

# **JavaTest™ User's Guide**






Sun Microsystems, Inc.  
4150 Network Circle  
Santa Clara, California 95054  
1-650-960-1300 or  
1-800-555-9SUN

Version 3.1.2  
October 2002



# Table Of Contents

1. What is the JavaTest Harness?	2
1.1. JavaTest Features	2
2. Before Starting the JavaTest Harness	4
3. Starting the JavaTest Harness	4
3.1. Welcome Dialog Boxes	5
3.1.1. Test Suite Dialog Box	5
3.1.2. Work Directory Dialog Box	5
4. JavaTest Online Help	7
4.1. Accessing Help	7
4.2. Navigation	7
5. Using the JavaTest Harness	9
5.1. Test Manager Window	10
5.1.1. Configure Menu	11
5.1.2. Run Tests Menu	12
5.1.3. Report Menu	13
5.1.4. View Menu	13
5.1.5. Tool Bar	14
5.2. Displaying JavaTest Help and Information	15
5.2.1. Help Menu	15
5.2.2. Help Buttons	15
5.2.3. F1 Key	15
5.3. Opening a Test Suite	15
5.4. Opening a Work Directory	15
5.5. Creating a Work Directory	16
6. Configuring a Test Run	17
6.1. Configuring All Values	18
6.1.1. Using the Configuration Editor: All Values View	18
Create or Edit a Configuration	18
Searching the Interview	19
Saving the Interview	19
6.1.2. Using the Menus	19
File Menu	19
Search Menu	20
View Menu	20
Help Menu	20
6.1.3. Using the All Values View Panes	21
Index Pane	21
Question Pane	21
More Info Pane	21
6.1.4. Find Questions	22
6.2. Changing Standard Values	23
6.2.1. Using the Configuration Editor: Standard Values View	23
Specifying Tests to Run	24
Using Exclude Lists	24
Using Keywords as a Filter	26
Using Prior Status as a Filter	27
Specifying the Test Environment	28
Setting Test Execution Values	29
6.3. Working with Multiple Configurations	31
6.3.1. Creating a New Configuration File	31
6.3.2. Opening an Existing Configuration File	31
6.4. Viewing the Configuration Checklist	31
6.5. Viewing the Test Environment	32
6.5.1. Test Environment Dialog Box	32
6.6. Viewing Exclude Lists	33

6.6.1. Exclude List Dialog Box . . . . .	33
Exclude List Contents . . . . .	33
Test Details . . . . .	33
6.7. Viewing the Question Log . . . . .	33
7. Running Tests . . . . .	35
7.1. Starting a Test Run . . . . .	36
7.2. Monitoring a Test Run . . . . .	37
7.3. Using the Test Tree . . . . .	37
7.4. Using the Test Progress Display . . . . .	38
7.4.1. Select a Monitor . . . . .	38
Elapsed Time . . . . .	38
Run Progress Meter . . . . .	38
7.5. Using the Progress Monitor . . . . .	38
7.5.1. Progress . . . . .	39
7.5.2. Time . . . . .	40
7.5.3. Memory . . . . .	40
7.5.4. Tests in Progress . . . . .	40
7.6. Stopping a Test Run . . . . .	40
7.7. Troubleshooting a Test Run . . . . .	41
7.7.1. Test Tree . . . . .	41
7.7.2. Folder View . . . . .	41
7.7.3. Test View . . . . .	41
8. Browsing Test Information . . . . .	43
8.1. Test Tree . . . . .	43
8.1.1. Folder Icons . . . . .	45
Result Status . . . . .	45
Run Status . . . . .	45
8.1.2. Test Icons . . . . .	46
Result Status . . . . .	46
Run Indicator . . . . .	46
8.1.3. Using the Test Tree Popup Menu . . . . .	47
"Quick Pick" Test Execution . . . . .	47
Refresh Test Suite Contents . . . . .	48
Clear Previous Test Results . . . . .	48
8.2. Folder View . . . . .	50
8.2.1. Summary Information . . . . .	50
8.2.2. Status Information . . . . .	51
 Passed (green) . . . . .	51
 Failed (red) . . . . .	52
 Error (blue) . . . . .	52
 Not Run (white) . . . . .	52
 Filtered Out . . . . .	52
8.3. Test View . . . . .	53
8.3.1. Test Description . . . . .	54
Name . . . . .	54
Value . . . . .	54
8.3.2. Files . . . . .	54
8.3.3. Configuration . . . . .	54
Name . . . . .	54
Value . . . . .	54
8.3.4. Test Run Details . . . . .	55
Name . . . . .	55
Value . . . . .	55
8.3.5. Test Run Messages . . . . .	55
Message List . . . . .	56
Message Area . . . . .	56

8.4. Using Filters . . . . .	57
8.4.1. The Current Configuration Filter . . . . .	58
8.4.2. The All Tests Filter . . . . .	59
8.4.3. The Custom Filter . . . . .	59
Editing the Custom Filter . . . . .	59
Using a Custom View Filter . . . . .	60
8.5. Test Manager Properties . . . . .	61
8.5.1. Test Suite . . . . .	61
8.5.2. Work Directory . . . . .	61
8.5.3. Configuration . . . . .	61
8.5.4. Plug-Ins . . . . .	61
8.6. Test Suite Errors . . . . .	62
9. Using Test Reports . . . . .	63
9.1. Generating New Reports . . . . .	63
9.2. Viewing Reports . . . . .	64
9.2.1. View Reports in the Report Browser . . . . .	64
9.2.2. View Reports Offline . . . . .	65
9.3. Moving Report Files . . . . .	66
10. Auditing a Test Run . . . . .	67
10.1. Auditing in GUI Mode . . . . .	67
10.2. Auditing in Batch Mode . . . . .	67
10.3. Setting Audit Options . . . . .	67
10.3.1. Test Suite . . . . .	68
10.3.2. Work Directory . . . . .	68
10.3.3. Configuration File . . . . .	69
10.3.4. Start Audit Button . . . . .	69
10.3.5. Cancel Button . . . . .	69
10.3.6. Help Button . . . . .	69
10.4. Audit Test Results Window . . . . .	69
10.4.1. Summary . . . . .	70
10.4.2. Bad Result File . . . . .	71
10.4.3. Bad Checksum . . . . .	71
10.4.4. Bad Test Description . . . . .	71
10.4.5. Bad Test Cases . . . . .	71
11. Customizing the JavaTest GUI . . . . .	71
11.1. The JavaTest GUI . . . . .	72
11.2. The JavaTest GUI Windows . . . . .	72
11.3. JavaTest Menus . . . . .	72
11.3.1. File Menu . . . . .	73
11.3.2. Tasks Menu . . . . .	74
11.3.3. Windows Menu . . . . .	75
11.3.4. Help Menu . . . . .	75
11.4. Setting JavaTest Preferences . . . . .	75
11.4.1. Changing Appearance Preferences . . . . .	76
Changing Window Styles . . . . .	77
Setting Tool Tip Options . . . . .	78
Changing Shutdown Options . . . . .	78
11.4.2. Changing Test Manager Preferences . . . . .	78
11.4.3. Changing Configuration Editor Preferences . . . . .	79
Set the Configuration Editor Default View . . . . .	79
Display/Hide the Configuration Editor More Info Pane . . . . .	80
11.5. Managing JavaTest Windows . . . . .	80
12. Using a JavaTest Agent . . . . .	81
12.1. Choosing the Type of Agent . . . . .	81
12.2. Starting an Agent . . . . .	83
12.2.1. Agent Application . . . . .	83
12.2.2. Agent Applet . . . . .	83
12.2.3. Using The GUI . . . . .	84

12.2.4. Starting an Agent Application . . . . .	84
Class Paths . . . . .	85
Application Classes . . . . .	85
Agent Options . . . . .	86
12.2.5. Starting an Agent Applet . . . . .	86
Agent Applet Tag . . . . .	86
Setting Parameters in the Applet Tag . . . . .	88
12.2.6. Specifying Active Agent Options . . . . .	88
Mode . . . . .	88
Host . . . . .	89
Port . . . . .	89
12.2.7. Specifying Passive Agent Options . . . . .	89
Mode . . . . .	89
Port . . . . .	90
12.2.8. Specifying Serial Agent Options . . . . .	91
Mode . . . . .	91
Port . . . . .	91
12.2.9. Specifying Additional Options . . . . .	92
Options Used to Display Help . . . . .	92
Options Used to Run and Monitor the Agent . . . . .	92
12.3. Monitoring JavaTest Agents . . . . .	94
12.3.1. Agent Monitor Window . . . . .	94
Agent Pool . . . . .	95
Agents Currently In Use . . . . .	95
12.3.2. Statistics Pane . . . . .	95
12.3.3. History Pane . . . . .	96
12.3.4. Selected Task Pane . . . . .	97
12.4. Troubleshooting JavaTest Agents . . . . .	99
12.4.1. Troubleshooting Active Agents . . . . .	99
12.4.2. Troubleshooting Passive Agents . . . . .	100
12.5. Installing Agent Classes on a Test System . . . . .	100
12.5.1. Classes Required to Run Agents Using a GUI . . . . .	101
12.5.2. Classes Required to Run Agents from the Command Line . . . . .	102
12.5.3. Classes Required to Run Agents as Applets . . . . .	102
12.6. Creating a Map File . . . . .	103
13. Using the JavaTest Command-Line . . . . .	105
13.1. Using Batch Mode . . . . .	105
13.1.1. Formatting Batch Commands . . . . .	106
Batch Options . . . . .	106
Single String Arguments . . . . .	106
Batch Command Files . . . . .	106
13.1.2. Initializing the Configuration . . . . .	107
open <i>name</i> . . . . .	107
testSuite <i>testsuite</i> . . . . .	108
workDirectory <i>work-directory</i> . . . . .	108
Shortcuts Used to Initialize the Current Configuration . . . . .	109
13.1.3. Setting the Standard Values . . . . .	109
concurrency <i>number</i> . . . . .	109
env <i>environment</i> . . . . .	109
envFile <i>environment-file</i> . . . . .	109
excludeList <i>exclude-list-file</i> . . . . .	110
keywords <i>keyword-expr</i> . . . . .	110
params <i>parameter-arguments</i> . . . . .	110
priorStatus <i>status-arguments</i> . . . . .	110
tests <i>test-name</i> . . . . .	110
timeoutFactor <i>number</i> . . . . .	110
13.1.4. Setting Other Configuration Values . . . . .	111
set <i>question-tag-name value</i> . . . . .	111

13.1.5. Running Tests in Batch Mode . . . . .	111
13.1.6. Writing Reports in Batch Mode . . . . .	111
13.1.7. Auditing Tests . . . . .	111
13.1.8. Index of Available Batch Commands . . . . .	112
13.2. Specifying Additional Options . . . . .	113
13.2.1. Using Parameter Commands . . . . .	114
13.3. Displaying JavaTest Information . . . . .	115
13.4. Examples of Batch Commands . . . . .	116
13.4.1. Obtaining the Question <i>tag-name</i> . . . . .	116
13.5. Editing in Batch Commands . . . . .	117
13.5.1. Open a .jti File and Change Values Before Running Tests . . . . .	117
13.5.2. Create a New Work Directory . . . . .	117
13.6. Modifying Settings in a Configuration . . . . .	118
13.6.1. Selecting Tests to Run . . . . .	118
13.6.2. Selecting an Exclude List . . . . .	119
13.6.3. Setting Configuration Values . . . . .	119
13.7. Using a Batch File . . . . .	119
13.7.1. Example of Using a Batch File . . . . .	120
14. Using Additional JavaTest Utilities . . . . .	121
14.1. Monitoring Results with HTTP Server . . . . .	121
14.1.1. HTML Formatted Output . . . . .	121
Accessing HTTP Server HTML Formatted Output . . . . .	121
Viewing HTTP Server Index Page . . . . .	122
Viewing HTTP Server Harness Page . . . . .	122
Viewing HTTP Server Test Result Index Page . . . . .	122
Viewing the Harness Environment Page . . . . .	122
Viewing the Harness Interview Page . . . . .	123
Using HTTP Server to Stop a Test Run . . . . .	123
14.1.2. Plain Text Output . . . . .	123
Accessing Version Information . . . . .	123
Accessing Harness Information . . . . .	123
14.2. Browsing Result (.jtr) Files . . . . .	125
14.3. Browsing Exclude List Files . . . . .	125
14.4. Editing Responses in a Configuration File . . . . .	125
14.4.1. Format of EditJTI Command . . . . .	125
14.4.2. Obtaining the Question <i>tag-name</i> . . . . .	127
14.5. Examples of Using EditJTI . . . . .	127
14.5.1. Edit a Configuration File . . . . .	127
Generate a Log of All Updates . . . . .	128
Preview But Not Change . . . . .	128
Echo Results of Your Edit . . . . .	128
Show Paths for Debugging . . . . .	128
Change Test Suites or Create a New Interview . . . . .	128
14.5.2. Change the HTTP Port and Overwrite Original Configuration File . . . . .	129
14.5.3. Change the HTTP Port and Create a New Configuration File . . . . .	129
14.5.4. Doing Escapes in a UNIX Shell . . . . .	129
14.6. Moving Test Reports . . . . .	130
14.6.1. Format of EditLinks Command . . . . .	130
14.6.2. Example of EditLinks Command . . . . .	131
15. Troubleshooting . . . . .	132
15.1. Problems in Running Tests . . . . .	132
15.2. Problems Using Agents . . . . .	132
16. Glossary . . . . .	133
17. Index . . . . .	140





## Copyright Notice

Copyright © 2002 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, U.S.A. All rights reserved.

Sun Microsystems, Inc. has intellectual property rights relating to technology embodied in the product that is described in this document. In particular, and without limitation, these intellectual property rights may include one or more of the U.S. patents listed at <http://www.sun.com/patents> and one or more additional patents or pending patent applications in the U.S. and in other countries.

This document and the product to which it pertains are distributed under licenses restricting their use, copying, distribution, and decompilation. No part of the product or of this document may be reproduced in any form by any means without prior written authorization of Sun and its licensors, if any.

Third-party software, including font technology, is copyrighted and licensed from Sun suppliers.

Sun, Sun Microsystems, the Sun logo, Java, JavaTest, the Duke logo and the Java Coffee Cup logo are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

Federal Acquisitions: Commercial Software - Government Users Subject to Standard License Terms and Conditions.

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

---

Copyright © 2002 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, Etats-Unis. Tous droits réservés.

Sun Microsystems, Inc. a les droits de propriété intellectuels relatants à la technologie incorporée dans le produit qui est décrit dans ce document. En particulier, et sans la limitation, ces droits de propriété intellectuels peuvent inclure un ou plus des brevets américains énumérés à <http://www.sun.com/patents> et un ou les brevets plus supplémentaires ou les applications de brevet en attente dans les Etats-Unis et dans les autres pays.

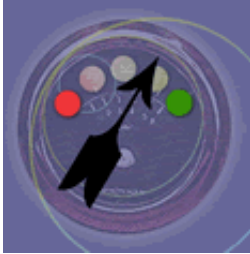
Ce produit ou document est protégé par un copyright et distribué avec des licences qui en restreignent l'utilisation, la copie, la distribution, et la décompilation. Aucune partie de ce produit ou document ne peut être reproduite sous aucune forme, par quelque moyen que ce soit, sans l'autorisation préalable et écrite de Sun et de ses bailleurs de licence, s'il y en a.

Le logiciel détenu par des tiers, et qui comprend la technologie relative aux polices de caractères, est protégé par un copyright et licencié par des fournisseurs de Sun.

Sun, Sun Microsystems, le logo Sun, Java, JavaTest, le logo Duke et le logo Java Coffee Cup sont des marques de fabrique ou des marques déposées de Sun Microsystems, Inc. aux Etats-Unis et dans d'autres pays.

LA DOCUMENTATION EST FOURNIE "EN L'ÉTAT" ET TOUTES AUTRES CONDITIONS, DECLARATIONS ET GARANTIES EXPRESSES OU TACITES SONT FORMELLEMENT EXCLUES, DANS LA MESURE AUTORISÉE PAR LA LOI APPLICABLE, Y COMPRIS NOTAMMENT TOUTE GARANTIE IMPLICITE RELATIVE A LA QUALITE MARCHANDE, A L'APTITUDE A UNE UTILISATION PARTICULIERE OU A L'ABSENCE DE CONTREFAÇON.

# 1. What is the JavaTest Harness?



The JavaTest harness is a powerful test harness that can run tests on a variety of test platforms (such as servers, workstations, browsers, and small devices) with a variety of test execution models (such as API compatibility tests, language/compiler tests, and regression tests).

## 1.1. JavaTest Features

- **GUI mode** To browse and run test suites and test results. The graphical user interface provides a set of windows and menus that you use to configure and run tests, monitor test and agent status, evaluate and analyze test results, and include or exclude tests from test runs.
- **Batch mode** Batch mode provides complete test execution functionality without using the GUI -- this allows you to use the JavaTest harness to run tests in build scripts and other automated processes.
- **HTML reports** That summarize test runs. HTML reports for the test run can be generated in batch mode or from the GUI
- **Auditing test runs** Auditing test runs can be performed in batch mode or from the GUI
- **Web server to monitor batch mode** The JavaTest harness provides a small web server that you can use to monitor and control test progress while tests are running.
- **Runs tests on small systems** The JavaTest harness provides an agent (a separate program that works in conjunction with the JavaTest harness) to run tests on systems that can't run the JavaTest harness.
- **Online help** JavaTest online help describes how to use the JavaTest harness to run test suites and evaluate test results. It also provides context sensitive help, full-text search, keyword search, and can be accessed without starting the JavaTest harness.

**Note:**

This User's Guide is a PDF version of the JavaTest HTML online help. It is provided in PDF format so that users can conveniently view and print the contents of the online help without starting the JavaTest harness.

However, there are several minor differences between the online help and the PDF versions -- for example, in some cases the contents of the online help have been resequenced to be more useful in book format; in the PDF format there are numerous page references embedded in the text that are hypertext links in the online help; and, extensive online help navigation links have been removed from the PDF format.

## 2. Before Starting the JavaTest Harness

Before starting the JavaTest harness, you must have a valid test suite and JDK 1.3 or later installed on your test system.

- Refer to your test suite documentation for information about installing the test suite and JavaTest harness on your test system.
- For information about creating a test suite, refer to the Test Suite Architect's Guide. The Test Suite Architect's Guide is available as part of the Java Compatibility Test Tools.
- For information about installing the current JDK on your test system, refer to <http://java.sun.com/products>.

## 3. Starting the JavaTest Harness

Start the JavaTest harness from a writeable directory where you intend to create files and store test results. You can use either a graphical user interface (GUI mode) or a command-line user interface (batch mode) to start the JavaTest harness. You must include the path of the directory *[jt\_dir]* where the *javatest.jar* file is installed. The *javatest.jar* file is usually installed in the TCK lib directory when the JavaTest harness is bundled with a TCK.

To start the JavaTest harness in GUI mode, you can use any of the following:

Start by:	Description
Running a Startup Script	Use a startup script if your test suite includes one. Refer to your test suite documentation for details about using a startup script.
Executing the JAR File	If you have access to a command line you can directly execute the <i>.jar</i> file from the directory where you intend to create files and store test results. At the command prompt start the harness:  <pre>java -jar [jt_dir]/javatest.jar</pre>
Double-Clicking the Icon	If you are using a GUI, your system may support double clicking the <i>javatest.jar</i> file icon to launch the harness.
Invoking the Class Directly	If you have access to a command line and the <i>jar</i> file is on the class path, you can directly invoke the class from the directory where you intend to create files and store test results. At the command prompt start the harness:  <pre>java com.sun.javatest.tool.Main</pre>

If you choose to run tests in the batch mode refer to Using Batch Mode [p 105] .

Unless you are starting the harness for the first time or use a command line option, the JavaTest harness restores your previous desktop when it starts the GUI. See Specifying Additional Options [p 112] for information about the optional arguments that can be included in the command string when starting the JavaTest harness.

If the JavaTest harness cannot restore an existing desktop, it displays a Welcome dialog box that guides you through the startup. See Welcome Dialog Boxes [p 5] for a detailed description of using the Welcome dialog box to start the JavaTest harness.

## 3.1. Welcome Dialog Boxes

When you start the JavaTest harness for the first time or instruct it to create a new desktop, it checks the following:

1. Has it been started in a work directory or test suite?
2. Has it been installed with a test suite?

The JavaTest harness then displays an appropriate Welcome to JavaTest dialog box:

- If the JavaTest harness is able to open a test suite, it displays a Welcome to JavaTest dialog box that guides you through the process of opening an existing work directory or creating a new work directory. Refer to Work Directory Dialog Box [p 5] for a description of this dialog box.
- If the JavaTest harness is unable to open a test suite, it displays a Welcome to JavaTest dialog box that guides you through the process of opening a test suite or work directory. Refer to Test Suite Dialog Box [p 5] for a description of this dialog box.



Test suites contain the tests that are run on the test system. Work directories are the locations that the JavaTest harness uses to store test results and other information about a test suite. See the Glossary for additional information about test suites and work directories.



The next time you start the JavaTest harness it automatically opens the test suite and work directory for you without displaying the Welcome to JavaTest dialog box.

After you specify a work directory, you can use Test Manager to configure and run tests. Refer to Using the JavaTest Harness [p 9] for a description of how to begin running tests.

### 3.1.1. Test Suite Dialog Box

If the JavaTest harness cannot locate and open a test suite when it creates a new desktop, it displays a dialog box that guides you through the process of opening a test suite or a work directory:

Button	Function
Open Test Suite	Click this button to locate and use a test suite.
Open Work Directory	Click this button to use an existing work directory. When you open an existing work directory, the JavaTest harness also opens the test suite associated with it.

