building Your Test	1-8
Running Your Test	1-26
Analyzing Your Test Results	1-28

Working With the Basic Elements

2

Working With the Sections of a Test	2-2
Pre Test	2-2
Main Test	2-3
Post Test	2-3
The Basic Elements	2-4
MATLAB Element	2-4
Limit Check Element	2-5
IF Element	2-7
Vector Plot Element	2-8
Scalar Plot Element	2-10
Stop Element	2-12
Subsection Element	2-13

Using the Simulink Element

3

Before You Begin	3-2
Mapping Test Vectors and Test Variables to a Simulink Model	3-4
Model	3-4
Adding a Simulink Element	3-4
Specifying the Simulink Model	3-5
Overriding Simulink Model Inputs	3-6
Mapping Simulink Model Outputs to Test Variables	3-10
Using Simulink Model Coverage	3-18

Using the Instrument Control Toolbox Elements

4

Introduction	4-2
Instrument Control Toolbox Elements	4-2
Accessing Resources	4-2
Example: Measuring a Generator's Frequency	4-3
Setting Up the Signal Generator	4-4
Setting Up the Oscilloscope	4-7
Taking the Measurement	4-9
Saving Test Results	4-10
Running the Test and Viewing Test Results	4-11

Using the Data Acquisition Toolbox Elements

5

Introduction	5-2
Data Acquisition Toolbox Test Elements	5-2

Example: Testing a Voltage Regulator	5-3
Sending Analog Stimulus Data to the DUT	5-4
Enabling the DUT with Digital Data	5-6
Receiving Analog Response Data from the DUT	5-8
Disabling the DUT with Digital Data	5-9
Performing Data Analysis	5-10
Defining Post Test Elements	5-12
Saving and Viewing Test Results	5-13

Using the Image Acquisition Toolbox Element

6

Introduction	6-2
Example: Acquiring Video Data in a Test	6-3
Adding the Video Input Element to a Test	6-3
Saving and Viewing Test Results	6-7
Running the Test	6-8

Using the Test Results Viewer

7

Before You Begin	7-2
A Quick Tour of the Test Results Viewer	7-4
Viewing Your Test Results	7-6
Reserved Keywords	7-6
Browsing Results	7-6
Generating Plots	7-7
Exploring Plots	7-11
Refining Your Test Results	7-24
Creating and Applying Constraints	7-24
Plotting Single Iterations	7-30

Viewing Simulink Time Series Data	7-32
Creating a Time Series Plot	7-32

SystemTest Hot Keys

A

Index

Getting Started With SystemTest

This section explains what SystemTest is and shows you how to use it. It contains the following topics:

What Is SystemTest? (p. 1-2)	Introduces SystemTest and the kinds of tasks it can perform
A Quick Tour of SystemTest (p. 1-3)	Provides a quick tour of the SystemTest Desktop
Building a Test: An Example (p. 1-7)	Describes building and running tests with SystemTest

What Is SystemTest?

SystemTest provides MATLAB® and Simulink® users a framework that integrates software, hardware, simulation, and other types of testing in one environment. You use predefined elements to build test sections that simplify the development and maintenance of standard test routines. You can save and share tests throughout a development project to ensure standard and repeatable test verification. SystemTest offers integrated data management and analysis capabilities for creating and executing tests, and saving test results in order to enable continuous testing across the development process.

SystemTest automates testing in MATLAB and Simulink products. With SystemTest you get:

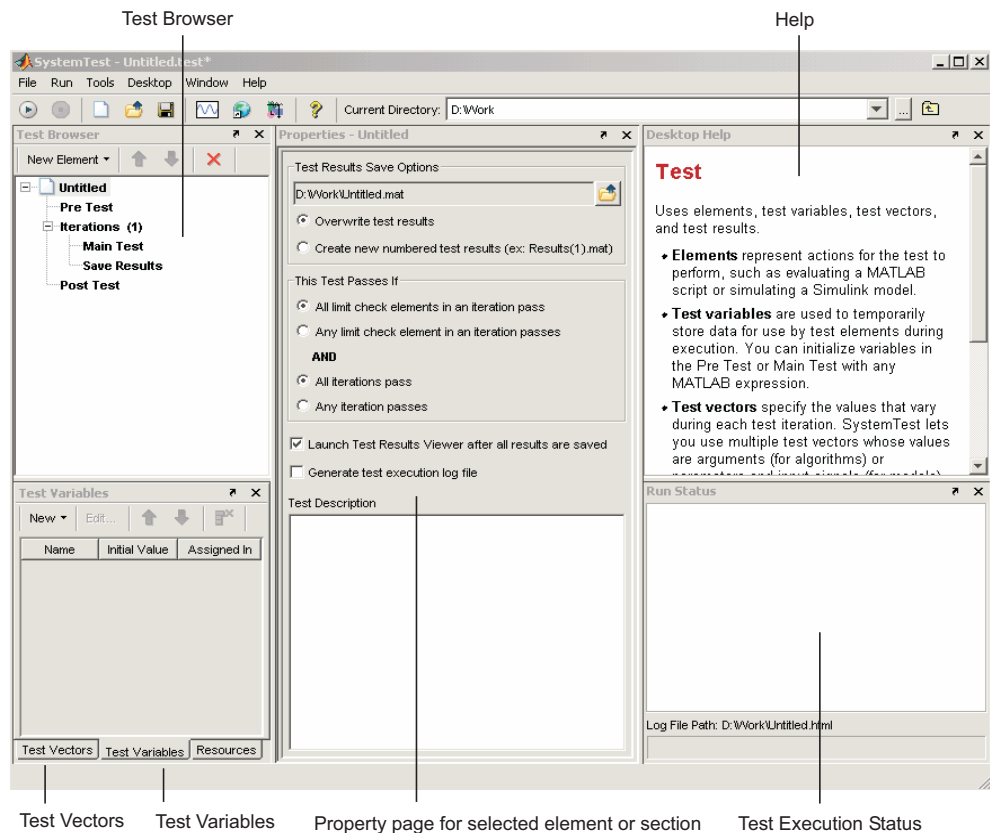
- Graphical test editing — Quickly edit your test within a graphical test development environment.
- Repeatable test execution — Tests developed with SystemTest all share the same execution flow, which provides a consistent test framework among tests.
- Parameterized testing — Create test vectors over which your test iterates.
- Reusability — After you design a test, you can save it for later use by you or others.
- Maintainability — Because you design and execute tests from within the SystemTest Desktop, you do not need to understand unfamiliar code or concepts.
- Integration — SystemTest integrates with MATLAB, Simulink, and other MATLAB and Simulink-based products.

A Quick Tour of SystemTest

The SystemTest Desktop is an integrated development environment that lets you perform all of your testing activities from one centralized location. This section provides a brief overview of the SystemTest environment. For more information about how to use SystemTest to build tests and run them, see “Building a Test: An Example” on page 1-7.

Getting Familiar with the Desktop

To get familiar with the SystemTest environment, open the SystemTest Desktop from MATLAB by selecting **Start > MATLAB > SystemTest > SystemTest Desktop** or typing `systemtest` at the MATLAB command line.



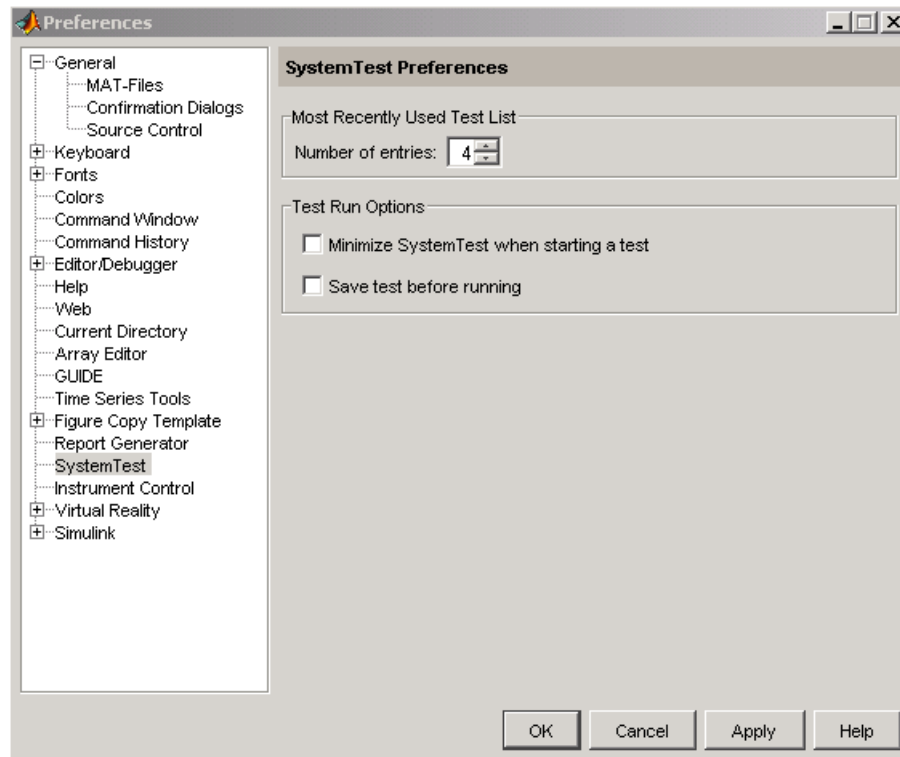
The desktop has a number of different panes that aide you in building and running your test.

- **Test Browser** — Shows the overall structure of a test. A test is made up of Pre Test, Iterations, Main Test, Save Results, and Post Test. Use the Test Browser to add elements to your test. These elements determine what actions your test performs.
- **Test Vectors** — Lets you define the parameters or test cases of your test. The test vectors you define determine the number of iterations performed by your test. Test vectors are automatically indexed during test execution.
- **Test Variables** — Lets you define temporary variables used by your test. Variables can serve both input and output functions in your test. You can define variables that are declared in the Pre Test section of your test or in the Main Test section of your test.
- **Properties** — Shows the properties of the test or the element you are editing. The contents of this pane change when you select a section or element in your test.
- **Desktop Help** — Shows help about the element or aspect of the test that is currently selected. For the full product Help, click **Help > SystemTest User's Guide**.
- **Run Status** — Shows a summary of the test's execution status.

Note Many areas of the user interface have shortcut menus. For example, if you right-click in the **Test Vectors**, **Test Variables**, **Run Status**, or **Desktop Help** panes, you can access these shortcut menus.

Setting SystemTest Preferences

You can set SystemTest preferences by selecting **File > Preferences...** on the SystemTest Desktop. This opens the MATLAB Preferences dialog box. Click **SystemTest** in the left pane if SystemTest Preferences are not showing in the right pane.



Most Recently Used Test List

This option determines how many tests will appear on the SystemTest **File** menu's most recent files list. The default is 4 tests. If you change it to 0, no recent tests will appear on the list. The maximum number is 9.

Test Run Options

Select **Minimize SystemTest when starting a test** if you want the SystemTest Desktop to minimize when a test starts running. This check box is cleared by default.

Select **Save test before running** if you want SystemTest to save your test before it runs. If this option is selected and you run a test that is not yet saved, you will be prompted to name and save the test. This check box is cleared by default.

Note You can save a test any time, before or after running it, by selecting **File > Save**.

Viewing Test Results

SystemTest includes a tool called the Test Results Viewer that you can use to view the results you have chosen to save for your test. You can launch the tool from the SystemTest **Tools** menu by selecting **Tools > Test Results Viewer**. You can also configure the SystemTest environment to launch the Test Results Viewer automatically after all test results you specified have been saved for each iteration and test execution has completed. For more information, see “Analyzing Your Test Results” on page 1-28.

Building a Test: An Example

This section builds a simple example to illustrate the four primary stages of testing: planning, building, and running the test, and viewing test results.

The example uses a simple MATLAB expression to emulate a scalar measurement during each iteration of the test. The example uses an arbitrary formula dependent on the test vector named `signal` to generate the Y data. The example tests each measurement to determine if it falls within certain specified limits. If a measurement exceeds these limits, that particular iteration of the test fails. By default, the test fails if any iteration fails, but you can configure other pass/fail criteria.

The following sections provide more information about each stage, building the example test along the way. If you prefer, instead of working through the following sections to build the example, you can load it into SystemTest by running the “MATLAB – Getting Started with SystemTest” demo from the **Demos** page in the MATLAB Help browser (under **MATLAB > SystemTest > General**) or by entering `systemtest Simple_Demo` at the MATLAB command prompt.

- “Planning Your Test” on page 1-7
- “Building Your Test” on page 1-8
- “Running Your Test” on page 1-26
- “Analyzing Your Test Results” on page 1-28

Planning Your Test

In this first stage, you must identify what it is you want to test. SystemTest lets you specify input data, such as measurements from a model or device, and compare this input data to some predefined limits. Based on this comparison, SystemTest can declare whether a test passes or fails. Keep the following in mind as you plan tests:

- Identify your test data and test vectors.
- Specify test limits and determine if these limits can be expressed as scalar values. (The Limit Check element operates only on scalar data.)

- Determine what operations your test must perform. Must certain operations happen first or exist for others to follow? Which test vectors should be indexed first?
- Determine pass/fail criteria for your test.
- Decide which test variables you want to save as test results.

After this planning, you can begin to construct your test, which is described in “Building Your Test” on page 1-8.

Building Your Test

SystemTest provides a graphical integrated environment that you can use to create and edit tests. Tests consist of elements, test vectors, and test variables. You can use each of these entities to create a variety of test scenarios ranging from a simple test that runs a series of elements once to a full parameter sweep that iterates over the values of test vectors that you define.

The following sections show how to construct a test.

- “Starting SystemTest” on page 1-8
- “Structuring Your Test” on page 1-9
- “Creating a Test Vector” on page 1-10
- “Defining Test Variables” on page 1-12
- “Adding Elements” on page 1-14
- “Defining Pass/Fail Criteria” on page 1-22
- “Saving Test Results” on page 1-23
- “Test Execution Log” on page 1-24
- “Saving Your Test” on page 1-25

Starting SystemTest

Start by opening the SystemTest Desktop using the MATLAB **Start** button. To open SystemTest, select **Start > MATLAB > SystemTest > SystemTest Desktop**.

Alternatively, you can execute the `systemtest` command from the MATLAB command line.

SystemTest displays the desktop on your screen. See “A Quick Tour of SystemTest” on page 1-3 for an overview.

Structuring Your Test

SystemTest divides tests into three *sections*.

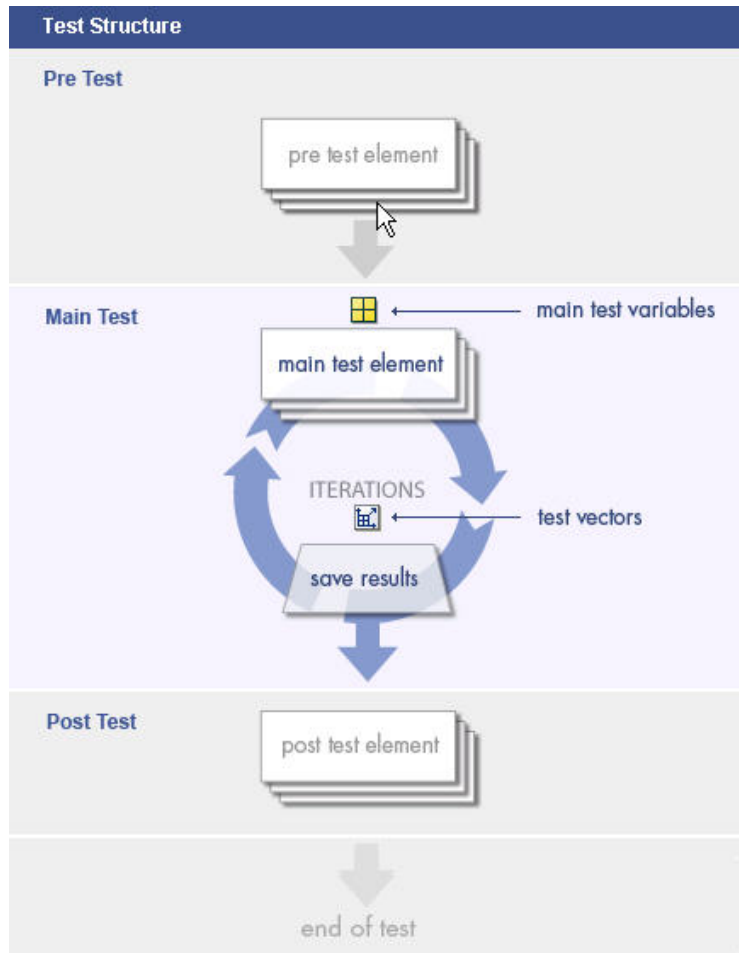
- **Pre Test** — This section is used to execute test elements in order to perform any test set-up operations, such as initializing variables, loading data from a file, and initializing system resources. Using Pre Test variables, you can assign an initial value to a test variable that persists between Main Test section iterations (unless another element in Main Test modifies the value). Pre Test is not mandatory, but it can be used if your test requires set-up operations to be performed.
- **Main Test** — Main Test defines the test elements that need to be performed across the parameter space defined by your test vectors. In this section Main Test variables are initialized before each Main Test iteration, which lets you assign an initial value to a test variable each time the Main Test runs. This is useful if your test variable has a derived value such as being indexed by a test vector or is the result of a MATLAB expression. You add elements in this section.

In the SystemTest desktop, Main Test is provided as part of **Iterations**. Iterations specifies the number of times the Main Test section will be run. This is determined from the test vectors you define. The SystemTest desktop also offers a **Save Results** area for you to specify which test variables you want to save as test results at the end of each Main Test iteration.

- **Post Test** — In this section you can perform any cleanup work necessary at the completion of the Main Test section, such as clearing workspace variables, closing a file, or freeing system resources.

For more detailed information about the sections of the test, see “Working With the Sections of a Test” on page 2-2.

The following figure illustrates the structure of a test in SystemTest.



Creating a Test Vector

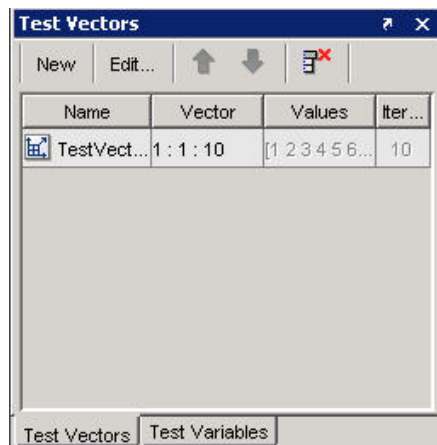
Test vectors are composed of values derived from a MATLAB expression. You can use any MATLAB expression that evaluates to a 1-by-N matrix or cell array to define your test vector. Using test vectors, you can iterate through a range of values to see how a system performs. Test vectors constitute parameterized testing in SystemTest. They are the test cases for your test.

For tests with multiple test vectors, the product of the lengths of the test vectors defines the number of iterations the test performs. For example, if you define the test vector [10 20 30], the test runs three times, using a value of 10 for the first run, 20 for the second, and 30 for the final run. If you add a second test vector with three other values, the total number of test runs would be nine. SystemTest iterates through each vector in combination with the other vector as though the test were a group of nested FOR loops—the outermost loop being the first test vector in your table and the innermost loop being the last test vector. The **Iterations** section in the **Test Browser** shows the total number of test iterations defined by your test vectors.

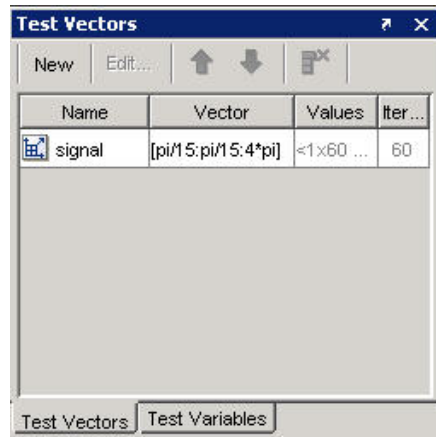
For the example, use the vector [pi/15:pi/15:4*pi] which defines 60 values for our test vector ranging from pi/15 to 4*pi in pi/15 increments. To specify this test vector in SystemTest:

- 1 Click the **New** button in the **Test Vectors** pane.

SystemTest adds a new test vector with default values to the **Test Vectors** pane.



- 2 Assign a name to the test vector by clicking the **Name** field. For this example, name the test vector signal.
- 3 Assign a value to the test vector by clicking the **Vector** field. Enter the test vector specified above for the pi values.



After you create the test vector, in the **Test Browser** pane, the **Iterations** section label updates to include the number of iterations defined by the test vector. It should say Iterations (60).

Defining Test Variables

SystemTest uses *test variables* to define temporary storage variables that a test acts on or generates. You assign test variables in the Pre Test or Main Test sections of your test.

You can define Pre Test variables or Main Test variables. Using Pre Test variables, you can assign an initial value to a test variable that persists between Main Test section iterations (unless another element in Main Test modifies the value). Pre Test is not mandatory, but it can be used if your test requires set-up operations to be performed.

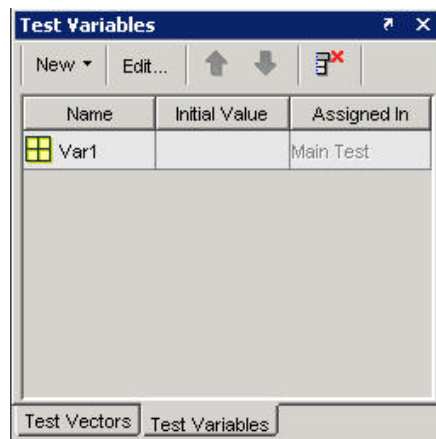
Main Test defines the test elements that need to be performed across the parameter space defined by your test vectors. Main Test variables are initialized before each Main Test iteration, which allows you to assign an initial value to a test variable each time the Main Test runs. This is useful if your test variable has a derived value such as being indexed by a test vector or is the result of a MATLAB expression. You add elements in this section.

The example test requires three test variables:

- **Y** — Contains a value that will be calculated from the signal test vector at each iteration
- **HiLimit** — Contains the upper limit for Y that you do not want the signal to exceed
- **LowLimit** — Contains the lower limit for Y that you do not want the signal to go below

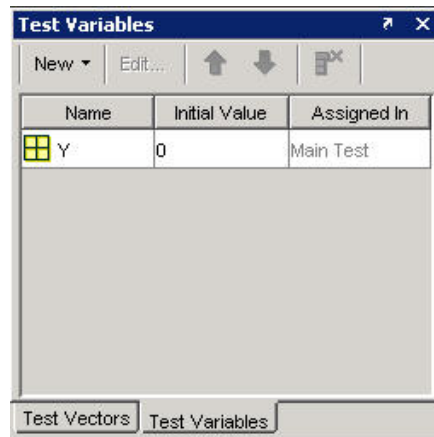
To create these test variables in SystemTest:

- 1 Click the **Test Variables** tab at the bottom of the SystemTest Desktop to access the **Test Variables** pane.
- 2 Click the **New** button to create a Pre Test or Main Test variable. Select **Main Test Variable**. SystemTest adds a new test variable, named Var1, to the **Test Variables** pane.



- 3 Assign a name to the test variable by clicking the **Name** field and entering the test variable name. For this example, enter Y.
- 4 Set the test variable's initial value by clicking the **Initial Value** field and entering a value. For the example test variable Y, enter 0.

Note If you do not provide an initial value, it will default to empty, that is `Var1 = [];` in MATLAB code.



5 Repeat steps 2 through 4 to create the remaining two test variables, using the settings listed in the following table:

Variable Name	Initial Value	Assign in
HiLimit	1	Main Test
LowLimit	-1	Main Test

Adding Elements

Elements are the actions that a test performs. SystemTest includes the following set of elements, listed in alphabetical order.

- IF — Implements a logic control operator
- Limit Check — Specifies the comparison to be performed of the value under test and the limit (or limits)
- MATLAB — Executes any MATLAB statements
- Plot — Graphically shows the value of any test variable or vector, as the test is executing, as either scalar or vector plot

- Simulink — Runs a Simulink model. Note that you need to have a license for Simulink to use this element.
- Stop — Implements a logic control operator
- Subsection — Creates a new section in a test that you can use to group elements within

Note Some MathWorks products, such as the Image Acquisition Toolbox, the Data Acquisition Toolbox, and the Instrument Control Toolbox, provide their own elements that integrate those products' capabilities within SystemTest. If you have licenses for those products, those elements will also appear in the elements list.

For more information about using the basic elements, see Chapter 2, “Working With the Basic Elements”.

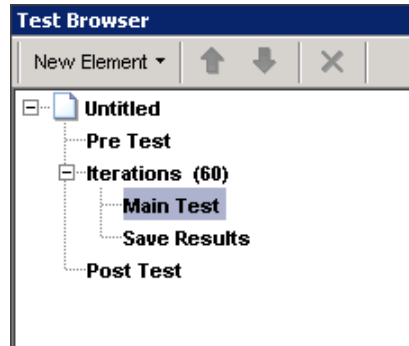
You add elements to a section in your test; however, not all elements are available in all sections. You can, for example, use a MATLAB element anywhere within a test, but you can only use the Limit Check element in the Main Test section.

To illustrate using elements, let's continue with our example. This test uses three elements in the Main Test section:

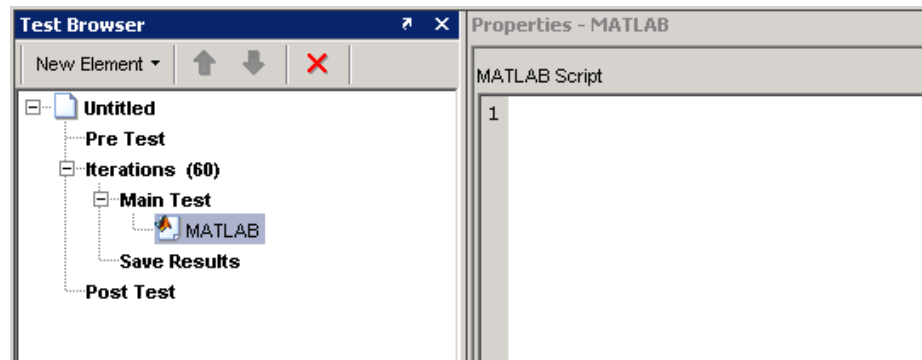
Element	Description
MATLAB	Use a MATLAB expression to assign data to Y that is dependent on the test vector signal.
Limit Check	Compare the value generated in the MATLAB element to the specified limit and see if the Y test variable exceeds the upper or lower limit you defined in your HiLimit and LowLimit test variables.
Scalar Plot	Plot the current test variable values and see whether the test variable exceeds the upper and lower limits.

To add these elements in SystemTest:

- 1 Select the section of the test in which you want to add the element. For this example, click **Main Test** in the **Test Browser**.

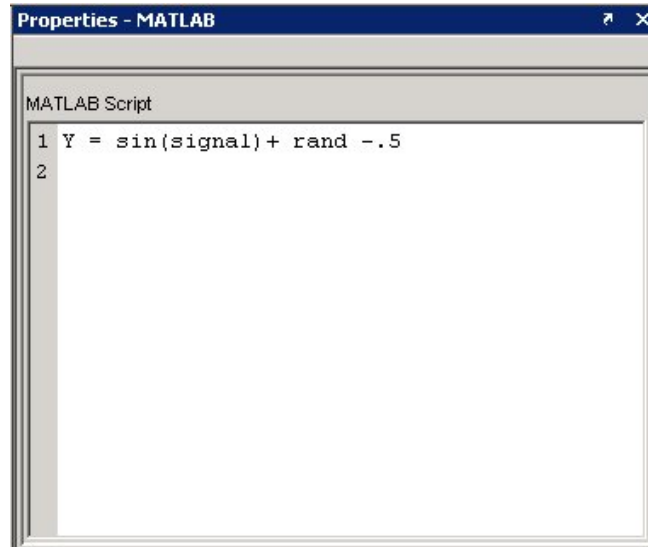


- 2 Specify the element you want to add to the test section. For this example, click the **New Element** button and select **MATLAB**. SystemTest adds a MATLAB element to the Main Test section of your test and opens the MATLAB element property page in the **Properties** pane of the SystemTest desktop.



- 3 In the **Properties** pane, type the following M-code in the MATLAB Script edit box. This MATLAB code calculates a value for Y that is dependent on the test vector signal.

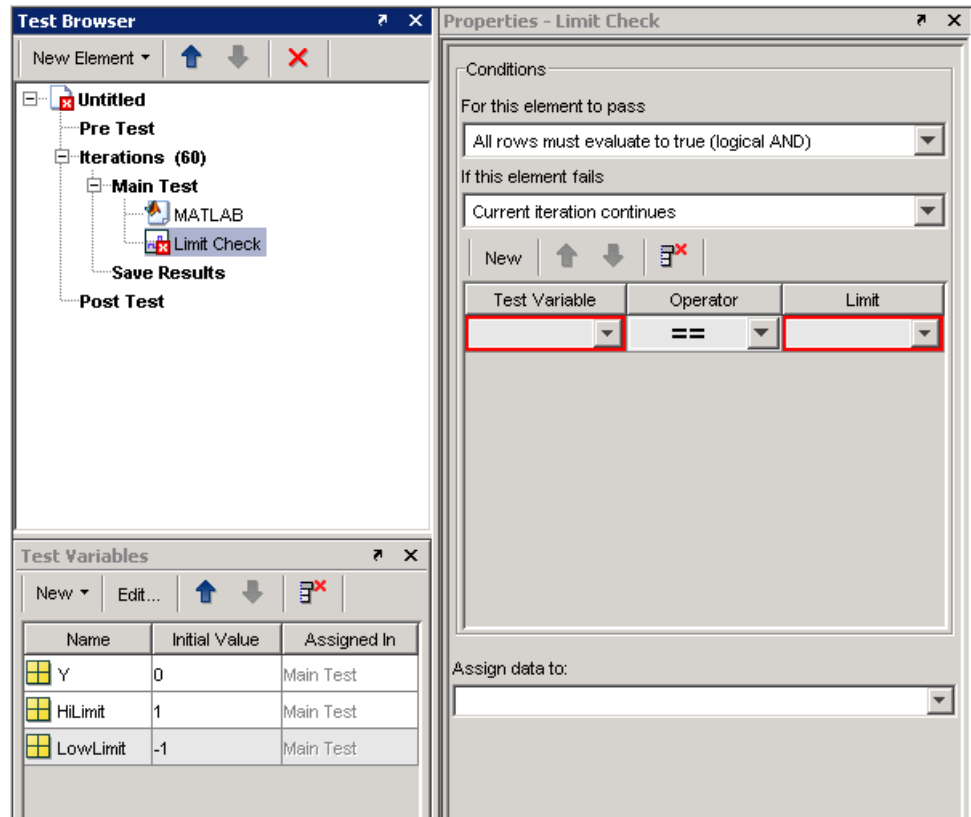
```
Y = sin(signal)+ rand -.5
```



For each iteration, SystemTest evaluates the MATLAB expression and assigns a value to Y.