You can also use the Test Manager window to change test suites or work directories after the JavaTest GUI is running:

- Open another test suite [p 15]
- Open an existing work directory [p 15]
- Create a new work directory [p 16]

### 3.1.2. Work Directory Dialog Box

If the JavaTest harness can locate and open a test suite when it creates a new desktop, it displays a dialog box that guides you through the process of either opening or creating a work directory:

<b>Button</b>	<b>Function</b>
Open Work Directory	Click this button to use an existing work directory. When you open an existing work directory, the JavaTest harness also opens the test suite associated with it.
Create Work Directory	Use Create Work Directory when you either do not have or do not want to use an existing work directory for the test suite.

You can also use the Test Manager window to change test suites or work directories after the JavaTest GUI is running:

- Open another test suite [p 15]
- Open an existing work directory [p 15]
- Create a new work directory [p 16]

## 4. JavaTest Online Help

JavaTest online help describes how to use the JavaTest harness to run test suites and evaluate test results. This page describes how to access online help and how to navigate through the help topics to find the information you want.


The JavaTest HTML online help is also provided in PDF format as the JavaTest User's Guide, available in your TCK installation directory at:

`doc/javatest/javatest.pdf`

The User's Guide is provided in PDF format so that you can use an appropriate viewer, such as Adobe Acrobat Reader from Adobe at [www.adobe.com](http://www.adobe.com), to easily view and print the contents of the online help without starting the JavaTest harness.

### 4.1. Accessing Help

You can get help and information about the JavaTest harness in the following ways:






Action	Description
F1 key	Press the F1 key to see information about the JavaTest window that has keyboard focus.  It is very important to establish keyboard focus in a window before pressing the F1 key. In some cases you may have to highlight something in the window in order to establish focus.
Help menu	Choose Help > Help to start the Help viewer with this page displayed. Use the navigators in the left pane to locate information.
Help buttons	Click the Help button in a dialog box for information about how to use that dialog box

You can also display online help without starting the JavaTest harness. Include the path of the directory `[jt_dir]` where the `javatest.jar` file is installed. The `javatest.jar` file is usually installed in the TCK lib directory when the JavaTest harness is bundled with a TCK. Type the following at a system prompt:

```
java -jar [jt_dir]/javatest.jar -onlineHelp
```

### 4.2. Navigation

The left pane of the help viewer contains four tabs that can help you navigate through the help information:

Icon	Description
	Table of contents. Topics are organized by task (where possible). Topics displayed in the right pane are highlighted in the table of contents.
	Keyword index. Index entries are listed in alphabetical order. To search for entries, enter a string of characters in the Find field and press Return -- if the string is found in the index it is highlighted (press Return again to repeat the search).
	Glossary. Glossary entries are listed in alphabetical order. You can scroll through the list or search for a term. To search for a term, enter a string of characters in the Find field and press Return -- if the term is found in the glossary it is highlighted in the list and its definition is displayed. Press Return to repeat the search.
	<p>Full-text search. Type a natural language phrase in the Find field and press return. A ranked list of topics are returned with the following information:</p> <ul style="list-style-type: none"> <li>• The circle in the first column indicates the ranking of the matches for that topic. The more filled-in the circle is, the higher the ranking. There are five possible rankings (from highest to lowest): </li> <li>• The number in the second column indicates the number of times the query was matched in the listed topic.</li> <li>• The title of the topic in which matches are found is listed as it appears in the table of contents.</li> </ul>




## 5. Using the JavaTest Harness

The JavaTest harness restores your previous desktop unless you are starting the harness for the first time or include a command line option when starting the harness.



If you do not use an existing desktop, the JavaTest harness displays a Welcome dialog box [p 5] that guides you through the startup.

When the JavaTest GUI starts, it displays the Test Manager window [p 10] used to configure a test run, run tests, monitor test runs, browse test information, and troubleshoot a test run.

After you have opened a work directory there are two ways you can start a test run:

- You can immediately click the  button on the tool bar or choose the Run Tests > Start menu item to start a test run.

Before it starts the test run, the JavaTest harness checks whether the required configuration information has been supplied. Refer to Starting a Test Run [p 36] for detailed information.

- You can open the configuration editor from the Configure menu [p 11] or click the  button on the tool bar to provide configuration information. Then, click the  button or choose the Run Tests > Start menu item to start a test run [p 36] .

During the test run you will notice that the icons in the test tree change color to reflect their test status. You can also browse information about the tests by choosing them in the test tree and viewing information in the Test Manager window.



Information about performing tasks in JavaTest harness can be found in the following topics:

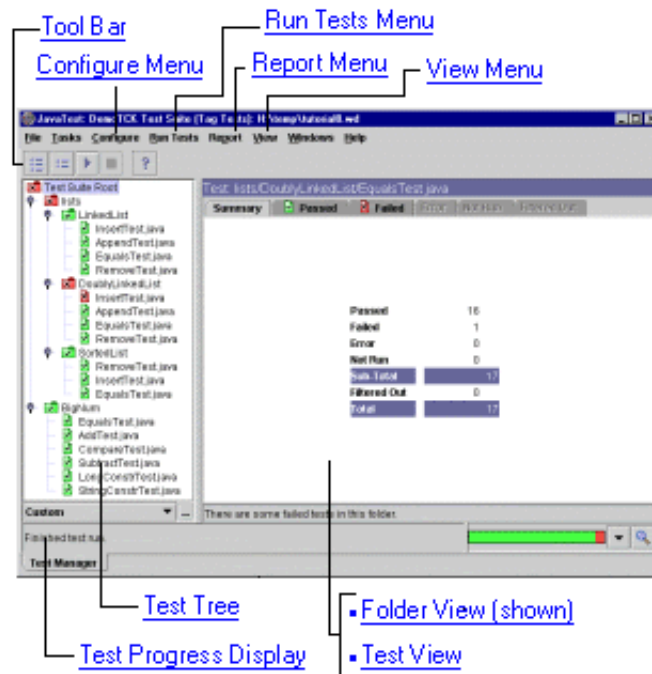
- Configuring a Test Run [p 17] - Using the configuration editor.
- Running Tests [p 35] - Using the Run Tests menu and Progress Monitor.
- Browsing Test Information [p 43] - Using the Test Manager window to view test information.

To locate detailed information about performing a specific task, use the full-text search and keyword search index tabs. Refer to JavaTest Online Help [p 7] for a description of how to use the full-text search and keyword search index tabs.

A Glossary of terms used in the documentation is also provided. Refer to JavaTest Online Help [p 7] for a description of how to use the Glossary tab.

## 5.1. Test Manager Window

Use the Test Manager window to run tests and browse test results.




The Test Manager window contains the following:

- Configure Menu [p 11]
- Run Tests Menu [p 12]
- Report Menu [p 13]
- View Menu [p 13]
- Tool Bar [p 14]
- Test tree [p 43]
- Folder View [p 50]
- Test View [p 53]
- Test Progress Display [p 38]

Depending on the window style that you are using, JavaTest standard menus can also be displayed in the menu bar. See JavaTest Menus [p 72] for a description of the JavaTest standard menus.

Use the Test Manager to:

Task	Description
Configure a test run	Provide configuration information required to run your test suite. Configuration as part of running tests is described in <a href="#">Configuring a Test Run [p 17]</a> .
Run tests	Start test runs by choosing the Run Tests > Start menu item or click the  button on the tool bar. See <a href="#">Running Tests [p 35]</a> for a detailed description of how to run tests.
Monitor test runs	Use the test tree with the folder and test views to monitor the status of the test run. For more information about monitoring test runs, see <a href="#">Monitoring a Test Run [p 37]</a> .
Browse test information	Use the test tree with the folder and test views to browse information about overall test status as well as what occurred during a test run. See <a href="#">Browsing Test Information [p 43]</a> for details about browsing test run information.
Generate and view test reports	Use the Report menu to generate and view about test run information. See <a href="#">Using Test Reports [p 63]</a> for details about generating and viewing test run information.
Troubleshoot a test run	Use the test tree with the folder and test views to troubleshoot a test run. See <a href="#">Troubleshooting a Test Run [p 41]</a> for help with troubleshooting test run problems.



### 5.1.1. Configure Menu

Use the Configure menu to load, create, modify, and view configuration data used for a test run. The Configure menu contains the following menu items:

Menu Item	Description
New Configuration	<p>Sets the current configuration to an empty configuration and opens the Configuration Editor.</p> <p>Use the configuration editor to create configuration data containing the test environment values and standard values required to run a test suite.</p> <p>See Configuring a Test Run [p 17] for detailed information.</p>
Load Configuration	<p>Opens the Load Configuration File dialog box.</p> <p>Use the dialog box to load an existing configuration interview (.jti) file into the Test Manager for use in running the test suite. The JavaTest harness does not open the Configuration Editor when it loads an existing configuration interview.</p>
Edit Configuration	<p>Opens the configuration editor and the current configuration interview.</p> <p>Use the configuration editor to edit data in the current configuration interview required to run a test suite.</p> <p>See Configuring a Test Run [p 17] for detailed information.</p>
Show Checklist	<p>Displays a checklist of tasks to be performed before running tests. See Viewing the Configuration Checklist [p 31] for detailed information.</p>
Show Exclude List	<p>Opens an Exclude List dialog box that contains the exclude list used to run the test suite</p> <p>You can use the Exclude List dialog box to review but not edit the contents of the exclude list. Use the configuration editor to add or remove exclude lists. See Configuring a Test Run [p 17] for detailed information.</p> <p>See Viewing Exclude Lists [p 32] for detailed information about using the Exclude List dialog box.</p>
Show Test Environment	<p>Opens a Test Environment dialog box that contains the environment values used when running the test suite</p> <p>You can browse but not edit values in the Test Environment dialog box. Use the configuration editor to edit the environment values. See Configuring a Test Run [p 17] for detailed information.</p> <p>See Viewing the Test Environment [p 32] for detailed information about using the Test Environment dialog box.</p>
Show Question Log	<p>Displays a log of the current configuration interview questions and answers. See Viewing the Question Log [p 33] for detailed information.</p>
<i>Configuration History</i>	<p>Displays a list of configuration interview files that have been opened. Choose a file from the list to open a new instance of it as the current configuration.</p>

### 5.1.2. Run Tests Menu

The Run Tests menu is used to start, stop, and monitor a test run. The Run Tests menu contains the following menu items:

Menu Item	Description
 Start	<p>When the JavaTest harness is not running tests, it enables the Start menu item. Choose the Start menu item to start a test run. Only one test run at a time can be active in the Test Manager window.</p> <p>See Starting a Test Run [p 36] for detailed information about starting a test run.</p>
 Stop	<p>When the JavaTest harness is running tests, it enables the Stop menu item. Choose the Stop menu item to end a test run after the current test is completed.</p> <p>See Stopping a Test Run [p 40] for detailed information about stopping a test run.</p>
Monitor Progress	<p>When the JavaTest harness is running tests you can use the Test Manager Progress Monitor Dialog box to monitor the progress of the test run and current resource information about the test system. Choose the Monitor Progress menu item to open the Test Manager Progress Monitor Dialog box.</p> <p>See Using the Progress Monitor [p 38] for detailed information about using the Test Manager Progress Monitor Dialog box.</p>

### 5.1.3. Report Menu

The Report menu contains menu items that generate and view reports about test run information. The Report menu contains the following menu items:

Menu Item	Description
New Report	<p>Opens the New Report dialog box used to generate reports of test results after a test run.</p> <p>See Generating New Reports [p 63] for detailed information.</p>
Open Report	<p>Opens the Report dialog box used to specify a report.</p> <p>See Viewing Reports [p 64] for detailed information.</p>
<i>Report History</i>	<p>Displays a list of reports that have been generated. Choose a report from the list to open a new instance of it in the Report browser.</p> <p>See Viewing Reports [p 64] for detailed information about the Report browser.</p>

### 5.1.4. View Menu

The View menu contains menu items that display information about a test run:

Menu Item	Description
Filters...	Displays the available view filters and allows you to create additional custom view filters.  See Using Filters [p 57] for detailed description.
Properties	Click the Properties menu item to display the Test Manager Properties dialog box containing the current settings of the Test Manager window.  See Test Manager Properties [p 61] for detailed information about using the Test Manager Properties dialog box.
Test Suite Errors	The JavaTest harness only enables the Test Suite Errors menu item when it detects errors in a test suite. Click the Test Suite Errors menu item to display a dialog box containing current errors detected in the test suite.  See Test Suite Errors [p 62] for detailed information.

### 5.1.5. Tool Bar

The tool bar contains buttons to perform routine tasks that also available as menu items from the menu bar. The JavaTest harness provides tooltips describing each button on the tool bar.

## 5.2. Displaying JavaTest Help and Information

You can display JavaTest help and information about the JavaTest harness in the following ways:

- Help menu
- Help buttons
- F1 key


### 5.2.1. Help Menu

The JavaTest desktop provides a Help menu that you can use to open JavaTest online help, open test suite documentation, display information about the JavaTest harness, display information about the test suite, or display information about the current Java runtime.

When multiple test suites are opened the Help menu displays a menu item for each document provided by the test suite. As test suites are closed, their documents are removed from the Help menu.

### 5.2.2. Help Buttons

The JavaTest GUI provides a Help button on all window tool bars and in all dialog boxes:

- Tool bar help buttons  open the help viewer and display JavaTest online help for that window.
- Help buttons on dialog boxes open the help viewer and display JavaTest online help for that dialog box.

### 5.2.3. F1 Key

Press the F1 key to see information about the JavaTest window that has keyboard focus. Establish keyboard focus in a window before pressing the F1 key. In some cases you may have to highlight something in the window in order to establish focus.

## 5.3. Opening a Test Suite

You can open a different test suite after starting the JavaTest harness by choosing File > Open Test Suite from the menu bar.

The JavaTest harness opens a file chooser dialog box. Use the dialog box to open the test suite.

When you choose a test suite, the JavaTest harness loads the test suite in either a new or empty Test Manager [p 10] window.



Before running tests you must open or create a work directory for the test suite.

## 5.4. Opening a Work Directory

You can open an existing work directory after starting the JavaTest harness by choosing File > Open Directory from the menu bar.

The JavaTest harness opens a file chooser dialog box. Use the dialog box to locate and open the work directory.



Each work directory is associated with a specific test suite and stores its test result files. The test result files contain all of the information gathered by the JavaTest harness during test runs.

When you open an existing work directory, the JavaTest harness only associates it with the current test suite if the test suite is both a match and has no other work directory already open.

If the JavaTest harness cannot associate the work directory with the open test suite, it opens a *new* Test Manager window and loads both the work directory and its test suite.

## 5.5. Creating a Work Directory

You can create a new work directory after starting the JavaTest harness by choosing File > Create Directory from the menu bar.

The JavaTest harness opens a file chooser dialog box. Use the dialog box to create a new work directory for the current test suite.



Each work directory is associated with a specific test suite and stores its test result files. The test result files contain all of the information gathered by the JavaTest harness during test runs.

When you create a new work directory, the JavaTest harness associates it with the current test suite.



## 6. Configuring a Test Run

In order for the JavaTest harness to execute the test suite, it requires information about how your computing environment is configured. The JavaTest harness uses the Configuration Editor to interview you for this information.

Because the quantity and scope of this information depends on the test suite -- some test suites run in diverse environments (different platforms, networks), while others run in very specific, well defined environments -- a test suite may include an interview for the Configuration Editor to use.

If your test suite includes a configuration interview, the JavaTest harness uses the Configuration Editor to interview you for this information.

If your test suite does not include a configuration interview, consult the test suite documentation for directions about how to supply any required configuration information.

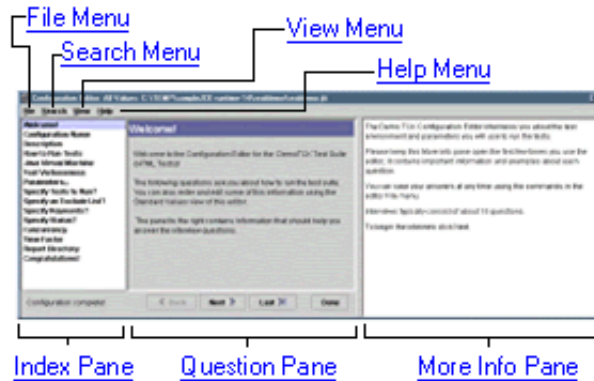
In all cases, you can use the Configuration Editor to specify runtime values that govern which tests in the test suite are run.

This chapter describes how to use the Configuration Editor, the configuration Checklist, the Exclude List browser, the Test Environment browser, and the Question Log:

Topic	Description
Configuring All Values [p 18]	Describes how to use the Configuration Editor's All Values view to create a configuration file for the test run and to search interview titles, questions, and answers for character strings.
Changing Standard Values [p 23]	Describes how to use the Configuration Editor's Standard Values view to modify specific runtime values.
Working with Multiple Configurations [p 31]	Describes how use multiple interview files to switch configurations between test runs
Viewing the Configuration Checklist [p 31]	Describes how to display the checklist of items generated by the configuration editor.
Viewing the Test Environment [p 32]	Describes how to view the environment variables and values used to run a test suite.
Viewing Exclude Lists [p 32]	Describes how to view the list of tests excluded from a test run.
Viewing the Question Log [p 33]	Describes how to browse (in HTML format) the text of all the completed questions asked in the configuration interview and their answers.

## 6.1. Configuring All Values

The Configuration Editor: All Values view consists of a menu bar and three panes:



- File Menu [p 19]
- Search Menu [p 20]
- View Menu [p 20]
- Help Menu [p 20]
- Index Pane [p 21]
- Question Pane [p 21]
- More Info Pane [p 21]

### 6.1.1. Using the Configuration Editor: All Values View

Use the All Values view to determine the characteristics of the test environment for your product -- these characteristics are required by the JavaTest harness in order to run the test suite.

You can use the All Values view to:


- Create or Edit a Configuration [p 18]
- Search an Interview [p 19]
- Save an Interview [p 19]



If the JavaTest harness displays the Standard Values view, choose View > All Values from the configuration editor menu bar.


### Create or Edit a Configuration

To create a new configuration for the test run do one of the following:

- Choose Configure > New Configuration from the menu bar
- Click the  button on the tool bar.

The JavaTest harness opens an empty configuration editor for you.

To edit the current configuration do one of the following:

- Choose **Configure > Edit Configuration** from the menu bar
- Click the  button on the tool bar.

The JavaTest harness opens the current configuration in the configuration editor.

Questions appear in the main text area of the editor (the Question Pane [p 21] ). Answer the questions by using controls such as text boxes, radio buttons, or combo boxes located beneath the question. After you answer each question, click the **Next** button to proceed to the next question.

If you are editing a configuration you can also search the interview for specific characters or character strings in interview titles, questions, and answers. See *Searching the Interview* [p 19] for a description.



Some "questions" provide information and do not require an answer. In these cases, click the **Next** button to proceed to the next question.

You can go backward and forward to any question to review or change your answer by doing one of the following:

- Clicking the **Back** button, the **Next** button, or the **Last** button
- Choosing a question directly from the Index pane [p 21]

As you move backward and forward, answers to questions you previously answered are preserved (until you change them).

After you complete the interview, click the **Done** button to save the interview [p 19] .

## Searching the Interview

When editing a configuration, you can search for a string of characters in interview titles, questions, and answers. The search functionality enables you to easily find answers that you want to change.

To find a character string in an interview, use the **Search > Find** menu item to open the Find Question dialog box.

Refer to the Find Question dialog box [p 22] for a detailed description.

## Saving the Interview

At any point in the interview you can use the **File > Save As**, or **File > Save** menu items to save your answers and your position in the interview. If you are using different configurations to run a test suite, you can save each interview using different file names. See *Working with Multiple Configurations* [p 31] for more information.

### 6.1.2. Using the Menus

The Configuration Editor: All Values View provides the following menus that you can use to manage the configuration interview as well as change Configuration Editor views.

#### File Menu

The File menu contains items to open/save interview files and a log file.

Menu Items	Description
New Configuration	Clears the current interview and starts a new interview from the beginning.
Load	Opens and uses a previously saved interview.
Save	Saves the current interview.  You can choose File > Save at any point in the interview to save your answers and your position in the interview. If the configuration is new, the editor opens the file chooser for you to use in naming and saving the interview. If you do not provide the <code>.jti</code> extension when you name the file, the editor adds the extension when it saves the file.
Save As	Activates a file chooser dialog box that you can use to choose a file in which save the current interview.  You can use the File > Save As command at any point in the interview to save your answers and your position in the interview. If you do not provide the <code>.jti</code> extension when you name the file, the editor adds the extension when it saves the file. When the editor makes that the default interview.
Revert	Discards your changes to the configuration and restores the last saved version of the current interview.
Close	Closes the configuration editor.

## Search Menu

Use the Search menu items to find the occurrence in an interview of a specific character string. When troubleshooting a test run, you can use the Search menu to quickly locate a question whose answer needs to be changed.

Menu Items	Description
Find	Opens the Find Question dialog box [p 22] to search for a string: <ul style="list-style-type: none"> <li>● In question titles</li> <li>● In question text</li> <li>● In your answers</li> <li>● Anywhere (all of the above)</li> </ul>
Find Next	Repeats the previous search.

## View Menu

Use the View menu to display the All Values view or the Standard Values view of the Configuration Editor. The More Info panels can also be hidden or displayed.

## Help Menu

Use the Help menu to display the online help for the All Values view, the Standard Values view, and the Configuration Editor.

### 6.1.3. Using the All Values View Panes

The Configuration Editor: All Values View provides the following panes that you can use to manage and view the configuration interview.

#### Index Pane

The Index pane lists the titles of the questions you have answered, are currently answering, or that the editor determines may need to be answered. The current question is highlighted.

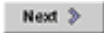


The title is also displayed at the top of the question pane when you are answering a question.