- 4 Add the Limit Check element to the **Main Test** section of the test. With the MATLAB element selected, click the **New Element** button, and click **Limit Check**. SystemTest adds a Limit Check element to the **Main Test** section of the test and opens the Limit Check properties page in the **Properties** pane. For this example, the Limit Check element must follow the MATLAB element in the test.

Note You can reposition an element in a test by selecting the element and then clicking the up and down arrows in the **Test Browser** toolbar. You can also drag and drop elements within **Main Test**. You cannot move elements between test sections.



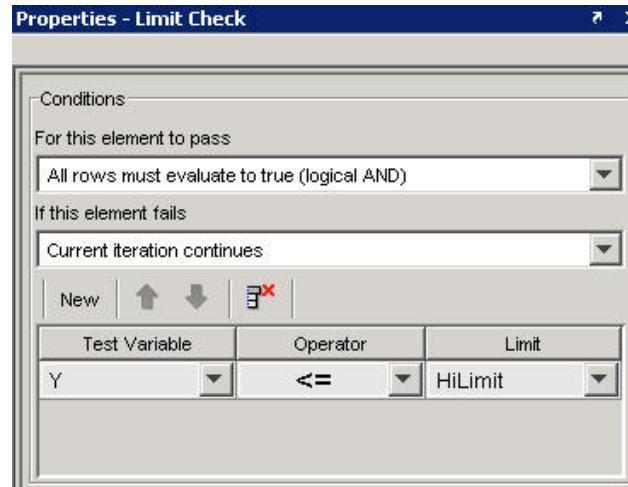
Notice that the Limit Check element icon in the **Test Browser** shows a red x, which indicates that information is missing. The corresponding red outlining in the **Properties** pane highlights any fields that require configuration.

5 Specify the limit comparison operations in the Limit Check element.

- a** In the **Test Variable** column, click the drop-down list and select a test variable you created in step 4. For this example, select Y.
- b** In the **Operator** column, click the drop-down list and select the comparison you want to perform. For this example, pick the less-than-or-equal-to operator, <=.

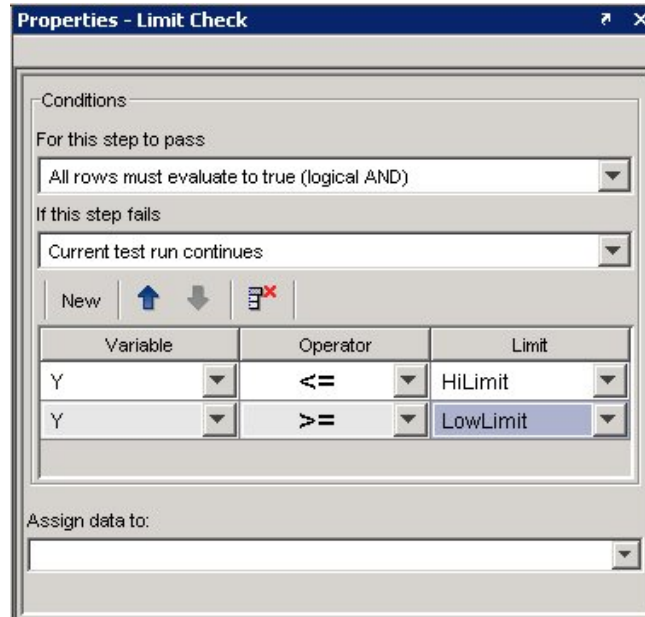
- c In the **Limit** column, click the drop-down list and select the test variable you want to compare to. For this example, select HiLimit, which is the test variable you created earlier.

The following figure shows the configuration of this limit:



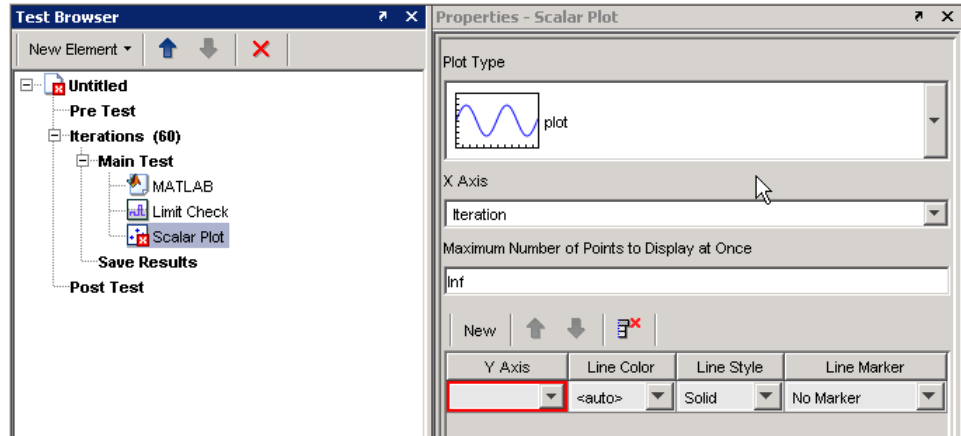
- 6 To add another limit comparison operation, click the **New** button in the **Properties** pane. SystemTest adds a new row below the last limit you specified. In this new row, set **Test Variable** to Y, set **Operator** to >=, and set **Limit** to LowLimit.

The following figure shows the configuration of this second limit:



For each iteration of the Main Test, the MATLAB element's expression is evaluated and a new value assigned to Y. When the Limit Check element runs, it determines whether the value of Y falls between the HiLimit and LowLimit values. If Y is outside this range, the test iteration fails. The default pass/fail criteria for the overall test passes the test only if both Y variable evaluations succeed.

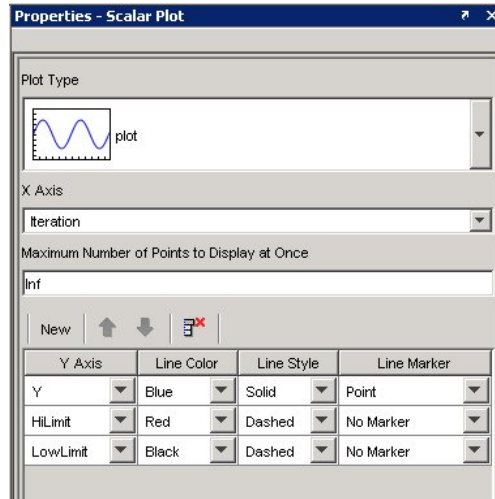
- 7 To view the test variables as the test runs, plot the data. To add a Plot element to the test, click the **New Element** button, and select **Plot > Scalar Plot**. SystemTest adds a Scalar Plot element to the Main Test section, and opens the property page for the element in the **Properties** pane.



With each Main Test iteration of the test, the Scalar Plot element updates a figure window with data you selected.

- 8 Click the **New** button twice in the **Properties** pane and set the three rows to match the following table:

Y Axis	Line Color	Line Style	Line Marker
<i>Y</i>	Blue	Solid	Point
<i>HiLimit</i>	Red	Dashed	No Marker
<i>LowLimit</i>	Black	Dashed	No Marker



Defining Pass/Fail Criteria

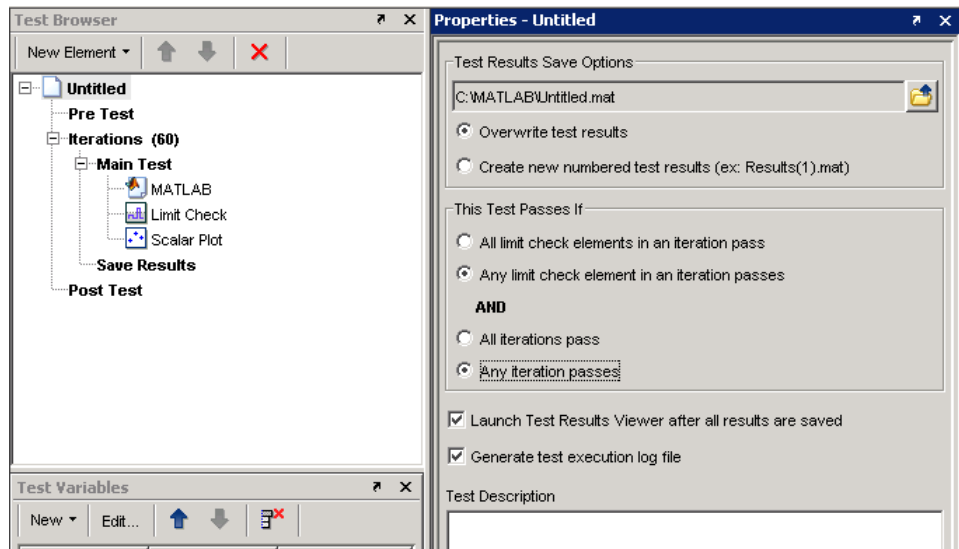
You can define whether your test passes or fails by making SystemTest monitor the outcome of any or all Limit Check elements during any or all Main Test iterations. Your test's threshold of success can range from the passing of any Limit Check in any single test iteration to the passing of all Limit Check elements in all test iterations. If your test contains no Limit Check elements, there is no notion of pass/fail and no pass/fail information is displayed. (Testing of this type is useful for experimenting with a system or to explore its behavior rather than validate its performance.)

You can set any of the following conditions to define when your test passes:

- All Limit Check elements pass in all test iterations.
- All Limit Check elements pass in any test iteration.
- Any Limit Check element passes in all test iterations.
- Any Limit Check element passes in any test iteration.

You can configure this behavior within the test's **Properties** pane. Click the test name in the **Test Browser** (named **Untitled** by default) to open the test's properties and look for the section labeled **This Test Passes If**.

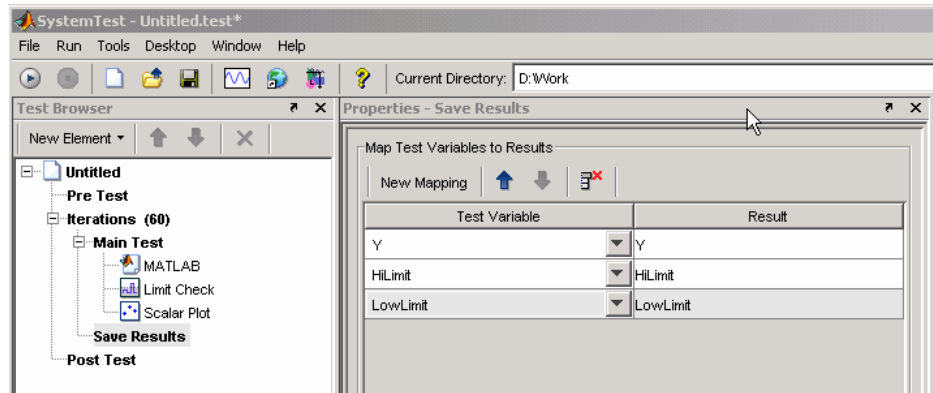
Using the signal test example that you constructed in this section, set the test to pass if all Limit Check elements pass in all test iterations.



Saving Test Results

SystemTest saves the results from the iterations of your test in a MAT-file. You must explicitly specify which test variables you want to save as test results.

SystemTest gives you the ability to save results at the end of each iteration. Before you run your test, select the **Save Results** section in your test and specify which test variables you want to save as test results. Click the **New Mapping** button and then select from the drop-down list the name of the test variable you want to map to a result. You can optionally specify a name for the results that you want to save. The following figure shows the mapping of test variables to test results:



After you specify which test variables you want to save as test results, you can specify the name of the MAT-file that SystemTest uses. Using this MAT-file you can reload the test results into the base workspace. By default, SystemTest names the file `Untitled_results.mat` and puts the file in the current working directory (visible in the SystemTest toolbar). To change the name or location of the MAT-file, click on the test name in the **Test Browser**, then in the **Properties** pane, use the **Test Results Save Options** field.

By default, each time you run the test you overwrite this file unless you select the **Create new numbered test results** option on the test **Properties** pane.

Note Test variables that are not saved as a test result will be lost at the end of the test execution.

Test Execution Log

When you run your test, SystemTest displays the status of the test in the **Run Status** pane. This display contains basic information about your test:

- When your test started running
- Which section your test is in
- How many test iterations have passed or failed as defined by any limit checks

You can make SystemTest generate and save more detail about the running test by enabling the test execution log file. This log is useful when you use limit checks in your test and you want to see specific test iterations that passed or failed. For example, instead of finding that a test iteration failed, the log shows how far a test variable varied from the upper or lower limit defined in a Limit Check element.

Note Because the test execution log renders HTML during the test, this option results in the test taking longer to execute.

To enable the test execution log, check the **Generate test execution log file** option on the **Properties** pane.

By default, SystemTest names the test execution log file `Untitled_results.html` and overwrites the file each time the test is run. The options you set for your test results MAT-file also apply to your test execution log file: the filename, location, and write options. See “Saving Test Results” on page 1-23 to learn how to change these options.

See “Viewing the Test Execution Log” on page 1-28 to see what information the test execution log generates.

Saving Your Test

SystemTest lets you save tests so that you can reuse them later. For example, to save the signal test:

- 1 Select **File > Save As** to open the Save file as dialog box.
- 2 Select a directory location and enter `mySavedTest` in the **File name** field.
- 3 Click **Save**.

SystemTest saves the test as `mySavedTest.test` and renames your test as it appears in the **Test Browser**. This does not rename the test results MAT-file or the test execution log file. Their names are controlled separately from the name of the test, as explained in “Saving Test Results” on page 1-23.

Running Your Test

After you build a test, you are ready to run it. At run time, SystemTest assigns values to test vectors and test variables in the order they appear in the **Test Vectors** and **Test Variables** panes. Each test section runs elements in the order that they appear in the **Test Browser**.

To execute your test, do one of the following:

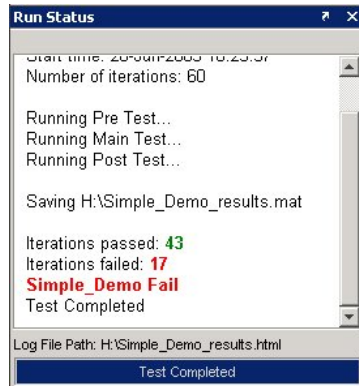
- Click the **Run** button.
- Select the **Run > Run** menu.
- Press the **F5** key.



Note While a test is running, you can stop its execution by pressing **Ctrl+C** or clicking the **Stop** button on the toolbar.

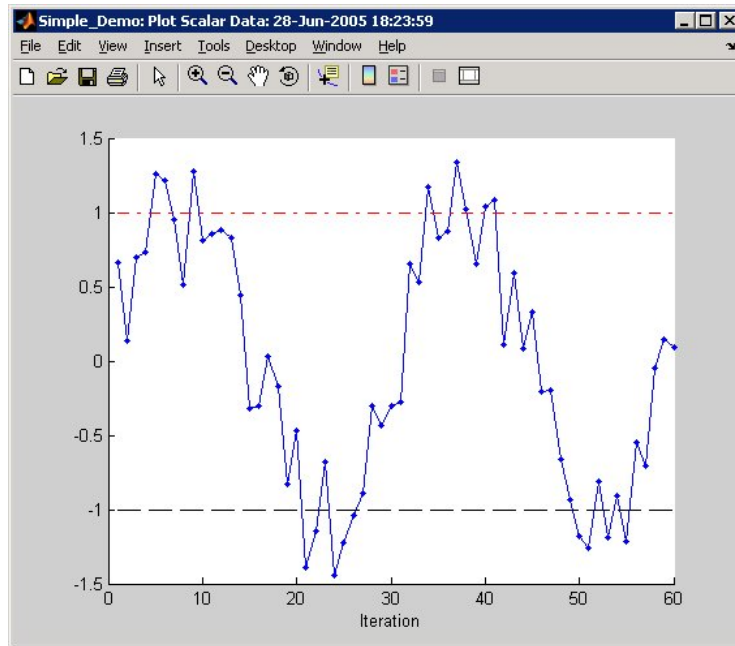
Tracking Output

While the test runs, the **Run Status** pane shows summary test output.



If your test includes a Plot element, SystemTest creates the plot and updates the plot during each iteration. Since Limit Check elements evaluate whether an iteration passed or failed, they directly affect the data that appears in the test execution log and the **Run Status** pane.

In the example test, the plot includes the high and low limits defined in the Limit Check element, to show which test iterations exceed the limits.



Analyzing Your Test Results

After SystemTest runs your test, you can explore the results that SystemTest generated. This section shows how to:

- View and interpret the test execution log.
- Inspect your test results with the Test Results Viewer.

Viewing the Test Execution Log

When you enable the test execution log, SystemTest saves information about each test iteration in an HTML file. You can see the contents of this file by clicking the **View Test Execution Log** button on the SystemTest toolbar. The generated output resembles the following:

C:/MATLAB/Untitled.html

File Edit View Go Debug Desktop Window Help

Location: C:/MATLAB/Untitled.html

Name: mySavedTest
Start time: 28-Feb-2006 14:34:17
Number of iterations: 60

Pre Test

Main Test 1 of 60

signal = 2.094395e-001

Limit Check					
Variable Name	Variable Value	Operator	Limit Value	Limit Name	Evaluates To
Y	-0.06095	<=	1	HiLimit	TRUE
Y	-0.06095	>=	-1	LowLimit	TRUE

Iteration 1 Passed

Main Test 2 of 60

signal = 4.188790e-001

Limit Check					
Variable Name	Variable Value	Operator	Limit Value	Limit Name	Evaluates To
Y	0.51358	<=	1	HiLimit	TRUE
Y	0.51358	>=	-1	LowLimit	TRUE

Iteration 2 Passed

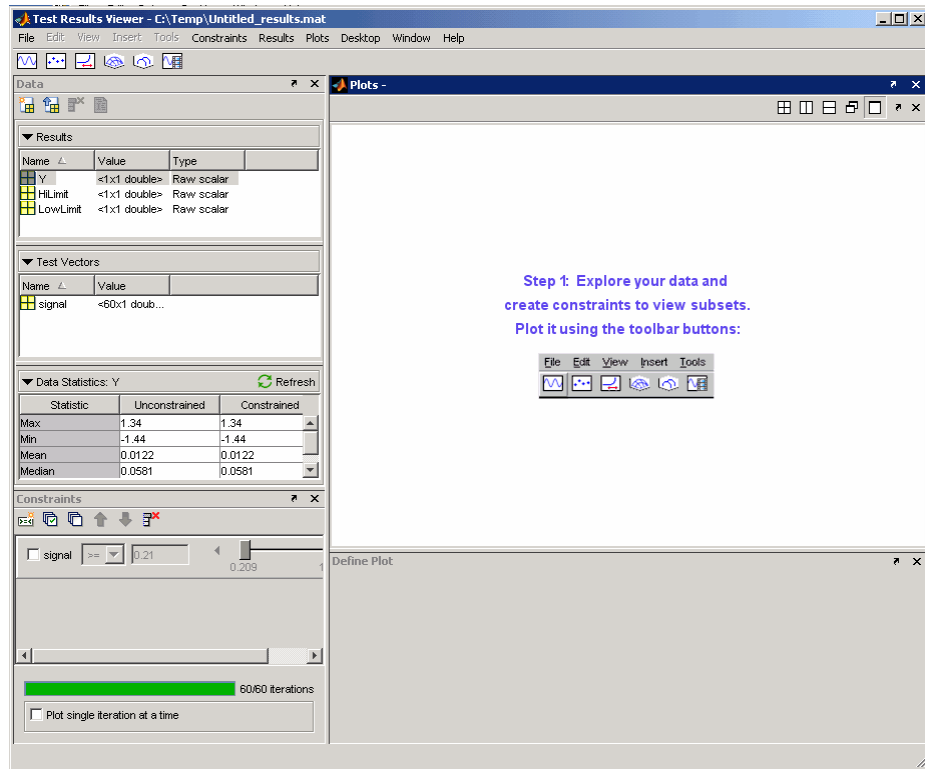
The Main Test section of the test execution log shows each iteration. You see the value of the test vector `signal` and determine the values the Limit Check element used in evaluating whether the test passed. For the first several iterations, the value of `Y` did not exceed either the high or low limits so the iterations passed. You can also see this in the scalar plot drawn while the test ran. For other iterations that failed, you can scroll through the test execution log to find the values of `Y`.

Viewing Test Results in the Test Results Viewer

To help you analyze your test results, SystemTest includes a tool, called the Test Results Viewer, that provides a variety of plotting tools and the ability to compare data. With this tool, you can see how your test results compare to the test vectors used as inputs to your test.

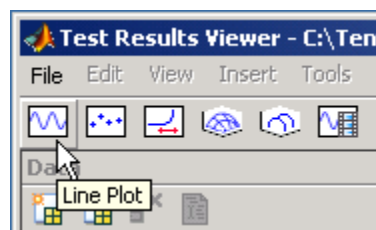
You can start the Test Results Viewer by selecting it from the **Tools** menu. You can also configure a test to launch the Test Results Viewer automatically when the test completes. To do this, select the test in the **Test Browser** and select the **Launch Test Results Viewer after all results are saved** option on the test's **Properties** pane.

The test vectors your test used and the test results you selected to be saved appear within the viewer so you can immediately start to explore your data. For any selected test vector or test result, you can see a summary of statistics for its values in the **Data** pane. For example, after running the test, the viewer opens showing the saved test results and signal test vector. Clicking the Y test result shows information such as the highest and lowest values that Y evaluated to during the test. It also shows the mean, median, and standard deviation for all values.



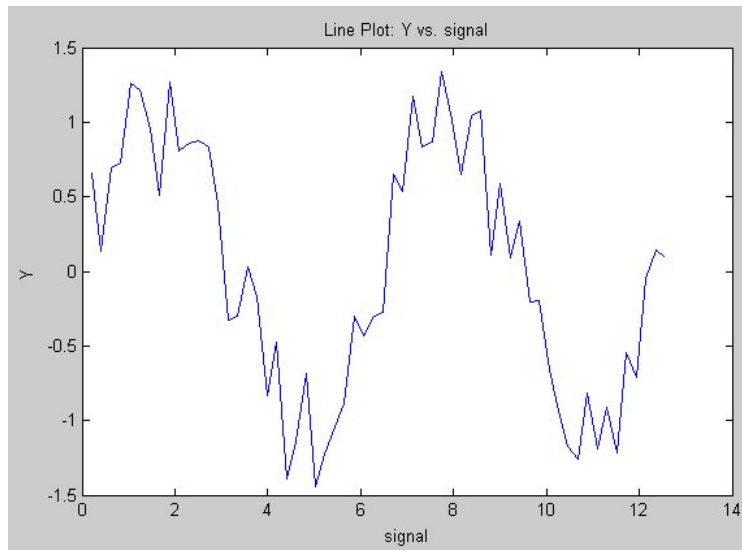
The viewer has a rich selection of plotting capabilities that you can use to visualize your test results. Plotting capabilities include line, scatter, time series, surf, waterfall, and image plots. Using the signal example, you can reproduce the line plot that your scalar plot element generated during the test.

1 Click the **Line Plot** button.



- 2 In the **Define Plot** pane, click the **X Axis** list and select signal. Note that you should choose signal or Auto values for the X-axis if you want to show test vectors; line plots that use the X-axis for test vectors let you see how each test iteration value corresponds to a test result.
- 3 Click the **Y Axis** list and select Y.
- 4 Click the **Plot** button.

You now have a line plot that resembles the scalar plot.



You can make this plot show the HiLimit and LowLimit test results too.

- 1 In the **Define Plot** pane, click the **Multiple Y Data** option. The **Y Axis** field expands to show all saved test results from your test.
- 2 Select the check boxes next to HiLimit and LowLimit.
- 3 Click the **Refresh Plot** button.

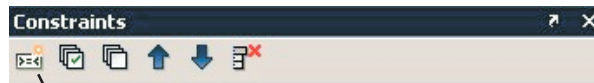
Constraining Data for Further Analysis

Using the Test Results Viewer constraint capability, you can filter out data. For example to see only the test iterations that passed the test, you can create two constraints that screen out data that exceeds the upper and lower limits.

Note By default, the Test Results Viewer provides a list of test vector constraints for you to choose from.

To create and configure the constraints:

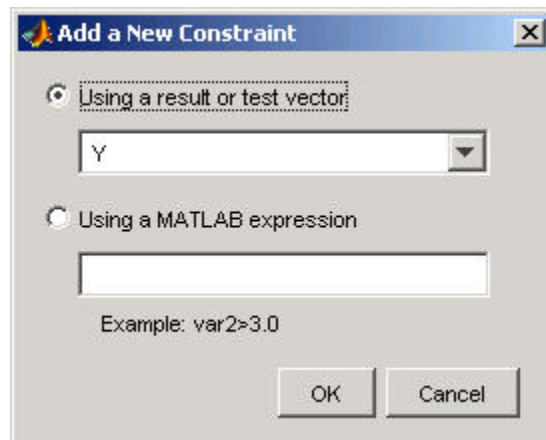
- 1 Click the **New Constraint** button in the **Constraints** pane.



New constraint button

The Add a New Constraint dialog box appears.

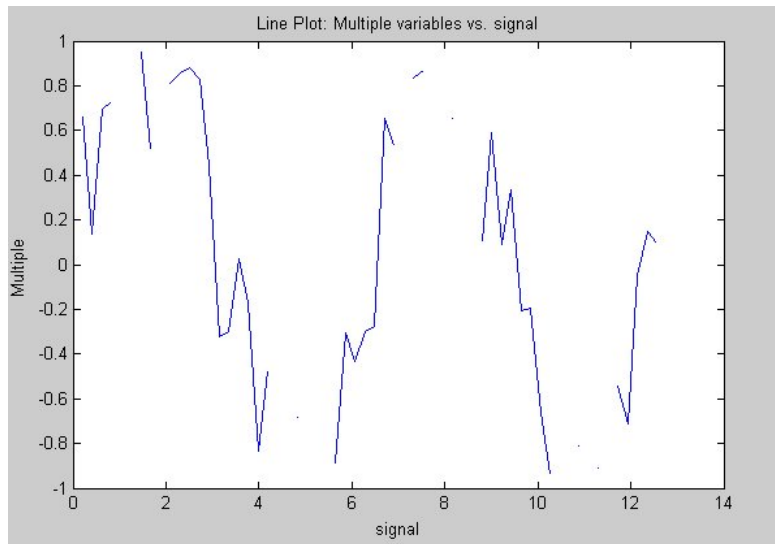
- 2 Click the **Using a result or test vector** option button.
- 3 Select Y from the list.



- 4 Click **OK**. A new constraint appears in the **Constraints** pane.

- 5** Repeat steps 1 to 4.
- 6** Click the check box next to the first constraint to activate it.
- 7** Make sure the operator is set to \geq .
- 8** Click the constraint's text field and change the constraint to -1.
- 9** Click the check box next to the second constraint to activate it.
- 10** Click the operator and set it to \leq .
- 11** Click the constraint's text field and change the constraint's value to 1.

The green indicator bar beneath the constraints shows how many iterations remain following the filter you applied. This is also reflected in the line plot, which the viewer redraws after you apply the new constraints. As you can see, only a subset of your test result remains.



You can just as easily show only the iterations of your test that failed by reconfiguring the constraints you created to filter out data that is < -1 or > 1 . Alternatively, you can create a new constraint that uses a MATLAB expression. For example:

- 1** Delete the constraints you just created.
 - a** Select the constraint row.
 - b** Click the **Delete** button.
- 2** Click the **New Constraint** button in the **Constraints** pane. The Add a New Constraint dialog box appears.
- 3** Click the **Using a MATLAB expression** option button.
- 4** Enter the expression $Y < -1 \mid \mid Y > 1$.
- 5** Click **OK**.
- 6** Click the check box next to the constraint.

The Test Results Viewer redraws the line plot to now show only those test iterations that failed.

For more information about the Test Results Viewer, see Chapter 7, “Using the Test Results Viewer”.

Working With the Basic Elements

Working With the Sections of a Test
(p. 2-2)

Describes the three sections of the test and which elements you can add to each one.

The Basic Elements (p. 2-4)

Describes the basic elements, which sections of the test you can add them to, how they work, and their options in the Properties pane.

Working With the Sections of a Test

Each section of the test serves a different purpose and has different properties that can be set in the **Properties** pane. Click on a part of the test or an element in the **Test Browser** to see the properties for that section or element.

The descriptions of the elements in this chapter include a list of which sections of the test you can use each element in. The following sections describe the sections of a test. They are followed by a description of how to use the basic elements.

Pre Test

The Pre Test runs once prior to any number of iterations through Main Test. Pre Test can be used to perform general test setup such as:

- Opening a model.
- Initializing variables.
- Accessing system resources, such as opening a file.
- Initializing external test equipment.

In Pre Test, only test variables defined as a Pre Test variable may be modified or assigned to. Pre Test variables are initialized once during Pre Test and persist throughout the Main Test and Post Test.

In Pre Test you can add the following element types: Simulink, MATLAB, Subsection, IF, Video Input, the three Instrument Control Toolbox elements, and the four Data Acquisition Toolbox elements.

With Pre Test you can initialize Pre Test variables and run elements that you only want to run once before any Main Test iterations. For example, you can:

- Add a Simulink element to run a model and assign baseline data to a Pre Test variable.
- Add a MATLAB element to load a MAT-file or perform some other test setup.

- Create conditions with the IF element and follow up with a Subsection element to define what to do when those conditions are met.

Main Test

The Main Test is run one or more times based on the number of iterations. It is used to:

- Execute elements multiple times in order to perform batch testing or sweep through a parameter space.
- Perform batch testing or parameter sweeps that require multiple independent iterations using different test conditions for each iteration.

The number of iterations is defined by the number and length of test vectors you specify. SystemTest executes Main Test once for each permutation of values in the test vectors specified.

In Main Test you can add all of the element types.

Post Test

The Post test runs once after all Main Test iterations have executed or when a run-time error occurs in Pre Test or Main Test. Post Test can be used to perform test cleanup, such as:

- Closing a model.
- Cleaning up your workspace.
- Releasing system resources, such as closing a file.
- Returning external test equipment to a safe state.

In Post Test you can add the following element types: MATLAB, Subsection, IF, Video Input, the three Instrument Control Toolbox elements, and the four Data Acquisition Toolbox elements.

The Basic Elements

The following sections describe how to work with the basic elements: MATLAB, Limit Check, IF, Vector Plot, Scalar Plot, Stop, and Subsection. The Simulink element is covered in detail in Chapter 3, “Using the Simulink Element”.

To see the MATLAB, Limit Check, and Scalar Plot elements used in an example, see “Adding Elements” on page 1-14.

Note You can rename any element or subsection by double-clicking its name.

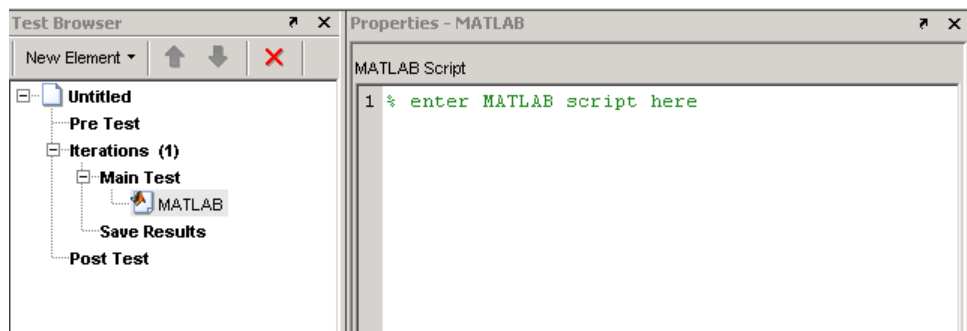
MATLAB Element

The MATLAB element lets you run MATLAB scripts from within a test. In addition to specifying any valid MATLAB script to execute, you can incorporate any test variable into your code, as well as access any variables residing in the MATLAB workspace.

Allowed Test Sections

The MATLAB element can be used in the following test sections:

- Pre Test
- Main Test
- Post Test



Properties Pane

In the **MATLAB Script** edit field, enter any valid MATLAB script.

Limit Check Element

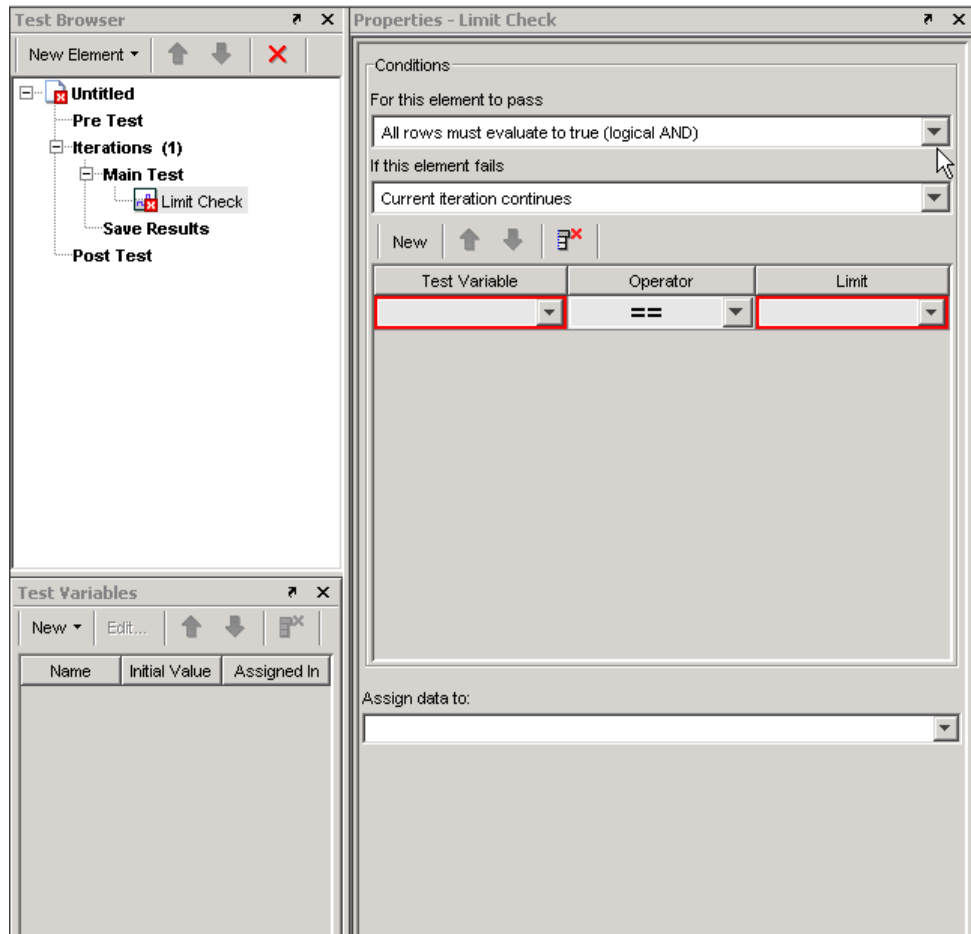
The Limit Check element determines test conditions are met by using scalar comparisons. It can be used to:

- Compare measured data to expected data.
- Stop an iteration or an entire test if a test constraint is violated.
- Assign a test variable the logical value derived from the comparison(s) for use by other elements.

Allowed Test Sections

The Limit Check element can be used in the following test section:

- Main Test



How to Use

1 Set your test's passing conditions.

- The element can pass if all comparisons complete successfully (a logical AND).
- The element can pass if one or more of the comparisons complete successfully (a logical OR).

- 2 Set your fallback procedure if the element fails. You can:
 - Allow the current iteration to continue executing.
 - Stop the current iteration and proceed onto the next iteration.
 - Stop the test by proceeding onto Post Test.
- 3 Identify the SystemTest test variable you want to assign the logical value derived from the comparison(s).

Note Aside from setting limit checks on individual elements, you can set these properties for the entire test, reachable by clicking the test name in the **Test Browser**, to determine pass/fail criteria for the test as a whole.

Properties Pane

You can set the following properties for the Limit Check element.

- **For this element to pass** — Choose between a logical AND (all comparisons must pass) or a logical OR (at least one comparison needs to pass) for the element to pass.
- **If this element fails** — Choose between continuing the test, stopping the current iteration, or stopping the entire test.
- **Test Variable** — Value to compare to limit using operator.
- **Operator** — Boolean operator used to compare test variable to limit.
- **Limit** — Value to compare to test variable using operator.
- **Assign data to:** — Test variable assigned the logical value of this evaluation. The logical value will be 1 if the element passes or 0 if the element fails. If the element does not run, the test variable will be assigned its initial value.

IF Element

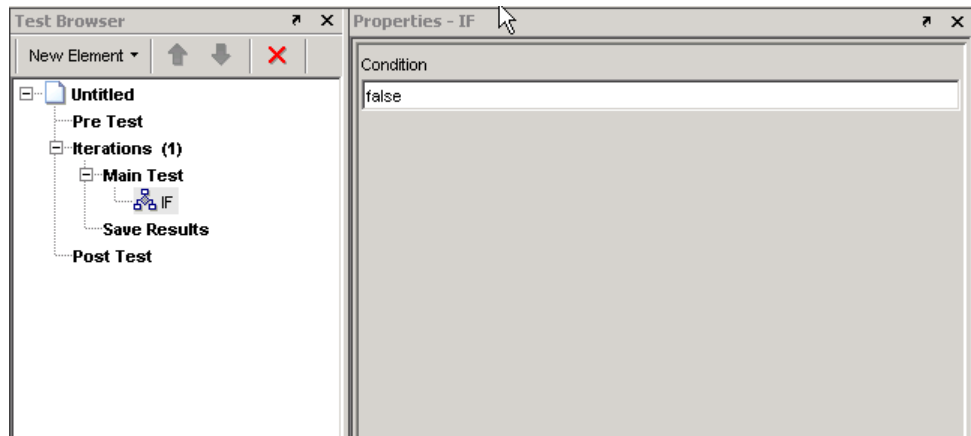
The IF element provides logical control of a test by evaluating a condition.

The IF element allows sub-elements to run only when the IF element's condition evaluates to true. After adding an IF element, you should add one or more elements to perform a specific task.

Allowed Test Sections

The IF element can be used in the following test sections:

- Pre Test
- Main Test
- Post Test



Properties Pane

You can set the following property for the IF element.

- **Condition** — Enter a valid MATLAB expression that will evaluate to true or false.

Vector Plot Element

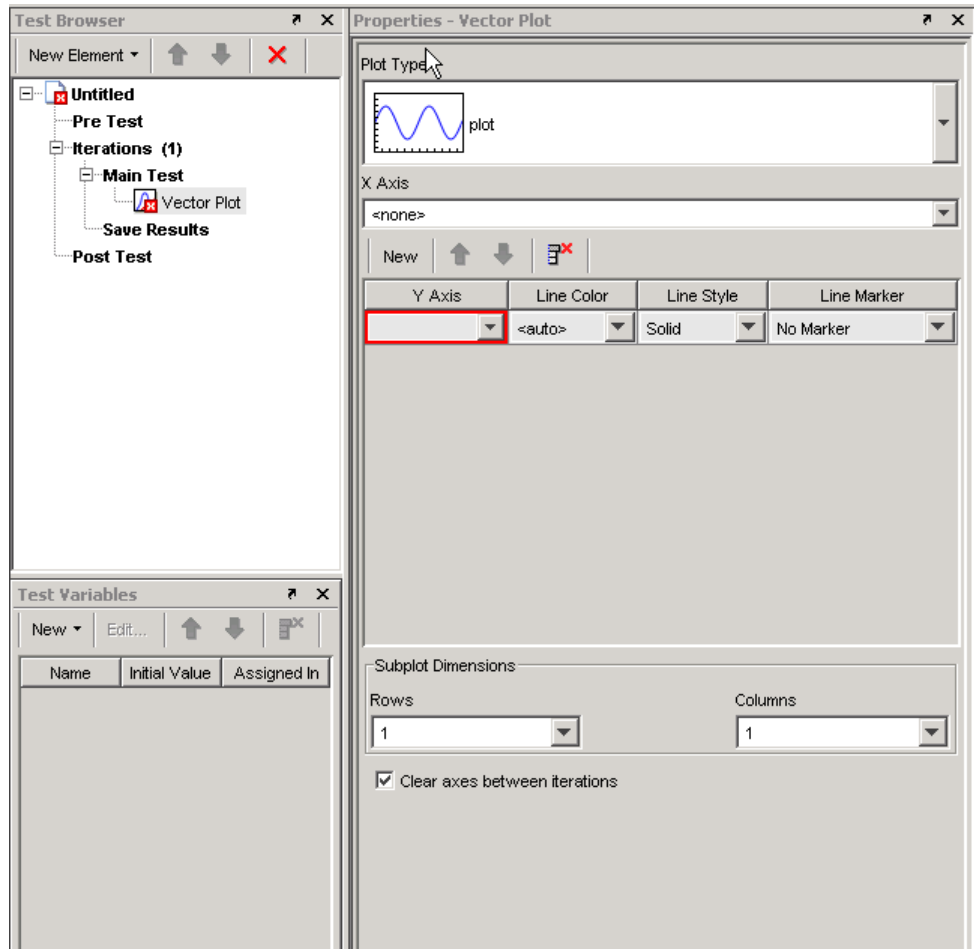
The Vector Plot element is used to plot array or vector data over multiple iterations. It is accessed off of **Plot** in the Elements list.

Use this element during the Main Test to generate plots of any test variables containing array or vector data. You can change the number of iterations displayed to as many as 16 (in a 4 x 4 matrix) using the **Subplot Dimensions** fields. The default is 1 iteration.

Allowed Test Sections

The Vector Plot element can be used in the following test section:

- Main Test



Plot Type

Choose one of the following plot types:

- **plot** — Standard plot of X and Y.
- **semilogx** — Semilogarithmic plot with logarithmic X-axis.
- **semilogy** — Semilogarithmic plot with logarithmic Y-axis.
- **loglog** — Log-log scale plot.
- **stem** — Lines extending from a baseline along the X-axis.

Properties Pane

You can set the following properties for the Vector Plot element.

- **X Axis** — Choose a test variable to use for an X-axis value.
- **Y Axis** — Choose a test variable to use for a Y-axis value.
- **Line Color** — Select a color to use for the line between each data point.
- **Line Style** — Set the type of line to be drawn between each data point.
- **Line Marker** — Choose a symbol to represent each data point.

Subplot Dimensions

- **Rows** — The number of rows you want displayed in the Subplots window.
- **Columns** — The number of columns you want displayed in the Subplots window.
- **Clear axes between iterations** — Applies only when you have one row and one column to display. Selecting this option (default) rewrites the plot with new data during each iteration. Clearing this option adds new data to the plot during each iteration.

Scalar Plot Element

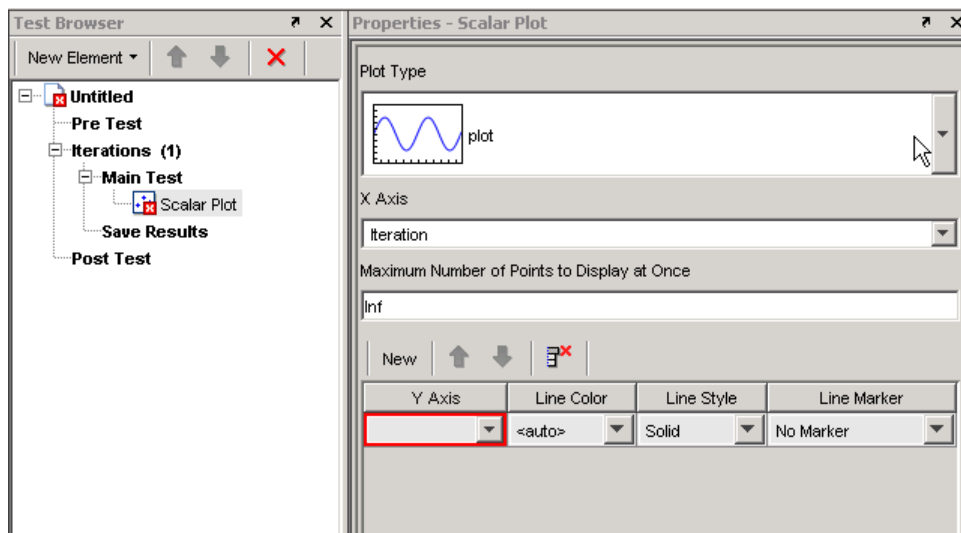
The Scalar Plot element is used to plot scalar data for each iteration. It is accessed off of **Plot** in the Elements list.

Use this element during the Main Test to generate a plot of one or more scalar test variables.

Allowed Test Sections

The Scalar Plot element can be used in the following test section:

- Main Test



Plot Type

Choose one of the following plot types:

- **plot** — Standard plot of X and Y.
- **semilogx** — Semilogarithmic plot with logarithmic X-axis.
- **semilogy** — Semilogarithmic plot with logarithmic Y-axis.
- **loglog** — Log-log scale plot.
- **stem** — Lines extending from a baseline along the X-axis.

Properties Pane

You can set the following properties for the Scalar Plot element.

- **Maximum Number of Points to Display at Once** — Determine how many points to show simultaneously. By default this is infinite such that all points will be plotted. Use a MATLAB numeric that evaluates to a positive, nonzero integer to set this field's value.
- **X Axis** — Choose a test variable to use for an X-axis value.
- **Y Axis** — Choose a test variable to use for a Y-axis value.
- **Line Color** — Select a color to use for the line between each data point.
- **Line Style** — Set the type of line to be drawn between each data point.
- **Line Marker** — Choose a symbol to represent each data point.

Stop Element

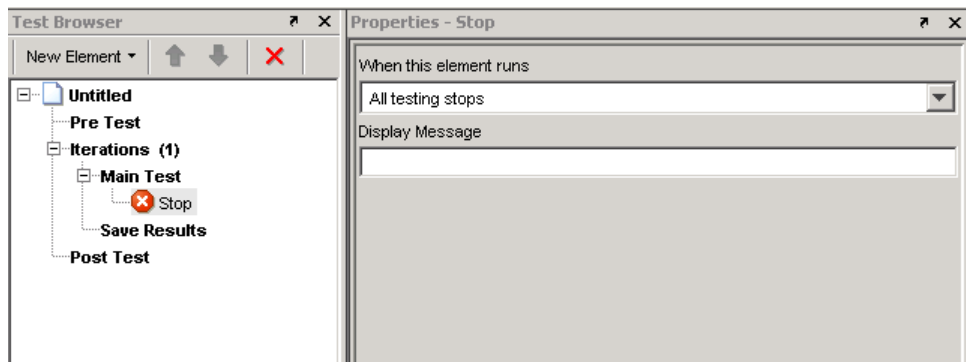
The Stop element stops an iteration or an entire test unconditionally.

You can use the Stop element with conditional logic elements, such as the IF element, to control the test's execution.

Allowed Test Sections

The Stop element can be used in the following test section:

- Main Test



Properties Pane

You can set the following properties for the Stop element.

- **When this element runs** — Select a stop action. The **Current iteration stops** option stops the current Main Test iteration. The **All testing stops** option stops all Main Test iterations and runs Post Test.
- **Display Message** — Enter a message to display in the **Run Status** pane when the test stops.

Subsection Element

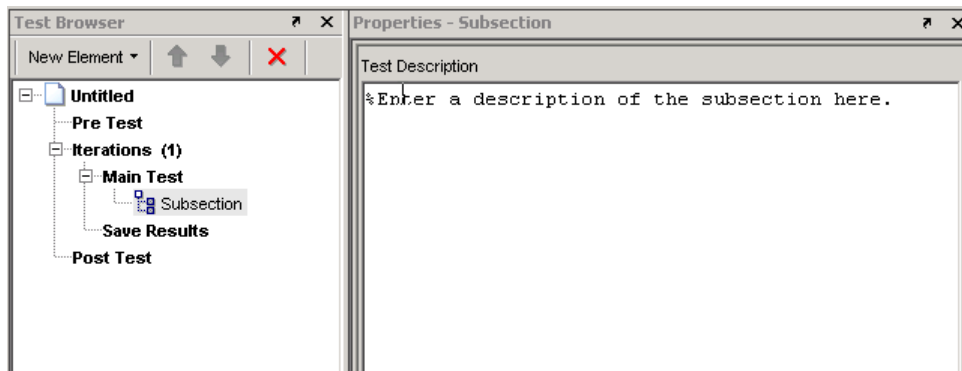
Use subsection elements to organize one or more elements to maintain readability of your test or to better manage complex test structures. Use a subsection to:

- Group elements under a single root element.
- Organize tests.
- Manage complex test structures.

Allowed Test Sections

The Subsection element can be used in the following test sections:

- Pre Test
- Main Test
- Post Test



Properties Pane

You can set the following properties for the Subsection element.

- **Test Description** — Type in your description of the subsection.

Using the Simulink Element

The Simulink[®] element allows you to override the inputs to a Simulink model with SystemTest test vectors and test variables. You can further map the model's outputs to SystemTest test variables for later processing by other test elements. This means that you can use SystemTest to define, generate or load input data, feed it into Simulink, run the model while iterating over the input data, and map the outputs back into SystemTest.

Note To use the Simulink element, you must have a license for Simulink.

Before You Begin (p. 3-2)

Prerequisite steps to take if you intend to use the examples provided with the procedures in this chapter

Mapping Test Vectors and Test Variables to a Simulink Model (p. 3-4)

Describes how to map test vectors and test variables to Simulink inputs and Simulink outputs to test variables

Using Simulink Model Coverage (p. 3-18)

Describes how to use the Simulink Verification and Validation model coverage feature

Before You Begin

This chapter explains the Simulink setup by having you recreate the Simulink element that is part of the Inverted Pendulum demo. Before continuing, you should load this demo from MATLAB and delete the Simulink element from the demo.

The following steps describe how to do this:

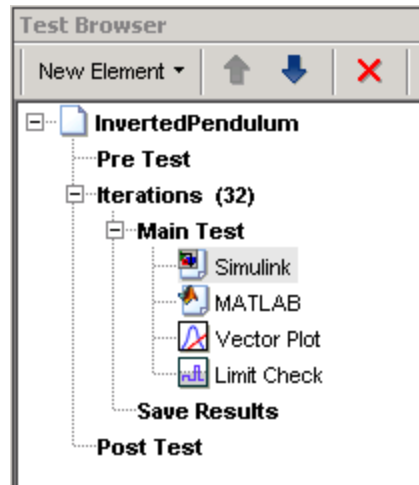
- 1 Start MATLAB.
- 2 Open the Inverted Pendulum demo.
 - a Select **Start > Demos** to open the Help browser.
 - b Expand the **MATLAB** list from the left frame of the browser.
 - c Select **SystemTest**. The SystemTest demos open in the right browser frame.
 - d Click “Simulink - Mapping and Overriding Simulink Data Using an Inverted Pendulum Model.” An overview of the demo opens.
 - e Click the link “Open the demo in the SystemTest Desktop” at the bottom of the page.

Alternatively, you can enter the following command at the MATLAB command line:

```
systemtest InvertedPendulum
```

The SystemTest Desktop opens with the Inverted Pendulum demo loaded.

- 3 Click the **Simulink** element in the **Test Browser**.



- 4 Click the **Delete element** button in the Test Browser button bar or press the **Delete** key.

Mapping Test Vectors and Test Variables to a Simulink Model

To help you learn how to use the Simulink element, this section walks you through the configuration of the Simulink element for the Inverted Pendulum test. The Inverted Pendulum demo includes both a model of the pendulum and a model of a controller that keeps the inverted pendulum balanced. Moving the bottom of the pendulum disturbs the equilibrium, causing the pendulum to move and the controller to rebalance it. The Inverted Pendulum test varies the mass of the pendulum, the mass of the cart the pendulum is on, and the distance to the pendulum's center of mass, testing the robustness of the controller as it attempts to return the pendulum to equilibrium. Using the Simulink element in a test lets you vary the model inputs and assess the model outputs.

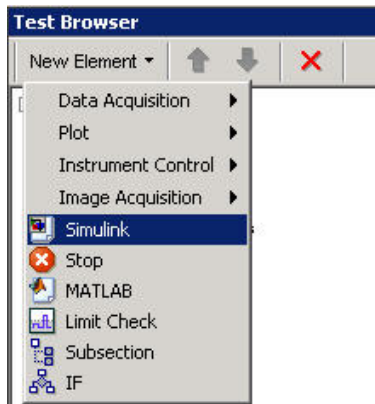
The following sections show you how to use a Simulink element in a test:

- “Adding a Simulink Element” on page 3-4
- “Overriding Simulink Model Inputs” on page 3-6
- “Mapping Simulink Model Outputs to Test Variables” on page 3-10

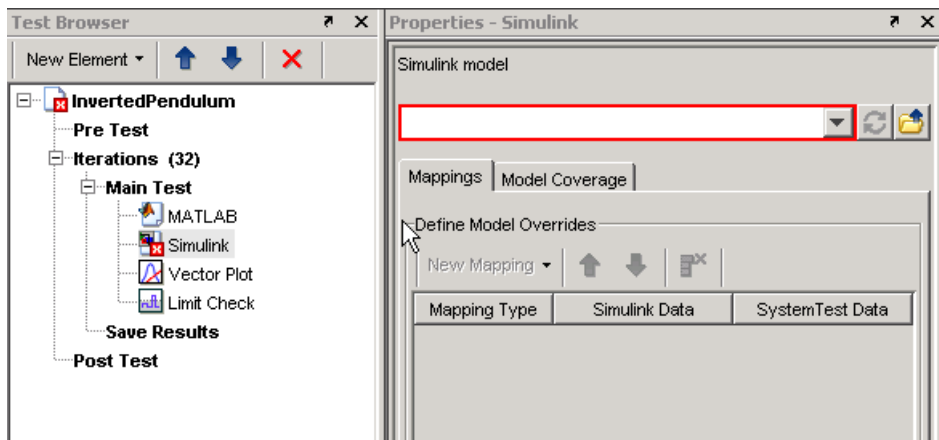
This section assumes you have loaded the Inverted Pendulum demo and deleted the Simulink element, as explained in “Before You Begin” on page 3-2.

Adding a Simulink Element

To add a Simulink element to a test, click the **New Element** button in the **Test Browser** and select the Simulink element. If you have a license for Simulink, the **New Element** list contains the Simulink element, as shown in the following figure.



SystemTest adds the Simulink element to the test and opens the Simulink element **Properties** pane.



Specifying the Simulink Model

When you first add the element, the icon in the **Test Browser** has a red x, meaning that the element requires some information. The **Simulink model** field in the Simulink element **Properties** pane is outlined in red, indicating that it is a required field. You must specify the model that the Simulink element will interact with. If the model is on the MATLAB path, you can type its name in the **Simulink model** field. If you are not sure of the name, or

the model is not on the path, you can browse to its location using the browse button.

For the Inverted Pendulum example, type `systemtestpendulum` in the **Simulink model** field and press Enter. SystemTest opens the `systemtestpendulum` model in Simulink and opens the **Pendulum Visualization** window.

Overriding Simulink Model Inputs

Using test vectors and test variables, you can override the following Simulink model inputs:

- Block parameters — Described in “Overriding Simulink Block Parameters” on page 3-6
- Model and base workspace variables — Described in “Overriding to Workspace Variables” on page 3-8
- Inport signals — Described in “Overriding Simulink Model Inport Signals” on page 3-10

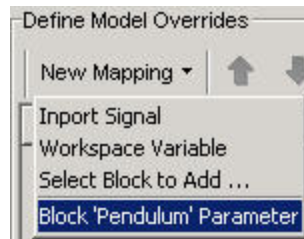
Overriding Simulink Block Parameters

You can override Simulink block parameters with SystemTest test vectors or test variables. When you run the test, Simulink runs the model using data provided by SystemTest. Overriding does not change your Simulink model file; it only overrides in the test. The procedure for creating block parameter overrides requires that you select your block in the Simulink model, but everything else you need to do happens within the Simulink element **Properties** pane.

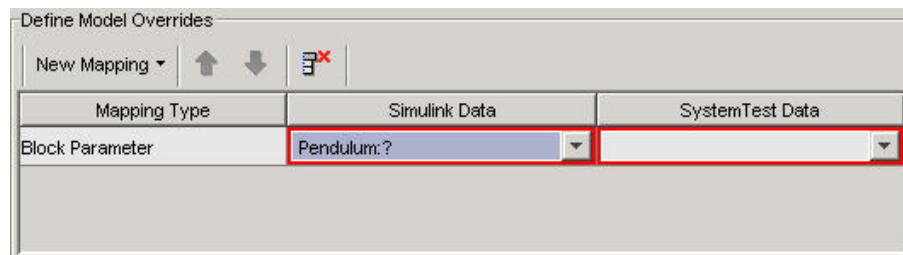
To override a Simulink block parameter:

- 1 In the **Mappings** tab of the **Properties** pane for the Simulink element in SystemTest, find the **Define Model Overrides** area and click the **New Mapping** button, and select **Select Block to Add...** This opens the model in Simulink, if it is not already open.

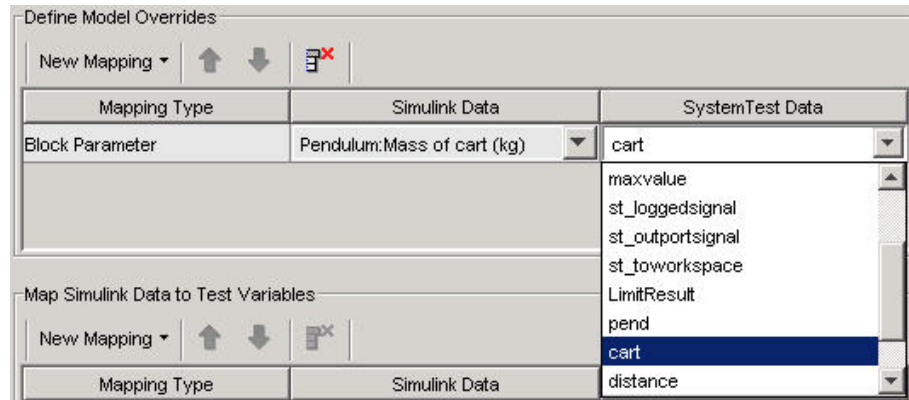
- 2 In the Simulink model window, click the block containing the parameter you want to override. For this example, click the Pendulum block in the **systemtestpendulum** model window.
- 3 In SystemTest, return to the Simulink element **Properties** pane and, in the **Define Model Overrides** area, you'll see that the **Block 'Pendulum' Parameter** was added. If you click the **New Mapping** button, you'll see that SystemTest also adds an entry to this menu for the block.



In the override table, the **Simulink Data** field shows that this entry is linked to the Pendulum block but the question mark (?) indicates that no parameter for the block has been mapped.



- 4 Select the parameter from the block that you want to map. Click the **Simulink Data** field for the block and select a parameter from the list. For the Inverted Pendulum demo example, click Pendulum:Mass of cart (kg).
- 5 Specify the SystemTest test vector or test variable you want to map to this block parameter. Click the **SystemTest Data** field for the block parameter. This shows you all defined SystemTest test vectors and test variables available for mapping. For this example, select **cart**.



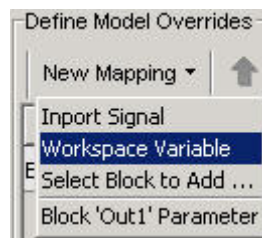
Overriding to Workspace Variables

You can use a SystemTest test vector or test variable to override either a MATLAB base workspace variable or a Simulink model workspace variable. This lets you define test values and conditions in SystemTest and have a Simulink model act on them.

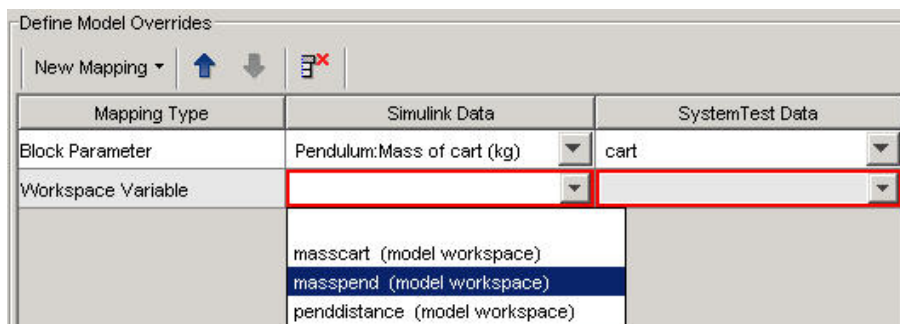
This section describes how you can use the values in the pend and distance test vectors to override the model workspace variables masspend and penddistance in the Inverted Pendulum demo.