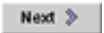

You can click on any question in the list to make it the current question. If you click on a question the list itself does not change; but, if you change an answer that alters your configuration options in the interview, the editor updates the list to reflect the change. You can also use the buttons at the bottom of the Question Pane to navigate through the interview. See Question Pane below for a description of the navigation buttons.

Answers to previously answered questions are always saved until you change them, even if the question does not currently appear in the list.

#### Question Pane

Interview questions appear in the main text area of the editor. You answer the questions using controls such as text boxes, radio buttons, or combo boxes located beneath the question. After you answer each question, click  at the bottom of the panel to proceed to the next question.

The buttons at the bottom of the Question pane control the following editor functions:

Button	Description
	Returns to the previous question.
	Advances as far as possible through the interview.
	Proceeds to the next question.
	Saves your interview answers and exits the editor.

For more information about using the Question pane to fill out the interview see [Create or Edit a Configuration \[p 18\]](#) .

#### More Info Pane

The More Info pane provides you with additional information about each question, such as:

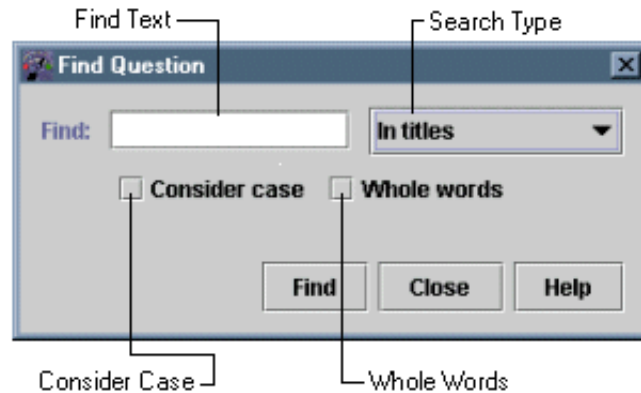
- Background information about the question
- Examples of answers
- Additional information about choosing an answer

To hide or expose the More Info pane choose View > More Info from the menu bar. By default, the More Info pane is exposed.

## 6.1.4. Find Questions

You can locate and display the interview panes containing a specific character string by choosing Search > Find. The Configuration Editor opens the Find Question dialog box for you to use in locating and displaying interview panes containing a specified character string.

The JavaTest harness can search interview titles, questions, and answers for matching characters.



Item	Description
Find text	Enter the character string that you are trying to find.
Search type	Choose the items in the interview that you want to search: <ul style="list-style-type: none"><li>• In titles</li><li>• In text of questions</li><li>• In answers</li><li>• Anywhere</li></ul>
Consider Case	Specifies that the search pattern match the case of the characters in the Find text field.
Whole Words	Specifies that the search pattern only match whole words from the Find text field.

Click the Find button to search the interview for the character string.

To repeat the search, either click the Find button again or use the Search > Find Next menu item from the configuration editor menu bar.

Click the Help button to display context sensitive help.

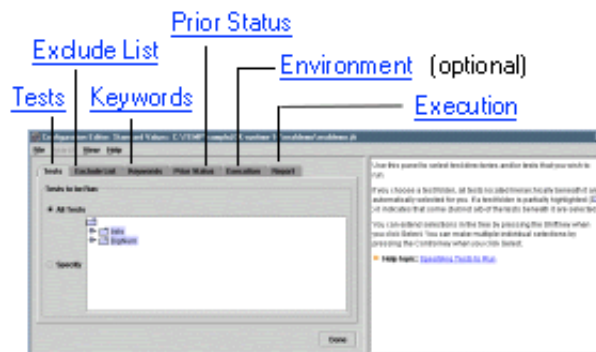
## 6.2. Changing Standard Values

The standard values in a configuration govern the tests in the test suite are run and can change from test run to test run. You can use the Configuration Editor: Standard Values view to quickly view and modify all runtime values in the current configuration.

To use the Configuration Editor: Standard Values View, you must have a current configuration loaded in the configuration editor.


### 6.2.1. Using the Configuration Editor: Standard Values View

You can use the Standard Values view to quickly change the runtime values of a test run.



Depending on you test suite, the Standard Values view can contain either five or six tabbed panes:

- Tests [p 24]
- Exclude List [p 24]
- Keywords [p 26]
- Prior Status [p 27]
- Environment [p 28] (displayed if required by the test suite)
- Execution [p 29]

Choose Configure > Edit Configuration from the menu bar or click the  button on the tool bar. The JavaTest harness opens the configuration editor and loads the current configuration for you.



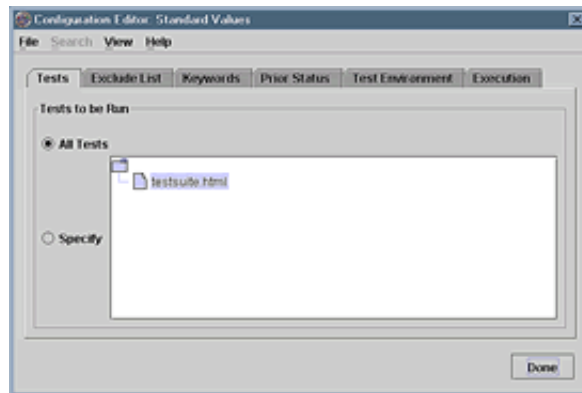
If the JavaTest harness displays the Full Values view, choose View > Standard Values from the configuration editor menu bar.

The Standard Values View allows you to change or specify the following:

- Specifying Tests to Run [p 24]
- Using the Exclude List [p 24]
- Using Keywords as a Filter [p 26] (only when provided by the test suite)
- Using Prior Status as a Filter [p 27]
- Specifying the Test Environment [p 28] (only when required by the test suite)
- Setting Test Execution Values [p 29]

## Specifying Tests to Run

You can use the Tests pane in the Standard Values view [p 23] to specify which tests in a test suite are run.



You can choose to run tests in one of the following:

- All Tests (default setting)
- Specify

If you choose Specify, you can use the test tree in the Tests pane to choose tests or folders containing groups of tests for the JavaTest harness to run. The JavaTest harness walks the test tree starting with the sub-branches and/or tests you specify and executes all tests that it finds (unless they are filtered out).



Restrictions are applied cumulatively. For example, you can specify the tests in a test suite, then restrict the set of tests using an exclude list, and then further restrict the set to only those tests that passed on a prior run.

If you choose a test folder, the JavaTest harness selects all tests in the test suite under that location for the test run.

If you choose one or more individual tests, the JavaTest harness selects those individual tests for the test run.

When a test or folder is highlighted, it indicates that it has been selected for the test run. You can make individual selections in the test tree by pressing the Control key when you click an icon or name in the test tree.

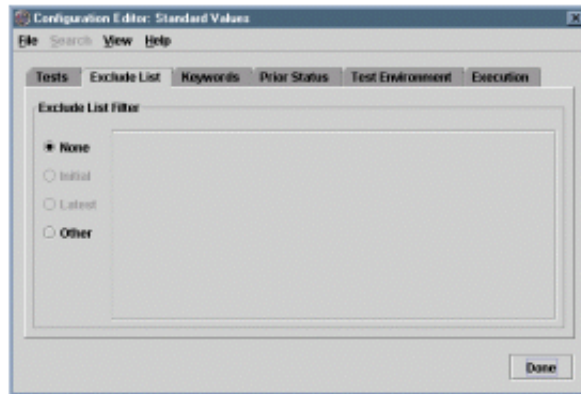
To select a series or sequence of tests or folders, press the Shift key and then click the first and the last icon or name in the sequence.

If a test folder is partially highlighted it indicates that you have selected some (but not all) of the tests beneath it.

## Using Exclude Lists

Exclude list files contain a list of tests in a test suite that are not run by the JavaTest harness. You can use the Exclude Lists pane in the Standard Values view [p 23] to specify one or more exclude list files.





Use the following to specify the exclude list option used to run tests:

- None - an Exclude List is not used
- Initial - enabled if the test suite provides an exclude list. If you choose Initial, the tests are run using the exclude list provided by the test suite.
- Latest - enabled if the test suite provides a location for the test suite. If you choose Latest, additional options are displayed. See Latest Exclude List [p 25] below for detailed information.
- Other - a custom exclude list can be used. See Other Exclude List [p 26] below for detailed information.

If a complete test is added to the exclude list, the JavaTest harness updates the test result status without requiring that the test be rerun. However, if only a test case from a test is added to the exclude list, the JavaTest harness requires that you rerun the test using the updated exclude list before it updates the test result status.

### ***Latest Exclude List***

When you choose Latest, the text field displays the following:

<b>Text and Controls</b>	<b>Description</b>
Location:	Displays the location of the exclude list specified by the test suite. This is a non-editable field.
Last updated:	Displays the date that the exclude list was last updated. This is a non-editable field.
Check For Updates Automatically	When checked the JavaTest harness automatically checks the location of the exclude list and compares the date/time stamps of the remote and local exclude lists. The Javatest harness then displays a dialog box advising you of the results. If a new exclude list is available, you can choose to download it.
Every _ Days	Allows you to choose an interval for the JavaTest harness to automatically check the remote location of the exclude list for updates.
Every Test Run	Allows you to have the JavaTest harness automatically check the remote location of the exclude list for updates before each test run.
Check Now	Click the Check Now button to have the JavaTest harness check the remote location of the exclude list for an update.

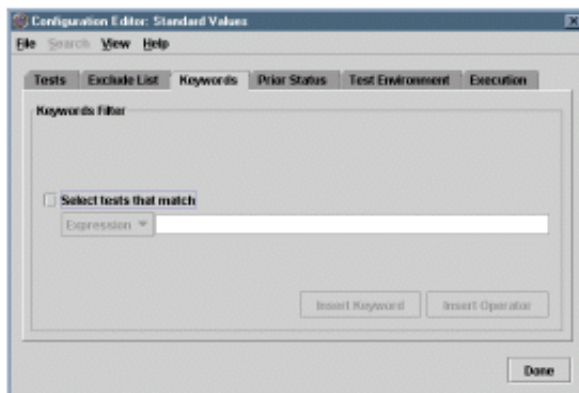
## Other Exclude List

When you choose Other, the text field displays the following:

Button	Description
Add	To select an exclude list file for your test suite, click Add. As you make selections with the file chooser dialog box, they are added to the list. After you have added an exclude list, you can modify the list.
Remove	Clears an item from the list. Select an item in the list and click Remove.
Move Up	Moves an item one position higher in the list. Select an item in the list and click Move Up.
Move Down	Moves an item one position lower in the list. Select an item in the list and click Move Down.

## Using Keywords as a Filter

The JavaTest harness does not display this tab if your test suite does not provide keywords. If your test suite does provide keywords, you can use the Keywords pane in the Standard Values view [p 23] to restrict the set of tests to be run.



To specify the keywords and how they are used to restrict the tests in a test run click Select tests that match. The JavaTest harness enables the Expression, and Insert Operator buttons.

Because The keywords in a test suite are project specific, the JavaTest harness disables the Insert Keyword button if your test suite does not use keywords. When the Insert Keyword button is enabled, you can click it to display the list of the keywords you can use with the tests in the test suite.

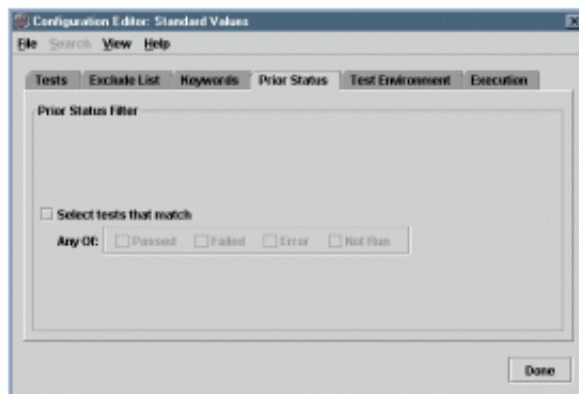
The Insert Operator button displays a list of operators that you can choose from when constructing a boolean expression in the text field.

The following table describes the expressions that can be constructed:

Expression	Description
Any Of	<p>Runs all tests in the test suite having <i>any of</i> the keywords entered in the text field.</p> <p>Example:</p> <p>A test suite uses the keyword "interactive" to identify tests that require human interaction, and "color" to identify tests that require a color display.</p> <p>To execute only the tests containing the "interactive" keyword, choose <i>Any Of</i> and then use the Insert Keyword button to choose the <i>interactive</i> keyword.</p>
All Of	<p>Runs all tests in the test suite having <i>all of</i> the keywords entered in the text field.</p> <p>Example:</p> <p>To execute only the tests containing both the "interactive" and "color" keywords, choose <i>All Of</i> and then use the Insert Keyword button to choose the <i>interactive</i> and <i>color</i> keyword.</p>
Expression	<p>Runs all tests in the test suite having the <i>expression</i> entered in the text field</p> <p>Use the Insert Keyword and the Insert Operator buttons to construct a boolean expression in the text field. Keywords stand as boolean predicates that are true if, and only if, the keyword is present in the test being considered. A test is accepted if the overall value of the expression is true; all other tests are rejected by the restriction.</p> <p>Example:</p> <p>A test suite uses the keyword "interactive" to identify tests that require human interaction, and "color" to identify tests that require a color display.</p> <p>To execute only the tests with the "color" keyword that do not also contain the "interactive" keyword, choose <i>Expression</i> and then use the Insert Keyword button to choose the <i>color</i> keyword, the Insert Operator button to choose the <i>!</i> operator, and the Insert Keyword button to choose the <i>interactive</i> keyword,</p>

## Using Prior Status as a Filter

You can use the Prior Status pane in the Standard Values view [p 23] to restrict the set of tests to be run.



By choosing Select tests that match, you can run tests with restrictions based on their result from a prior test run:

Prior Status	Action
Passed	Selects tests that passed the last time the test was executed.
Failed	Selects tests that failed the last time the test was executed.
Error	Selects tests that the JavaTest harness could not execute the last time it was included in a test run.
Not Run	Selects tests without results in the current work directory.

If a Prior Status filter is selected and the Current Configuration view filter is set, the test tree only displays the status of tests and folders that match the Prior Status filter. The test tree displays all other tests and folders as gray status icons.

During a test run, when the status of a test or folder changes and no longer matches the Prior Status filter, the test tree displays it as a gray icon.

Example:

A test run has failed tests. You set the Prior Status filter to "any of failed" and repeat the test run. As tests pass, the test tree displays gray folder and test status icons, indicating that they no longer match the "any of failed" Prior Status filter.



It is often useful to choose all of the status values except "passed" for the first few test runs, then refine the filtering to reduce the number of tests in subsequent runs.

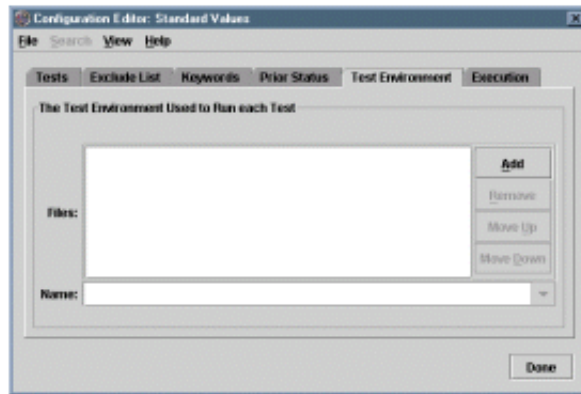
Prior status is evaluated on a test-by-test basis using information stored in result files (.jtr) written in the work directory. Unless overridden by a test suite, a result file is written in the work directory for every test that is executed.

If you change the work directory between test runs, the result files will not be found. If the new work directory is empty, the JavaTest harness behaves as if the test suite was never run.

## Specifying the Test Environment

**NOTE:** The JavaTest harness does not display this tab if your test suite does not use a test environment (.jte) file.

You can use the Test Environment pane in the Standard Values view to create or modify the list of environment files used to run tests in your computing environment.



The Test Environment pane contains two areas:

- Files [p 29]
- Name [p 29]

## ***Files***

Use the buttons in the Files area to create or modify a list of environment files used to run tests in your computing environment:

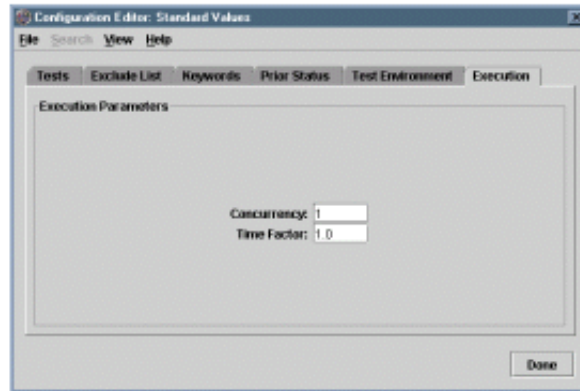
Button	Description
Add	Adds an environment file to the list. Select an environment file (.jtc) for your test suite, click Add. As you make selections with the file chooser dialog box, they are added to the list. After you have added an environment file, you can modify the list.
Remove	Clears an item from the list. Select it in the list and click Remove.
Move Up	Moves an environment file one position higher in the list. Select it in the list and click Move Up.
Move Down	Moves an environment file one position lower in the list. Select it in the list and click Move Down.

## ***Name***

Click the drop-down arrow on this field to see the list of test environments in the environment file(s) listed in the Env Files list. Click the Show button to view the contents of the selected test environment.

## **Setting Test Execution Values**

You can use the Execution pane in the Standard Values view [p 23] to specify how tests in a test suite are run.



Use the Execution pane to set:

- Concurrency [p 30]
- Time Factor [p 30]

## ***Concurrency***

The JavaTest harness can run tests concurrently. If you are running the tests on a multi-processor computer or are using multiple agents on a test system, concurrency can speed up your test runs. Refer to Using a JavaTest Agent [p 81] for detailed information about using agents to run tests.

When using multiple agents to run tests, the concurrency value must not exceed the number of agents. If the concurrency value exceeds the total number of available agents, an error will occur in the test run.

If you have unexpected test failures, run the tests again, one at a time. Some test suites may not work correctly if you run tests concurrently.

For your first test run you should leave this field set to "1." After you have the tests running properly you can experiment with increasing this value.

## ***Time Factor***

To prevent a stalled test from stopping a test run, most test suites use a timeout limit for each test. The timeout limit is the amount of time that the JavaTest harness waits for a test to complete before moving on to the next test.

If you are running the tests on a particularly slow CPU or slow network, you can increase the time limit by specifying a value in the time factor field. Each test's timeout limit is multiplied by the time factor value.

Example:

If you specify a value of "2", the timeout limit for tests with a 10 minute basic time limit becomes 20 minutes.

Try running without a time factor first and increase the value as necessary.

## 6.3. Working with Multiple Configurations

In some testing situations it is useful to use multiple configuration files to switch between different configurations for different test runs.

For example, if you test your product under different versions of the JVM you could specify a different Java launcher command in each configuration.

### 6.3.1. Creating a New Configuration File

Use the editor's File > Save As command to save a configuration to a file name of your choosing. You can save the configuration file anywhere in your file system. Generally, however, the work directory should not be used to save configuration files. Clearing the work directory would also result in deleting the configuration file.

### 6.3.2. Opening an Existing Configuration File

Use the editor's File > Load command to load a previously saved configuration file into the configuration editor. Click the Done button at the bottom of the panel to make that file the new default configuration and use it for test runs. If the Done button is not enabled, you must complete the interview before making it the new default configuration.

## 6.4. Viewing the Configuration Checklist

Configuration interviews may produce a checklist of steps that must be performed before running tests. The checklist is dynamically generated by the JavaTest harness according to the current interview path and/or the answers on the interview path. Questions on the current path may generate items to go on the checklist while different paths through the interview may produce different checklist items.

To view the Configuration Checklist of the current configuration interview choose Configure > Show Checklist. If the interview does not produce a checklist, the JavaTest harness disables the Show Checklist menu item.

To close both the Configuration Checklist and the viewer, click the Close button at the bottom of the window.

## 6.5. Viewing the Test Environment

During troubleshooting you can view the test environment attributes, values, and sources that the JavaTest harness used to run a test suite.

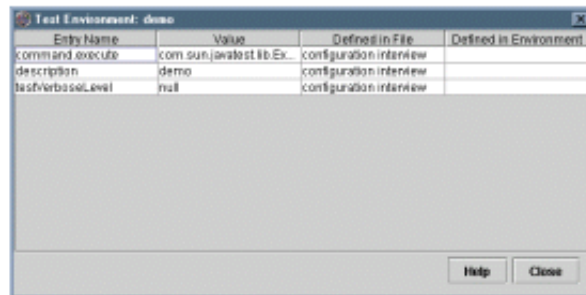
To view the contents of a test environment, open the Test Environment dialog box from the Test Manager window by choosing:

Configure > Show Test Environment

The JavaTest harness opens the Test Environment dialog box [p 32] containing the environment values used to run the tests.

### 6.5.1. Test Environment Dialog Box

To browse the environment variables and values used to run a test suite, open the Test Environment dialog box [p 32] .



The Test Environment dialog box contains a four column table that displays the test environment variables, values, and files used in configuring the test suite. You can rearrange the order and change the size of the columns by clicking and dragging the column headers or their separators.

The Test Environment dialog box contains:

Name	Description
Entry Name	Provides a list of the environment variables specified by the configuration data and used to run the tests.
Value	Provides a list of environment variable values specified by the configuration data and used to run the tests.
Defined in file	Identifies the path and name of the file containing the environment variable information used to run the tests.
Defined in environment	Identifies the environment containing the variables and values used to run the tests.



## 6.6. Viewing Exclude Lists

You can use the Exclude List dialog box to view the list of tests that were excluded from a test run. The dialog box also displays details about individual tests selected in the list.

Open the Exclude List dialog box from the Test Manager window by choosing:

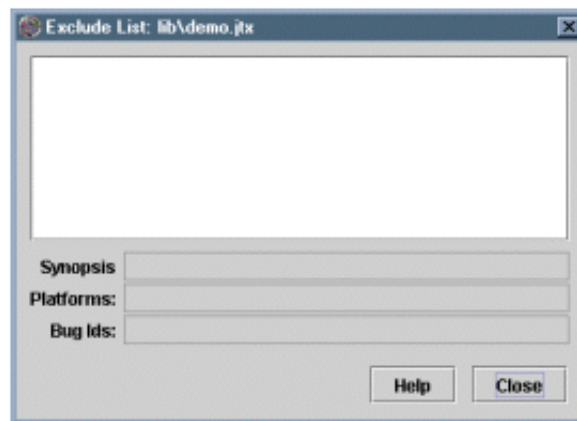
Configure > Show Exclude List

JavaTest opens the Exclude List dialog box [p 33] containing the list of tests excluded from the test run.

### 6.6.1. Exclude List Dialog Box

To view the list of tests that were excluded from a test run, open the Exclude List dialog box [p 32] from the Test Manager window. You can view but not edit the contents of the exclude list.

To add or remove exclude lists in a configuration, use the configuration editor. See *Configuring a Test Run* [p 17] for detailed information.



### Exclude List Contents

Individual tests in the exclude list are displayed in a single column. Click a test in the list to display its details in the text fields at the bottom of the panel.

### Test Details

The JavaTest harness displays the following details about individual tests highlighted in the Exclude List contents area:

Field	Description
Synopsis	Provides annotated information about the excluded test
Platforms	Provides a list of keywords that describe why the test was excluded
Bug Ids	Lists the bug tracking ids for the excluded test

## 6.7. Viewing the Question Log

The JavaTest harness creates a log file of all the completed questions asked in the current, saved configuration interview with their answers. The JavaTest harness does not update the question log with changes to a configuration interview until you save the interview or click the Done button in the

Configuration Editor.

Choose Configure > Show Question Log to view the Question Log of the current, saved configuration interview.

The log provides a list of all questions in the saved configuration interview with links to the following details about each question:

- Interview question
- Question tag
- Question description
- Response (if appropriate)

## 7. Running Tests


You can use the Run Tests menu, Test Manager tool bar, or the test tree popup menu to start a test run. To stop a test run, use either the Run Tests menu or the Test Manager tool bar.

See the Test Manager window [p 10] for a description of the Run Tests menu, Test Manager tool bar, and the test tree popup menu.

This chapter contains the following sections, presented in a sequence that you can use when running tests:


- ▶ Starting a Test Run [p 36]
- ▶ Monitoring a Test Run [p 37]
- ▶ Stopping a Test Run [p 40]
- ▶ Troubleshooting a Test Run [p 41]

## 7.1. Starting a Test Run

When the JavaTest harness is not running tests, it enables both the  button on the tool bar and the Run Tests > Start menu item.



Only one test run at a time can be active in a Test Manager window.

To start a test run using the current configuration data, test suite, and work directory, either click the  button on the tool bar or choose Run Tests > Start in the menu bar. You can also use the test tree popup menu to run a specific test or group of tests in a folder. See Using the Test Tree Popup Menu [p 47] .

Before the JavaTest harness attempts to run the test suite, it determines whether the required configuration information has been supplied. You can view the configuration state in the Test Manger Properties [p 61] dialog box:

- If the configuration state is complete, the JavaTest harness displays a dialog box that identifies the configuration used and allows you to start or cancel the test run. To use a different configuration, choose Cancel and perform Configuring a Test Run [p 17] .
- If the configuration information is not provided or the state is incomplete, the JavaTest harness starts the configuration editor for you to use in Configuring a Test Run. You can also open the configuration editor directly from the Configure menu [p 11] .



To change the test suite or work directory before running tests, refer to:

- Opening a Test Suite [p 15]
- Opening a Work Directory [p 15]
- Creating a Work Directory [p 16]



If the JavaTest Agent [p 81] is used to run the tests for your product, you must start the agent before you begin the test run.

If the JavaTest harness issues a request before the active agent is started, the harness waits for an available agent until its timeout period ends. If the timeout period ends before an agent is available, the JavaTest harness reports an error for the test.

## 7.2. Monitoring a Test Run


After the test run begins, the Test Manager window displays the progress of the test run in the following areas:

Area	Description
Test Tree	<p>The test tree uses colored icons to display the current run and test results status of the folders and tests. As individual tests are completed, the JavaTest harness changes the color of each test tree icon to indicate its status.</p> <p>See Using the Test Tree [p 37] for additional information about using the test tree to monitor the progress of the test run.</p>
Test Progress Display	<p>The test progress display (located at the bottom of the Test Manager window) contains two fields that monitor the progress of the test run.</p> <p>See Using the Test Progress Display [p 38] for detailed information about using the test progress display to monitor the progress of the test run.</p>
Progress Monitor dialog box	<p>The Progress Monitor dialog box displays current, detailed information about the progress of the test run.</p> <p>See Progress Monitor [p 38] for detailed information.</p>
Information Area	<p>As tests run, you can display information about the run in the information area to the right of the test tree. The information area provides two views:</p> <ul style="list-style-type: none"><li>• Folder view - displays information about a folder and its descendents when you click its folder icon in the test tree. The JavaTest harness displays a pane containing a Summary tab, five status tabs, and a status field. For detailed information about browsing folder information see Folder View [p 50] .</li><li>• Test view - displays information about a specific test when you click its icon in the test tree or double-click its name in the information pane. For detailed information about browsing test information see Test View [p 53] .</li></ul>

## 7.3. Using the Test Tree

After the test run begins, you can track its progress using the test tree. The test tree uses specific icons to display the current run and test results status of the folders and tests. As individual tests are completed, the JavaTest harness changes the icons of the test tree to indicate its status.

See Test Tree [p 43] for detailed information about the icons and other features used in the test tree.

The goal of a test run is for the root test suite folder to become  signifying all tests in the test suite (that are not filtered out) have passed. This may require multiple runs. Use the Current Configuration or your test suite specific filter if one is provided.

You can click the test suite icon in the test tree to display information about the run in the Test Manager information area. By browsing the tabbed pane and the test tree, you can find all folders that contain tests that have not passed.

Select a view filter [p 57] from the test tree pane to filter the displayed test tree icons.

## 7.4. Using the Test Progress Display

The test progress display located at the bottom of the Test Manager window contains (from left to right) two areas and two buttons:



- The resizable text area displays information about Test Manager activities, such as the state of the test run and the name of the test being run.
- The monitor area displays either a run progress meter or the elapsed time of the test run, the ▼ button and the 🔍 button.
- The ▼ button opens a drop-down list used to select a monitor for display. See Select a Monitor [p 38] for a detailed description.
- The 🔍 button opens or closes the Progress Monitor dialog box. See Using the Progress Monitor [p 38] for a detailed description.

### 7.4.1. Select a Monitor

Click the ▼ button to open a drop-down list of monitors you can use in the test progress display. You can display either the elapsed time of the test run or a run progress meter.

#### Elapsed Time

Displays the time since the start of the current test run.

#### Run Progress Meter

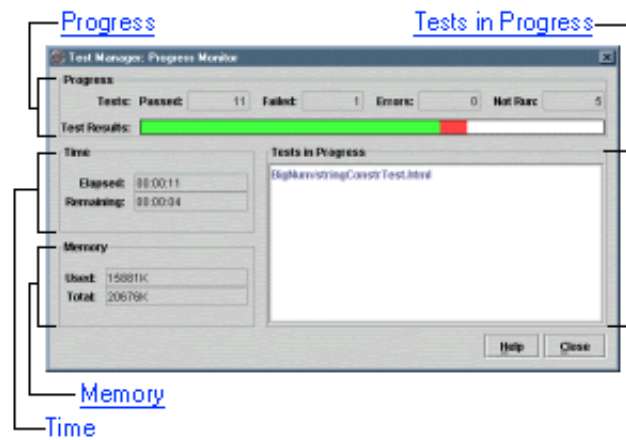
A colored progress bar of the tests in the test suite. As tests are run, they are displayed as colored segments in the progress bar.

The colors used in the progress bar represent the relative quantities of completed tests by their status. The test results are displayed from left to right in the following order:

Color	Status	Description
Green	Passed	Tests with passing results when they were executed.
Red	Failed	Tests with failed results when they were executed.
Blue	Error	The JavaTest harness could not execute these tests. Errors usually occur because the test environment is not properly configured.
White	Not run	Tests that are not yet executed by the JavaTest harness and are not excluded from the test run.

## 7.5. Using the Progress Monitor

The JavaTest harness displays the Test Manager: Progress Monitor dialog box when you choose Run Tests > Monitor Progress from the Test Manager menu bar or click the 🔍 icon in the test status display.



The following information is displayed:

- Progress [p 39]
- Time [p 40]
- Memory [p 40]
- Tests in Progress [p 40]

The information in the dialog box is only for the current configuration and is not changed by the view filter settings. The dialog box only displays information when the harness is running tests.

### 7.5.1. Progress

The Progress area contains the following information:

Name	Description
Passed	Displays the number of tests in the test suite that were run and had passing results
Failed	Displays the number of tests in the test suite that were run and had failing results
Errors	Displays the number of tests in the test suite could not be run
Not Run	Displays the number of tests in the test suite have not yet been run
Test Results	<p>A colored progress bar representing the results of the tests in the test suite</p> <p>As tests are completed, they are displayed as colored segments in the progress bar. The colors used in the progress bar represent the current status of the tests. The progress bar is the same as that displayed at the bottom of the Test Manager window.</p>

The colors used in the progress bar are displayed from left to right in the following order:

Color	Status	Description
Green	Passed	Tests that passed when they were executed
Red	Failed	Tests that failed when they were executed
Blue	Error	The JavaTest harness could not execute these tests. Errors usually occur because the test environment is not properly configured.
White	Not yet run	Tests that have not yet been executed and are not excluded from the test run.

You can also display the progress bar in the test progress display by clicking the ▼ button and then choosing Run Progress Meter from the selectable list. See Using the Test Progress Display [p 38] for detailed description.

### 7.5.2. Time

The Time area contains two fields that are continuously updated throughout a test run:

Name	Description
Elapsed:	The actual elapsed time from the start of the test run
Remaining:	The estimated time required to run the remaining tests.

These numbers can be adversely affected by tests that timeout or have execution times much longer relative to the the other tests being run.

### 7.5.3. Memory

The Memory area contains two text fields and a bar graph.


Name	Description
Used:	The memory used to run the test
Total:	The total memory available for use by the Java virtual machine

### 7.5.4. Tests in Progress

The Tests in Progress text box displays the names of the tests that the JavaTest harness is currently running. The text box is empty when idle. The number of items displayed is directly affected by concurrency settings and /or agents.

You can click on this list to display the appropriate test view in the Test Manager window.

## 7.6. Stopping a Test Run

When the JavaTest harness is running tests, it enables both the  button on the tool bar and the Stop menu item.



Either click the  button or choose Run Tests > Stop to stop a test run.


The JavaTest harness displays a confirmation dialog box, and after it receives your confirmation, finishes currently running tests before stopping the test run.


As it completes each test, the JavaTest harness writes the test results (.jtr files) in the work directory.

Stopping a test run may cause the tests in progress to indicate an error.

## 7.7. Troubleshooting a Test Run

Normally, the goal of a test run is for all tests in the test suite that are not filtered out to have passing results. If the root test suite folder contains tests with errors or failing results, you must troubleshoot and correct the cause to satisfactorily complete the test run.





Tests with errors  are tests that could not be executed by JavaTest. These errors usually occur because the test environment is not properly configured or the software under test is defective.

Tests that failed  are tests that were executed but had failing results.

The Test Manager window provides you with the following facilities for effectively troubleshooting a test run:

- Test Tree
- Folder View
- Test View

### 7.7.1. Test Tree

Use the test tree [p 43] and view filters to identify specific folders and tests that had errors or failing results. Open the red  and blue  folders until the specific tests that failed  or had errors  are displayed.

### 7.7.2. Folder View

When you click a folder [p 50] icon in the test tree pane, the JavaTest harness displays the contents of the folder in the Test Manager information area.

Click the Error and the Failed tabs to display the lists of all tests in and under a folder that were not successfully run. You can double-click a test in the lists to view its detailed test information. Refer to Test View below for a description of the test information that the JavaTest harness displays.

### 7.7.3. Test View

When you click a test icon in the test tree or double-click its name in the folder view, the JavaTest harness displays detailed information about the test in the information area.

The test view contains detailed test information [p 53] and a brief status message at the bottom of the pane that identifies the type of result. This message may be sufficient for you to identify the cause of the error or failure.

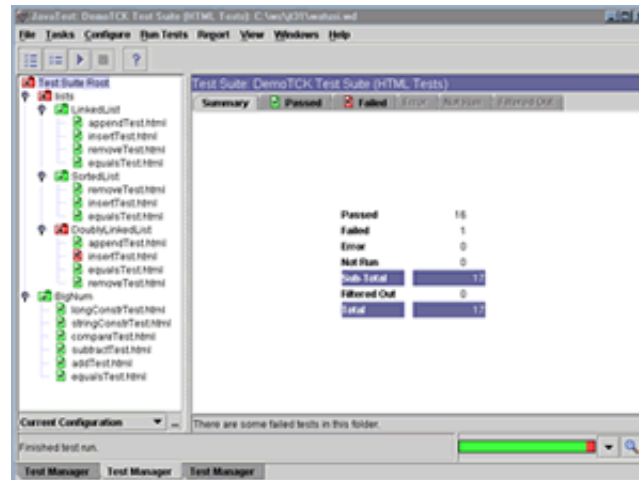
If you need more information to identify the cause of the error or failure, use the following panes listed in order of their importance:

- Test Run Messages [p 55] contains a Message list and a Message pane that display the messages produced during the test run
- Test Run Details [p 55] contains a two column table of name/value pairs recorded when the test was run

- Configuration [p 54] contains a two column table of the test environment name/value pairs actually used to run the test derived from the configuration data

## 8. Browsing Test Information

You can quickly browse test information in the Test Manager window by clicking folder and test icons in the test tree.



This chapter is divided into the following topics:

Topic	Description
Test Tree [p 43]	Describes the test tree and the view filter combo box used to display the test suite, its folders, tests and status icons
Folder View [p 50]	Describes how to display folder information in the Test Manager window
Test View [p 53]	Describes how to display test information in the Test Manager window
Test Manager Properties [p 61]	Describes how to use the View menu to display the Test Manager Properties dialog box
Test Suite Errors [p 62]	Describes how to use the View menu to display the Test Suite Errors dialog box

### 8.1. Test Tree

Use the test tree to view run and test result status of folders and tests in a test suite. The JavaTest harness uses colored icons in the test tree to indicate both the current run status and the test result status of the:

- Folders [p 45]
- Tests [p 46]

The goal of a test run is for the root test suite folder to become green signifying all tests in the test suite that are not filtered out have passed. Use the filters [p 57] to specify tests or groups of tests whose results are displayed in the test tree. Using filters when browsing the tree makes it possible to easily monitor folders containing tests that have not passed.

When you click a folder icon [p 50] in the test tree, the JavaTest harness displays its folder view in the Test Manager information area. The information in the folder view is changed by the view filter.

When you click a test icon [p 53] in the test tree, the JavaTest harness displays its test view in the Test Manager information area. The information in the test view is changed by the view filter.

The test tree also provides a popup menu for each folder and test icon, that allows you to:

Execute these tests

- Performs a "quick pick" execution of the folder or test.

Refresh

- Performs an "on-demand" refresh scans for new folders, new tests and updated test descriptions.

Clear Results

- Performs an "on-demand" clearing of the contents of the selected folder, test, or entire work directory

Folders and tests do not have to be highlighted in the test tree for you to use the popup menu.






See Using the Test Tree Popup Menu [p 47] for additional information.

### 8.1.1. Folder Icons

The test tree uses colored icons to indicate folder run status and result status. The folder icon displayed in the test tree is determined by the result of all its tests (see Test Icons [p 46] ) and by the selected view filter (see Using Filters [p 57] ). Folder icons are updated as individual tests are completed.

#### Result Status

The folder icons displayed in the test tree indicate the highest priority result of any test hierarchically beneath it. The priority order is:

Icon	Result	Color and Description
	Error	A blue folder containing a ! symbol indicates that it and/or one or more of its child folders contains tests with a result of <i>error</i> . Note that this folder <i>may</i> also contain tests and folders that are Failed, Not Run, Passed, and Filtered out.
	Failed	A red folder containing a x symbol indicates that it and/or one or more of its child folders contains tests with a result of <i>failed</i> . Note that this folder <i>may</i> also contain tests and folders that are Not Run, Passed, and Filtered out.
	Not run	A white folder containing a - symbol indicates that it and/or one or more of its child folders contains tests with a result of <i>not run</i> . Note that this folder <i>may</i> also contain tests and folders that are Passed and Filtered out.
	Passed	A green folder containing a ✓ symbol indicates that it and all of its children have a result of <i>passed</i> . Note that this folder <i>may</i> also contain tests and folders that are Filtered out.
	Filtered out	A gray folder containing no symbols indicates that it and all of its children have been filtered out.

#### Run Status

When activity is occurring in a folder such as loading or running tests, the JavaTest harness displays an arrow over the folder icon. The folder icon used indicates the last known test result and does not change until its tests are completed.






After the JavaTest harness completes the tests in a folder, it displays the appropriate result status icon.

### 8.1.2. Test Icons

The JavaTest harness uses icons to indicate test result status. The color of the icon is determined by the test status and the selected view filter (see Using Filters [p 57] ).

#### Result Status

The JavaTest harness uses colored test icons to indicate the result of each test:

Icon	Result	Color and Description
	Error	A blue icon containing a ! symbol indicates that the test is not filtered out and that JavaTest could not execute it. These errors usually occur because the test environment is not properly configured.
	Failed	A red icon containing a x symbol indicates that the test is not filtered out and failed the last time it was executed.
	Not run	A white icon containing a - symbol indicates that the test is not filtered out but has not yet been executed.
	Passed	A green icon containing a ✓ symbol indicates that the test is not filtered out and passed the last time it was executed.
	Filtered out	A gray icon containing no symbols indicates that the test is currently not selected to be run.

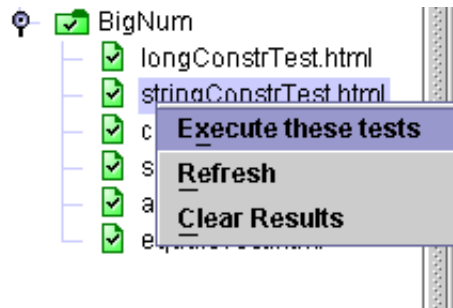
#### Run Indicator

When running a test, the JavaTest harness displays an arrow over the test icon. The test icon used indicates the last known test result and does not change until the test is completed.

After the JavaTest harness completes the test, it displays the appropriate result status icon.

### 8.1.3. Using the Test Tree Popup Menu

The test tree provides a popup menu for each folder and test icon.



**Execute these tests**

Performs a "quick pick" execution of the folder or test.

**Refresh**

Performs an "on-demand" refresh scans for new folders, new tests and updated test descriptions.

**Clear Results**

Performs an "on-demand" clearing of the contents of the selected folder, test, or entire work directory

Operations using the popup menu are not allowed when the JavaTest harness is running tests. The JavaTest harness displays an error dialog box if you attempt to perform an operation using the popup menu when tests are running.

#### "Quick Pick" Test Execution

You can use the Execute these tests menu item to run either a single test or all of the tests in a folder. However, multiple tests and multiple folders cannot be run using the popup menu.

To perform a "Quick Pick" test execution of a folder or test:

1. Display the popup menu for the folder or test. This is a platform specific operation (such as right clicking on the folder or test icon in the test tree).
  - Selecting a single test executes only that test.
  - Selecting a folder executes all tests currently known to the test manager in and below that folder.
2. Choose Execute these tests from the test tree popup menu.

Except for the initial tests setting, which is overridden by using the "Quick Pick" test tree selection, the JavaTest harness uses the current configuration to run the tests.

- If the test manager does not contain a completed configuration interview, the JavaTest harness displays an advisory message and does not start the test run.
  - If the test manager contains a completed configuration interview, the JavaTest harness displays an advisory message to confirm the execute operation.
3. The JavaTest harness updates all icons and progress monitors during test execution, as it does during a normal test run.

The JavaTest harness does not automatically perform a refresh operation before running the tests. If changes have been made to a test suite, you must perform a refresh before running tests. See Refresh Test Suite Contents [p 48] for a description of the refresh operation.

## Refresh Test Suite Contents

When developing tests, changes in a test suite are not automatically detected by the JavaTest harness. The first time tests are run, the JavaTest harness uses the test finder to read test descriptions. If the harness loads tests from an existing work directory, the test descriptions contained in those results will be used by default.

The refresh operation allows test developers to load changes they may have made in a test suite without restarting the JavaTest harness or reloading the test suite.

The JavaTest harness does not require a work directory to perform a refresh of the test suite.

If you are viewing the test panel after refreshing a test or folder, you must choose a different test or folder icon and then repeat your test tree choice to update the test tree.

### ***Refreshing a Single Test***

To refresh the contents of a test:

1. Display the popup menu for a test. This is a platform specific operation (such as right clicking on a test icon).
2. Choose Refresh from the popup menu.
3. The JavaTest harness checks the time stamp of the file containing the test description.
4. If the time stamp has changed, it compares the test descriptions.
5. If the properties of the test descriptions are different, the JavaTest harness:
  - Removes the test result from the work directory and the test manager.
  - Loads a test containing the new test description into the test manager and displays it in the "not run" state.