To override workspace variables:

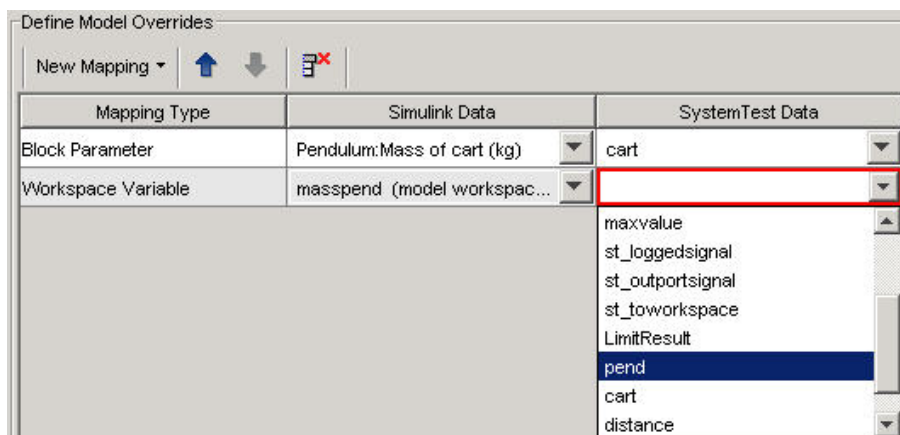
- 1 In the **Define Model Overrides** area of the Simulink element **Properties** pane, click the **New Mapping** button and select **Workspace Variable**. SystemTest adds a row for a new mapping of this type.



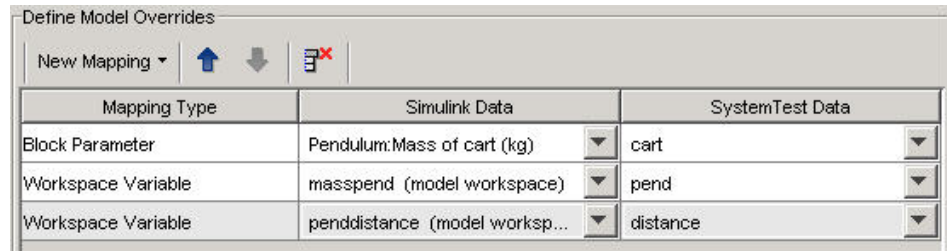
- 2 Select the workspace variable you want to override. Click the **Simulink Data** field of this row to see all available base workspace variables and Simulink model workspace variables. For the Inverted Pendulum example, select **masspend**.



- 3 Specify which SystemTest test vector or test variable you want to map to the Simulink workspace variable. Click the **SystemTest Data** field of this row to see all available test vectors and test variables. For this example, select **pend**.



- 4 Repeat steps 1 to 3 to override the Simulink model workspace variable penddistance with the SystemTest test vector distance.



Overriding Simulink Model Inport Signals

As with block parameters and workspace variables, you can use SystemTest to override a model's inport signals. This lets you externally manipulate the input signal of a Simulink model.

You use the same procedure to map an inport signal that you use for mapping block parameters and workspace variables. Click the **New Mapping** button and select **Inport Signal**. SystemTest creates a new row for the mapping. Then you can choose the inport signal to override from the **Simulink Data** field and select the SystemTest test vector or test variable to be mapped from the **SystemTest Data** field.

The Inverted Pendulum demo example does not override any inport signals.

Mapping Simulink Model Outputs to Test Variables

Using test variables you can assign the output from the following types of Simulink model data:

- Logged signals — Described in “Mapping Simulink Logged Signals to Test Variables” on page 3-11
- Output signals — Described in “Mapping Simulink Output Signals to Test Variables” on page 3-13
- To Workspace blocks — Described in “Mapping Simulink To Workspace Blocks to Test Variables” on page 3-14

After you map model outputs to test variables, you can incorporate the model data into SystemTest. This section shows you how to map this data for the Inverted Pendulum example.

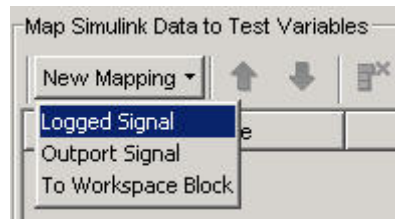
Note The output from Simulink models can only be mapped to SystemTest test variables. You cannot map this output to SystemTest test vectors.

Mapping Simulink Logged Signals to Test Variables

Logged signals are a way to obtain outputs from a model without adding more outputs. Using logged signals, you can identify a particular signal and map the output to a SystemTest test variable.

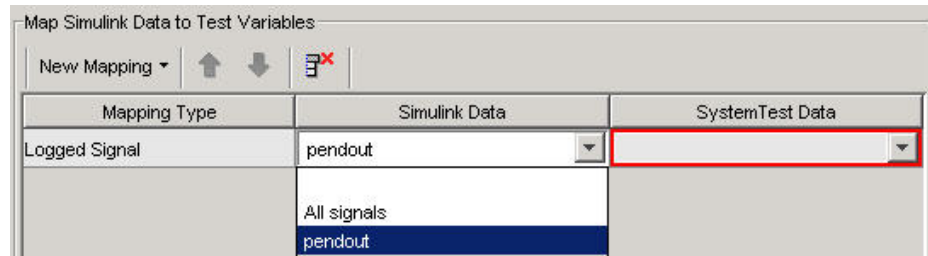
To map logged signals to a SystemTest test variable:

- 1 In the **Map Simulink Data to Test Variables** area of the Simulink element **Properties** pane, click the **New Mapping** button. From the list, select **Logged Signal**. SystemTest adds a row for a new mapping of this type.

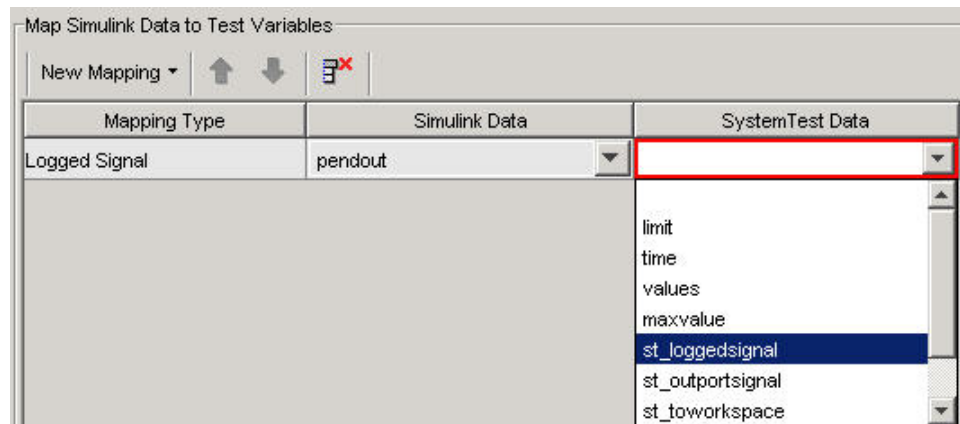


- 2 Specify the signal you want to capture. Click the **Simulink Data** field to see all the signals in the model. For the Inverted Pendulum example, select **pendout**.

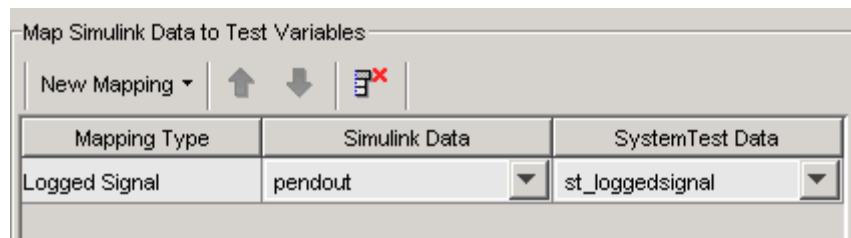
Note If you added logged signals to your model and they do not appear in this list, click the refresh button, on the **Properties** pane next to the model name, to update the list.



- 3** Specify the SystemTest test variable to which you want to map the output. Click the **SystemTest Data** field and select a test variable. For the Inverted Pendulum example, select `st_loggedsignal`.



SystemTest creates the mapping to the test variable.

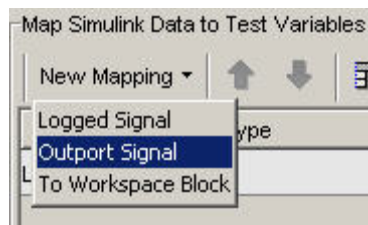


Mapping Simulink Output Signals to Test Variables

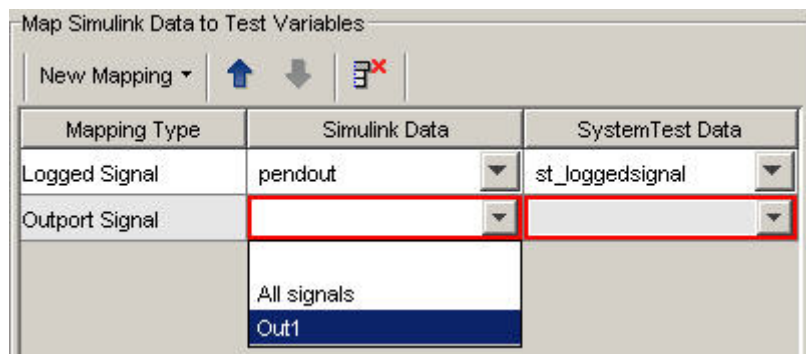
SystemTest lets you map all output signals to a test variable for further processing in SystemTest.

To map Simulink output signals to a test variable:

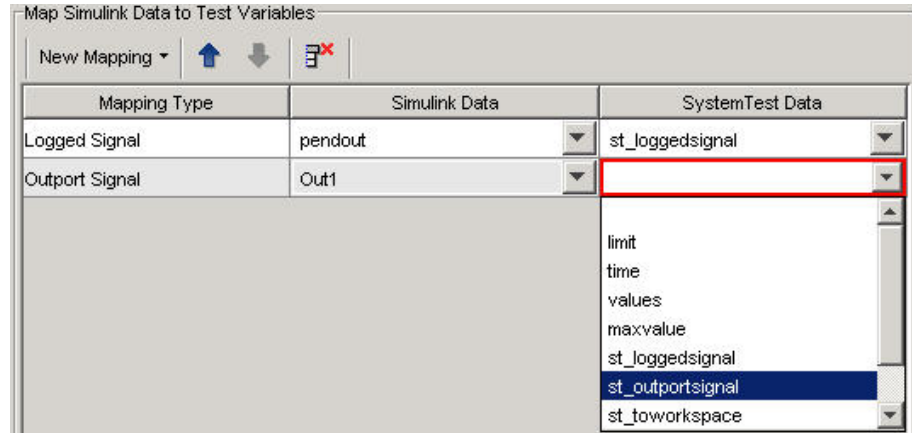
- 1 In the **Map Simulink Data to Test Variables** area of the Simulink element **Properties** pane, click the **New Mapping** button. From the list, select **Output Signal**. SystemTest adds a row for a new mapping of this type.



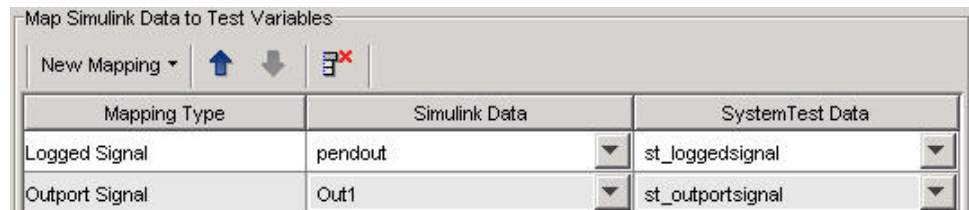
- 2 Specify the output signal you want to capture. Click the **Simulink Data** field and select a signal. For this example, click **Out1**.



- 3 Specify the SystemTest test variable to which you want to map the output signals. Click the **SystemTest Data** field and select a test variable from the list. For this example, select **st_outportsignal**.



SystemTest creates the mapping to the test variable.



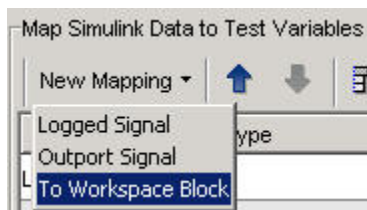
Mapping Simulink To Workspace Blocks to Test Variables

When Simulink runs a model with To Workspace blocks, these blocks save model information in the MATLAB workspace as variables. Using SystemTest, this data can be mapped to SystemTest test variables.

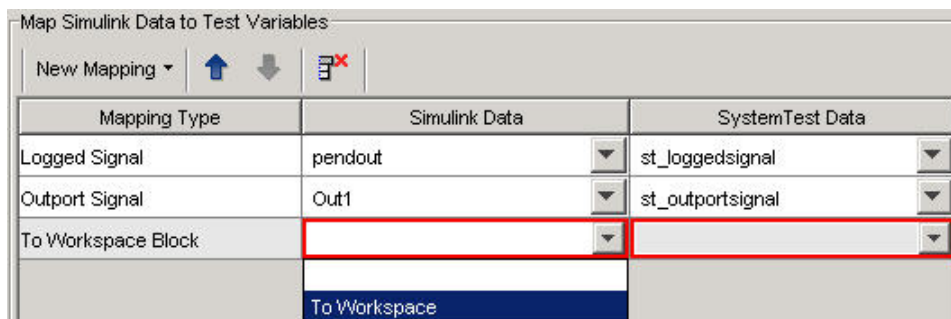
This section shows how you create To Workspace block mappings in SystemTest using the Inverted Pendulum demo as an example.

To map the To Workspace block:

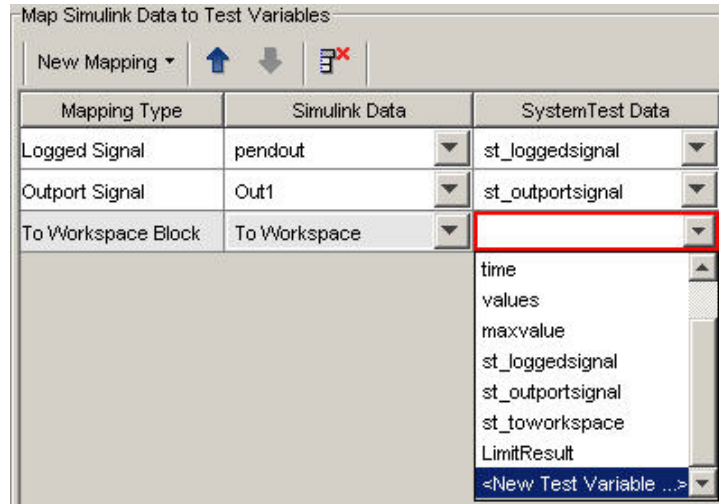
- 1 In the **Map Simulink Data to Test Variables** area of the Simulink element **Properties** pane, click the **New Mapping** button. From the list, select **To Workspace Block**. SystemTest adds a row for a new mapping of this type.



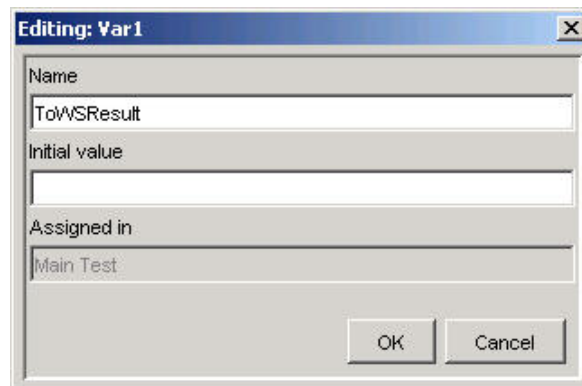
- Specify the To Workspace block in the model that you want to capture. Click the **Simulink Data** field and select the block from the list. For this example, select **To Workspace**.



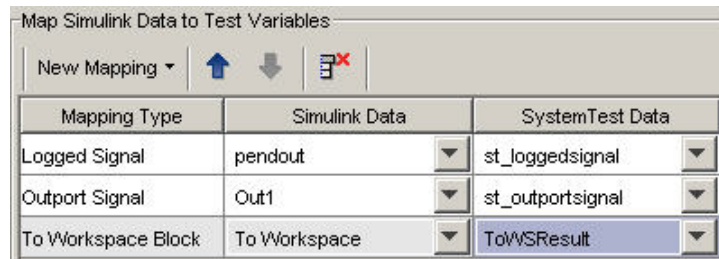
- Specify the SystemTest test variable to which you want to map the To Workspace block. Click the **SystemTest Data** field and select a test variable from the list. For this example, select **New Test Variable...** to create a test variable.



SystemTest opens the Editing Variable dialog box. Assign a name to the test variable and optionally an initial value, and then click **OK**. Name the test variable ToWSResult.



SystemTest creates the mapping to the new test variable and adds the new test variable to the list in the **Test Variables** pane.



Mapping Type	Simulink Data	SystemTest Data
Logged Signal	pendout	st_loggedsignal
Output Signal	Out1	st_outportsignal
To Workspace Block	To Workspace	ToWSResult

Using Simulink Model Coverage

The model coverage feature provided by Simulink Verification and Validation allows you to generate coverage analysis metrics for a Simulink model, which can be incorporated directly into your SystemTest test. Model coverage metrics allow you to validate your model by identifying unexecuted subsystems, unselected switch positions, or untaken conditional transition paths. You can generate a cumulative coverage report, specify individual coverage options, or inherit a model's coverage settings.

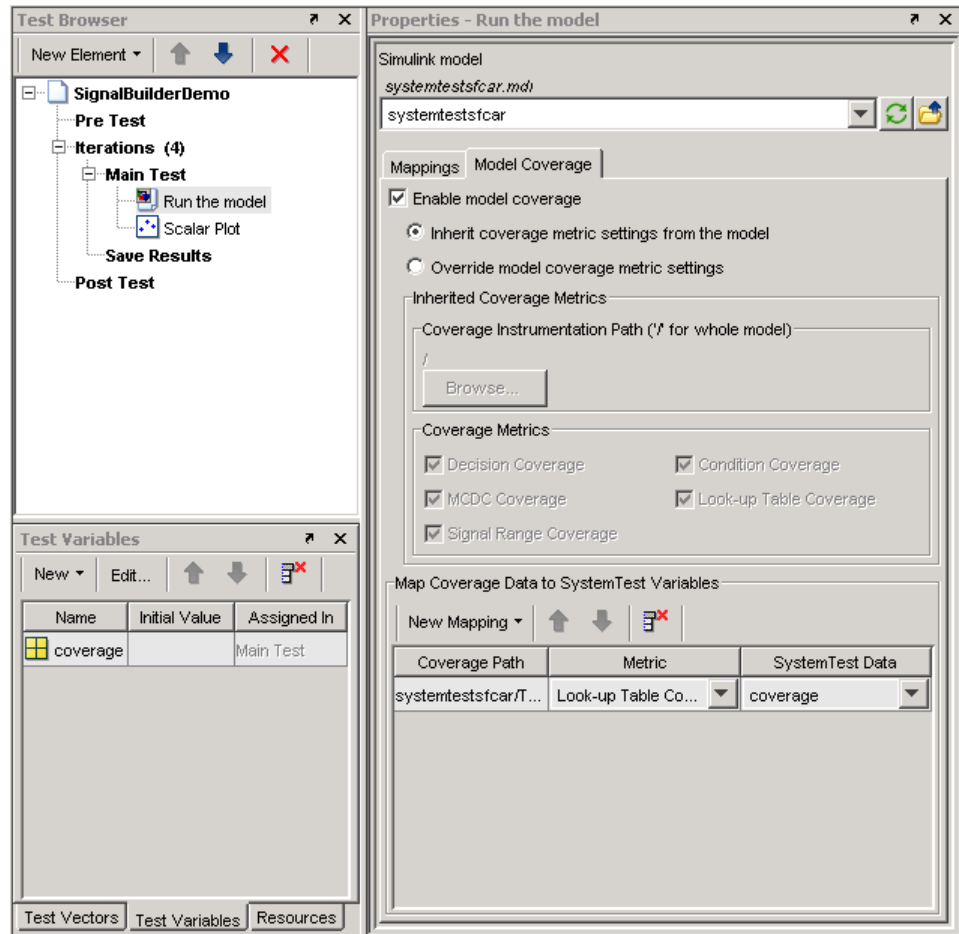
Note To use the model coverage feature, you need a license for Simulink Verification and Validation.

The following basic steps describe the typical work flow to use this feature:

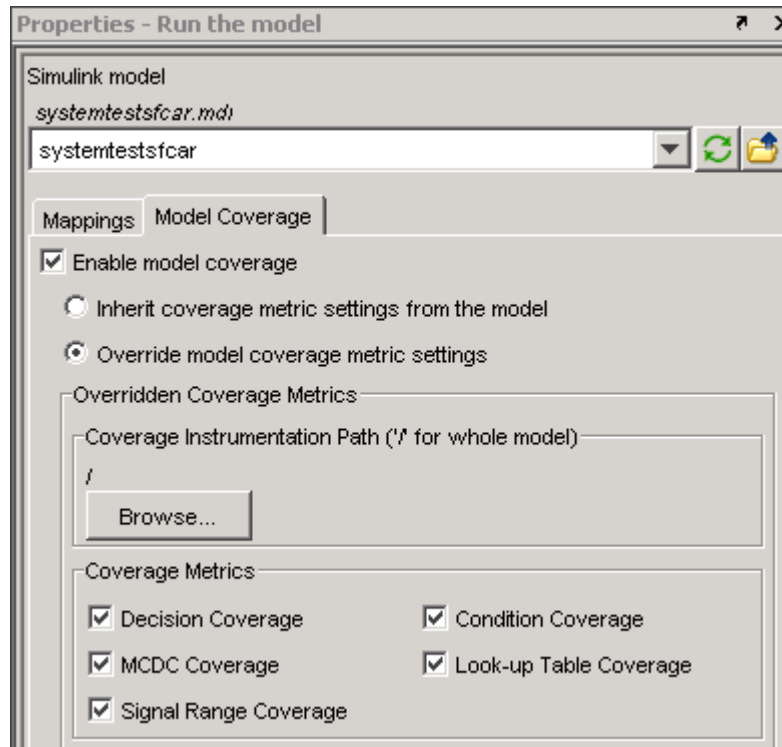
- 1** Use an existing Simulink element or add one by clicking the **New Element** button and selecting **Simulink**.
- 2** On the **Properties** pane, browse for your Simulink model using the browse button next to the **Simulink model** field.

To see an example, you can run the Signal Builder demo by typing `systemtest SignalBuilderDemo` in MATLAB.

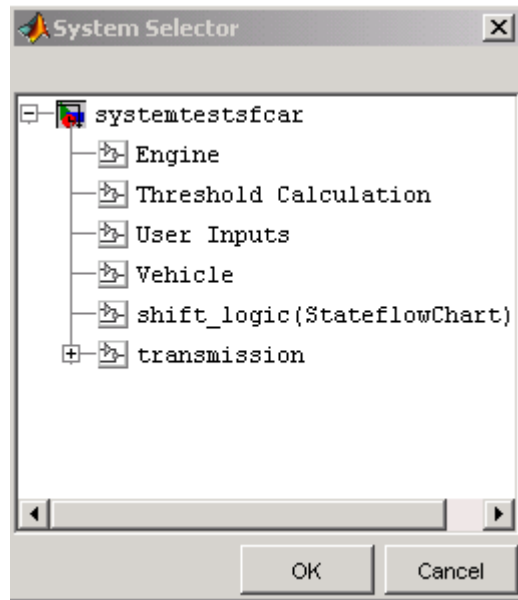
- 3** Configure the Simulink element as described in this chapter, using the **Mappings** tab of the **Properties** pane to define model overrides and map Simulink data to test variables.
- 4** On the **Model Coverage** tab, which appears if you have a license for Simulink Verification and Validation, select the **Enable model coverage** check box. The following figure shows the Signal Builder demo.



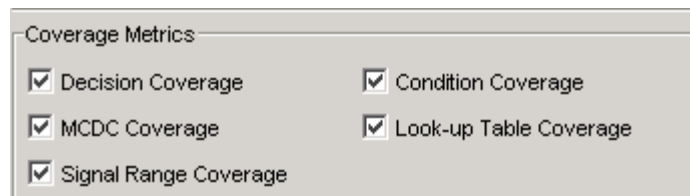
- 5 If you want to use the model coverage settings you already have on the Simulink model, select the **Inherit coverage metric settings from the model** option. Then go to step 9.
- 6 If you want to override the existing settings, select the **Override model coverage metric settings** option.



7 Click the **Browse** button to specify the **Coverage Instrumentation Path**.

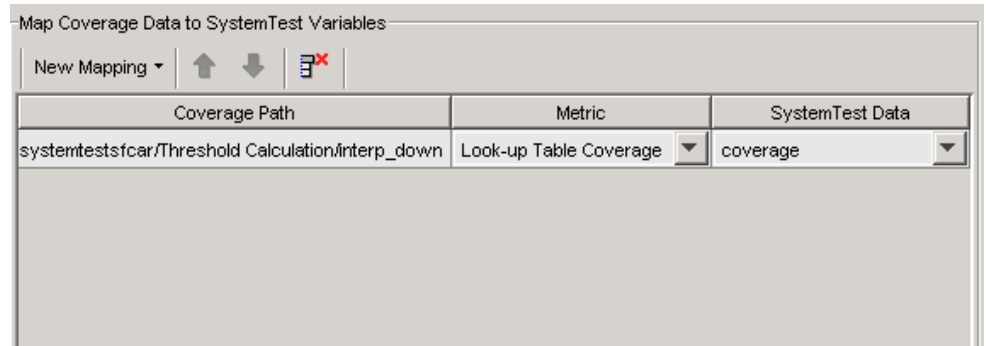


- 8** In the **Coverage Metrics** field, select the types of coverage your test requires. The selected coverage types will be generated and shown in the coverage report.

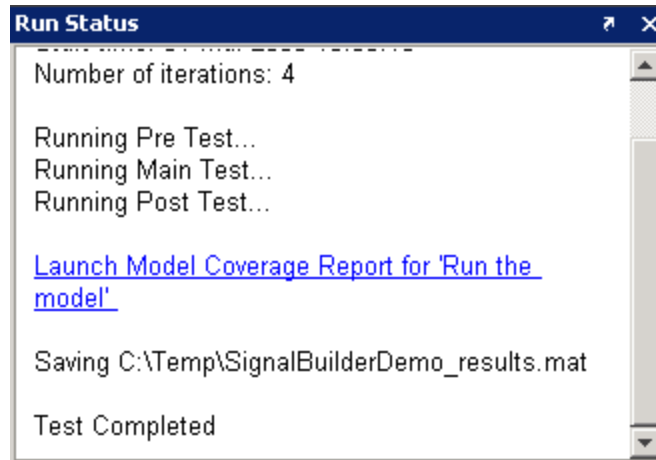


- 9** Use the **Map Coverage Data to SystemTest Variables** field to map coverage metrics to test variables. Click **New Mapping** and select **Full Coverage Instrumentation Path** if you want coverage data below the root you specified under **Coverage Instrumentation Path**, or select **Select Path to Map...** if you want to pick an alternate coverage path, which must be within the coverage instrumentation path. If you select the latter, your Simulink model will open and you can select a block to specify an alternate root for your coverage path.

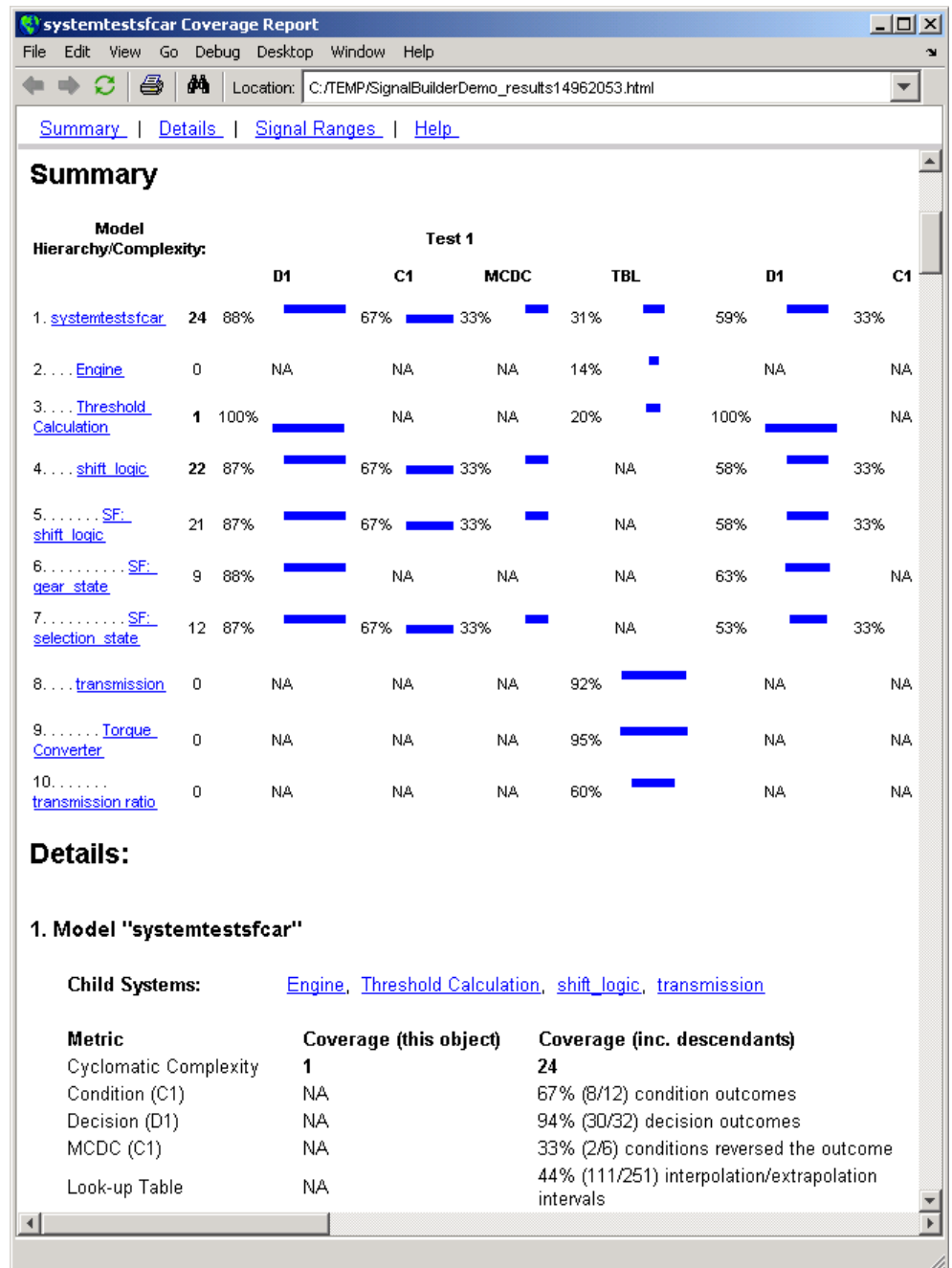
- 10 Select the **Metric** you wish to map to a test variable, and specify the test variable to use under the **SystemTest Data** column.



- 11 Run your test.
- 12 View the coverage report by clicking the link in the **Run Status** pane.



For more information on the model coverage feature, see the Using Model Coverage section of the Simulink Verification and Validation documentation.



Using the Instrument Control Toolbox Elements

The Instrument Control Toolbox provides several elements to use in SystemTest.

Introduction (p. 4-2)

Introduces the Instrument Control Toolbox elements

Example: Measuring a Generator's Frequency (p. 4-3)

Provides a basic example of using Instrument Control Toolbox elements to test a device

Introduction

Instrument Control Toolbox Elements

This chapter describes how to use the Instrument Control Toolbox elements with SystemTest.

The Instrument Control Toolbox elements provide a way to bring data from instruments into a SystemTest test, or to transmit data from your instrument. You can use these elements along with the other elements in SystemTest to create tests for Simulink models and other applications.

Note To use the Instrument Control Toolbox elements, you need a license for the Instrument Control Toolbox. These three elements will not appear in SystemTest without this license.

The Instrument Control Toolbox provides three of elements that you can use in SystemTest:

- To Instrument — For sending commands or data to your instrument
- From Instrument — For reading data from your instrument
- Query Instrument — For querying your instrument status or properties

You can configure these elements to communicate with your instruments by using SystemTest resources supported by the Instrument Control Toolbox.

Accessing Resources

If your MATLAB installation includes elements that require resources, the SystemTest Desktop includes a **Resources** pane that lets you access the resources available through these toolboxes. For example, if your MATLAB installation includes the Instrument Control Toolbox, you will see the **Resources** pane. Resources are toolbox-specific. For example, an Instrument resource might be configured to connect to a device over your computer's serial port.

Example: Measuring a Generator's Frequency

To illustrate how to use some of the Instrument Control elements in SystemTest, this section provides a step-by-step example.

In this example a SystemTest element configures a signal generator to produce signals of various frequencies, which are measured by an oscilloscope configured by other SystemTest elements.

The signal generator is a Hewlett-Packard 33120A at GPIB address 5, and the oscilloscope is a Tektronix TDS 210 at GPIB address 4. For this example, the generator output is fed directly to the scope input. The generator will be programmed to generate signals of 1500, 5000, and 7500 Hz, while the oscilloscope will measure each signal's frequency.

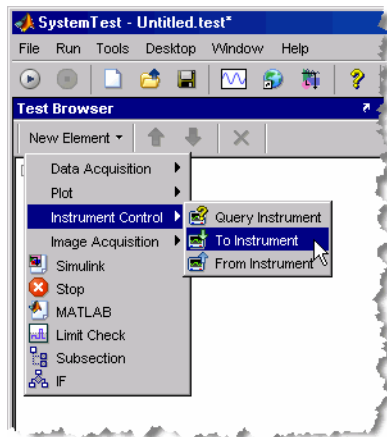
The steps in this example are:

- “Setting Up the Signal Generator” on page 4-4
- “Setting Up the Oscilloscope” on page 4-7
- “Taking the Measurement” on page 4-9
- “Saving Test Results” on page 4-10
- “Running the Test and Viewing Test Results” on page 4-11

Setting Up the Signal Generator

The first element in the test programs the generator to output signals of various frequencies. To test at three frequencies, the test be comprised of three test cases, i.e., three iterations. This is a one-way communication to the generator, so you use a To Instrument element.

- 1 Open SystemTest from MATLAB by selecting **Start > MATLAB > SystemTest > SystemTest Desktop**. You can also just type `systemtest` at the MATLAB command line.
- 2 When SystemTest opens, ensure that the **Launch Test Results Viewer after all results are saved** check box is selected in the **Properties** pane.
- 3 No set-up is required in the Pre Test, so the elements of this test are all in the main test, so click **Main Test** in the **Test Browser**.
- 4 Add an element by clicking **New Element > Instrument Control > To Instrument**.

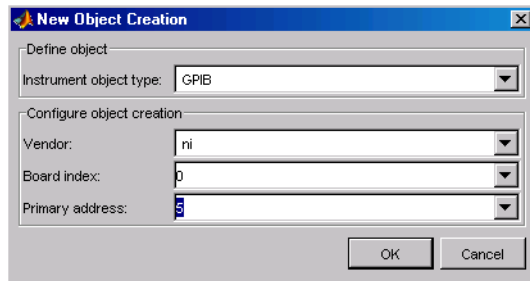


The element appears in the browser as To Instrument.

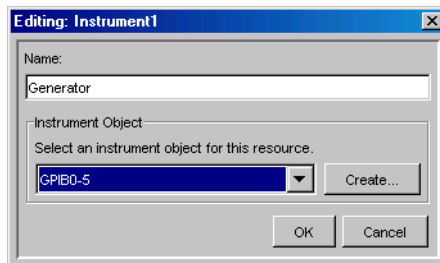
- 5 Double-click To Instrument, rename it Set Generator, and press **Enter**.
- 6 From the **Properties** pane's **Select an instrument resource** list, select New Instrument Resource. The instrument resource is the

communication channel between MATLAB and your instrument, in this case the generator at GPIB address 5.

- 7 In the Editing: Instrument1 dialog box, enter Generator in the **Name** field.
- 8 Click **Create** to create an instrument resource.
- 9 In the New Object Creation dialog box, select GPIB in the **Instrument object type** list. Select the appropriate **Vendor** (in this example, ni for National Instruments), **Board index** (0), and instrument **Primary address** (in this example, 5).



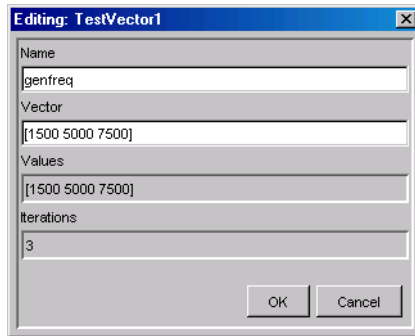
- 10 Click **OK** to return to the Editing: Instrument1 dialog box, where the instrument object is now filled in and selected for this resource (GPIB0-5).



- 11 Click **OK** to apply this resource and return to the **Properties** pane in the SystemTest Desktop.
- 12 In the **Command text** field, enter frequency followed by a space to separate the text from the variable that will follow. This is the command to

set the frequency of the 33120A generator, as described in the instrument's reference manual proved by the vendor.

- 13 Click **Data source** and select **New Test Vector**. The name of the vector you create for setting the generated frequencies is called `genfreq`. Enter that text in the **Name** field, and set the **Vector** field to `[1500 5000 7500]`, including the brackets.

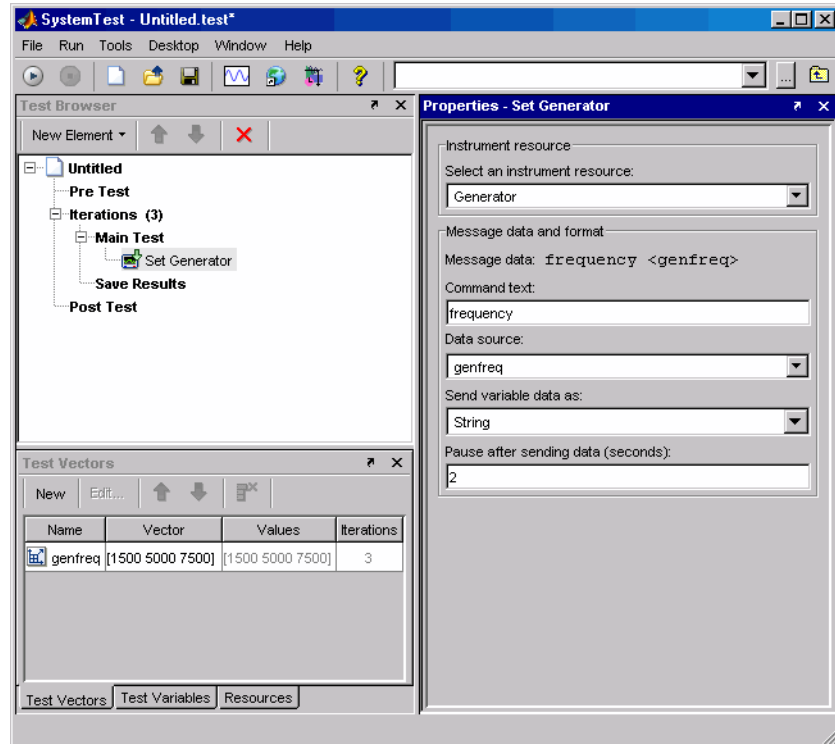


- 14 Click **OK** to return to the SystemTest Desktop.

Notice that the **Iterations** node in the tree now says (3). Because you entered three values in the test vector, the test is comprised of three iterations, one for each frequency value in the test vector.

- 15 Keep the **Send variable data as** setting as **String**. The generator is expecting string values for its commands.
- 16 Set a pause value of 2 seconds. This allows the generator to settle before you measure its output.

The element should now resemble the following figure:



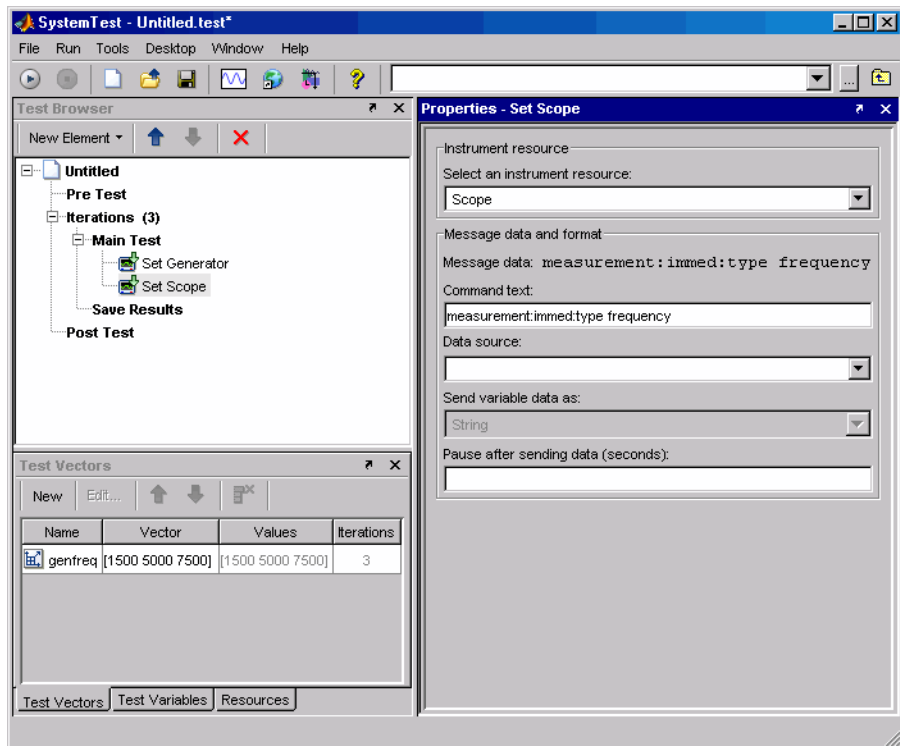
Setting Up the Oscilloscope

You use a To Instrument element, which provides a one-way communication to the oscilloscope, to program the scope to measure frequency.

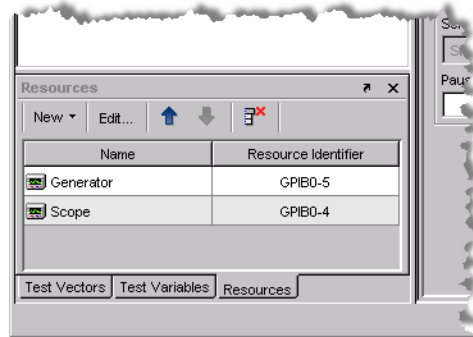
- 1 Add an element by clicking **New Element > Instrument Control > To Instrument**.
- 2 Double-click **To Instrument** in the tree, rename it Set Scope, and press Enter.
- 3 As before, create a new instrument resource, but this time call it Scope. Create a new instrument object for it using Board index 0, and GPIB primary address 4.

- 4 For the command text, enter `measurement:immed:type frequency`. This puts the scope in the frequency measurement mode, as described in the instrument's reference manual provided by the vendor.

There is no test variable or pause required for this element, so the element looks like the following figure:



To see the resources you created for communications with your two instruments, click the **Resources** tab at the bottom of the SystemTest window. You can see the Generator and Scope resources, along with their GPIB settings.

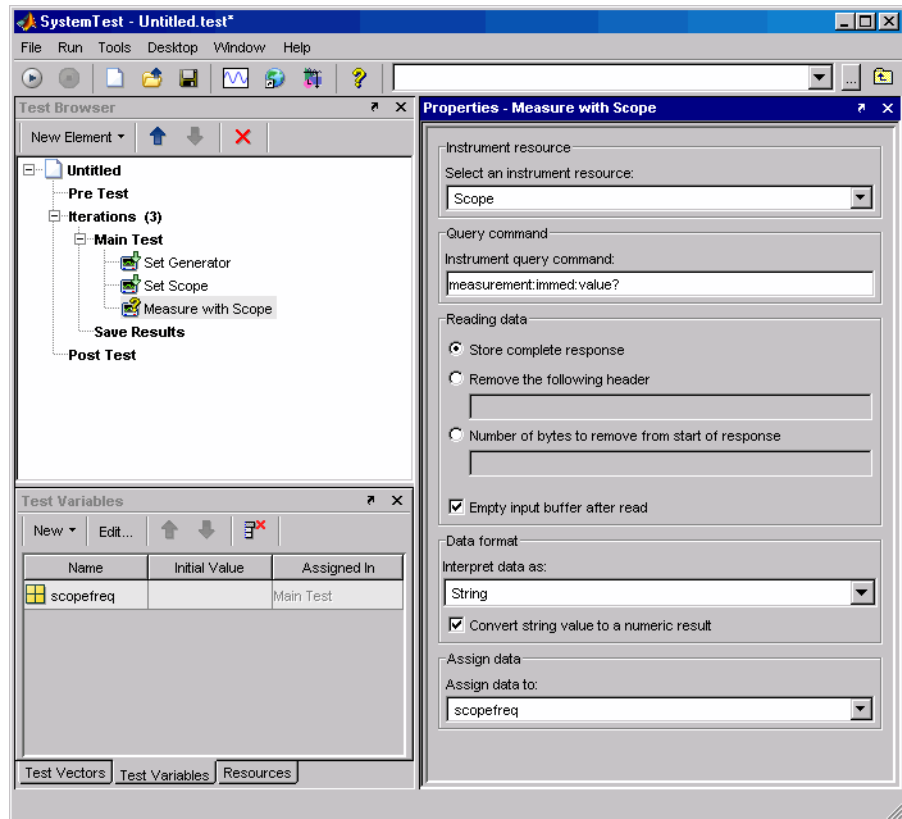


Taking the Measurement

With the generator and scope set up, you can take the measurement with the scope using a Query Instrument element, which sends the command to the scope for taking the measurement.

- 1 Add an element by clicking **New Element > Instrument Control > Query Instrument**.
- 2 Double-click **Query Instrument** in the tree, rename it Measure with Scope, and press Enter.
- 3 Use the existing instrument resource called Scope, by selecting it in the **Instrument resource** list.
- 4 Enter the command to query for a measurement by typing `measurement:immed:value?` in the **Instrument query command** field.
- 5 Select **Store complete response**, and select the **Empty input buffer after read** check box.
- 6 From the **Interpret data as** list, select String (this scope returns ASCII strings), and select the **Convert string value to a numeric result** check box.
- 7 From the **Assign data to** list, select New Test Variable. For the oscilloscope's frequency measurement, name the test variable scopefreq. It needs no initial value.

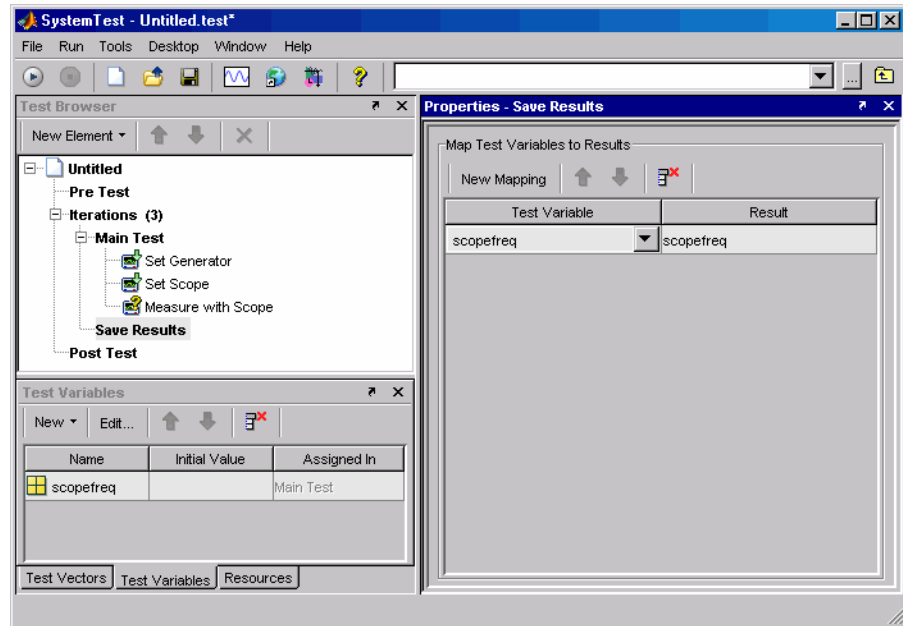
The element now looks like the following figure:



Saving Test Results

To view the results of your test, you must first specify the test variables you wish to save as test results. This is done in the **Save Results Properties** pane.

- 1 Click **Save Results** in the test browser tree.
- 2 In the **Properties** pane, click **New Mapping**.
- 3 From the **Test Variable** list, select `scopefreq`. This test variable contains the frequency measurements provided by the oscilloscope during each Main Test iteration, as shown in the following figure:

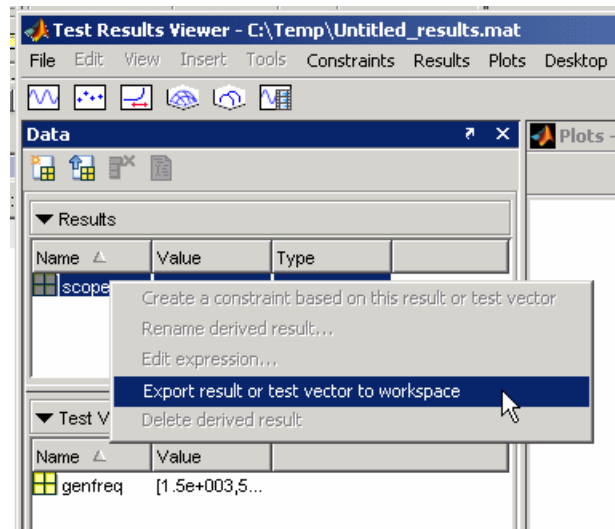


Running the Test and Viewing Test Results

Now that the test elements are all created, you can run the test.

- 1 Run your test. When the test is complete, the Test Results Viewer displays your test results.
- 2 You can explore and plot your test results using the Viewer. Alternatively, in the **Data** pane, right-click the name `scopefreq` and select **Export result or test vector to workspace**. This makes the variable available in your MATLAB workspace.

4 Using the Instrument Control Toolbox Elements



3 To see the measurement results, at the MATLAB prompt type

```
format short g
scopefreq
scopefreq =
    1501.5
    5000
    7500
```

This verifies that the signal generator is producing the expected signal frequencies.

Using the Data Acquisition Toolbox Elements

The Data Acquisition Toolbox provides several elements to use in SystemTest.

Introduction (p. 5-2)

Introduces the Data Acquisition Toolbox elements

Example: Testing a Voltage Regulator (p. 5-3)

Provides a basic example of using Data Acquisition Toolbox elements to test a device

Introduction

Data Acquisition Toolbox Test Elements

This chapter describes how to use the Data Acquisition Toolbox elements with SystemTest.

The Data Acquisition Toolbox elements provide a way to bring analog and digital data from a data acquisition device into a SystemTest test, or to send analog or digital data from your device. You can use these elements along with the other elements in SystemTest to create tests for Simulink models and other applications.

Note To use the Data Acquisition Toolbox elements, you need a license for the Data Acquisition Toolbox. These four elements will not appear in SystemTest without this license.

The Data Acquisition Toolbox provides four elements that you can use in SystemTest:

- Analog Input — For reading analog data from your data acquisition device
- Analog Output — For sending analog data to your data acquisition device
- Digital Input — For reading digital data from your data acquisition device
- Digital Output — For sending digital data to your data acquisition device

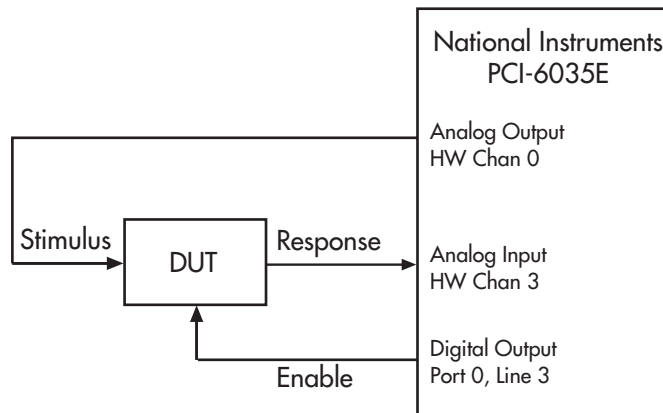
You can configure each test element to communicate with your data acquisition devices for sending or receiving digital or analog data.

Example: Testing a Voltage Regulator

To illustrate how to use some of the Data Acquisition Toolbox test elements in SystemTest, this section provides a step-by-step example. The example shows how to use the elements that send data to a device under test (DUT) and receive data from a device under test, using both analog channels and digital lines.

This example samples the response of a 5-V voltage regulator that is stimulated with three different voltages of 4, 5, and 7.5 volts. The regulator has an enable function controlled by a digital signal. In this example, you collect 22,000 samples per second of the DUT response for 2 seconds.

All data going to and from the DUT is handled by a National Instruments PCI-6035E data acquisition card. The example uses this card's analog output for the DUT stimulus, analog input for capturing the DUT response, and digital output for controlling the DUT's enable line. The test configuration is shown in the following figure:



The steps in this example are

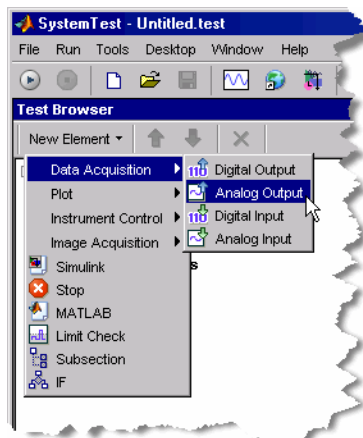
- “Sending Analog Stimulus Data to the DUT” on page 5-4
- “Enabling the DUT with Digital Data” on page 5-6
- “Receiving Analog Response Data from the DUT” on page 5-8

- “Disabling the DUT with Digital Data” on page 5-9
- “Performing Data Analysis” on page 5-10
- “Defining Post Test Elements” on page 5-12
- “Saving and Viewing Test Results” on page 5-13

Sending Analog Stimulus Data to the DUT

Stimulus data is sent to the DUT from an analog output channel of your data acquisition card.

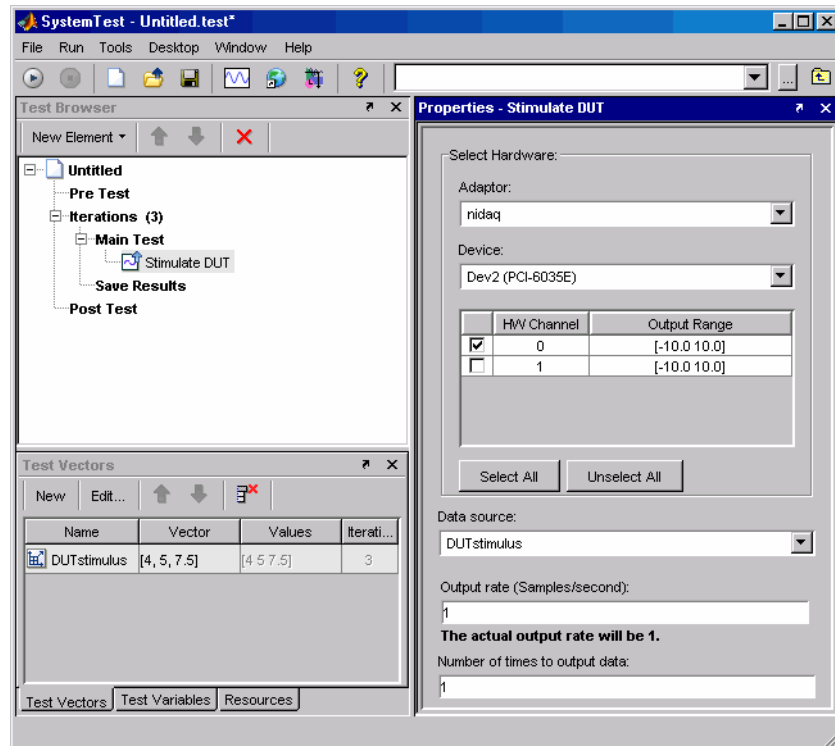
- 1 Open SystemTest in MATLAB by selecting **Start > MATLAB > SystemTest > SystemTest Desktop**. You can also type `systemtest` at the MATLAB command line.
- 2 This example does not use the Pre Test section, so select the **Main Test** section in the **Test Browser** pane.
- 3 Add an Analog Output element by clicking **New Element > Data Acquisition > Analog Output**.



The new element appears in the browser tree, and its properties appear in the **Properties** pane. SystemTest scans your computer for installed data acquisition adaptors and devices. This can take several seconds.

- 4** Double-click the new Analog Output node in the browser tree, and enter a new name for this element, such as Stimulate DUT.
- 5** Since we have three test cases, we need to create a test vector containing the three voltage settings to test against. Click the **Test Vectors** tab. The voltage values for the stimulus to the DUT are held in a test vector. Click **New** to create a new test vector.
- 6** Click the name TestVector1 and enter a new name for your vector, such as DUTstimulus.
- 7** Click the default 1 : 1 : 10 entry in the **Vector** column, and replace it with the values for your test: [4, 5, 7.5] (be sure to include the brackets) and press **Enter**. Notice that because there are three values in your vector, the browser tree now indicates that the Main Test will run three iterations. Each iteration will use one of the three values in the vector for the DUT stimulus voltage.
- 8** In the **Properties** pane, select the adaptor and device to use for the test. This example uses the nidaq adaptor, and the device is a PCI-6035E.
- 9** The example hardware configuration uses the card's analog output hardware channel 0 to provide the DUT's stimulus. So select the check box for this channel. The element will generate signals of 4, 5, and 7.5 volts, so keep the default output range of [-10.0 10.0].
- 10** From the **Data source** list, select the DUTstimulus test vector.
- 11** Enter a value of 1 for **Output rate**. You are using a single static value rather than a sampled waveform, so this is not critical.
- 12** Enter a value of 1 for **Number of times to output data**. The card will hold its last programmed value, so you need to send it only once.

The **Properties** pane now looks like the following figure:



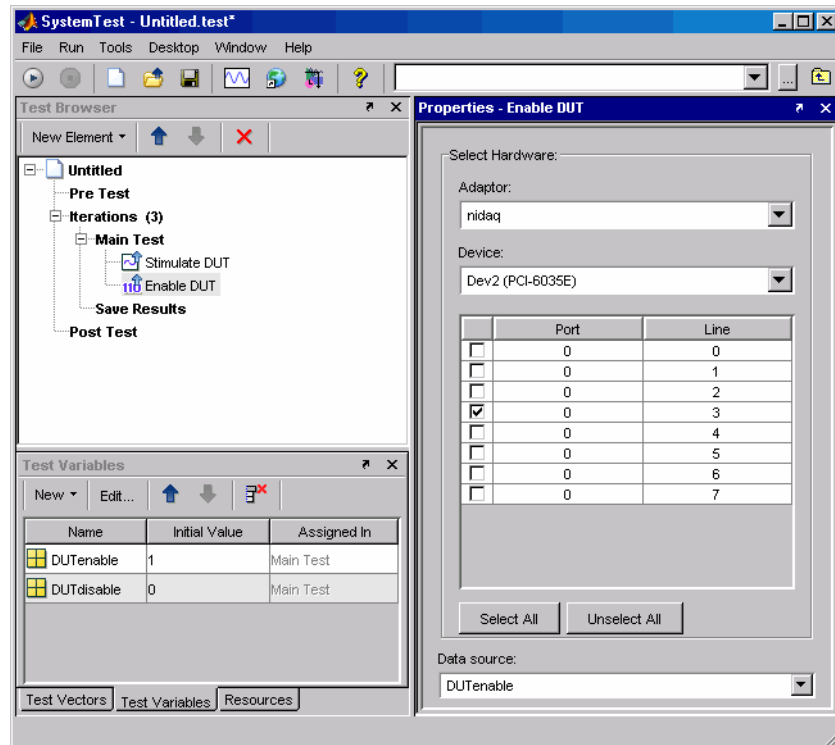
Enabling the DUT with Digital Data

To send a digital enable signal to the DUT, use a digital output element.

- 1 Select **New Element > Data Acquisition > Digital Output**.
- 2 Double-click the new Digital Output element in the browser tree, and type a new name for this element, such as Enable DUT.
- 3 Click the **Test Variables** tab.
- 4 Select **New > Main Test Variable** to create a new variable. You will create two variables: one for enabling and one for disabling the DUT.
- 5 Click the name Var1, and replace it with the text DUTenable.
- 6 Click its empty **Initial Value** entry, and enter 1.