### ***Refreshing a Folder***

To refresh the contents of a folder:

1. Display the popup menu for a folder. This is a platform specific operation (such as right clicking on a folder icon).
2. Choose Refresh from the test tree popup menu.
3. The JavaTest harness scans for new folders and tests. This operation may take place on any folder, including the root folder.
4. The JavaTest harness checks the time stamps of the files in a folder.
5. If a time stamp has changed, the JavaTest harness compares the test descriptions.
6. If the properties of the test descriptions are different, the JavaTest harness:
  - Removes the test result from the work directory and the test manager.
  - Loads the test containing the new test description into the test manager and displays it in the "not run" state.

## Clear Previous Test Results

You can use the Clear Results menu item to remove existing test results for a single test or for all of the tests in and below a folder.

To clear test results, you must have an open work directory.

### ***Clear a Single Test Result***

To clear a test result:



1. Display the popup menu for a test. This is a platform specific operation (such as right clicking on a test icon).
2. Choose Clear Results from the test tree popup menu.
3. The JavaTest harness:
  - Removes the .jtr file from the work directory for that test.
  - Refreshes the test description for that test.
  - Displays the test in the "not run" state.

### ***Clear Test Results in a Folder***

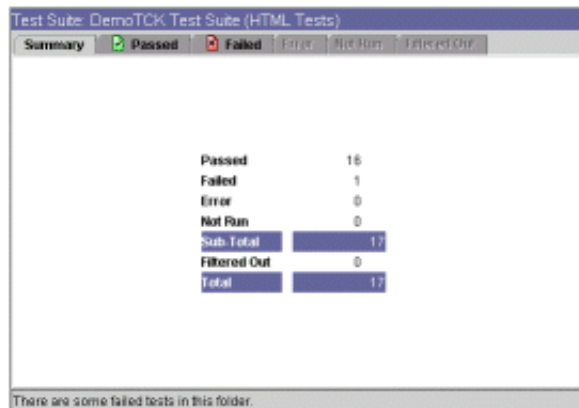
To clear the test result in a folder:

1. Display the popup menu for a folder. This is a platform specific operation (such as right clicking on a folder icon in the test tree).
2. Choose Clear Results from the test tree popup menu.
3. The JavaTest harness:
  - Removes all .jtr files from the work directory for all tests in and below that folder.
  - Deletes all other files in and below the folder in the work directory.
  - Deletes all other directories corresponding to the folders in and below the folder in the work directory.
  - Displays the folder and its tests in the "not run" state.

The JavaTest harness does not display an error message if it is unable to delete a folder or file from the work directory.

## 8.2. Folder View






Click a folder icon in the test tree [p 43] to display its information in the information pane. The folder view displays a Summary tab, five status tabs, and a status display.



During a test run, you can use the folder view to monitor the status of a folder and its tests. You can also use the folder view during troubleshooting to quickly locate and open individual tests that had errors or failed during the test run. When a status pane is empty, the JavaTest harness disables its tab.

See Summary Information [p 50] for a description of the information displayed in the Summary pane.

See Status Information [p 51] for a description of the folder information displayed by clicking the following status tabs:

-  Passed (green)
-  Failed (red)
-  Error (blue)
-  Not Run (white)
-  Filtered Out (gray)

The status display at the bottom of the pane displays messages about the selected tab. The messages may indicate that tests in the folder are loading or may provide detailed status information about a selected test.

### 8.2.1. Summary Information

When you click the Summary tab, it displays the following information about the tests in a folder that are selected by the current configuration:

Field	Description
Passed	The number of tests in a folder and all of its subordinate folders that were run and passed
Failed	The number of tests in a folder and all of its subordinate folders that were run and failed
Error	The number of tests in a folder and all of its subordinate folders that were run but had errors
Not run	The number of tests in a folder and all of its subordinate folders that have not been run and were not filtered out
Sub-Total	The total number of tests that were selected to run
Filtered Out	The total number of tests in a folder and all of its subordinate folders that were filtered out. Tests that were filtered out include tests that you omitted from the test run by using keywords, prior status, or exclude lists.
Total	Total number of tests in a folder and its subordinate folders.

When using the Current Configuration view filter, the numbers are recalculated anytime you use the configuration editor to make a change that effects the view filter. Click the appropriate status information [p 51] tab to identify the individual tests in a category.



The default view filter is Current Configuration. If you set Prior Status as a filter in the Current Configuration or are using a custom filter with "Prior Status" set as a filter, tests will "move" during test execution from their previous status (i.e. failed ) to the filtered out category, not to their new status (i.e. passed). This is the correct, expected behavior because the GUI only displays those tests that match your view filter setting.

Example:

If you choose to run only the failed tests, the tree and statistics show you only the tests currently in the failed state. When tests are no longer in the failed state, they move to the filtered out category. Consequently, when you choose to run only failed tests, as each test passes, it is moved to the filtered out category and the appropriate nodes in the tree turn grey.

To avoid this behavior and view all test results, you should use a different view filter:

- The All Tests view filter displays all test results in the tree and in the Summary tab statistics.
- If your TCK provides a certification filter you can use it to display all test results in the tree and Summary tab statistics.
- You can also configure a Custom view filter to display the same view as the Current Configuration without using the prior status filter.

## 8.2.2. Status Information

In addition to Summary information, the folder view [p 50] contains five status tabs that group and list the tests by their results.



### Passed (green)

Displays the test names of all tests in the folder and all of its subordinate folders that had passing results when they were run.

Click a test in the list to display its test status message in the status display at the bottom of the pane. Double-click a test in the list to display it in the test tree and view its detailed test information [p 53] .

### **Failed (red)**

Displays the path names of all tests in the folder and all of its subordinate folders that were run and had failing results.

Click a test in the list to display its test status message in the status display at the bottom of the pane. Double-click a test in the list to display it in the test tree and view its detailed test information [p 53] .

### **Error (blue)**

Displays the path names of all tests in the folder and all of its subordinate folders with errors that prevented them from being executed.

Click a test in the list to display its test message in the status display at the bottom of the pane. Double-click a test in the list to display it in the test tree and view its detailed test information [p 53] .

### **Not Run (white)**

Displays the path names of all tests in the folder and all of its subordinate folders that are selected by the view filter but have not been run.

Click a test in the list to display its test result message in the status display at the bottom of the pane. Double-click a test in the list to display it in the test tree and view its detailed test information [p 53] .

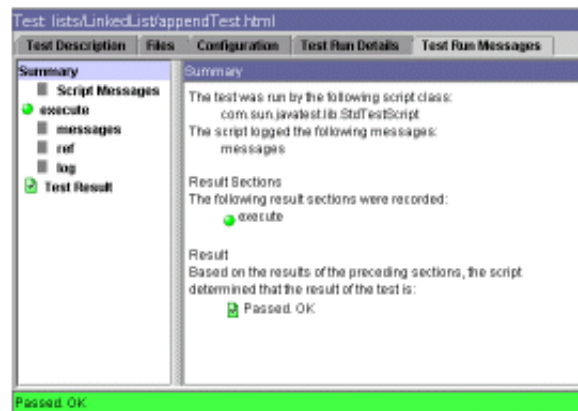
### **Filtered Out**

Displays the path names of all tests in the folder and all of its subordinate folders that were omitted from the test run by the view filter.

Click a test in the list to display its test status message in the status display at the bottom of the pane. Double-click a test in the list to display it in the test tree and view its detailed test information [p 53] .

## 8.3. Test View

Click a test icon in the test tree [p 43] or double-click a test name in the information pane [p 51] to display detailed test information in the information pane. The information pane displays the test view, which contains five tabs and a colored test status message display.



The color of the message display matches the colored test result icon [p 46] and displays a description of the test result. The status field is visible in all of the following:

Tab	Description
Test Description [p 54]	Displays the name/value pairs contained in the test description. The contents are input data and always available.
Files [p 54]	Contains a drop down list of source files from the test description. Click a file name from the drop down list to display its contents. The contents are input data and always available.
Configuration [p 54]	Displays a table of JavaTest environment values used to run a specific test. The contents are output data and only enabled if the test has been run.
Test Run Details [p 55]	Displays the name/value pairs that were recorded when the test was run. The contents are output data and only enabled if the test has been run.
Test Run Messages [p 55]	Contains a tree and message panel of output from sections of the test. Click a name to display its contents. The contents are output data and only enabled if the test has been run.

When an information pane is empty, the JavaTest harness disables it.

### 8.3.1. Test Description

Click the Test Description tab to display a two column table of name/value pairs derived from the test descriptions cached in the work directory. Each test in a test suite has a test description.

When you open the test suite in the JavaTest harness, the test finder reads the test descriptions and caches them in the work directory. Test descriptions in the cache and the Test Description pane are not updated until you close and reopen the test suite.

#### **Name**

The names displayed in the table identify the attributes and properties contained in the test description.

#### **Value**

The values displayed in the table are the attribute and property values that the JavaTest harness used to run the test. The values are read from the files in the test suite.



Refer to your test suite documentation for detailed descriptions of the name/value pairs contained in the Test Description pane.

### 8.3.2. Files

Click the Files tab to display a drop down list of source files and the test description file for the test:

- For compiler tests, the source files are those files that were compiled during the test run.
- For runtime tests, the source files are those files that were previously compiled to create the test class files.

Choose a file from the drop down list to display its contents. You can browse but not edit source files in this panel.

### 8.3.3. Configuration

Click the Configuration tab to display a two column table of test environment name/value pairs derived from the configuration data actually used to run the test.

Because the table contains values that were used when the test was run, it may provide valuable information when troubleshooting a test run.

#### **Name**

The names in the table identify the test environment properties used by the JavaTest harness to run the test.

Because the test environment properties are a function of the configuration data, the contents of this pane vary for each test suite.

#### **Value**

The values displayed in the table are the test environment values that the JavaTest harness used to run the test.



Refer to your test suite documentation for detailed descriptions of the environment name/value pairs for your test.

## 8.3.4. Test Run Details

Click the Test Run Details tab to display a two column table of name/value pairs that were recorded when the test was run and may provide valuable information when troubleshooting a test run.

### Name

The JavaTest harness derives property names from the test results file and displays them in the table:

- Information about the version of the JavaTest harness used to run the test
- Information about the operating system used to run the test
- Date and time the test started
- Date and time the test ended
- Additional details recorded by the test script used to run the test

Because the properties listed in the table are a function of the test script that you are running, the contents vary for each test suite.

Information written by commands, tests, and scripts as they are executing are displayed in the Test Run Messages pane [p 55] .

### Value

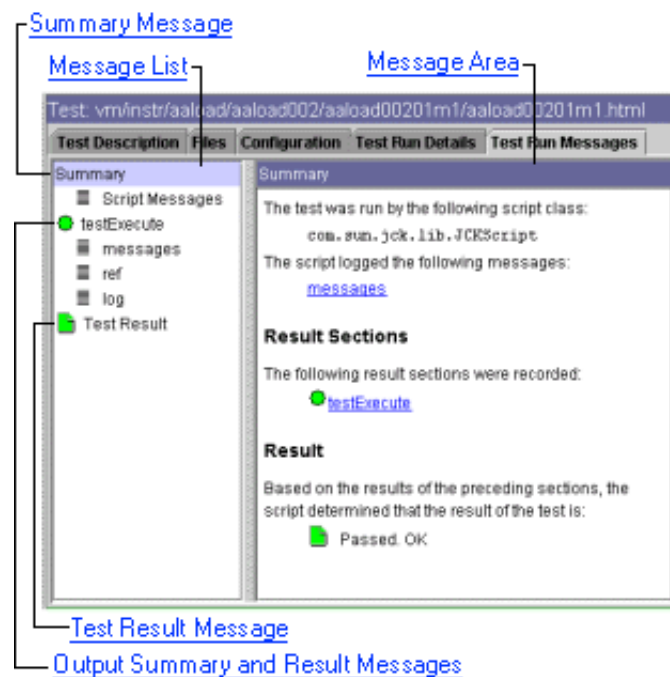
The values displayed in the table are from the test results file created by JavaTest after running the test.



Refer to your test suite documentation for detailed descriptions of the result property name/value pairs for your test.

## 8.3.5. Test Run Messages

Click the Test Run Messages tab to display detailed messages describing what happened during the running of each section of the test. This information is especially useful when troubleshooting a test run.



The Test Run Messages tab contains the following:

- Message List [p 56]
- Message Area [p 56]
- Summary Message [p 56]
- Output Summary and Result Messages [p 56]
- Test Result Message [p 57]

## **Message List**

The message list provides a detailed list of messages issued during a test run. Click an item in the list to display its contents in the messages area.

## **Message Area**

Displays the messages issued during a test run. The number, names, and content vary for each test suite and may also vary for different tests in the same test suite.

## ***Summary Message***

Only one per test, summarizes all of the messages generated during a test run and provides hypertext links to their detailed contents:

- The test script used to run the test
- The messages logged by the test script
- The individual test result sections
- The test result and its result icon

## ***Script Messages***

Only one per tests, passed up from the Script that executed the test. Script messages vary for each test script. Refer to your test suite documentation for detailed descriptions of its script messages when troubleshooting a test run.

## ***Output Summary and Result Messages***

Each test result section has an Output Summary and Result message that provides summary messages and hypertext links to its detailed messages. The name of the Output Summary message is a function of the test suite and varies for each test suite.

Some tests have only one result section, while others have multiple sections. Refer to your test suite documentation for detailed descriptions of the tests when troubleshooting a test run.

The following lists and describes the message types:



Message Type	Description								
Output Summary	<p>A two column table listing the name and size of each output section. Each output section contains text generated while executing the test section:</p> <table> <tr> <th>Name</th><th>Description</th></tr> <tr> <td>messages</td><td>Provides the command string used by the test script to run the test section</td></tr> <tr> <td>ref</td><td>Can provide standard output information from the test section</td></tr> <tr> <td>log</td><td>Can provide standard error information from the test section</td></tr> </table> <p>The ref and the log output are two streams that a test can use when writing test information. These are <i>only example names</i>, although the messages filed always exists in a section.</p> <p>Many tests only use the log stream and include tracing as well as standard error information when writing to the log output. If there are no details in an output section, the JavaTest harness does not create its hypertext link and indicates in the Size (chars) column that it is "empty."</p> <p>The contents of each output section varies from test suite to test suite. Refer to your test suite documentation for detailed descriptions of the test section messages when troubleshooting a test run.</p>	Name	Description	messages	Provides the command string used by the test script to run the test section	ref	Can provide standard output information from the test section	log	Can provide standard error information from the test section
Name	Description								
messages	Provides the command string used by the test script to run the test section								
ref	Can provide standard output information from the test section								
log	Can provide standard error information from the test section								
Result	Contains a colored status icon and a brief description of the results of the specific test section. The color of the circle indicates the result of the test section.								

### ***Test Result Message***

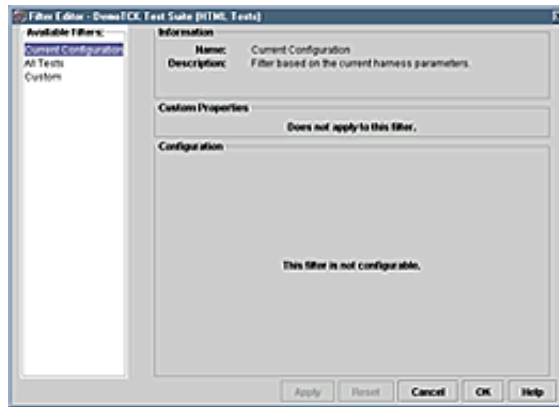
Only one per test, the Test Result Message indicates the cumulative result of the test, determined by the test script from the results of the preceding test sections (if any).



For negative tests, the Test Result correctly indicates "passed" when all of its test sections have failed.

## **8.4. Using Filters**

The JavaTest harness provides a special filtering facility that allows you to filter both the folders and tests displayed by the test tree and the test results printed in the reports. See Generating New Reports [p 63] for a description of using the filters when generating reports.



To select the status of the folders and tests displayed by the test tree, either choose View > Filters from the menu bar or select a view filter from the list at the bottom of the test tree.

Selecting a view filter only filters the status (the colors and counters of the folders and their tests) displayed in the Test Manager window, not the tests that are run. You must use the configuration editor [p 24] to specify the tests that are run.

The JUnit harness provides three filters:

- Current Configuration [p 58]
- All Tests [p 58]
- Custom [p 59]

Edit the Custom filter by choosing View > Configure Filters and using the Filter Editor. See Custom Filter [p 59] for a description of how to edit the custom filter.

Additional filters, such as certification filters, can also be added to the list by the test suite. Refer to your test suite documentation for detailed descriptions of any additional filters displayed in the list of view filters.

### 8.4.1. The Current Configuration Filter

The Current Configuration filter is the default filter and only displays the status of those folders and tests selected in the Configuration Editor.

Examples:

1. You only want to view results for tests in the "api" folder. In the Tests tabbed pane of the Configuration Editor Standard Values view click Specify and the "api" folder. All folders and tests except those under "api" turn gray in the test tree. Those folder and test icons under "api" are displayed in the test tree.
2. You only want to view results for failed tests. In the Prior Status tabbed pane of the Standard Values view click:
  - Select tests that match
  - Any Of: Failed

All folders and tests except those with "failed" test results turn gray in the test tree. Those folder and test icons with "failed" test results are displayed in the test tree. Repeat the test run. As failed tests pass their icons turn from red to gray because they are no longer selected by the configuration criteria.

## 8.4.2. The All Tests Filter

The All Tests filter displays status icons for all folders and tests in the test suite, including those in any exclude list associated with the test suite. This is effectively an unfiltered view of the contents of the work directory.

Example:

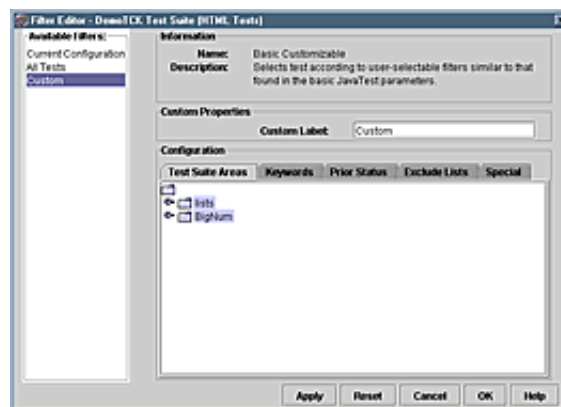
A test run has failed tests. You set the Prior Status filter in the Configuration Editor to run only failed tests and then repeat the test run. The tests pass and the test tree displays green status icons for all folders and tests.

Because you set the All Tests filter, the test tree displays all test and folder status icons regardless of the configuration set in the Configuration Editor.

## 8.4.3. The Custom Filter


When the Current Configuration and the All Test view filters do not provide a suitable view of the test tree, you can use the Filter Editor to edit the Custom filter for the test suite.

The Custom filter is unique to a test suite. When the JavaTest harness opens the test suite in the Test Manager window, it also restores its Custom filter. You can also use the Custom filter to generate reports. See Generating New Reports [p 63] for a description of using the Custom filter when generating reports.



## Editing the Custom Filter

To edit the Custom filter:

1. Choose View > Filters > Configure Filters from the menu bar or click the  button at the bottom of the test tree.
2. Choose Custom in the Available Filters panel. You can provide a name for the filter in the Custom Label field.
3. Use the Test Suite Areas, Keywords, Prior Status, Exclude Lists, and Special tabs to set the view filter properties:
  - Specifying Tests to View [p 60]
  - Using the Exclude List as a Filter [p 60]
  - Using Keywords as a Filter [p 60]
  - Using Prior Status as a Filter [p 60]
  - Using Special Settings as a Filter [p 60]
4. Click one of the following buttons:
  - Apply - saves but does not dismiss the dialog box. Apply updates the GUI if the filter is selected.
  - Reset - discards all changes and restores the last saved Custom filter.

- Cancel - closes the dialog box without saving any changes.
- OK - saves the current changes, updates the Custom filter, and closes the dialog box.
- Help - displays online help for the Filter Editor.

Using Test Suite Areas, Keywords, Prior Status, Exclude Lists, and Special settings to filter the displayed test results does not stop the tests from running. To specify which tests are run, use the Configuration Editor Standard Values View [p 23] .

## ***Specifying Tests to View***

Click the Test Suite Areas tab and use the tree to choose the results of test folders or individual tests that you want displayed in the Test Tree. The JavaTest harness walks the test tree starting with the sub-branches and/or tests you specify and displays the results of all tests that it finds.

## ***Using Keywords as a View Filter***

If your test suite provides keywords and you want to use them to filter the test results displayed in the Test Tree, click the Keywords tab.

See Specifying Tests to Run [p 24] for a description of the behavior of this view filter.

See Using Keywords as a Filter [p 26] for a description of how to select the keywords available for your test suite.


## ***Using Prior Status as a View Filter***

Click the Prior Status tab and choose the test results from the previous test run that you want displayed in the Test Tree.

See Specifying Tests to Run [p 24] for a description of the behavior of this view filter.

## ***Using the Exclude List as a View Filter***

To use the exclude list specified in the configuration interview as a filter, click the Exclude Lists tab and its check box.

Any test in the exclude list is filtered out and displayed as a  icon in the test tree.

## ***Using Special Settings as a Filter***

The test suite architect has the option of providing a default filter for the test suite. The default filter is automatically used in the Current Configuration view.

Select this filter setting when you are simulating the current configuration without actually changing the interview.

Example:

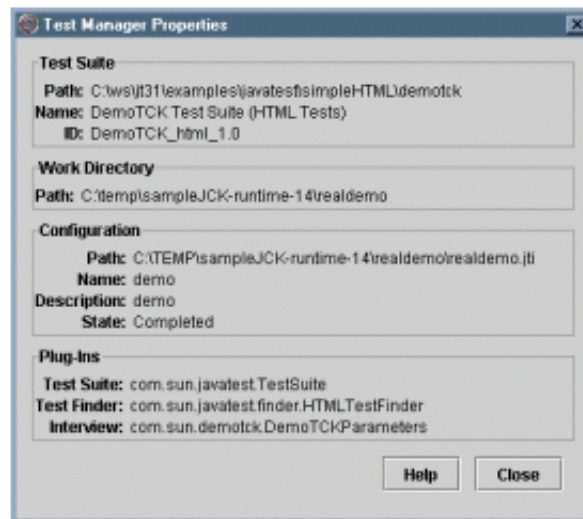
You want to display the tests in the GUI that are marked with a keyword and do not want to change your interview, click Enable test suite filter.

## ***Using a Custom View Filter***

To use a custom filter, choose it from the list of view filters below the test tree or from the View > Filters menu. The JavaTest harness displays the status of the folders and tests in the test tree that match the filter settings of the custom filter.

## 8.5. Test Manager Properties

To view the properties of a test manager, choose View > Properties. The JavaTest harness opens the Test Manager Properties dialog box.



The Test Manager Properties dialog box contains four areas:

- Test Suite [p 61]
- Work Directory [p 61]
- Configuration [p 61]
- Plug-Ins [p 61]

### 8.5.1. Test Suite

The Test Suite properties area displays the Path, Name, and ID of the current test suite opened by the test manager.

### 8.5.2. Work Directory

The Work Directory properties area displays the path of the current work directory opened by the test manager.

### 8.5.3. Configuration

The Configuration properties area displays the Path, Name, Description, and State of the current configuration interview opened by the test manager. The State field indicates whether the configuration is complete and tests can be run as well as identifies the availability of special filters.

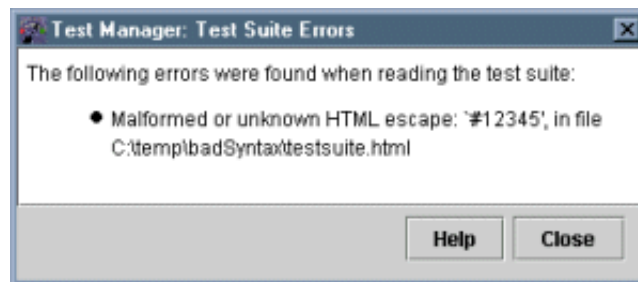
### 8.5.4. Plug-Ins

The Plug-Ins properties area displays the name of the plugins used by the Test Manager. The plug-ins are provided by the test suite architect.

Property	Description
Test Suite	The fully qualified name of the test suite class used by the test manager
Test Finder	The fully qualified name of the test finder class used by the test manager
Interview	The fully qualified name of the interview class used by the test manager

## 8.6. Test Suite Errors

The JavaTest harness displays the Test Manager: Test Suite Errors dialog box when you choose View > Test Suite Errors from the Test Manager menu bar.



The dialog box displays a list of errors detected in the test suite. Unless instructed otherwise, you should report any test suite errors to the owner of the test suite.

## 9. Using Test Reports

You can use the JavaTest harness to generate and view reports about test run information such as:

- Tests grouped by test result
- Configuration information used
- Test environment names and values used

The JavaTest harness does not automatically generate reports at the end of a test run. See [Generating New Reports \[p 63\]](#) for a description of how to generate test reports.

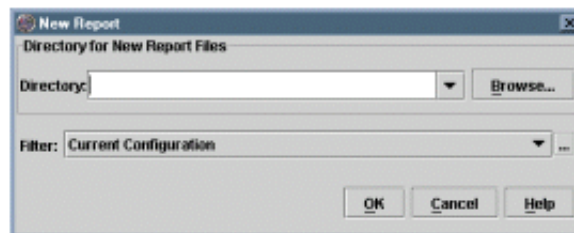
To view reports in the JavaTest Report Browser, choose **Report > Open Report** from the menu bar. See [Viewing Reports \[p 64\]](#) for a description of how to view reports.

The test reports contain relative and fixed links to other files. See [Moving Reports \[p 66\]](#) for a description of how to move reports to another directory.

### 9.1. Generating New Reports

The JavaTest harness does not automatically generate reports of test results after a test run. You can generate test reports either from the command line in batch mode (see [Writing Reports in Batch Mode \[p 105\]](#)) or from the JavaTest GUI. To generate a test report from the JavaTest GUI, you must:

1. Choose **Report > New Report** from the Test Manager menu bar. The JavaTest harness opens the New Report dialog box.



2. Type the name of a report directory in the Directory field or click the Browse button to specify where to put the new reports. You can either specify a new directory or an existing directory. If you ran reports earlier, it displays the directory from the previous run. If you use an existing report directory, the JavaTest harness saves the previous reports as backups and then writes the new reports.
3. Choose a filter option to generate reports. You can use the Current Configuration filter, All Tests filter, or a custom filter that you have created. Using a custom filter allows you to generate test reports for a specific set of test criteria.
  - See [Using Filters \[p 57\]](#) for a description of the filters.
  - See [Custom Filters \[p 59\]](#) for a description of how to create custom filters.
4. After specifying a report directory and choosing a filter, click the OK button to generate new reports.

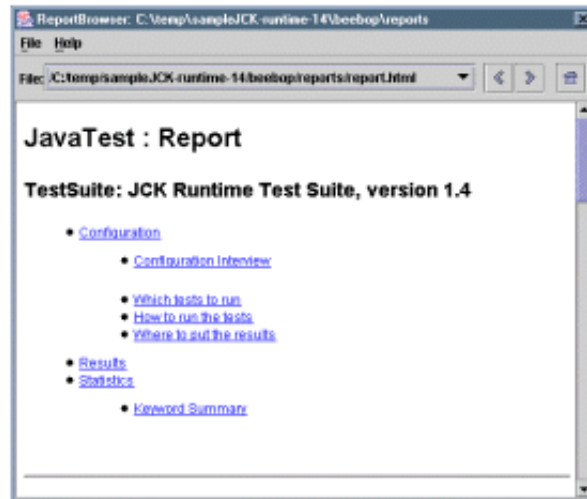
The JavaTest harness writes the reports and then displays a dialog box that gives you the option of either viewing the new reports in the report browser or returning to the Test Manager window.

## 9.2. Viewing Reports




You can use the JavaTest Report browser or a web browser to view reports.

### 9.2.1. View Reports in the Report Browser

Choose Report > Open Report from the menu bar. The JavaTest harness opens a file chooser dialog box for you to specify a report directory. When you choose a report directory, the JavaTest harness opens the Report Browser and displays the report.html file in that directory.



The Report Browser consists of:

Component	Description
Menu bar	<p>The menu bar contains a File and a Help menu.</p> <ul style="list-style-type: none"><li>● Use the File menu to generate new reports, open existing reports, and to close the Report Browser.</li><li>● Use the Help menu to display Report Browser online help.</li></ul>
Tool bar	<p>The tool bar contains a File field and three navigation buttons.</p> <p>The File field displays the name of the current report and provides a drop down list of reports previously opened in the browser. As reports are opened the JavaTest harness adds their names to the drop down list allowing you to navigate to any previously displayed report.</p> <div> Returns to the previously displayed report page.</div> <div> Opens the next report page that was displayed.</div> <div> Returns to the report.html file. The report.html file is the root page and links to all of the other test report pages.</div>
Contents area	<p>The Report Browser displays the report file contents in the area below the tool bar. The Report Browser displays text files as well as html files. For html files, you can use hyperlinks in the report to display additional related reports.</p>



## 9.2.2. View Reports Offline

If you choose to view the reports offline, use a web browser to open the report.html file located in the appropriate report directory.

The report.html file is the root file and links to all of the other HTML reports.

Report Files	Description
config.html	Contains the configuration interview questions and your answers used for the test run.
env.html	Contains the test environment names and values that were used for the test run.
error.html	Contains a list of the tests that had errors and could not be run.
excluded.html	Contains a list of the tests that were excluded from the test run.
failed.html	Contains a list of the tests that were executed during the test run but failed.
notRun.html	Contains a list of all tests that were not excluded from the test run but were not run.
passed.html	Contains a list of the tests that were executed during the test run and passed.
report.html	The root file that links to all of the other HTML reports.

The JavaTest harness also generates a summary.txt report that contains a list of all tests that were run, their test results, and their status messages. You can open the summary.txt file in any text editor.

## 9.3. Moving Report Files

The test reports contain relative and fixed links to other files that may be broken when moving reports to other directories.

The JavaTest harness provides a utility that you can use to move and update the links in the test reports when moving the files to another directory.

A detailed description of the utility is available in Appendix A of the JavaTest User's Guide or in your TCK at:

`doc/javatest/editlinks.html`

## 10. Auditing a Test Run

The JavaTest harness includes an audit tool that you can use to analyze the test results in a work directory. The audit tool verifies that all tests in a test suite ran correctly and identifies any audit categories of a test run that had errors.

You can audit a test run in GUI mode or in batch mode.

### 10.1. Auditing in GUI Mode

Choose Tasks > Audit Test Results from the menu bar.

The JavaTest harness opens the Audit Test Results window. If you are running the audit for that first time, the JavaTest harness also opens the Options dialog box for you.

Open the Options dialog box and specify the reference test suite, the work directory to audit, and the reference configuration file. See Setting Audit Options [p 67] for a description of the Options dialog box.

Click the Start Audit button at the bottom of the Options dialog box.

When the JavaTest harness completes the audit it displays the results in the Audit Test Results window [p 69] .

To repeat the audit, choose Audit > Options from the menu bar to open the Options dialog box and click the Start Audit button.

To close the Audit Test Results window, choose File > Close from the menu bar.

### 10.2. Auditing in Batch Mode

See Using Batch Mode [p 105] for a detailed description of using batch mode to audit a test run.

### 10.3. Setting Audit Options

Use the Options dialog box to specify the reference test suite, the audited work directory, the reference configuration, and to start the audit.

To display the Options dialog box, choose Audit > Options from the menu bar.





The Options dialog box contains:

- Test Suite [p 68]
- Work Directory [p 68]
- Configuration File [p 69]
- Start Audit button [p 69]
- Cancel button [p 69]
- Help button [p 69]

### 10.3.1. Test Suite

You can use the drop down list or the chooser to specify a test suite. You can also clear a previous entry in the Test Suite field by choosing the empty line from the drop down list. A blank field indicates a test suite is not set.

Click the  button to open the list of test suites currently loaded in the JavaTest harness. You are not limited to using these tests suites. You can either choose a test suite from the list or click the  button to open the dialog box used to choose another test suite.


If you choose a reference test suite, the JavaTest harness sets the entries in the work directory drop down list to the work directories that are currently loaded and match the specified test suite. If you have multiple test suites and work directories, specifying a test suite can simplify choosing the options.


The JavaTest harness always uses the test suite associated with the work directory that you choose to audit. See Work Directory below for a description of how to choose a work directory to audit.

### 10.3.2. Work Directory

The JavaTest harness audits the work directory named in the Work Directory field. A blank field indicates the work directory is not set.

You can use the drop down list or the chooser to specify the work directory to audit. You can also clear a previous entry in the work directory field by choosing the empty line in the drop down list.

Click the  button to open the list of work directories identified by the JavaTest harness. If you choose a reference test suite, the JavaTest harness only lists the work directories associated with it in the drop down list.

You are not limited to using these work directories. You can either choose a work directory from the list or click the  button to open the dialog box used to choose another work directory.


If you choose a work directory, the JavaTest harness uses the test suite associated with the work directory and sets the entries in the configuration files drop down list to those most recently used with the work directory.


If you do not choose a work directory, the JavaTest harness uses the work directory associated with the configuration file that you specify. See Configuration File below for a description of how to choose a reference configuration file.

### 10.3.3. Configuration File

You can use the drop down list or the chooser to specify a reference configuration file. You can also clear a previous file from the Configuration File field by choosing the empty line in the drop down list.

A blank field indicates that the default configuration file for the chosen work directory is used. If a work directory is not chosen, you can choose a reference configuration and the JavaTest harness opens its work directory.

Click the  button to open the list of configuration files identified by the JavaTest harness. The JavaTest harness lists the configuration files associated with the work directory.

Choose a file from the list or click the  button to open the dialog box used in choosing a configuration file.

If you specify a configuration file it must be associated with the work directory. If the configuration file is not associated with the work directory the JavaTest harness displays an error message without performing the audit.

### 10.3.4. Start Audit Button

After you set the audit options, click the Start Audit button to audit the work directory. The JavaTest harness closes the Option dialog box and displays a message in the Audit Test Results window that it is performing the audit.

### 10.3.5. Cancel Button

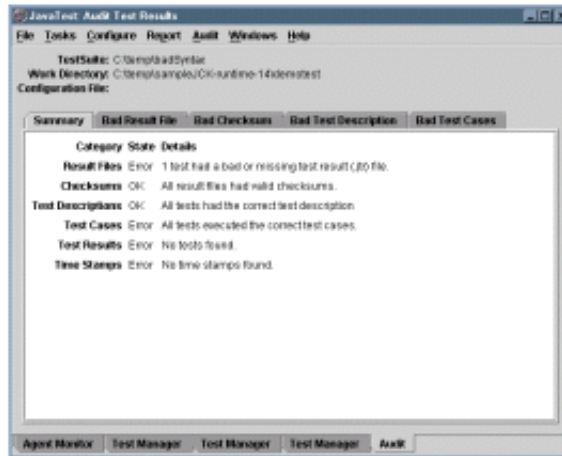
Closes the Options dialog box without accepting any changes to the option fields.

### 10.3.6. Help Button

Displays online help for the Options dialog box.

## 10.4. Audit Test Results Window

Use the Audit Test Results window to generate and view the audit report of the tests in a work directory. To display the Audit Test Results window choose Tasks > Audit Test Results from the menu bar.



The Audit Test Results window contains:

Component	Description
Menu bar	<p>The Audit menu bar contains an Audit and a Help menu. The menu bar also contains JavaTest standard menus when the Tabbed and SDI window styles are used. See Changing Window Styles [p 77] for a description of the different styles of windows that the JavaTest harness provides.</p> <ul style="list-style-type: none"> <li>• Use the Audit menu to set options used to generate an audit report. See Setting Audit Options [p 67] for a description of the Options dialog box.</li> <li>• Use the Help menu to display online help for the Audit Test Results window.</li> <li>• See JavaTest Menus [p 72] for a description of the JavaTest standard menus that can also be displayed on the menu bar.</li> </ul>
Text fields	<p>Three text fields are displayed at the top of the window:</p> <ul style="list-style-type: none"> <li>• Test Suite: contains the name and location of the reference test suite.</li> <li>• Work Directory: contains the name and location of the audited work directory.</li> <li>• Configuration File: contains the name and location of the reference configuration interview.</li> </ul>
Tabbed panels	<p>The JavaTest harness displays the audit report in five tabbed panels:</p> <ul style="list-style-type: none"> <li>• Summary [p 70]</li> <li>• Bad Result File [p 71]</li> <li>• Bad Checksum [p 71]</li> <li>• Bad Test Description [p 71]</li> <li>• Bad Test Cases [p 71]</li> </ul>

### 10.4.1. Summary

The Summary panel provides the state and details for each of the following audit categories:

Category	Description
Result Files	Displays a summary of the audit for corrupted and missing test result files. See Bad Result File [p 71] for a description of the detailed tab.
Checksums	Displays a summary of the audit of the test result files checksums. See Bad Checksum [p 71] for a description of the detailed tab.
Test Descriptions	Displays a summary of the audit of the test descriptions. See Bad Test Description [p 71] for a description of the detailed tab.
Test Cases	Displays a summary of the audit of the exclude list. See Bad Test Cases [p 71] for a description of the detailed tab.
Test Results	Displays a summary of the test results audit. To pass the audit, all tests must have passing results.
Time Stamps	Displays the date and time when the first and the last tests were run.

### 10.4.2. Bad Result File

The Bad Result File panel lists any corrupted or missing test result files.

A corrupted or missing test result file indicates that the result file was edited.

### 10.4.3. Bad Checksum

The Bad Checksum panel lists all test result files with invalid checksums.

The result file checksums must match the reference checksums in the reference test suite. An invalid checksum indicates that a result file was edited.

### 10.4.4. Bad Test Description

The Bad Test Description panel lists all test files having edited test descriptions.

Test descriptions must match the reference test descriptions in the reference test suite. An invalid test description indicates that the test file was edited.

### 10.4.5. Bad Test Cases

The Bad Test Cases panel lists all tests failing to execute the required test cases.

The test cases run must match the exclude list for the reference test suite. Tests failing to execute the required test cases indicates that the exclude list was edited.

## 11. Customizing the JavaTest GUI

This topic describes the graphical user interface (GUI) that the JavaTest harness provides for running, managing, and auditing tests and how you can customize it for your use.

Refer to Using a JavaTest Agent [p 81] for a description of the user interfaces that the JavaTest harness provides for agents running tests on remote systems.

## 11.1. The JavaTest GUI

The JavaTest GUI contains a set of windows [p 72] and menus [p 72] that you use to configure and run tests, monitor test and agent status, evaluate and analyze test results, and include or exclude tests from test runs.

You can display the graphical user interface in one of three styles:

- A single top-level window that displays all JavaTest windows as tabbed panes
- A single top-level window that contains all JavaTest windows
- A separate top-level window for each JavaTest window

Differences between the window styles are described in the Changing Appearance Preferences [p 76] .

## 11.2. The JavaTest GUI Windows

The JavaTest graphical user interface provides a set of windows for you to use when running and managing tests:

Window	Description
Test Manager [p 10]	Contains panels, menus, and controls that you use to: <ul style="list-style-type: none"><li>● Open and create work directories</li><li>● Create or modify the information that the JavaTest harness uses when running your tests</li><li>● Run the tests of a test suite</li><li>● Monitor tests and test results while they are being run</li><li>● View test environment settings of your configuration</li><li>● View the contents of an exclude list</li><li>● Browse completed tests and test results</li></ul>
Agent Monitor [p 94]	Contains controls that you use to set up the JavaTest agent pool for monitoring active agents and to provide feedback about the status of both passive and active agents.
Audit Test Results Window [p 69]	Contains panels and menus that you use to generate and view audit reports of the tests in a work directory.

## 11.3. JavaTest Menus

The JavaTest GUI provides two types of menus for you to use when performing tasks:



Menu Type	Description
Tool Menus	<p>These menus are only available for use in a specific window and are described in the section describing that window.</p> <ul style="list-style-type: none"> <li>● See Test Manager Window [p 10] for a description of the Test Manager menus.</li> <li>● See Audit Test Results Window [p 69] for a description of the Audit Test Results menus.</li> <li>● See Agent Monitor Window [p 94] for a description of the Agent Monitor menus.</li> </ul>
Standard Menus	<p>These menus are available for use in all JavaTest windows and include:</p> <ul style="list-style-type: none"> <li>● File [p 73]</li> <li>● Tasks [p 74]</li> <li>● Window [p 75]</li> <li>● Help [p 75]</li> </ul>

### 11.3.1. File Menu

Use the File menu to open files, set user preferences, close windows, and exit from the JavaTest harness. The contents of the File menu change dynamically, based on the context of the desktop. The JavaTest harness only enables menus when they can be used.

Menu Item	Description
Open Test Suite	<p>Opens a file chooser dialog box you can use to choose a test suite.</p> <p>When you choose a test suite, the JavaTest harness loads the test suite in an empty Test Manager [p 10] window.</p>
Open Work Dir	<p>Opens a file chooser dialog box you can use to choose an existing work directory.</p> <p>The JavaTest harness associates the work directory with an open test suite if the test suite is both a match and has no other work directory already open.</p> <p>If the JavaTest harness cannot associate the work directory with an open test suite, it opens a new Test Manager window and loads both the work directory and its associated test suite.</p>
New Work Dir	<p>Opens a file chooser dialog box for you to use in creating a new work directory for the current test suite.</p> <p>When you create a work directory, the JavaTest harness associates it with the current test suite.</p> <p>Each work directory is associated with a specific test suite and stores its test result files in a cache. The test result files contain all of the information gathered by the JavaTest harness during test runs.</p>
Preferences	<p>Opens the JavaTest Preferences [p 75] dialog box for you to set the display and functional options of the JavaTest harness.</p>
<i>File History</i>	<p>Displays a list of work directories and test suites that have been opened.</p> <p>Choose a file from the list to open a new instance of it in the current session.</p>
Close	<p>Shown in the File menu when window styles is set to Tabbed or SDI. Closes the current window without exiting from the JavaTest harness.</p> <p>Closing a Test Manager window closes a test session. To start another session using the same test suite and work directory, use the Open Work Directory menu above.</p>
Exit	<p>Exits from the JavaTest harness. When you exit, your current desktop is saved so that all open windows can be restored in your next JavaTest session.</p>

### 11.3.2. Tasks Menu

Use the Tasks menu to open the windows required to perform specific tasks:

Menu Item	Description
Monitor Agent Activity	<p>Opens a new Agent Monitor window [p 94] for you to:</p> <ul style="list-style-type: none"> <li>• Setup the Agent Pool for Active Agents.</li> <li>• Monitor communication between the JavaTest harness and JavaTest Agents.</li> </ul>
Audit Test Results	<p>Opens the Audit Test Results [p 69] window. Only one Audit Test Results window can be open.</p>

### 11.3.3. Windows Menu

Use the Windows menu in all window styles to select the active window by clicking on it's name in the window list.

In MDI and SDI window styles, you can also use the Windows menu to manage the layout of the open windows. See Managing JavaTest Windows [p 80] for a description of how you can use the Windows menu to manage the window layout.