- 7 Repeat steps 4 to 6 to create a second test variable, but name it DUTdisable with an initial value of 0.
- 8 In the **Properties** pane for the Enable DUT element, select the adaptor and device for sending this data. Again, you are using the nidaq adaptor, and the device is a PCI-6035E.
- 9 The hardware configuration uses the card's digital output port 0, line 3 for the enable signal, so select the check box for this line.
- 10 From the **Data source** list, select the variable DUTenable.

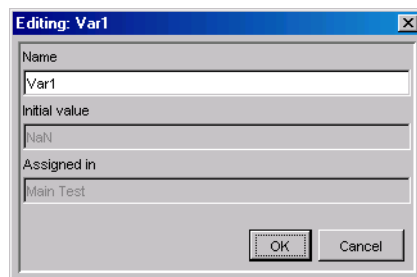
The **Properties** pane now looks like the following figure:



Receiving Analog Response Data from the DUT

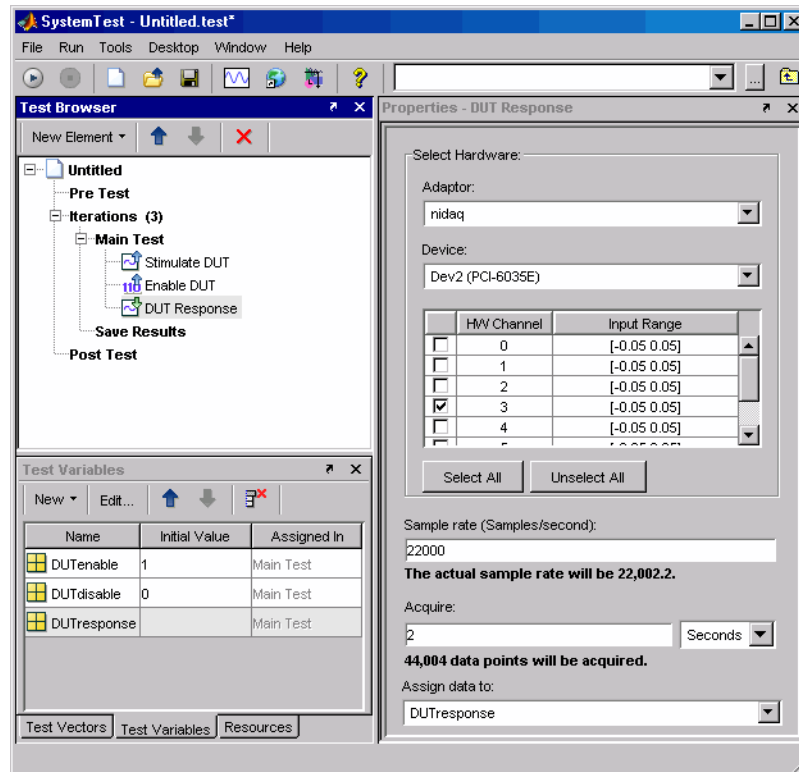
The next element in the test samples the output from the DUT and assigns the acquired data to a test variable.

- 1 Select **New Element > Data Acquisition > Analog Input**.
- 2 Double-click the new **Analog Input** element in the browser tree, and enter a new name for this element, such as DUT Response.
- 3 In the **Properties** pane, select the adaptor and device to use for the test. This example uses the nidaq adaptor, and the device is a PCI-6035E.
- 4 The hardware configuration uses the card's analog input hardware channel 3 to read the DUT's response, so select the check box for this channel. The expected signal will be about 5 volts, so keep the default output range of [-10.0 10.0].
- 5 Set a sample rate of 22000. Because of hardware limitations, the actual sample rate may not exactly match the value you specify.
- 6 In the **Acquire** field, specify to acquire data for 2 seconds. Set seconds in the unit list to the right of the value field.
- 7 In the **Assign data to** field, select New Test Variable from the list. This is where you specify what test variable to assign the acquired data to. The Editing dialog box appears.



- 8 Enter a name for the test variable, such as DUTresponse, then click **OK** to create the test variable.

The **Properties** pane now looks like the following figure:



Disabling the DUT with Digital Data

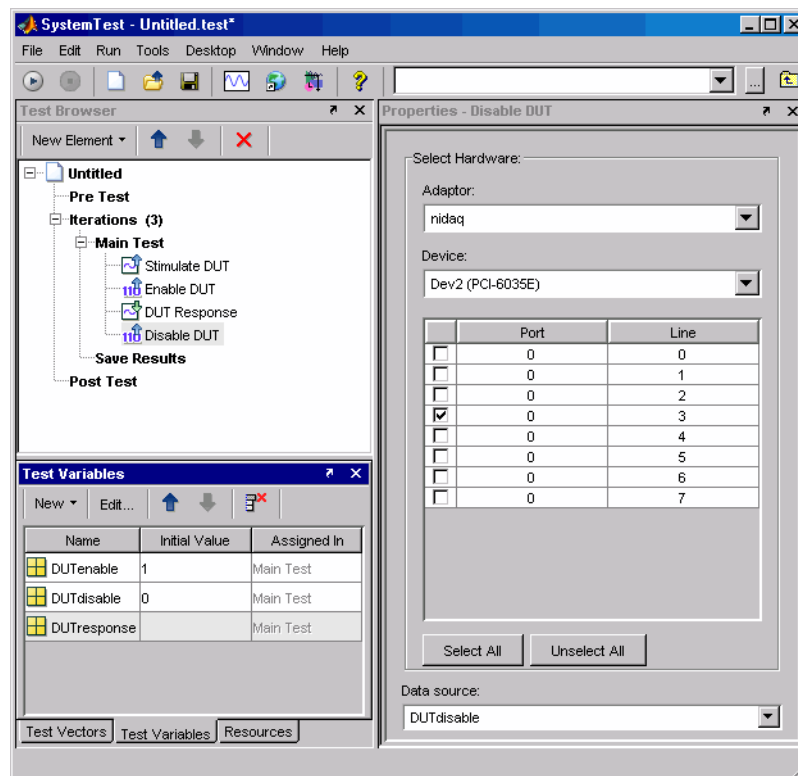
The next step is to disable the DUT with a digital output element that turns off the DUT's enable line. This element is similar to the Enable DUT element, except it sends a different value to the DUT.

- 1 Select **New Element > Data Acquisition > Digital Output**.
- 2 Double-click the new **Digital Output** element in the browser tree, and enter a new name for this element, such as `Disable DUT`.

You already created the test variable `DUTdisable`, which you will use in this element.

- 3 In the **Properties** pane for the Disable DUT element, select the adaptor and device for sending this data. Again, you are using the nidaq adaptor, and the device is a PCI-6035E.
- 4 The hardware configuration uses the card's digital output port 0, line 3 for the enable signal, so select the check box for this line.
- 5 From the **Data source** list, select the variable DUTdisable.

The **Properties** pane now looks like the following figure:

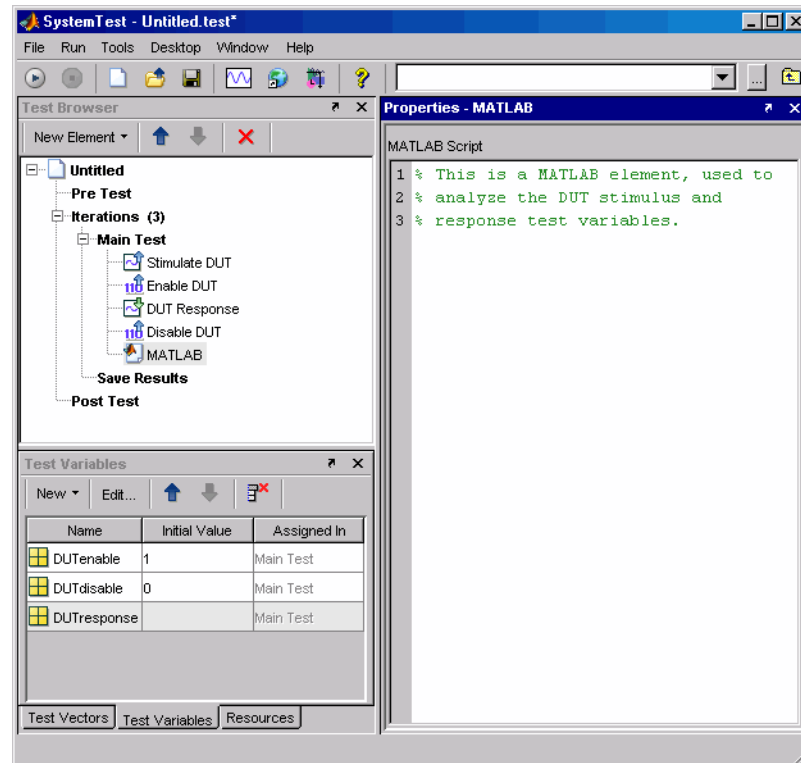


Performing Data Analysis

At this stage, you might perform any analysis or visualization routines on the data generated by the DUT. You can do this in a MATLAB element.

- 1 Select **New Element > MATLAB**.
- 2 Double-click the new **MATLAB** element in the browser tree, and enter a new name for this element, such as Process Data.
- 3 In the **MATLAB Script** edit field of the **Properties** pane, enter any MATLAB code that you need for analyzing your test variables. You might be interested in measuring ripple, noise, regulation, or many other characteristics. You can access the DUT response by referring to the test variable DUTresponse. The stimulus data is available in the test variable DUTstimulus.

The following figure shows a MATLAB element with only some comments added in the **Properties** pane.

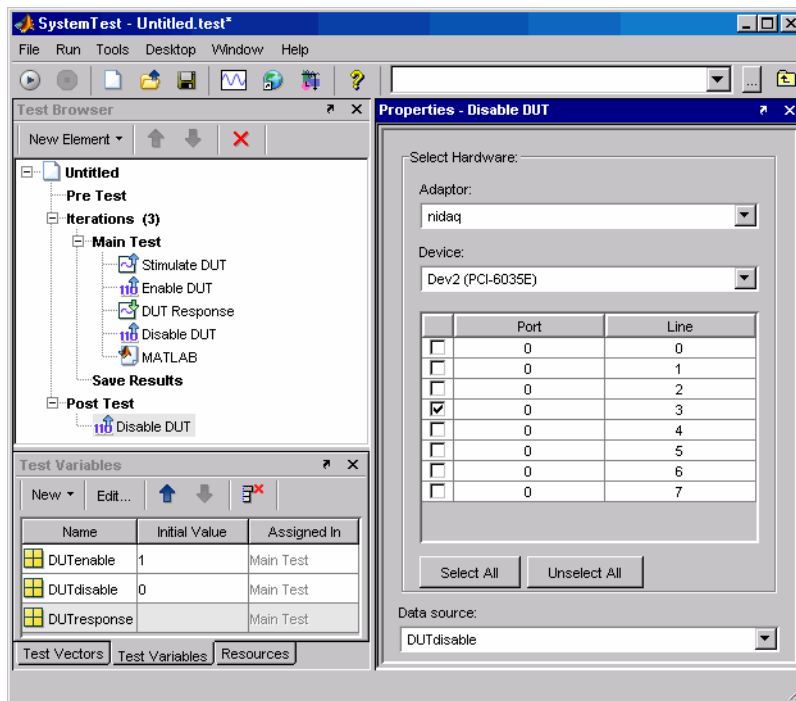


Defining Post Test Elements

In this example, it is recommended to include an element in the Post Test section to disable the DUT.

- 1 Click the **Post Test** section in the browser tree.
- 2 Create a digital output element set up like the element you made in “Disabling the DUT with Digital Data” on page 5-9.

With the extra Disable DUT element, the test now looks like the following figure:



The Post Test section of the test could also perform any analysis that requires completion of all the iterations of the Main Test.

Saving and Viewing Test Results

Before running a test, you must specify which test variables you want to save as a test result. In the **Save Results Properties** pane, you select the test variable that you want to save and map it to a test result name.

Saved test results will be viewable with the Test Results Viewer. To launch the Test Results Viewer, click on the test name in the **Test Browser**. In the **Properties** pane, make sure the **Launch Test Results Viewer after all results are saved** option is checked.

Using the Image Acquisition Toolbox Element

The Image Acquisition Toolbox includes a SystemTest element that you can use to bring live video data into a SystemTest test.

Introduction (p. 6-2)

Introduces the Image Acquisition Toolbox element

Example: Acquiring Video Data in a Test (p. 6-3)

Shows how to acquire video data

Introduction

This chapter describes how to use the Image Acquisition Toolbox element with SystemTest.

The Image Acquisition Toolbox element, called Video Input, provides a way to acquire live video data in a SystemTest test. You can use this element along with the other elements in SystemTest to create tests for Simulink models and other applications.

To learn how to use the Image Acquisition Toolbox element in SystemTest, see “Example: Acquiring Video Data in a Test” on page 6-3.

Note To use the Image Acquisition Toolbox element, you need a license for the Image Acquisition Toolbox. The Video Input element will not appear in SystemTest if you do not.

Example: Acquiring Video Data in a Test

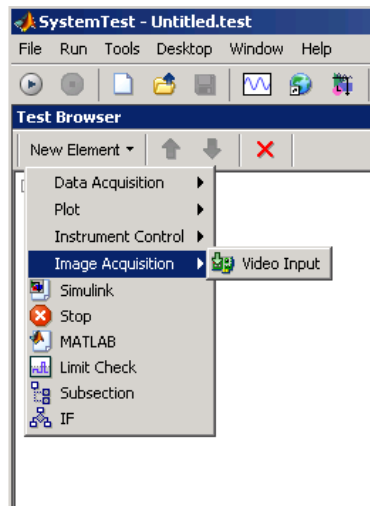
This section provides an example that illustrates how to use the Video Input element in SystemTest. The example uses the Video Input element to acquire a single frame of video for each iteration of the test and uses the MATLAB element to display the acquired image. The steps in this example are:

- “Adding the Video Input Element to a Test” on page 6-3
- “Saving and Viewing Test Results” on page 6-7
- “Running the Test” on page 6-8

Adding the Video Input Element to a Test

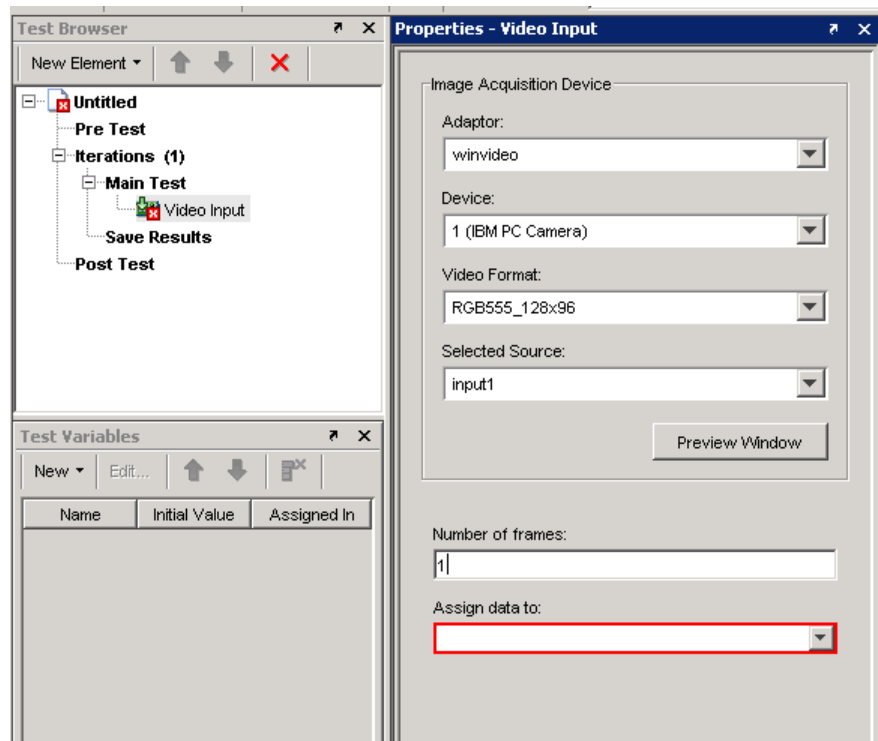
To create a test using the Video Input element:

- 1 Open SystemTest by selecting **Start > MATLAB > SystemTest > SystemTest Desktop** in MATLAB. You can also just type `systemtest` at the MATLAB command line.
- 2 In the SystemTest Desktop, start to create your test by selecting **Main Test** and adding the Video Input element. In the **Test Browser**, click **New Element > Image Acquisition > Video Input**.



SystemTest adds the Video Input element to the Main Test section of the test and displays the **Properties** pane for the Video Input element. (You can also add elements to the Pre Test or Post Test sections of a test but this example does not require it.)

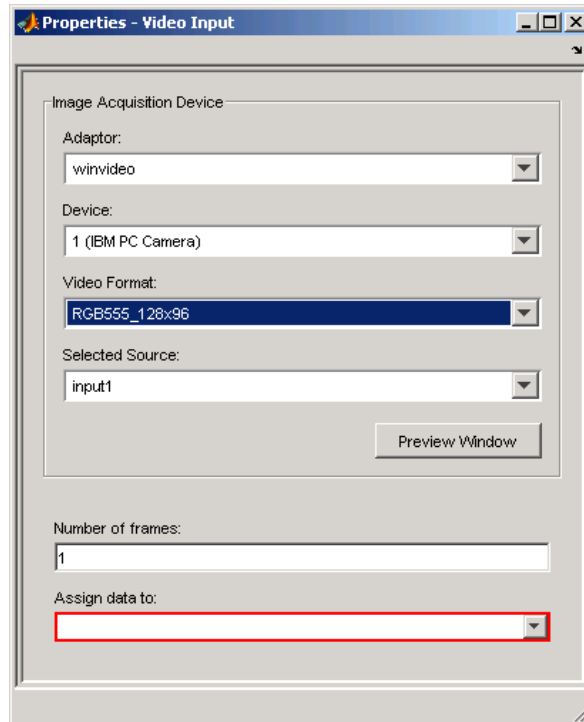
In the following figure, note the red x in the Video Input element icon in the Test Browser. This red x indicates that the element is in an error state. SystemTest outlines the required fields in red in the **Properties** pane.



- 3 Specify the device you want to use to acquire image data in the **Properties** pane for the Video Input element. You must specify the name of the adaptor you want to use in the **Adaptor** field, which is a required field. (SystemTest uses red outlining to indicate required fields that are not filled in yet.) SystemTest can detect any image acquisition devices supported by the Image Acquisition Toolbox that are connected to your system and fills in this field with a default value based on the alphabetical list of devices, if one

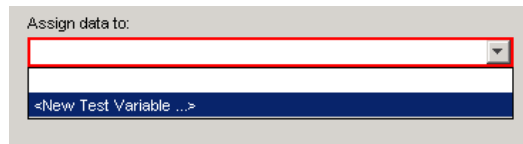
is available. For our example, in the figure, SystemTest sets the **Adaptor** field to winvideo. If your system has other adaptors that can connect to devices, select the adaptor that you want to use from the **Adaptor** list.

After the **Adaptor** field is set, SystemTest fills in the **Device**, **Video Format**, and **Selected Source** fields with default values. SystemTest populates the drop-down lists associated with each field with all available options for the field. Adaptors can support multiple devices and devices can support multiple formats. SystemTest preselects the default values for these fields but lists all available options in the drop-down lists associated with these fields. The following figure shows the list for the **Video Format** field:

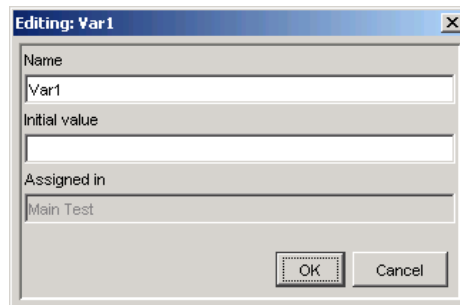


- 4 Specify the number of frames you want to acquire at each iteration of the test in the **Number of frames** field, which is a required field. For this example, we only need to acquire one frame for each iteration, so set this field to 1.

- 5 Specify the name of the SystemTest test variable that the acquired video data will be assigned to at each iteration. This is a required field. You can select a test variable from the list or create a new test variable by selecting **New Test Variable**.

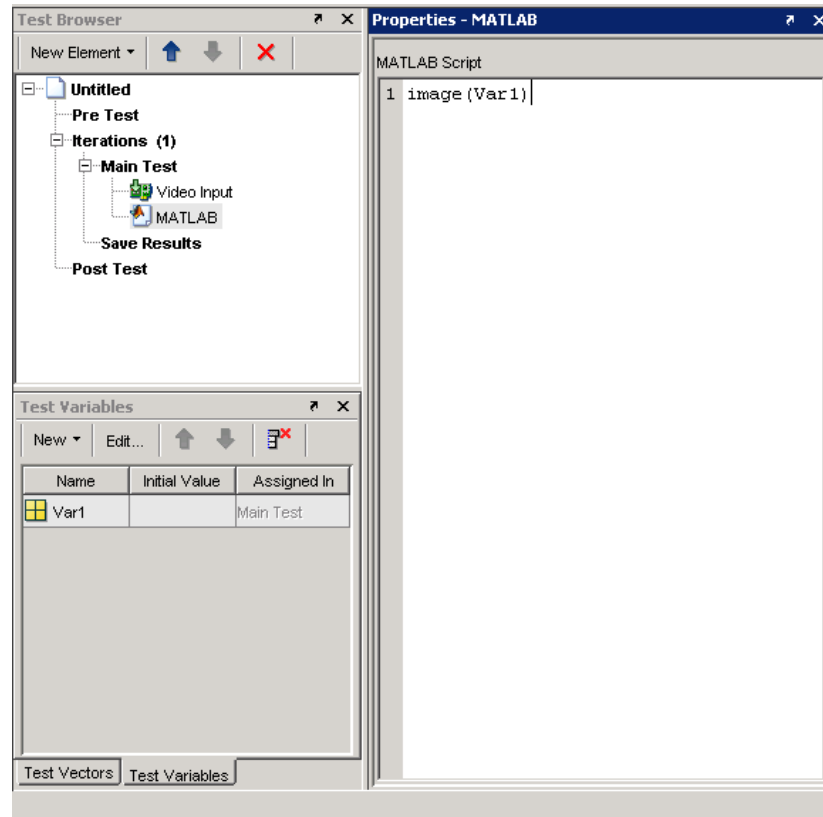


If you select **New Test Variable**, SystemTest opens the Editing dialog box. Assign a name to the test variable, or accept the default name, and click **OK**. You do not need to assign the test variable an initial value.



SystemTest adds the new test variable to the list in the **Test Variables** pane.

- 6 Optionally, verify the Video Input element settings by clicking the **Preview Window** button. SystemTest opens a Video Preview window and displays a live video stream from your camera. You can use this to verify that your hardware is configured correctly. You should close the preview window before running the test.
- 7 To complete this example test, add a MATLAB element to the Main Test section. In this MATLAB element, call the MATLAB image function to display the image frame acquired at each iteration.

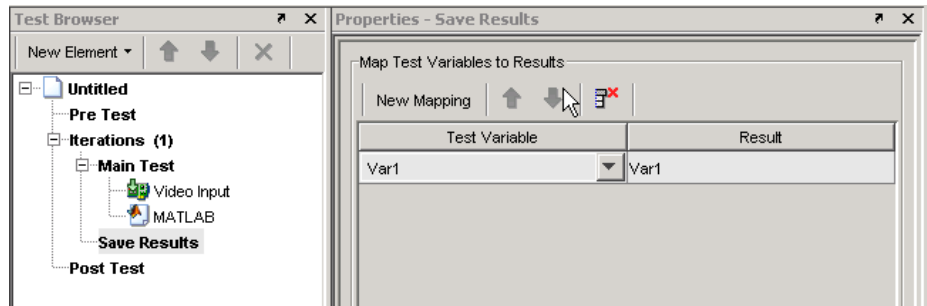


This completes this example test illustrating how to incorporate image data into SystemTest. In a real testing application, you can define test vectors that determine the number of iterations of your test that SystemTest performs. You can also compare test variables against defined limits in the Limit Check element and specify pass/fail criteria.

Saving and Viewing Test Results

Before running a test, you must specify which test variables you want to save as a test result. In the **Save Results Properties** pane, you select the test variable that you want to save and map it to a test result name.

Saved test results will be viewable with the Test Results Viewer. To launch the Test Results Viewer, click the test name in the **Test Browser**. In the **Properties** pane, make sure the **Launch Test Results Viewer after all results are saved** option is selected.

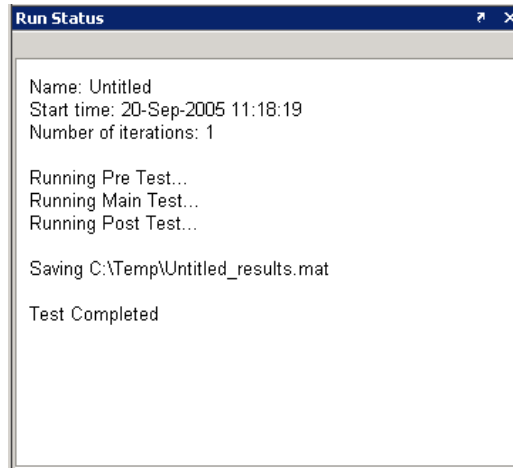


Running the Test

To run the test, do one of the following:

- Click the **Run** button.
- Select the **Run > Run** menu.
- Press the **F5** key.

While the test executes, SystemTest reports on the progress of the test in the **Run Status** pane.



After the test runs, the Test Results Viewer will launch. In the Viewer, select the type of plot you want to create. For this example, select **Image Plot** from the **Plots** menu or click the **Image Plot** button in the Test Results Viewer toolbar.

Using the Test Results Viewer

This chapter explains how to use the Test Results Viewer to explore and analyze your test results.

Before You Begin (p. 7-2)

Prerequisite steps to take if you intend to perform the examples provided with the procedures in this chapter

A Quick Tour of the Test Results Viewer (p. 7-4)

A quick overview of the Test Results Viewer

Viewing Your Test Results (p. 7-6)

How to render plots of your saved test results

Refining Your Test Results (p. 7-24)

How to use constraints to filter the test results shown in your plots

Viewing Simulink Time Series Data (p. 7-32)

How to plot Simulink values with time data

Before You Begin

The examples in this chapter use saved test results from the Throttle demo. You can follow the explanations by loading and running the Throttle demo from the MATLAB command line. The Throttle demo is configured to open the Test Results Viewer upon completing a test.

See the SystemTest Demos page for an explanation of the Throttle demo.

Note This demo will not be listed if you do not have Simulink installed.

To prepare for the rest of this chapter:

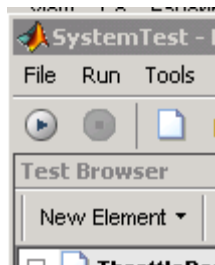
- 1 Start MATLAB.
- 2 In MATLAB, select **Start > Demos** to open the Help browser.
- 3 Expand the **MATLAB** list from the left frame of the browser.
- 4 Click **SystemTest**. The SystemTest demos open in the right browser frame.
- 5 Click “Simulink - Validating a Throttle Body Model.” An overview of the demo opens.
- 6 Click the link “Open the demo in the SystemTest Desktop” at the bottom of the page.

Alternatively, you can enter the following command at the MATLAB command line:

```
systemtest ThrottleDemo
```

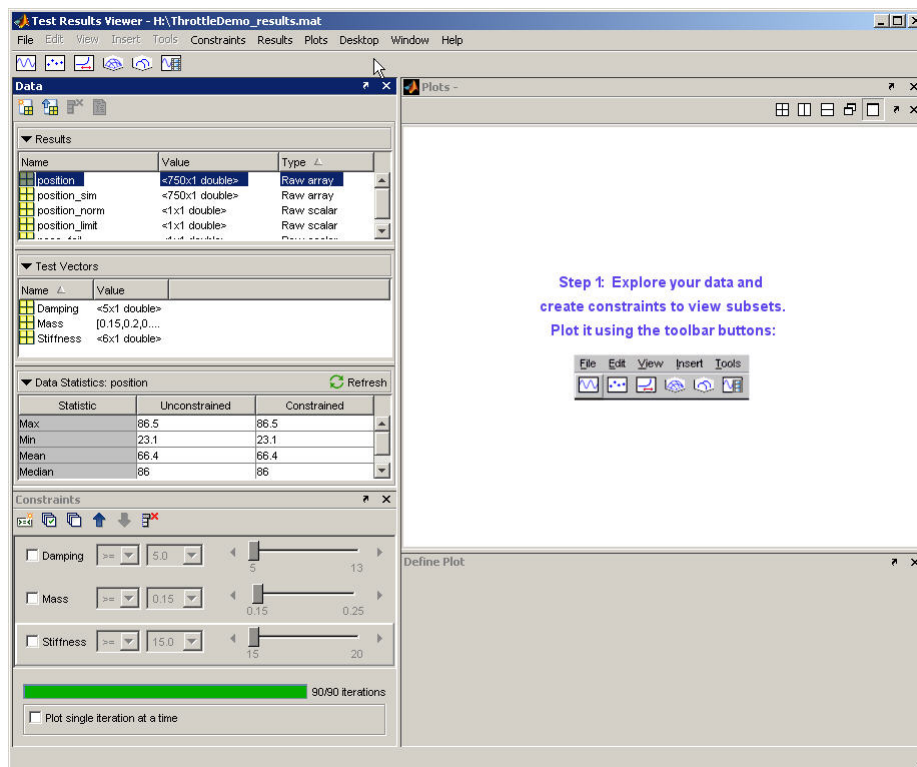
After the SystemTest Desktop appears, run the loaded test. Do one of the following:

- Click the **Run** button.