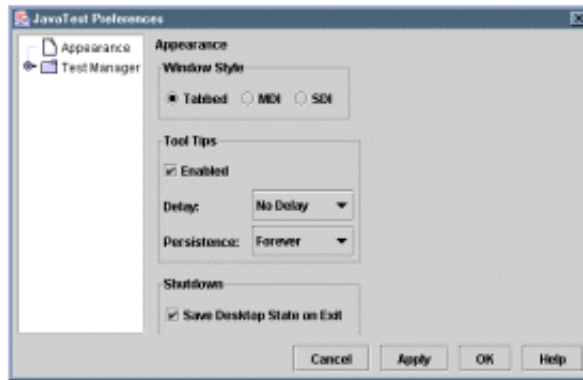
### 11.3.4. Help Menu

Use the Help menu to display online help for the window, JavaTest online help, available test suite documentation, JavaTest harness information, and current Java runtime information:

Menu Item	Description
active window	Opens the help viewer and displays online help for the active window.
JavaTest	Opens the help viewer and displays online help for the JavaTest harness.
User's Guide	Displays information about the location of the JavaTest User's Guide.
About JavaTest	Displays information about this release of the JavaTest harness.
About Java	Displays information about the Java runtime used to run the JavaTest harness.

## 11.4. Setting JavaTest Preferences

You can use the JavaTest Preferences dialog box to set the display and functional options of the JavaTest harness.



Open the JavaTest Preferences dialog box from the menu bar by choosing:

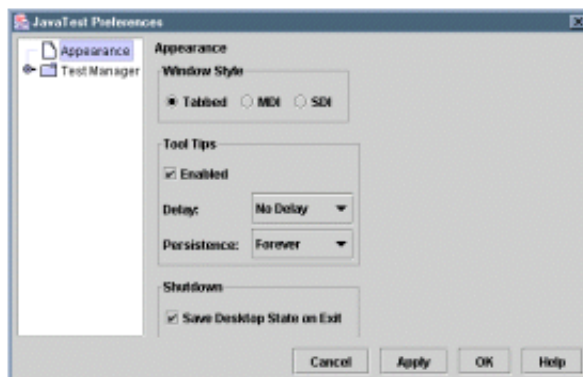
File > Preferences

The JavaTest Preferences dialog box contains the following categories:

Preference Category	Description
Appearance	Use the Appearance category to set how the JavaTest graphical user interface displays its windows and tool tips. See Changing Appearance Preferences [p 76] for detailed information.
Test Manager	Use the Test Manager category to set the Test Manager tool bar property. See Changing Test Manager Preferences [p 78] for detailed information.
Configuration Editor	Use the Configuration Editor category to set the Configuration Editor properties. See Changing Configuration Editor Preferences [p 79] for detailed information.

### 11.4.1. Changing Appearance Preferences

To change the appearance preferences for the JavaTest GUI, choose File > Preferences from the menu bar to open the JavaTest Preferences dialog box.



Click Appearance in the left panel and use the dialog box to set the following appearance preferences:

- Changing Window Styles [p 77]
- Setting Tool Tip Options [p 78]
- Changing Shutdown Options [p 78]

## Changing Window Styles

The JavaTest graphical user interface displays its windows and menus in one of three user specified styles:

Window Style	Description
Tabbed [p 77]	A single top-level window that displays individual tool windows as tabbed panes
MDI [p 77]	A single top-level desktop window that contains individual tool windows
SDI [p 78]	A JavaTest console window and individual tool windows displayed as top-level windows

### ***Tabbed***

When you choose the Tabbed window style, the JavaTest harness displays the opened tool windows as a set of tabbed panes within a single frame or desktop window.

Tabs at the bottom of each pane allow you to choose the pane that is active and displayed on the top of the stack.

The desktop window contains the complete set of menus including any tool menus used to perform tasks from the active pane. See JavaTest Menus [p 72] for a description of the two types of menus that JavaTest provides.

Advantages of using the Tabbed window style:

- Multiple open windows are managed from a single location
- The small footprint of the desktop window minimizes obstruction of windows from other applications
- If multiple windows of multiple applications are open, tabbed windows provide visual organization

Disadvantages of using the tabbed style:

- Only one window can be viewed at a time
- Performing tasks on multiple open test suites can be confusing

### ***MDI***

When you choose the Multiple Document Interface (MDI) window style, all of the tool windows that the JavaTest harness opens to perform a task are contained within a single desktop window.

The desktop window contains the standard menus and each tool window contains only the tool menus used to perform tasks appropriate for that window. See JavaTest Menus [p 72] for a description of the two types of menus that JavaTest provides.

Advantages of using the MDI window style:

- Open windows are contained in a top level window allowing simple window management
- Multiple windows can be viewed at the same time
- Multiple test suites can be open at the same time and easily monitored

Disadvantages of using the MDI style:

- The single top level desktop window has a large footprint that may obstruct open windows of other applications and slightly decrease usable space
- Open windows cannot be positioned outside the boundary of the desktop window

## ***SDI***

When you choose the Single Document Interface (SDI) window style, the JavaTest harness opens a console window and individual tool windows as separate top-level windows.

Each window contains the complete set of menus including any tool menus used to perform tasks from that window. See JavaTest Menus [p 72] for a description of the two types of menus that JavaTest provides.

Advantages of using the SDI window style:

- Individual windows can be positioned anywhere on the screen
- Obstruction of windows from other applications is minimized
- Multiple test suites can be opened and easily monitored at the same time

Disadvantages of using the SDI style:

- Windows must be managed individually
- When multiple windows of multiple applications are open, the display can be visually confusing

## **Setting Tool Tip Options**

You can set the tool tip options from the Appearance category of the JavaTest Preferences dialog box.

The Tool Tips area contains combo boxes and a check-box that you can use to specify how tool tips function in the GUI.

Option	Description
Enabled	Use the check-box to enable or disable tool tips for the JavaTest GUI.
Delay	Use the combo box to select the delay interval before displaying tool tips.
Persistence	Use the combo box to select the duration that the JavaTest GUI displays a tool tip.

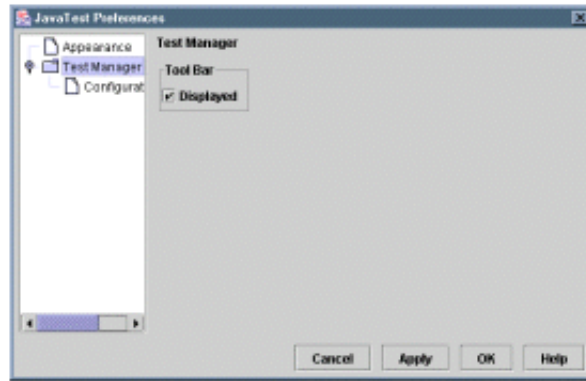
## **Changing Shutdown Options**

Click Save Desktop State on Exit to save and restore your current desktop in your next JavaTest session. If you chose not to save the the current desktop on exit, the JavaTest harness restores the last saved desktop for you to use.

This option is used each time you start the JavaTest harness.

### **11.4.2. Changing Test Manager Preferences**

The Test Manager window default setting is to display a tool bar containing task accelerator buttons.



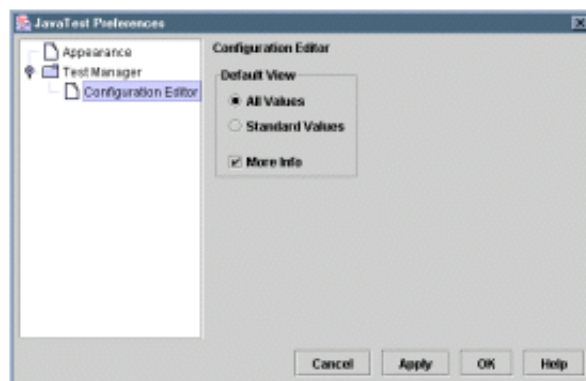
To turn off the tool bar:

1. Choose File > Preferences from the menu bar to open the JavaTest Preferences dialog box.
2. Click on the Test Manager folder in the preferences tree on the left.
3. Click the Tool Bar Displayed to set the Test Manager Tool Bar preferred style.

This preference is used each time you start the JavaTest harness.

### 11.4.3. Changing Configuration Editor Preferences

You can use the JavaTest Preferences dialog box to set the default view used when opening the Configuration Editor and to show or hide the More Info pane.



#### Set the Configuration Editor Default View

The JavaTest graphical user interface opens the Configuration Editor in one of two user specified views:

- All Values
- Standard Values

To change the view used when opening the Configuration Editor:

1. Choose File > Preferences from the menu bar to open the JavaTest Preferences dialog box.
2. Open the Test Manager folder in the tree on the left and then click the Configuration Editor icon.
3. Click the Configuration Editor Default View to set the Configuration Editor's opening view.

This preference is used each time you start the JavaTest harness and open the Configuration Editor.

### **Display/Hide the Configuration Editor More Info Pane**

The More Info pane contains detailed information about the questions and required settings presented in the Configuration Editor. When More Info is checked the Configuration Editor displays the More Info pane.

To hide the More Info pane, click More Info so that it is no longer checked.

## **11.5. Managing JavaTest Windows**

In MDI and SDI window styles, use the Windows menu to manage the layout of the open windows on the desktop:

<b>Menu Item</b>	<b>Description</b>
Tile	Arranges the open windows edge-to-edge in a tiled pattern.
Cascade	Arranges the open windows so that their top left corners form a cascading pattern from the top left corner to the bottom right corner of the desktop.



## 12. Using a JavaTest Agent

An agent is a separate program that works in conjunction with the JavaTest harness to run tests on a system other than the one that is running the JavaTest harness.

Depending on your test suite, agents are typically used to run tests on small devices that do not support online help.



You can either use custom agents or the agent provided with the JavaTest harness. The topics in this chapter describe how to configure and run the agent provided with the JavaTest harness. If you are using a custom agent, refer to your test suite documentation for a description of how to configure and run it.

To run tests using an agent on a test system, perform the following:

1. Choose the type of agent required to connect to the JavaTest harness. Refer to Choosing the Type of Agent [p 81] for a detailed description of active, passive, and serial modes.
2. Start the agent. Refer to Starting an Agent [p 83] for a detailed description how agents can be run either as applications or as applets.
  - If you choose to run the agent as an application, perform the procedure in Starting an Agent Application [p 84] .
  - If you choose to run the agent as an applet, perform the procedure in Starting an Agent Applet [p 86] .
3. Run tests and monitor agent activity. Refer to Monitoring Agents [p 94] for a detailed description how agents can be monitored when running tests.

This chapter contains the following sections:

- Choosing the Type of Agent [p 81]
- Starting an Agent [p 83]
- Monitoring Agents [p 94]
- Troubleshooting Agents [p 99]
- Installing Agent Classes on a Test System [p 100]
- Creating a Map File [p 103]

### 12.1. Choosing the Type of Agent

The JavaTest agent is a lightweight program compatible with JDK 1.1 that uses a bidirectional serial connection supporting both TCP/IP and RS-232 protocols to communicate between the test system and the JavaTest harness.

You can use the agent provided by the JavaTest harness if your test system meets the following minimum requirements:

- The device supports a communication layer that can last the duration of a test (couple of minutes)
- The device must be able to have the agent classes loaded on it

The type of agent that you use depends on the communication protocol used between your test system and the JavaTest harness and on the type of initial connection made between the agent and the JavaTest harness:

Mode	Description
Active	<p>Use active mode (active agent) when you want the agent to initiate the connection to the JavaTest harness via TCP/IP.</p> <p>Agents using active communication allow you to:</p> <ul style="list-style-type: none"> <li>• Run tests in parallel using many agents at once</li> <li>• Specify the test machines at the time you run the tests</li> </ul> <p>Active agents are used for network connections and are recommended. If the security restrictions of your test system prevent incoming connections then you must use an active agent.</p> <p>The JavaTest harness should be running and agent pool listening should be enabled before starting an active agent. Use Agent Monitor window [p 94] in the JavaTest harness GUI to enable listening.</p> <p>If listening is not enabled when the agent starts, it returns an error message and waits until its timeout period ends before re-contacting the JavaTest harness.</p>
Passive	<p>Use passive mode (passive agent) when you want the agent to wait for the JavaTest harness to initiate the connection via TCP/IP.</p> <p>Because the JavaTest harness only initiates a connection to a passive agent when it runs tests, passive communication:</p> <ul style="list-style-type: none"> <li>• Requires that you specify the test machine as part of the test configuration - not at the time you run the tests</li> <li>• Does not allow you to run tests in parallel</li> </ul> <p>Passive agents are used for network connections and must be started before the harness attempts to run tests. If the JavaTest harness issues a request before the passive agent is started, the harness waits for an available agent until its timeout period ends. If the timeout period ends before an agent is available, the JavaTest harness reports an error for the test.</p>
Serial	<p>Use serial mode (serial agent) when you want the agent to use an RS-232 serial connection. Serial agents wait for the JavaTest harness to initiate the connection. Infrared, parallel, USB, and firewire connections can also be added through the JavaTest API by modeling the existing serial system.</p> <p>Because the JavaTest harness only initiates a connection to serial agent when it runs tests, serial communication:</p> <ul style="list-style-type: none"> <li>• Requires that you specify the test machine as part of the test configuration - not at the time you run the tests</li> <li>• Does not allow you to run tests in parallel</li> </ul>
Other	<p>If your system does not meet the minimum requirements or if you have unique performance requirements, you can use the JavaTest API to create a custom agent. Refer to your test suite documentation for a description of how to configure and run it.</p>

#### Next:

Starting the Agent [p 83] - start an agent on your test system.

## 12.2. Starting an Agent

You can start an agent either as an application or as an applet. While the application provides you with the option of using either a GUI or a command line to configure and run the agent, the applet requires that you use a GUI:

	Interface	Application	Applet
GUI		Supported	Supported
Command Line		Supported	Not Supported

### 12.2.1. Agent Application

You can either use the application GUI or command line to configure and start an agent if the test system provides AWT support.

If a test platform is unable to or does not provide AWT support, you must use the command line to configure and start the agent. When using the command line to directly configure and run an agent, you:

- Must specify all agent options in the command line
- Cannot monitor agent performance during a test run
- Cannot modify agent properties without killing the agent and starting a new agent from the command line

If you use the GUI to run the agent, you can:

- Include options in the command line or start the GUI without specifying agent options
- Configure or reconfigure the agent after the GUI starts
- Monitor agent performance during a test run

The GUI used by the application is the same as that used by the applet. Refer to Using the GUI [p 84] below for a description of the tabbed panes.

### 12.2.2. Agent Applet

You can use either an applet or an application to run the agent on any test system that supports a web browser. However, you must use the applet when testing JVMs that run in web browsers.

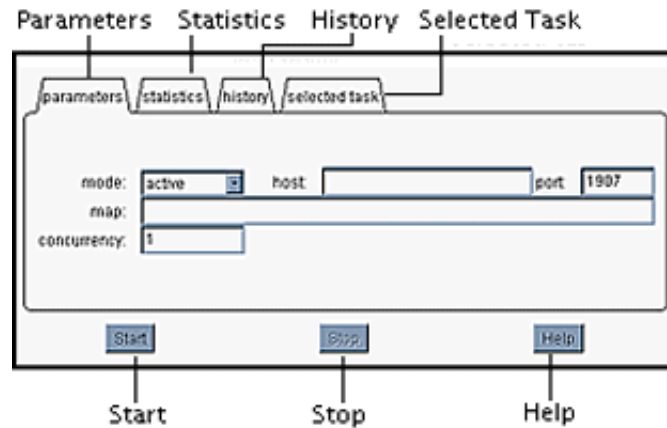
The GUI used by the applet is the same as that used by the application. Refer to Using the GUI [p 84] below for a description of the tabbed panes.

When using the applet, you can:

- Include parameters in the applet tag or start the GUI without specifying any parameters
- Configure or reconfigure the agent after the GUI starts
- Monitor agent performance during a test run.

### 12.2.3. Using The GUI

The GUI contains four tabbed panes and three buttons used to configure, control, and monitor the agent.



- The parameters tabbed pane allows you to configure, start, and stop the agent.
- The statistics tabbed pane displays detailed information about the tests that the agent is running.
- The history and selected task tabbed panes allow you to monitor tasks performed by the agent.
- The Start and Stop buttons control the agent.

**Next:**

Starting an Agent Application [p 84] - start an agent application on your test system.

Starting an Agent Applet [p 86] - start an agent applet on your test system.

### 12.2.4. Starting an Agent Application

Before you can start an agent application, the required classes must be installed on your test system. Refer to Installing Agent Classes on Test Systems [p 100] for the location and list of classes required to start the agent directly from the command line or using the application GUI.

1. Start the JavaTest harness and perform the following:
  - a) Open the Configuration Editor and configure the JavaTest harness to use an agent. In most cases, the Configuration Editor provides detailed instructions about configuring the JavaTest harness to run tests using an agent.
  - b) If you are starting an active agent, open the Agent Monitor window and enable agent pool listening. Refer to Agent Monitor Window [p 94] .

If the agent pool is not listening when an active agent starts, the agent cannot contact the harness. The agent returns an error message and then waits until its timeout period ends before recontacting the JavaTest harness.

2. Use the following application command template to enter the appropriate agent command at the command prompt:

```
java -cp class path [application class] [options]
```

- The `-cp` option sets the class paths required to run the agent. Use the ";" or ":" separator appropriate for your test system when more than one class path is included in the command string. Refer to Class Paths [p 85] below for detailed descriptions of the classes that your agent requires.

- The *[application class]* sets the class used to run the agent application. Refer to Application Classes [p 85] below for a list and description of the classes used to start an agent application.
- The *[options]* can be included in the command line to specify the agent parameters. Refer to Agent Options [p 86] below for a list and description of the parameters that you can use to configure and start an agent.

Example:

```
java -cp javatest.jar com.sun.javatest.agent.AgentFrame
```

3. If you are using the application GUI to run the agent, use the Parameters tabbed pane to verify the agent settings and start the agent:

- Specifying Active Agent Options [p 88] - the parameter settings required to run an active agent.
- Specifying Passive Agent Options [p 89] - the parameter settings required to run a passive agent.
- Specifying Serial Agent Options [p 91] - the parameter settings required to run a serial agent.

## Class Paths

The following class paths are required in the command line:

Classes	Description
Agent Classes	<p>The location of the agent classes installed on your test system.</p> <p>The agent classes are either located in the <code>javatest.jar</code> file or in the directory containing the minimum set of classes required to run the agent from the GUI.</p> <p>Some test suites include additional <code>.jar</code> files containing classes needed for an agent to run tests. These <code>.jar</code> files must also be included in the command string. Refer to Installing Agent Classes on a Test System [p 100] for a description of how agent classes can be installed.</p>
Test Classes	<p>Test classes are located in the classes directory of the test suite.</p>

The most common error in setting up a test platform to use an agent is entering the wrong class paths in the command string. Configuring your test platform to use the simplest class paths increases the reliability of the test run.

## Application Classes

An application class is required in the command line to run the agent. Two application classes are available:

Mode	Application Class
no GUI	<p><code>com.sun.javatest.agent.AgentMain [options]</code></p> <p>Used when the GUI is not wanted or not available. In this mode, all options must be fully specified on the command line. The agent automatically starts when the Return key is pressed. Refer to Agent Options [p 86] below for the <i>[options]</i> that are included on the command line.</p>
with GUI	<p><code>com.sun.javatest.agent.AgentFrame [options]</code></p> <p>Used to start the GUI. In this mode, options may either be given on the command line or in the GUI. The GUI is used to start and stop the agent. Refer to Agent Options [p 86] below for the <i>[options]</i> that are included on the command line.</p>

## Agent Options

There are two types of options used in the command line:

Type of Option	Description
Agent Parameters	<p>Set the parameters for the type of agent that you are using:</p> <ul style="list-style-type: none"><li>● Specifying Active Agent Options [p 88] - the parameter settings required to run an active agent.</li><li>● Specifying Passive Agent Options [p 89] - the parameter settings required to run a passive agent.</li><li>● Specifying Serial Agent Options [p 91] - the parameter settings required to run a serial agent.</li></ul> <p>If you are using the command-line application class (<code>com.sun.javatest.agent.AgentMain</code>) to directly configure and run the agent, you must include all options in the command line that are used to run the agent.</p> <p>If you are using the GUI application class (<code>com.sun.javatest.agent.AgentFrame</code>) you can either set the agent options in the command line or in the GUI before running the agent.</p>
Additional Parameters	<p>Display help, run the agent, or configure other agent properties.</p> <p>Refer to Specifying Additional Options [p 92] for a description of the additional parameters that can be set.</p>

### 12.2.5. Starting an Agent Applet

Before you can start an agent applet, the required classes must be installed on your test system. Refer to Installing Agent Classes on Test Systems [p 100] for the location and list of classes required to start the agent applet.

1. If an HTML page containing the required applet is not available, create it in your test suite root directory. Refer to Agent Applet Tag [p 86] below for a detailed description of an applet tag.
2. Use a web browser to open an HTML page containing the agent applet tag. The applet tag must be compatible with your browser's VM.
3. Use the Parameters tabbed pane to configure and run the agent:
  - Specifying Active Agent Options [p 88] - the parameter settings required to run an active agent.
  - Specifying Passive Agent Options [p 89] - the parameter settings required to run a passive agent.
  - Specifying Serial Agent Options [p 91] - the parameter settings required to run a serial agent.

### Agent Applet Tag

Because some browsers use built-in VMs to run applets, you must use a compatible applet or object tag. Refer to your VM documentation for a description of the tags required to run applets on your browser.

The following example is compatible with the Netscape browser VM. It calls the agent applet and sets the parameters of the applet GUI.

Agent parameters and run options can also be set in the applet tag. Refer to Setting Parameters in the Applet Tag [p 88] below.