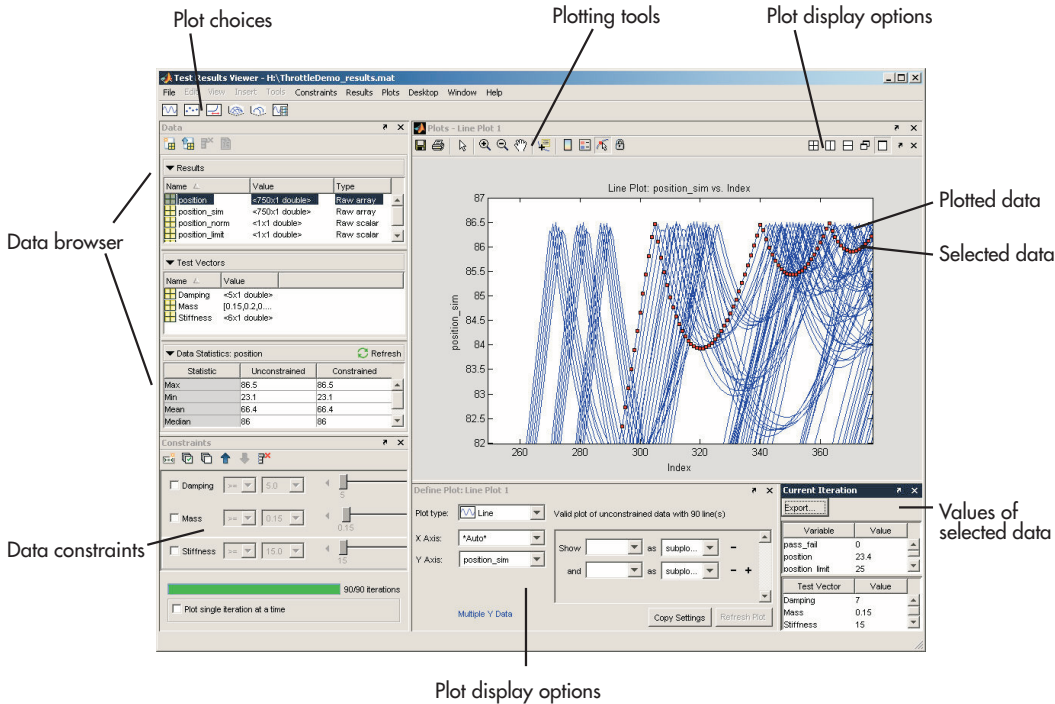
- Or press the **F5** key.
- Or select the **Run > Run** menu.

SystemTest runs the Throttle demo test, saves the specified test results, and opens the Test Results Viewer when it finishes.



A Quick Tour of the Test Results Viewer

The Test Results Viewer is organized to show you the test vectors you specified as inputs to your test, the results saved from your test, and tools you can use to plot and examine your test results.



The test results and test vectors from your test are available in the **Data** pane, which is a compact data browser. You choose your plot type, set your display options to include what appears on the different axes, and plot your data. The plotting tools let you select data from the plot to examine, and you can see the actual values that resulted in individual plot points in the **Current Iteration** pane, which will open automatically when you select a plot point. If this pane is not visible, select **Desktop > Current Iteration**.

“Viewing Your Test Results” on page 7-6, “Refining Your Test Results” on page 7-24, and “Viewing Simulink Time Series Data” on page 7-32 explain how to use the Test Results Viewer to plot and examine your test results.

Viewing Your Test Results

This section explains how you analyze your test results by using the Test Results Viewer. It contains the following sections:

- “Reserved Keywords” on page 7-6
- “Browsing Results” on page 7-6
- “Generating Plots” on page 7-7
- “Exploring Plots” on page 7-11

Reserved Keywords

The Test Results Viewer has several reserved keywords that you cannot use as a test result name or as a derived result name. These keywords are:

- time
- testrun
- testruns
- metadata
- data

If any of these keywords are used as a test result name, they will be prepended with "st_" when loaded in the Test Results Viewer. If you try to use these keywords as a derived result name in the Test Results Viewer, you will get an error dialog.

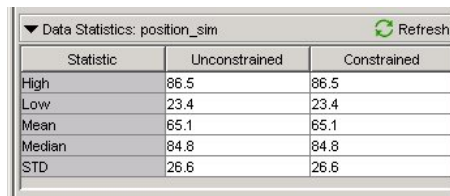
Browsing Results

“Viewing Test Results in the Test Results Viewer” on page 1-30 notes that the Test Results Viewer contains a data browser within the **Data** pane. This area of the viewer is one of the first things you see when the Test Results Viewer opens. It shows you the test variables and test vectors your test used, and it shows information about their values in the **Data Statistics** area.

These data statistics summarize the values of a test result or test vector across all of the tests. For example, the Throttle demo varies the parameters for mass, damping, and stiffness of a Simulink model. Test vectors vary

Simulink block parameters for 90 test iterations, and SystemTest saves how these changes affect the position of a simulated throttle opening in the `position_sim` test result.

If you click `position_sim` in the **Results** area of the **Data** pane, the **Data Statistics** area shows you a summary of statistical information for all 90 iterations. In this example, you have not defined any constraints on your data, so statistical information for the constrained and unconstrained columns is the same. See “Creating and Applying Constraints” on page 7-24.



Statistic	Unconstrained	Constrained
High	86.5	86.5
Low	23.4	23.4
Mean	65.1	65.1
Median	84.8	84.8
STD	26.6	26.6

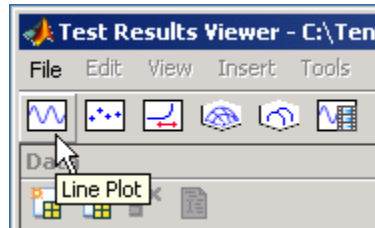
“Generating Plots” on page 7-7 explains how you can further explore your test results.

Generating Plots

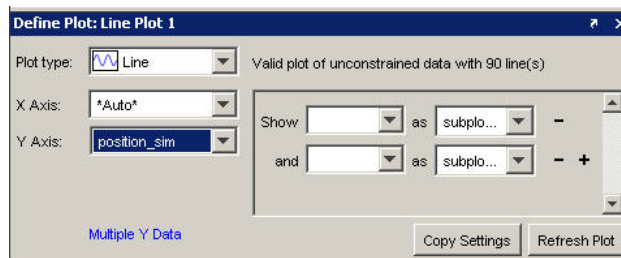
The Test Results Viewer has a plotting capability that helps you understand your test results. You can determine how values of different inputs (test vectors) affect the overall test results.

To generate any plot:

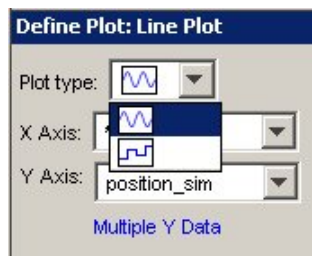
- 1 Click the button corresponding to the type of plot you want to generate. The plot buttons are below the menu bar. For example, click the **Line Plot** button. See “Choosing a Plot” on page 7-9 for an explanation of your choices. You can also use the **Plots** menu to generate plots.



- 2 Choose the data to use for your X-axis and Y-axis in the **Define Plot** pane. For example, select ***Auto*** from the **X Axis** list and **position_sim** from the **Y Axis** list to show the simulated throttle position trajectories at each test iteration. See “Choosing a Plot” on page 7-9 to understand which data types are available on each axis.

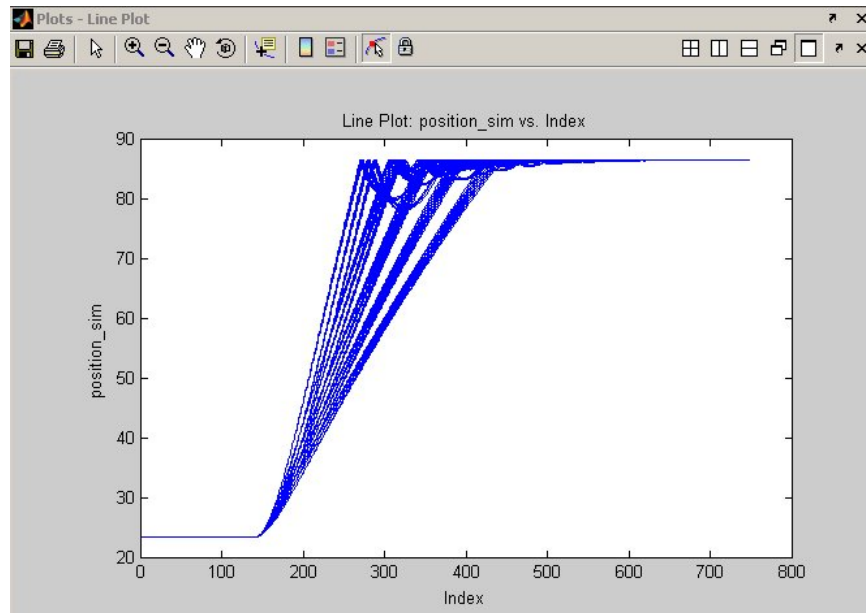


- 3 Choose a different plot type if you do not want to use the default. To choose a different plot type:
 - a Click **Plot type** in the **Define Plot** pane.



- b Click the plot type you want to use. For the Throttle demo example, use the default sine wave.

- 4 Click the **Plot** button. The Test Results Viewer renders a plot based on your selections.



Each line in the plot represents a test iteration. If it appears that there are not as many lines as you had test iterations, it is possible that two or more iterations generated similar enough results that they overlap.





Now you can analyze the plot. To help you with this task, you can:



- Explore the plot using the plotting tools available to you as explained in “Exploring Plots” on page 7-11.
- Refine what results are shown in your plot as explained in “Refining Your Test Results” on page 7-24.

Choosing a Plot

There are six types of plots. The line plot, mesh plot, and time series plot types have additional subtypes available. Additionally, the Test Results Viewer has rules for determining which test results you can plot on the X-axis,

Y-axis, and Z-axis. These rules vary by plot type. The following table explains these selections:

Plot	Description
<p>Line</p> 	<p>Standard line plot of Y versus X. Represents scalar or vector data. The default is a wave line, but you can choose a square line sub type. The following data are allowed on each axis:</p> <ul style="list-style-type: none"> • X — Numeric test vectors • Y — Numeric test results
<p>Surf</p> 	<p>Wireframe surf plot based on X, Y, and Z coordinates. Optional surface sub type available. The following data are allowed on each axis:</p> <ul style="list-style-type: none"> • X — Numeric test vectors • Y — Numeric test vectors • Z — Numeric test results
<p>Scatter</p> 	<p>Standard scatter plot of X and Y where either axis can have numeric test vectors or numeric test results.</p>
<p>Time Series</p> 	<p>Plots time series data Y against time (X is always time). Designed to represent Simulink time series object data. The default is a wave line, but you can choose a square line sub type. See “Viewing Simulink Time Series Data” on page 7-32 for more information about this plot type.</p>

Plot	Description
Waterfall 	<p>Waterfall plot for vectors or time series. One vector or time series can be displayed on each waterfall plot. The meaning of the X, Y, and Z axes is as follows:</p> <ul style="list-style-type: none"> • X — Is automatically selected to be “*Auto*” if the Z axes is assigned to a vector-valued test result, or “Time” if Z axes is assigned to a time series test result. • Y — You can select either Test Run or Iteration. In the former case, if a test is excluded by application of constraints a gap will appear in the waterfall plot at the Y position corresponding to that test. In the latter case, lines representing the test result displayed on the Z axis are always placed in consecutive Y positions. • Z — You can select either a single vector-valued numeric test result or a single time series test result.
Image 	<p>Lets you look at individual frames from an image sequence saved during a test iteration. Data must be a supported MATLAB Image format, and must be numeric test results whose size is compatible with an image, namely that:</p> <ul style="list-style-type: none"> • It has three or four dimensions. • The third dimension has a length of 1 or 3.

Exploring Plots

This section describes the tools the Test Results Viewer makes available to help you understand its generated plots. It contains the following topics:

- “Plotting Tools” on page 7-12 describes the tools available to help you examine and understand the contents of a generated plot.
- “Viewing Individual Iteration Values” on page 7-12 shows how to focus on specific iteration test results in a plot.
- “Highlighting Values in Your Plot” on page 7-16 shows how to distinguish test results in a plot.

- “Exposing Overlapping Plot Lines” on page 7-20 explains how you can view individual lines in a plot that shows multiple test result values as the same line.

Plotting Tools

The Test Results Viewer integrates the MATLAB Figure Toolbar, which lets you examine and distinguish the test results shown in your plots. See “Plotting Tools—Interactive Plotting” and “Data Exploration Tools” in the MATLAB Graphics documentation for more information.

In addition, the viewer also supports the desktop arrangement tools available in the MATLAB editor. See “Arranging the Desktop — Overview” in the MATLAB documentation.

The Test Results Viewer adds the following features to the MATLAB Figure Toolbar:

- Test run selection — Lets you click different test runs in the plot and see the test vector and test results for that iteration in the **Current Iteration** pane. “Viewing Individual Iteration Values” on page 7-12 shows an example of how to use this.
- Lock the plot — Prevents constraints from changing the test results displayed in the plot.

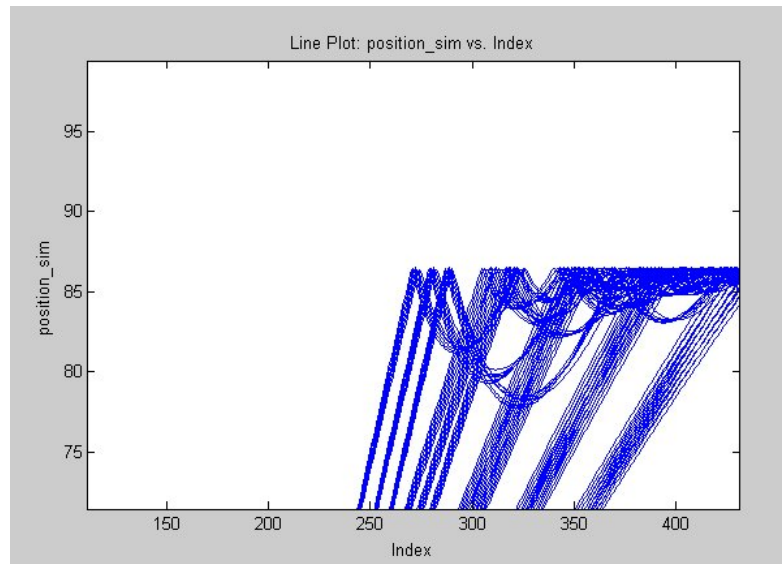
Viewing Individual Iteration Values

Every test iteration has its own representation in a plot unless you screened it out with a constraint (“Refining Your Test Results” on page 7-24 explains constraints). By clicking a line, marker, or surface in a plot with the test run selection tool, you can see the information associated with that test iteration in the **Current Iteration** pane.

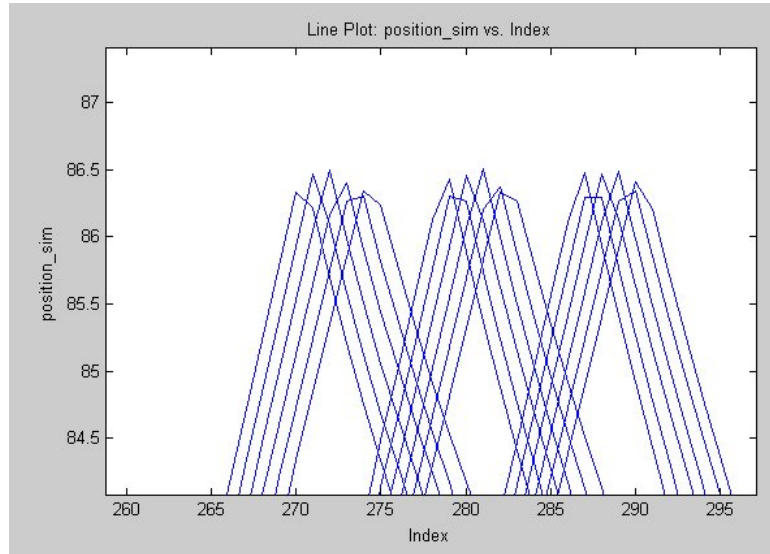
For example, “Generating Plots” on page 7-7 demonstrates how to generate a plot showing all test iteration results of the Throttle demo. You can use the Test Results Viewer plotting tools to zoom in on areas of the plot and determine which iteration was responsible for the result.

- 1 Click the **Zoom In** button.

- 2** Move the mouse pointer over an area of the plot you want to investigate further.
- 3** Left-click your mouse or click and drag over the area you want to see. The plot redraws to show this area.



You can repeat zooming in until you have the level of detail you want.

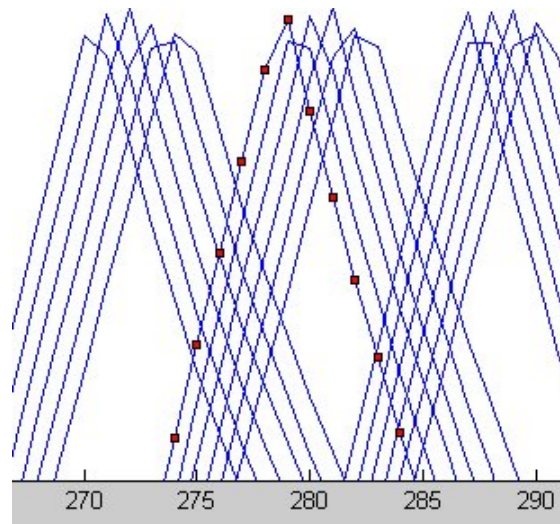


4 To turn off the Zoom, click the **Zoom In** button again.

5 Click the **Select an iteration** button in the Figure Toolbar.



6 Click one of the plotted lines in the line plot. The viewer marks the line.



The viewer simultaneously populates the **Current Iteration** pane with information about the values for all test vectors and test results for your selected test iteration. This lets you easily see what test conditions generated a specific result.

Current Iteration	
Result	Value
pass_fail	0
position	<750x1 double>
position_limit	25
position_norm	97.9

Test Vector	Value
Damping	5
Mass	0.2
Stiffness	15

Highlighting Values in Your Plot

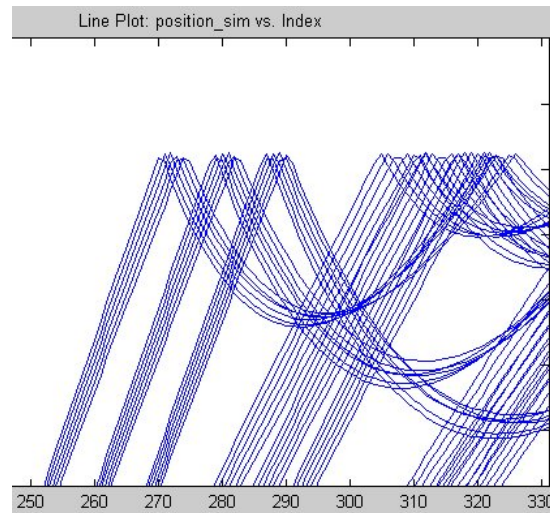
The Test Results Viewer lets you further distinguish your test results for any given plot by letting you control how a plot renders the data on each axis. This is useful in deciphering test results on a plot—especially when the initial plot has a large number of test results closely grouped together. This section explains how you use the **Define Plot** pane to modify the appearance of your plot without modifying the underlying test results. (See “Refining Your Test Results” on page 7-24 for information about modifying the test results used to render a plot.)

The **Define Plot** pane provides four ways to distinguish plotted test results:

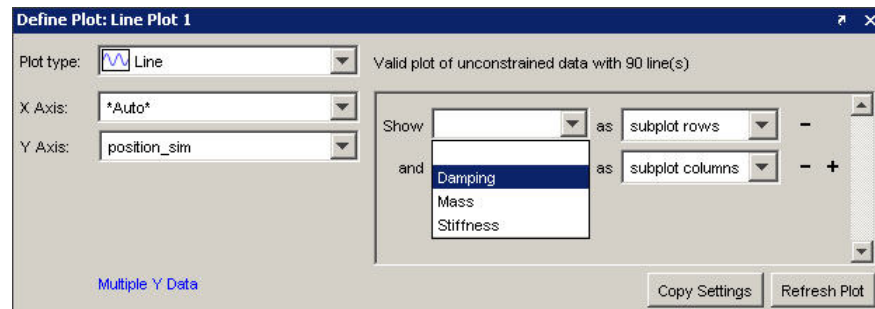
- Color
- Markers
- Subplot rows
- Subplot columns

For example, the Throttle demo shows the effect of variations in mass, damping, and stiffness on a component of a Simulink model. The plot you generated in “Generating Plots” on page 7-7 shows test results for of all test iterations, but it is impossible to determine how changes to each test vector affected this outcome. To distinguish the test results on the plot:

- 1 Zoom in on an area of the line plot so that you can see individual test iterations (as explained in “Viewing Individual Iteration Values” on page 7-12).

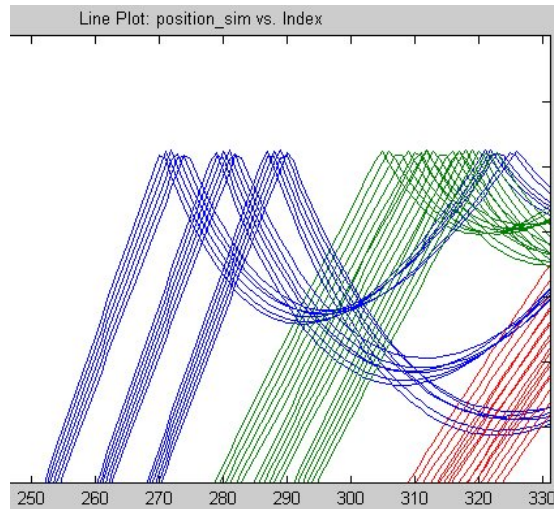


2 In the **Define Plot** pane, click **Show > Damping**.



3 Select **color** from the **as** list.

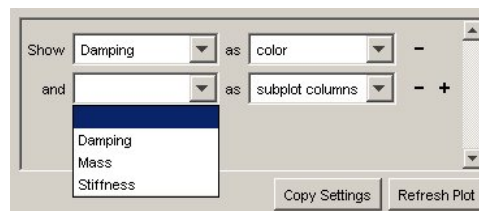
4 Click the **Refresh Plot** button. The plot lines change to show a range of colors.



You now have some idea how damping has affected the test results. You have a cluster of blue, green, and red indicating that damping is the same value in each cluster, which you can confirm by using the test selection tool to choose lines and by viewing the value for the Damping test vector in the **Current Iteration** pane.

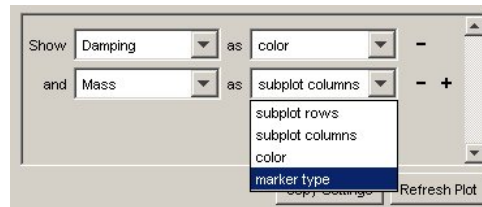
You can modify the appearance of another set of test vectors to further understand the test results. For example, the menu below **Damping** can be used to distinguish variations in mass with markers.

1 Click the menu next to **and**.



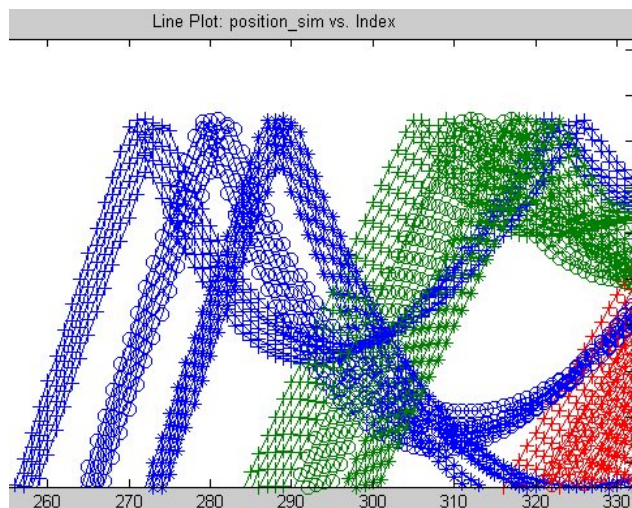
2 Select **Mass** from the list.

3 Click the menu next to **as** and select **marker type**.

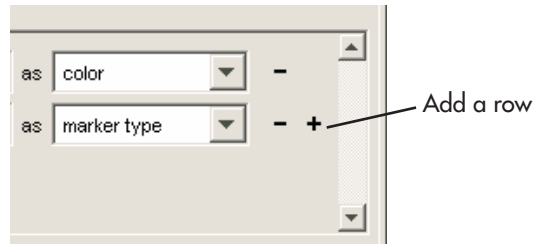


4 Click the **Refresh Plot** button.

The viewer redraws the plot to show markers distinguishing variations in mass. Notice how each cluster of lines has its own unique color and marker, which shows that variations in damping and mass have a visible effect when you run the model.



You can add two more rows using the **+** button in the **Define Plot** pane to distinguish your test results further.

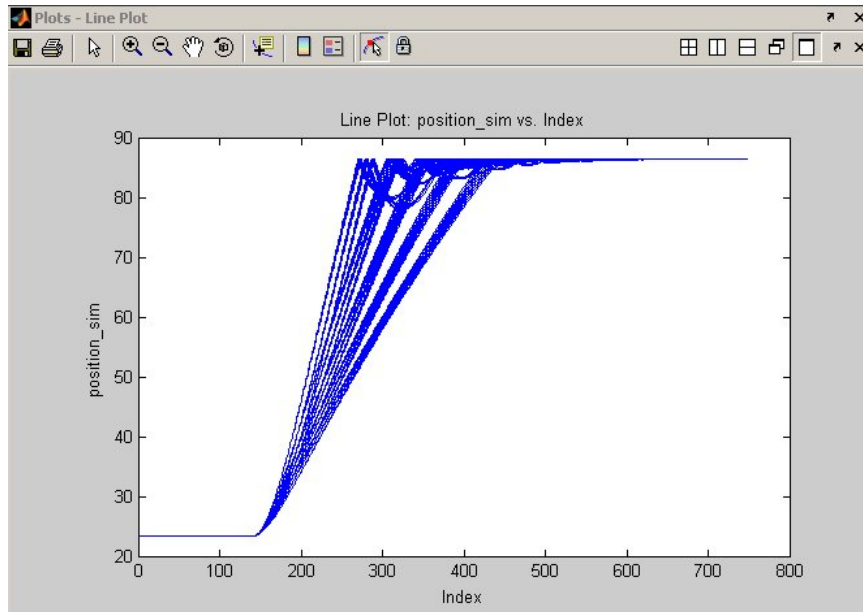


Note These colors and markers do not necessarily show the same value throughout the overall plot. The viewer cycles through all colors and markers in the palette making it possible for different test result values to have the same color or marker.

Exposing Overlapping Plot Lines

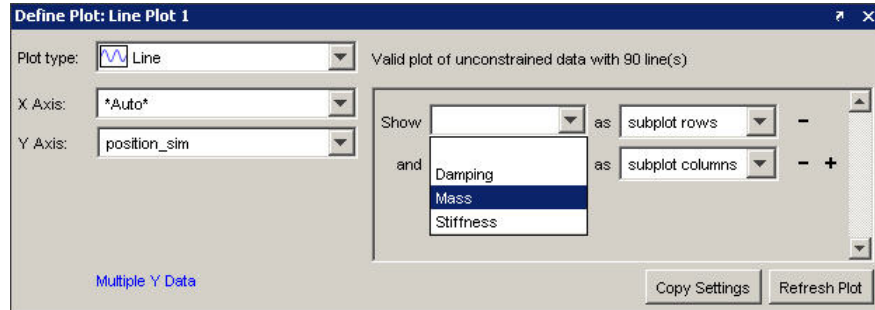
It is possible for plot lines and points to overlap and appear undistinguishable. When multiple lines overlap, you can create subplots to distinguish the data points.

For example, if you create a line plot for the Throttle demo with the X-axis set to ***Auto*** and the Y-axis set to **position_sim**, the Test Results Viewer renders a plot with plot lines in close proximity.

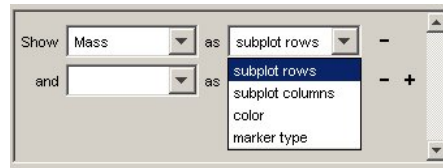


This plot has 90 lines that are too close together to be able to discern clear patterns. You can use the **Define Plot** pane to distinguish plots of test results by placing the generated lines of a test in individual subplots. Each subplot shows the test vector values associated with the test results being plotted. The number of runs per test vector value determines how many subplots you can generate. Using the Throttle demo, you can generate subplots based on changes in damping, mass, or stiffness. For example, what effect did changes in mass have on these test results? To show its effect:

1 In the **Define Plot** pane, select **Show > Mass**.

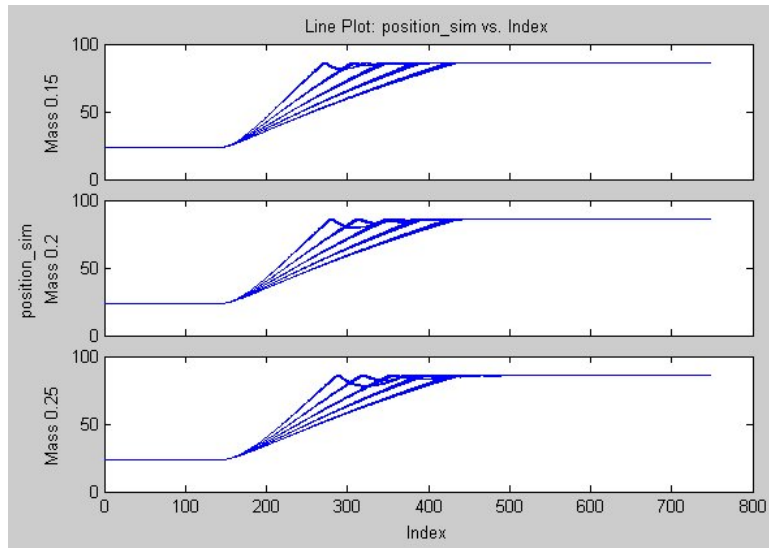


2 Select **subplot rows** from the **as** list.



3 Click the **Refresh Plot** button.

The viewer now shows three subplot diagrams, one for each value of the Mass test vector.



Refining Your Test Results

This section explains how you create and apply constraints to restrict the test results to a subset of test iterations. You also see how to use a constraint to walk through a set of test results. This section contains the following topics:

- “Creating and Applying Constraints” on page 7-24
- “Plotting Single Iterations” on page 7-30

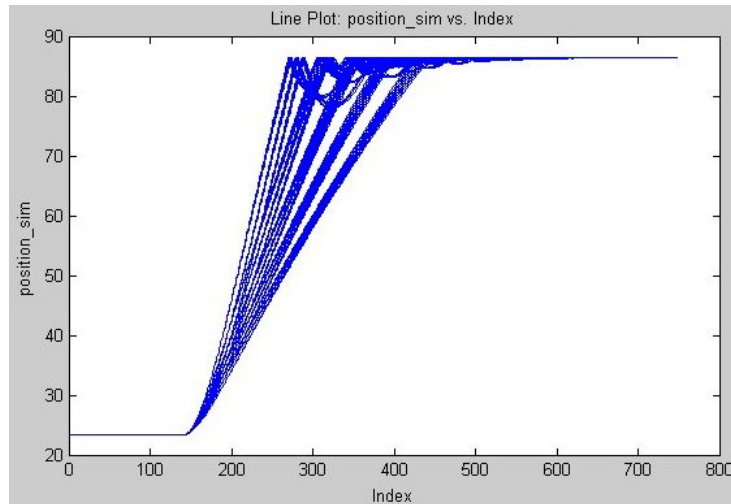
Creating and Applying Constraints

Constraints are a Test Results Viewer mechanism that screen out test result values. Constraints can be a single value, a range, or an evaluated expression. Applied constraints result in plots rendered from a subset of test iterations, and the viewer applies constraints immediately to all plots. This is useful when you want to screen out or filter test results in your attempts to find or understand the results of a test.

Using Default Constraints

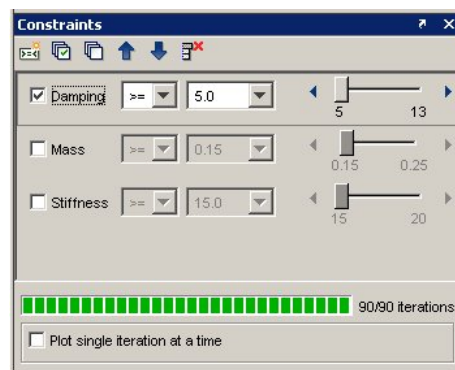
The Test Results Viewer, when opened after a test run, has constraints present but not applied. The viewer creates a constraint for each test vector and defines the constraint’s range as a function of the full range of values in the test vector. These default constraints let you see the immediate effect of your test’s test vectors on the results of the test.

For example, the Throttle demo has three test vectors corresponding to changes in damping, mass, and stiffness to a Simulink model. If you display a line plot as explained in “Generating Plots” on page 7-7, you get a plot similar to the following:



This output shows that the test results group in small clusters. You can use a constraint to see which of the test vectors cause this clustering.

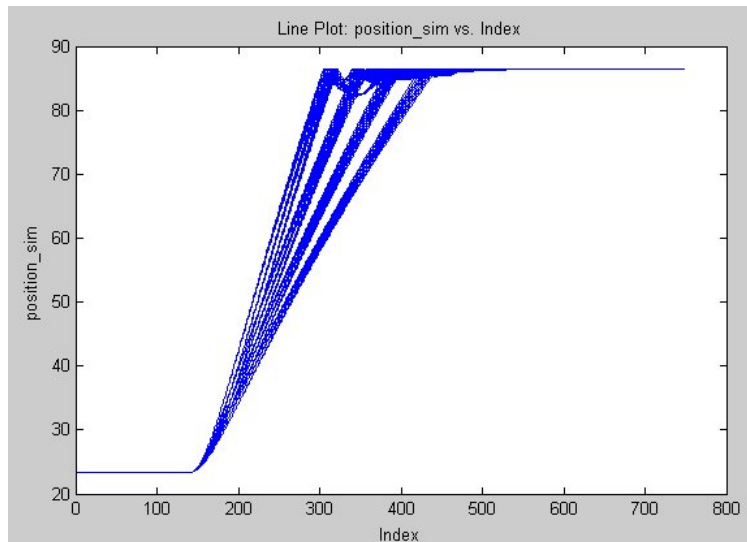
- 1 Return the plot to the previous state by clicking the menu next to **as** and clicking **color**, then click the **Refresh Plot** button.
- 2 In the **Constraints** pane, select the check box next to the **Damping** constraint. The constraint becomes active showing all tests with a Damping greater than or equal to 5.0, which is the lowest value in the range of test vectors. All test results remain in the plot.



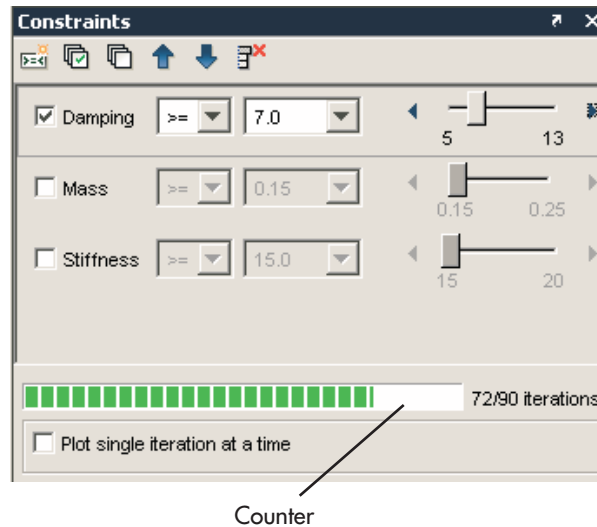
- 3 Click the right-pointing arrow at the end of the **Damping** constraint's slider.



This advances the constraints slider by one value of the test vector, which causes the first value of the Damping test vector to be removed from the test results used in generating the plot. The viewer immediately applies this constraint to the plot, which, in this case, removes the left-most cluster of test results from the plot.



The constraint counter gives another way for you to see whether the constraint affected the test results. In this case, if you set the constraint value to 7, the bar shows that there are only 72 of 90 test iterations visible because of the constraint you just created. Thus these 18 test iterations that are screened out have a Damping test vector value greater than or equal to 7 (see “Creating a Test Vector” on page 1-10 to understand test vector values).



Creating a Constraint

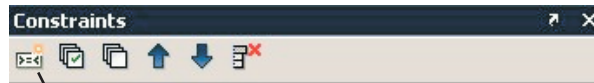
The Test Results Viewer lets you create a custom constraint based on the following:

- A mathematical expression
- Scalar logical test results
- Scalar numeric test results
- String test results that have a value for each test iteration
- Test vectors

You can see an example for creating a constraint based on a mathematical expression in “Viewing Test Results in the Test Results Viewer” on page 1-30.

A constraint you might want to create regularly would isolate test results that have passed or failed. This is useful if your test contains a Limit Check element that assigns data to a test variable that you choose to save as a test result. When this test variable is saved, SystemTest records the test iteration and whether the test passed or failed (represented by a 1 or 0); you can create a constraint based on these test results. For example:

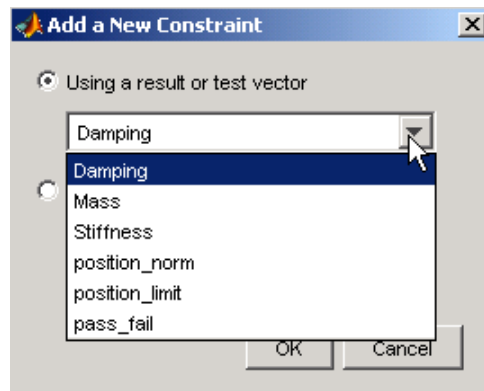
- 1 If you activated the **Damping** constraint in “Using Default Constraints” on page 7-24, deactivate it now by clearing the check box next to **Damping**, or delete it.
- 2 Click the **New Constraint** button.



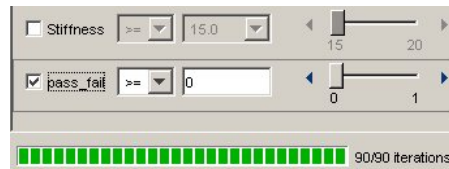
New constraint button

The Add a New Constraint dialog box appears.

- 3 Click the list beneath the **Using a result or test vector** field to show the list of test vectors and test results available for basing a constraint on.

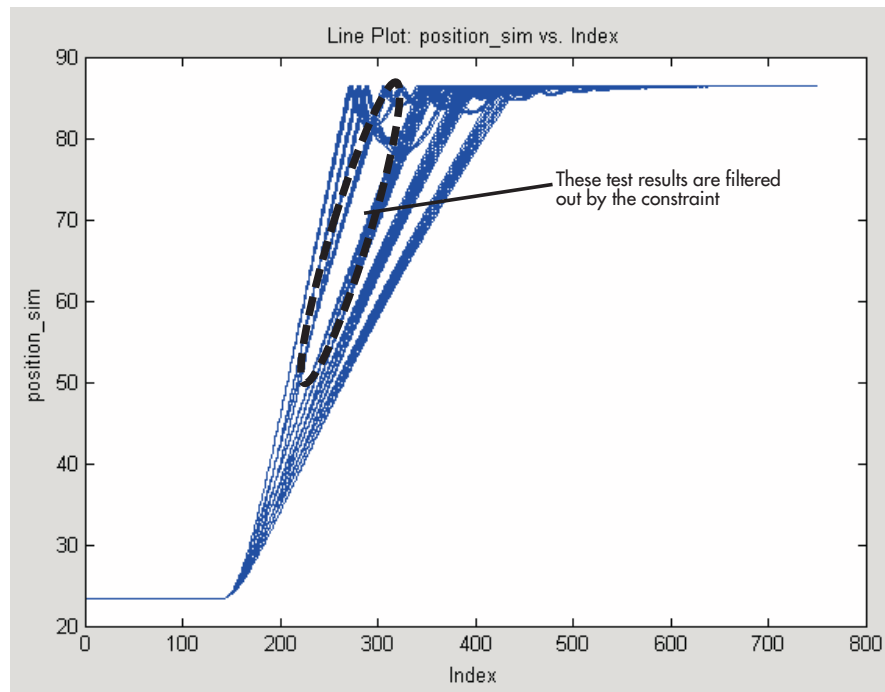


- 4 Scroll down and select **pass_fail** in this list. This is the name of the test result that is used to save the Throttle demo’s Limit Check element’s output.
- 5 Click **OK**. The viewer adds the new constraint to the **Constraints** pane, but it is not active.
- 6 Select the check box next to the **pass_fail** constraint to apply it.

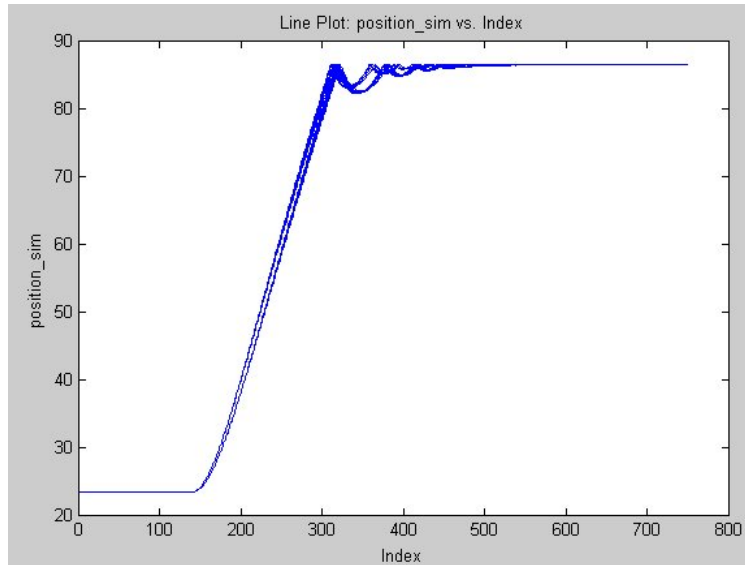


7 Change the operator to `==`. The value is already set to 0, representing failed test iterations.

You now have a constraint set to show only those test iterations that failed.



If you change the value of the constraint to 1 using the slider, you will show only those test results that passed the Limit Check element in your test.

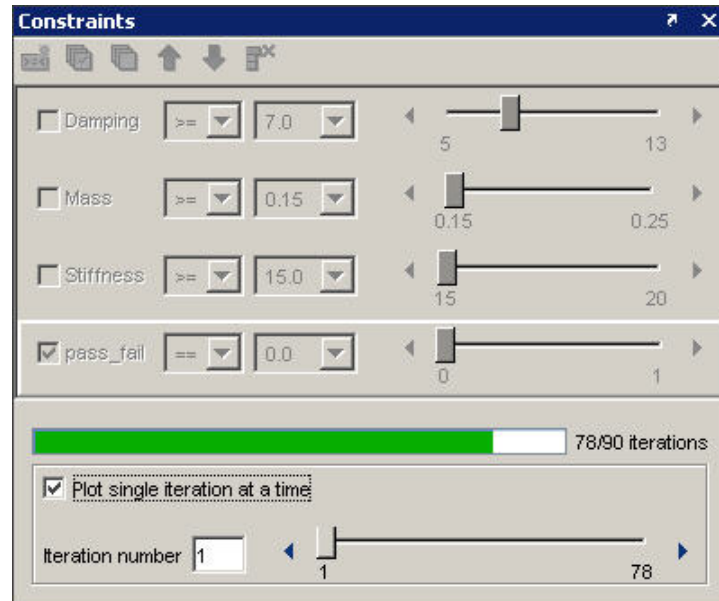


Plotting Single Iterations

The constraint option **Plot single iteration at a time** lets you step through and see individual test results within the subset defined by the active constraints. The plot shows only one test iteration until you choose to show the next or previous one. The specific values for that test iteration's test vectors and test results appear in the **Current Iteration** pane. This is useful when you want to know what combination of test vectors allow a test to pass, or what values can lead to failure.

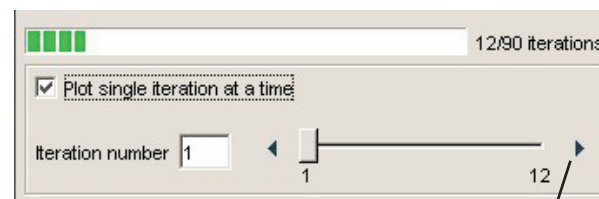
For example, if you follow “Creating a Constraint” on page 7-27, by the end you have created a constraint that shows you all test iterations that have passed. To see each iteration individually:

- 1 Move the slider for the **pass_fail** constraint back to 0.
- 2 Select the **Plot single iteration at a time** check box in the **Constraints** pane.



The **Constraints** pane changes to show a slider and the currently displayed test iteration.

- 3 Move the slider or click the advance button to see the next iteration. You see only those test results that match any defined constraints, which, in this case includes only those tests that have passed.



Click here to advance

The **Plots** pane updates to show only the plotted line for that iteration.

Viewing Simulink Time Series Data

The Test Results Viewer lets you plot test results over time. Simulink can generate time series data when it runs a model, and SystemTest can use this data to generate time series plots. Instead of knowing simply that a change in a test vector resulted in a specific test result value, you can now know when during the test that the test vector caused that test result value to be achieved.

This section shows how you plot test results containing time series data. The examples in this section use the model from the Inverted Pendulum demo; if you want to load this model and follow the examples in this section, see “Before You Begin” on page 3-2.

Creating a Time Series Plot

Time series plots require that you have time series data. Your test results will contain time series data because of any of the following:

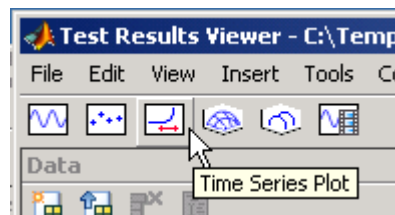
- Time series data is generated from Simulink Output Signals, Simulink Logged Signals, and Simulink To Workspace signals.
- The time series data was explicitly created in a MATLAB element and assigned to a test variable that was saved as a test result.
- The viewer created a derived result that represents time series data constructed from Simulink structs (with time data) or log signals. These new derived results have names derived from their original test result name and value.

You can verify whether your test generated time series data by reviewing the test results list in the Test Results Viewer’s **Data** pane. The viewer labels time series test results as being of type `Simulink.Timeseries` (Simulink saves time series data within the workspace in Model Data Logs objects).

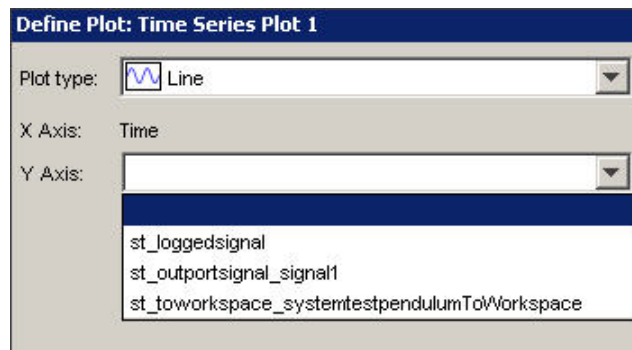
Name	Value	Type
limit	<1x1 double>	Raw scalar
st_time	<201x1 double>	Raw array
values	<201x1 double>	Raw array
maxvalue	<1x1 double>	Raw scalar
st_loggedsignal	{1007x1 Simulink.Timeseries}	Object/struct
st_outportsignal	{1x1 struct}	Object/struct
st_toworkspace	{1x1 struct}	Object/struct
limitResult	<1x1 double>	Raw scalar

To create a time series plot:

- 1 Run the test in SystemTest.
- 2 Click the **Time Series Plot** button in the viewer.



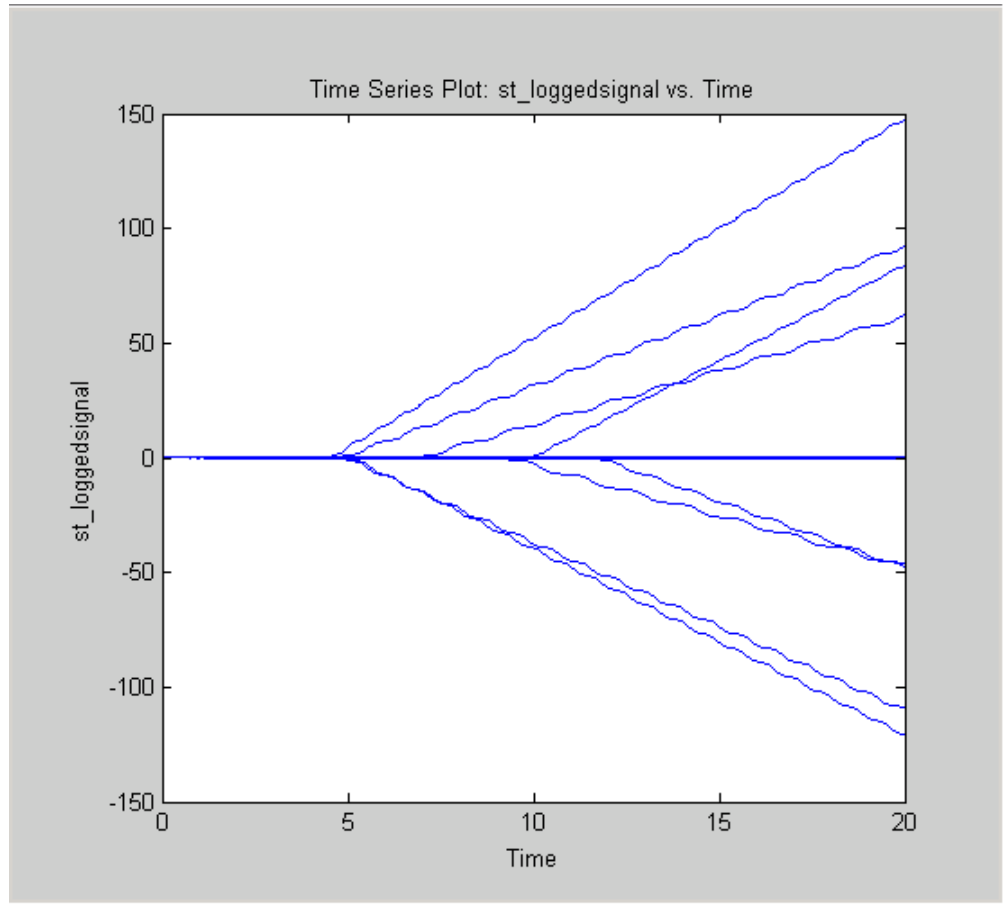
- 3 In the **Define Plot** pane, click the **Y Axis** menu to show a list of test results with time series data. The **Y Axis** field shows only test results with time series values. The **X Axis** field is always set to **Time** in a time series plot.



4 Click the test result you want to use. For the Inverted Pendulum example, click **st_loggedsignal**.

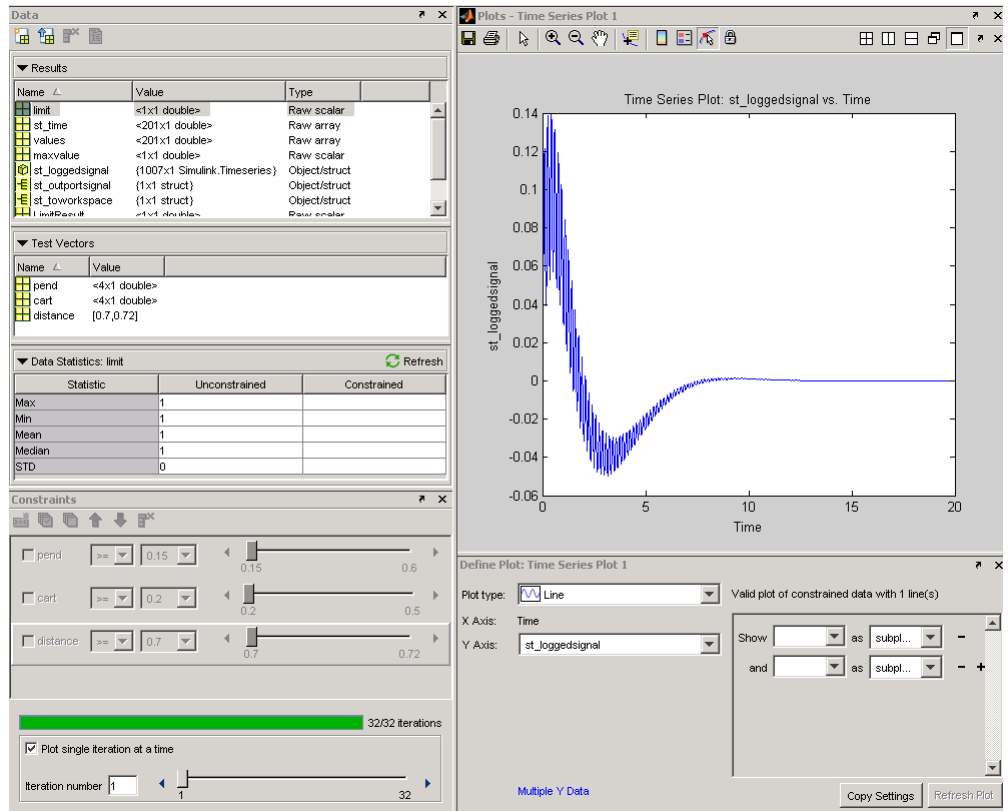
5 Click the **Refresh Plot** button.

The Test Results Viewer generates a time series plot with your selected data.



At this point, you can use the data exploration and refinement tools explained in “Viewing Your Test Results” on page 7-6 and “Refining Your Test Results” on page 7-24 to make more sense of the test results in the plot.

For example, you can use a constraint to step through each individual iteration, by selecting the **Plot single iteration at a time** check box.



As this example shows, the time series test result for a single test iteration is composed of many values over time. There are many points with uneven spacing reflecting the actual values of the signal over the time period.

SystemTest Hot Keys

The following keyboard shortcuts are available in SystemTest.

Key	Description
Alt+N	Activates the New button to create a new test vector.
F1	Opens Help.
F5	Runs a test.
Ctrl+C	While a test is running, stops the test.
Ctrl+C	When a test is not running, copies selection in some parts of the user interface.
Ctrl+N	Adds a new untitled test.
Ctrl+Q	Closes SystemTest.
Ctrl+V	Pastes the copied selection.
Ctrl+W	Closes a test.
Ctrl+X	Cuts a selection in some parts of the user interface.
Ctrl+0	Gives focus to the Test Browser .
Ctrl+1	Gives focus to the Properties pane.
Ctrl+2	Gives focus to the Test Vectors pane.
Ctrl+3	Gives focus to the Test Variables pane.
Ctrl+4	Gives focus to the Resources pane.
Ctrl+5	Gives focus to the Run Status pane.

Key	Description
Ctrl+6	Gives focus to the Desktop Help pane.
Ctrl+7	Gives focus to the Elements .

A

- adaptors
 - specifying in Video Input element 6-4
- adding
 - elements 1-14
 - Simulink element 3-4
 - Simulink model 3-5

B

- block parameter override 3-6
- browsing
 - test results 7-6

C

- constraints
 - counter 7-26
 - creating 7-27
 - default 7-24
 - defined 7-24
 - limit check 7-27
 - MATLAB expression 1-33
 - time series data 7-35
- counter 7-26
- creating
 - constraints 7-27
 - test variables 1-12
 - test vectors 1-10

D

- data
 - browsing 7-6
- Data Acquisition Toolbox elements 5-1
 - example 5-3
- Define Plot pane 7-16
- defining
 - iterations 1-11
- demos

- Getting Started 1-7
- Inverted Pendulum 3-2 7-32
- Signal Builder 3-18
- Simple 1-7
- Throttle 7-2
- Desktop 1-3

E

- elements 2-4
 - adding 1-14
 - Analog Input 5-8
 - Analog Output 5-4
 - Data Acquisition Toolbox 5-1
 - Digital Output 5-6 5-9
 - IF 2-7
 - Image Acquisition Toolbox 6-1
 - incorrectly configured example 1-18
 - Instrument Control Toolbox 4-1
 - Limit Check 2-5
 - MATLAB 2-4
 - Query Instrument 4-9
 - Scalar Plot 2-10
 - Simulink 3-1
 - Stop 2-12
 - Subsection 2-13
 - To Instrument 4-4
 - Vector Plot 2-8
 - Video Input 6-3
- examples
 - adding elements 1-14
 - applying constraints 1-33
 - building a test 1-7
 - creating constraints 7-27
 - creating time series plot 7-32
 - Data Acquisition Toolbox elements 5-2
 - generating plots 7-7
 - Image Acquisition Toolbox element 6-3
 - Instrument Control Toolbox elements 4-2
 - Limit Check element 1-17

- MATLAB element 1-16
- Scalar Plot element 1-20
- Simulink element 3-4
- viewing individual plot iterations 7-12

G

- generating
 - plots 7-7
- Getting Started demo 1-7

H

- HTML log
 - sample output 1-28

I

- IF element 2-7
- Image Acquisition Toolbox element
 - acquiring video data 6-1
 - example 6-3
- image data
 - importing into a test 6-1
- image plot 7-11
- inport signal override 3-10
- Instrument Control Toolbox elements 4-1
 - example 4-3
- Inverted Pendulum demo 3-2 7-32
- iteration
 - current 7-30
- iterations
 - defining 1-11
 - specifying number of frames acquired 6-5

L

- limit check
 - constraint 7-27
 - pass/fail 1-22
- Limit Check element 2-5

- example 1-17
- line plot 7-10
- log file
 - test execution 1-24
- logged signal override 3-11

M

- Main Test 1-9 2-3
- markers 7-19
- MAT-file 1-23
- MATLAB element 2-4
 - example 1-16
- MATLAB expression
 - constraint 1-33
 - test vector 1-10
- model
 - adding 3-5
 - input overrides 3-6
- model coverage 3-18

O

- outport signal override 3-13
- override
 - block parameter 3-6
 - inport signal 3-10
 - logged signal 3-11
 - model input 3-6
 - model outputs 3-10
 - outport signal 3-13
 - To Workspace block 3-14
 - workspace variable 3-8

P

- pass/fail 1-22
- plots
 - constraint 7-24
 - exploring 7-11
 - generating 7-7

- highlighting data 7-16
- image plot 7-11
- line plot 7-10
- markers 7-19
- overlapping lines 7-20
- plotting tools 7-12
- scatter plot 7-10
- single iterations 7-30
- subplot 7-21
- surf plot 7-10
- time series 7-32
- time series plot 7-10
- types 7-9
- waterfall plot 7-11
- plotting tools 7-12
- Post Test 1-9 2-3
- Pre Test 1-9 2-2
- Preferences 1-5
- product elements 1-15

R

- results
 - distinguish 7-16
- Run Status 1-26
- Run Status pane 1-24
- running
 - test 1-26

S

- saving
 - test 1-25
- Scalar Plot element 2-10
- scatter plot 7-10
- sections 1-9
- Signal Builder demo 3-18
- Simple demo 1-7
- Simulink element
 - adding 3-4

- block parameter 3-6
 - description 3-1
 - inport signal 3-10
 - logged signal 3-11
 - model coverage 3-18
 - model input overrides 3-6
 - model output overrides 3-10
 - model overrides 3-6
 - outport signal 3-13
 - To Workspace block 3-14
 - workspace variable 3-8
- Simulink model coverage 3-18
- starting
 - SystemTest 1-8
- Stop element 2-12
- stopping
 - test 1-26
- subplot rows 7-21
- Subsection element 2-13
- summary statistics 7-6
- surf plot 7-10
- SystemTest
 - benefits 1-2
 - Desktop 1-3
 - Preferences 1-5
 - runtime actions 1-26
 - starting 1-8

T

- test
 - analyzing results 1-28
 - components 1-8
 - constraints 7-24
 - construction workflow 1-8
 - elements 1-14
 - FOR loop 1-10
 - HTML output 1-24
 - pass/fail 1-22
 - planning 1-7

- plots 7-9
- running 1-26
- save results 1-23
- saving 1-25
- Simulink model 3-1
- stopping 1-26
- test vectors 1-10
- variables 1-12
- viewing results 1-30
- Test Browser
 - overview 1-4
- test execution log
 - activating 1-24
 - iteration results 1-29
- test execution log file 1-24
- test results
 - browsing 7-6
 - plot 7-7
- Test Results Viewer 1-30 7-1
 - constraint example 1-33
 - constraints 7-24
 - data browser 1-30
 - overlapping plot lines 7-20
 - overview 7-4
 - plot procedure 7-7
 - plot types 7-9
 - sample plot 1-31
- test sections 2-2
 - Main Test 2-3
 - Post Test 2-3
 - Pre Test 2-2
- test variables
 - creating 1-12
 - specifying in Video Input element 6-6
- test vector

- constraint 7-24
- creating 1-10
- workspace variable override 3-8
- tests
 - running in SystemTest 6-8
 - specifying image acquisition device 6-4
- Throttle demo 7-2
- time series
 - data 7-32
 - plot 7-10
- To Workspace block override 3-14

V

- Vector Plot element 2-8
- video
 - importing into a test 6-1
- Video Input element
 - running a test 6-8
 - specifying image acquisition device
 - properties 6-4
 - specifying number of frames per iteration 6-5
 - specifying test variable 6-6
 - using 6-1
- viewing
 - test results 1-30 7-6

W

- waterfall plot 7-11
- workflow 1-8
 - in SystemTest 1-7
- workspace variable override 3-8