### Example agent applet tag:

```
<APPLET/  
code=applet-class-path/  
archive=JavaTest-classes/  
width=display-width/  
height=display-height/  
>  
Applets have not been enabled  
on your browser. You must enable  
applets on you browser to display  
the applet GUI used to run the agent.  
</APPLET>
```

The following table describes the tags used in the applet:

Tag	Description
code	<p>The agent applet class installed on your test system.</p> <p>Example:</p> <pre>code=com.sun.javatest.agent.AgentApplet</pre>
archive	<p>The URL of the classes required to run the agent applet on your test system. The classes are either located in the <code>javatest.jar</code> file or in a directory containing the minimum set of classes required to run the agent applet.</p> <p>In the following example, the classes are contained in the <code>javatest.jar</code> file located in the same directory as the HTML page. Refer to <a href="#">Installing Agent Classes on a Test System [p 100]</a> for a description of how the agent applet classes can be installed.</p> <p>Example:</p> <pre>archive=javatest.jar</pre>
width	<p>Sets the width of the GUI. An initial value of 600 is suggested; however, adjust the value based on your screen size and resolution.</p> <p>Example:</p> <pre>width=600</pre>
height	<p>Sets the height of the applet. An initial value of 600 is suggested; however, you may need to adjust the value based on your screen size and resolution.</p> <p>Example:</p> <pre>height=600</pre>

## Setting Parameters in the Applet Tag

Parameters can also be set in the applet tag. Parameters in the applet tag are included as `<param name/value>` pair tags:

Example agent applet tag:

```
<APPLET
  code=applet-class-path
  archive=JavaTest-classes
  width=display-width
  height=display-height
>
... <param name=parameter-name value=parameter-value> Applets have not been enabled on your browser.
    You must enable applets on you browser to display
    the applet GUI used to run the agent.
</APPLET>
```

Two types of parameters can be included in the applet tag:

- Agent Parameters - used to specify the agent type. Can be set either in the applet tag or in the GUI. Anytime the agent is not running, you can also use the Parameters tabbed pane to change the agent parameters:

Specifying Active Agent Options [p 88] - parameter settings required to run an active agent.

Specifying Passive Agent Options [p 89] - parameter settings required to run a passive agent.

Specifying Serial Agent Options [p 91] - parameter settings required to run a serial agent.

- Additional Parameters - used to specify how an agent is run, Specifying Additional Options [p 92] .

### 12.2.6. Specifying Active Agent Options

Active agents can be configured and run from the application command-line, the application or applet GUI, or the applet tag. Refer to Starting the Agent [p 83] for a description of the different features and functions that each provides.

Depending on how you choose to start the agent, you must set the following minimum set of parameters either in the command line, the GUI Parameter pane, or the applet tag:

- Mode [p 88]
- Host [p 89]
- Port [p 89]

## Mode

Specifies the type of agent. The type of agent that you use determines how the agent communicates with the JavaTest harness and the protocol that is used. An active agent initiates the connection to the JavaTest harness via TCP/IP.

To specify an active agent, use the appropriate setting or option from the following table:


Interface	Option or Setting
default	active
command line	-active
applet tag	<code>&lt;param name=mode value=active&gt;</code>
GUI Parameter Pane	mode: <input type="text" value="active"/> host: <input type="text"/> port: <input type="text" value="1907"/>



## Host

Identifies the system running the JavaTest harness. Because an active agent initiates the connection to the JavaTest harness, the location of the system running the JavaTest harness must be set before it can run.

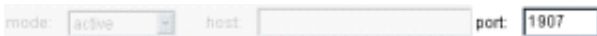
To specify the system running the JavaTest harness, use the appropriate setting or option from the following table:

Interface	Option or Setting
default	none
command line	-activeHost <i>host-name</i>
applet tag	<param name=activeHost value= <i>host-name</i> >
GUI Parameter Pane	

## Port

Specifies the port used by the active agent to communicate with the JavaTest harness. The agent and JavaTest harness must use the same port. If the ports are not the same, the agent cannot communicate with the JavaTest harness. The default value for active agents is 1907.

To specify a port other than 1907, use the appropriate setting or option from the following table:

Interface	Option or Setting
default	1907
command line	-activePort <i>port-number</i>
applet tag	<param name=activePort value= <i>port-number</i> >
GUI Parameter Pane	

### Next:

Specifying Additional Options [p 92] - additional parameters to display help or configure other agent properties.

## 12.2.7. Specifying Passive Agent Options

Passive agents can be configured and run from the application command-line, the application or applet GUI, or the applet tag. Refer to Starting the Agent [p 83] for a description of the different features and functions that each provides.



Depending on how you choose to start the agent, you must set the following minimum set of parameters either in the command line, the GUI Parameter pane, or the applet tag:

- Mode [p 89]
- Port [p 90]

## Mode

Specifies the type of agent. The type of agent that you use determines how the agent communicates with the JavaTest harness and the protocol that is used. A passive agent waits for the JavaTest harness to initiate the connection via TCP/IP.



To specify a passive agent, use the appropriate setting or option from the following table:

Interface	Option or Setting
default	active
command line	-passive
applet tag	<param name=mode value= <i>passive</i> >
GUI Parameter Pane	mode:  port: 

## Port

Specifies the port that the passive agent uses to listen for the JavaTest harness. The JavaTest harness and agent must use the same port. If the ports are not the same, the JavaTest harness cannot communicate with the agent. The default value for passive agents is 1908.

To specify a port other than 1908, use the appropriate setting or option from the following table:

Interface	Option or Setting
default	1908
command line	-passivePort <i>port-number</i>
applet tag	<param name=activePort value= <i>port-number</i> >
GUI Parameter Pane	mode:  port: 

## Next:

Specifying Additional Options [p 92] - additional parameters to display help or configure other agent properties.

## 12.2.8. Specifying Serial Agent Options

Serial agents can be configured and run from a command-line, GUI, or applet tag. Refer to Starting the Agent [p 83] for a description of the different features and functions that each provides.


Depending on how you choose to start the agent, you must set the following minimum set of parameters from the command line, GUI Parameter pane, or applet tag:

- Mode [p 91]
- Port [p 91]

### Mode

Specifies the type of agent. The type of agent that you use determines how the agent communicates with the JavaTest harness and the protocol that is used. A serial agent waits for the JavaTest harness to initiate the connection via an RS-232 serial connection or a connection added through the JavaTest API that models the serial system.


To specify a serial agent, use the appropriate setting or option from the following table:

Interface	Option or Setting
default	active
command line	-serial
applet tag	<param name=mode value=serial>
GUI Parameter Pane	

### Port

Specifies the com port that the serial agent uses to listen for the JavaTest harness. The JavaTest harness and agent must use the same port. If the ports are not the same, the JavaTest harness cannot communicate with the agent.

To specify a port, use the appropriate setting or option from the following table:

Interface	Option or Setting
command line	-serialPort <i>port-number</i>
applet tag	<param name=serialPort value=port-number>
GUI Parameter Pane	

### Next:

Specifying Additional Options [p 92] - additional parameters to display help or configure other agent properties.

## 12.2.9. Specifying Additional Options

Two types of additional options can be used:

- Options Used to Display Help [p 92]
- Options Used to Run and Monitor the Agent [p 92]

### Options Used to Display Help

The help option only displays command-line help for the agent regardless of the application class used in the command line. To start an agent application or applet after displaying command-line help, perform Starting the Agent [p 83] .

The following options are only used on the command line to display help:

Option	Function
<code>-help</code> or <code>-usage</code>	Displays command-line help.
Example:	
<pre>java -cp javatest.jar com.sun.javatest.agent.AgentMain -help</pre>	

### Options Used to Run and Monitor the Agent


The following options can be set in the application command-line, the application or applet GUI, or the applet tag.

- Specify a Map File [p 92]
- Set Concurrency [p 92]
- Set Number of Tasks in the History Tabbed Pane [p 93]
- AutoStart the Agent [p 93]
- Set Tracing [p 93]

#### *Specify a Map File*

Specifies that the agent use a map file to translate host specific values. Refer to Creating a Map File [p 103] for additional information about map files.

To specify a map file, use the appropriate setting or option from the following table:

Interface	Option or Setting
default	none (empty)
command line	<code>-map <i>map-file</i></code>
applet tag	<code>&lt;param name=map value=<i>map-file-url</i>&gt;</code>
GUI Parameter Pane	

#### *Set Concurrency*

To run tests concurrently, set the maximum number of simultaneous requests handled by the agent. Each request requires a separate connection to the JavaTest harness and a separate thread inside the agent. The request may also require a separate process on the test system running the agent. The default setting is one.

To run concurrent tests, use the appropriate setting or option from the following table:

Interface	Option or Setting
default	one
command line	<code>-concurrency <i>number-of-tests</i></code>
applet tag	<code>&lt;param name=concurrency value=<i>number-of-tests</i>&gt;</code>
GUI Parameter Pane	concurrency: <input type="text" value="1"/>

### ***Set Number of Tasks in the History Tabbed Pane***

Specifies the maximum number of tasks displayed in the history tabbed pane. Refer to History Tabbed Pane [p 96] for a description of the history tabbed pane and how it is used to monitor an agent.


To set the tasks displayed in the history tabbed pane, use the appropriate setting or option from the following table:

Interface	Option or Setting
default	one
command line	<code>-history <i>number-of-items</i></code>
applet tag	<code>&lt;param name=history value=<i>number-of-items</i>&gt;</code>
GUI	Not Supported

### ***AutoStart the Agent***

This option is only used with the application GUI class or as a parameter in the applet tag. When used, the autostart option automatically starts the agent after all command line options are validated and the GUI is displayed. The agent must be completely configured in the command line or applet tag. When the `-start` option is not used, click the Start button in the agent GUI to start testing.

To autostart the agent when the GUI is displayed, use the appropriate setting or option from the following table:

Interface	Option or Setting
default	false
command line	<code>-start</code>
applet tag	<code>&lt;param name=autostart value=<i>true</i>&gt;</code>
GUI	

### ***Set Tracing***

Sends detailed information about agent activity to the system output stream.

To start tracing when the agent is run, use the appropriate setting or option from the following table:

Interface	Option or Setting
default	false
command line	-trace
applet tag	<param name=trace value=true>
GUI	Not Supported

**Next:**

Monitoring JavaTest Agents [p 94] - monitor an agent while it runs tests.

## 12.3. Monitoring JavaTest Agents

There are two ways that you can monitor JavaTest agents. You can:

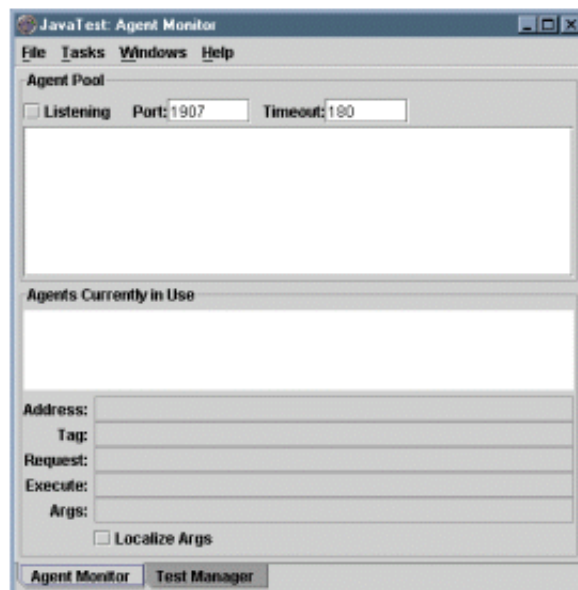
1. View all agents in a test system that are running tests. Refer to Agent Monitor Window [p 94] for detailed information about opening and using the Agent Monitor window to view all agents in a test system that are running tests.
2. Monitor specific information about an agent and the tests that it runs. To display information about the agent, you must use the tabbed panes in the application or applet GUI:
  - Statistics Pane [p 95] - displays the current status of the tests that the agent is running.
  - History Pane [p 96] - displays a list of tasks performed by the agent.
  - Selected Task Pane [p 97] - displays details about a specific task or test chosen in the history tabbed pane.

### 12.3.1. Agent Monitor Window

Open the Agent Monitor window by using the Test Manager menu bar to choose:

Tasks > Monitor Agent Activity

The Agent Monitor window contains two sections, the Agent Pool and the Agents Currently In Use.



## Agent Pool

The agent pool lists the active agents that are available to run tests. When active agents connect to the JavaTest harness they are added to the agent pool. When the JavaTest harness requires an active agent to run a test, it moves the agent from the Agent Pool to the Agents Currently In Use section until the test is completed.

The following table lists and describes the contents of the Agent Pool GUI:

Field	Description
Listening	Click the check box to enable listening for active agents. If listening is not enabled when an agent starts, the agent issues a message that it cannot connect to the JavaTest harness and then waits for its timeout period to end before attempting to re-contact the harness.
Port:	Port 1907 is the default port used by active agents. If your agent uses a different port, you must either change the value used by the agent or change this value to match the agent.
Timeout:	When the agent pool is empty, the timeout value sets the number of seconds that the JavaTest harness waits between tests for an available agent before reporting the test result as an error. If you run tests with one agent, there is usually a latent period between the time when the agent completes the test and when it returns to the agent pool. The timeout value must be greater than the agent's latent period. The default value of 180 seconds is usually sufficient.

## Agents Currently In Use

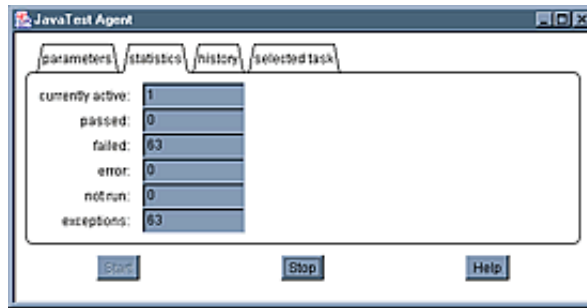
Lists all agents currently used by the JavaTest harness to run tests. When agents are not running tests they are removed from the list (active agents re-register with the agent pool). Click on an agent in the list to display detailed information about the agent and the test it is running. The detailed information is displayed in the text fields at the bottom and can be used to troubleshoot problems using an agent to run tests.

The following table lists and describes the contents of the Agents Currently In Use GUI:

Field	Description
Address	The network address of the agent
Tag	The test executed by the agent
Request	The function executed by the agent
Execute	The class executed by the agent
Args	The arguments passed to the class executed by the agent
Localize Args	Checked if the agent uses a map file

### 12.3.2. Statistics Pane

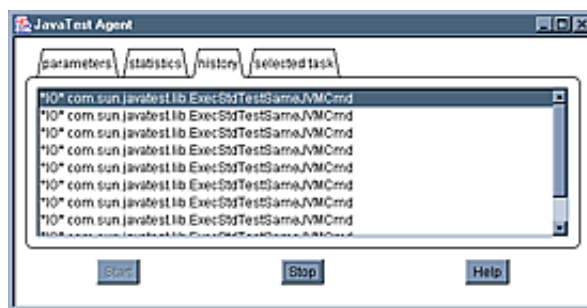
The statistics tabbed pane in the agent GUI displays the cumulative statistics for the tests in the test suite:



Field	Description
currently active	The number of tests being run by the agent
passed	The number of tests that were run by the agent and had passing results
failed	The number of tests that were run by the agent and had failing results
error	The number of tests that were run by the agent and had errors
not run	The number of tests that were not run by the agent and were not filtered out by the JavaTest harness
exceptions	The number of tests that were filtered out of the test run by the JavaTest harness

### 12.3.3. History Pane

The history tabbed pane in the agent GUI displays a list of the tasks performed by the agent.



To view the details about a task, click on it in the list. The GUI displays the task details in the selected task pane.

Refer to Selected Task Pane [p 97] for a description of the task information that is displayed.

The history pane displays a list of tasks that:

- Are currently being executed by the agent
- Have recently been completed

Each entry in the list contains a code indicating the current state of the task and additional information about the state of the task. A task in the history list will be in one of the following states:



Current State	Description
CONN host:port	This state shows that the JavaTest Agent has an open connection to JavaTest, at the specified network address, and that the JavaTest Agent is waiting for a request to be sent over the connection. If the JavaTest Agent is running in active mode, it will wait until JavaTest sends the request. If the agent is running in passive mode, this state will usually appear only temporarily because JavaTest will normally initiate a connection and then immediately send the request. The host will normally be identified by its host name; if JavaTest cannot determine the host name, the IP address of the host will be shown instead.
EXEC tag	This state shows that the JavaTest Agent is executing a task on behalf of JavaTest. The tag is an identification of the task supplied by JavaTest as part of the request.
*IO* tag	This state shows that the JavaTest Agent was executing a task on behalf of JavaTest but that some exception occurred while trying to send the results back to JavaTest.

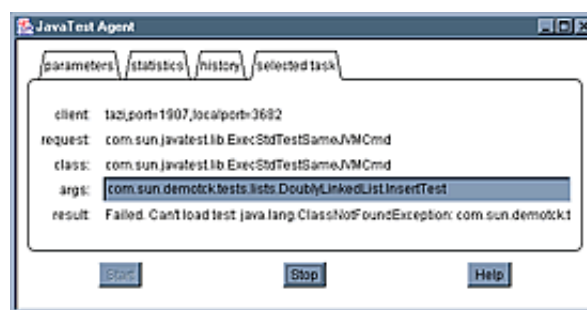
The following states show that the JavaTest Agent completed a request for JavaTest; the states correspond to the various possible outcomes of the task. These outcomes are exactly the same as the outcomes JavaTest gets when it runs tests directly, without the assistance of the JavaTest Agent.

State	Description
PASS:	The task completed successfully.
FAIL:	The task indicated that it failed.
ERR:	The task encountered some error before it could properly be executed.
!RUN:	This should never occur; if it were to occur, it would mean that a task has inappropriately indicated that it has not been run.

### 12.3.4. Selected Task Pane

The selected task tabbed pane in the agent GUI displays detailed information about a task selected from the task list in the history pane:

Refer to History Pane [p 96] for a description of the task list.



<b>Field</b>	<b>Description</b>
client	Displays the network address (host and port) of the source of the task request. The host will normally be identified by its host name, but if JavaTest cannot determine the host name, the IP address of the host will be shown instead.
request	Displays the tag that was supplied with the request in order to identify itself.
class	Displays the name of the class that was specified in the request. This is the class that will be loaded and run in fulfillment of the request.
args	Displays the arguments that were specified in the request. These arguments will be passed to the class that will be executed.
result	If and when the task is completed, this field will contain the outcome of the task, as indicated by a JavaTest Status object.

## 12.4. Troubleshooting JavaTest Agents

To troubleshoot JavaTest agents, verify that:

- The agent has been correctly configured and started
- The test harness is configured correctly for the agent and test system that the agent runs on
- Map files, if required, correctly translate host specific value into values that the agent can use

Because active agents initiate the connection with the JavaTest harness while passive agents wait for a request from the JavaTest harness, troubleshooting each type of agent requires a different approach:

- Active Agents [p 99]
- Passive Agents [p 100]

### 12.4.1. Troubleshooting Active Agents

Active agents initiate the connection with the JavaTest harness. You must set up the JavaTest agent pool so that the connection is made before running tests.

Errors in configuring, synchronizing, or implementing the connection between the agent and the JavaTest harness are the most probable causes of failure.

Use the Agent Monitor window, the Test Manager window, the agent GUI, and the following guide when troubleshooting problems running active agents:

1. In the Agent Monitor window, verify that the agent is listed in the agent pool. If the agent is not listed in the agent pool:
  - a) Verify that the Listening check box is selected.
  - b) Verify that the agent is configured to contact the correct active host and that the port value of the harness matches the port value used by the agent.
  - c) Check the physical connection between the JavaTest platform and the test platform.
2. Verify that the agent moves to Agents Currently in Use when tests are running. If the agent does not move to Agents Currently in Use when tests are running:
  - a) Use the Configuration Editor to verify that the harness is configured to use agents when running tests.
  - b) If you are running the tests using multiple JVMs, use the Configuration Editor to verify that the path you provided in the Java Launcher question is the path of the Java launcher for the *agent* running tests.
3. If tests are failing or have errors, check the error messages displayed in the Test Manager window. If the error indicates that tests are failing because of missing classes:
  - a) Verify that the class paths used to start the agent are correct.
  - b) Use the Configuration Editor to verify that the harness is correctly configured to use the agent on the test system.
  - c) Run the agent using the `-trace` option to verify that the paths in the stream messages for the test are correct. If the paths are not correct for the test system, create a map file [p 103] for the agent to use in translating host specific values into values that the agent can use.
  - d) If a map file was used to run the test, use the Test Run Messages pane to verify that the `-mapArgs` command is present in the stream messages. If the `-mapArgs` command is not present verify that both the agent *and* the harness are configured to use the map file. Use the Configuration Editor to verify that the harness has been configured to use the agent map file.

## 12.4.2. Troubleshooting Passive Agents

Because passive agents must wait for a request from the JavaTest harness before running tests, the port that the passive agent uses must be the same as that used by the JavaTest harness to send requests.

Errors in configuring, synchronizing, or implementing the connection between the agent and the JavaTest harness are the most probable causes of failure.

Use the Agent Monitor window, the Test Manager window, the agent GUI, and the following guide when troubleshooting problems running passive agents:

1. Verify that the agent was started before the JavaTest harness started the test run. If not, repeat the test run.
2. Verify that the port value used when starting the agent matches the port value used by the JavaTest harness to send requests.
3. Check the physical connection between the JavaTest platform and the test platform.
4. Use the Configuration Editor to verify that the harness is configured to use agents when running tests.
5. If you are running the tests using multiple JVMs, use the Configuration Editor to verify that the path you provided in the Java Launcher question is the path of the Java launcher for the *agent* running tests.
6. If tests are failing or have errors, check the error messages displayed in the Test Manager window. If the error indicates that tests are failing because of missing classes:
  - a) Verify that the class paths used to start the agent are correct.
  - b) Use the Configuration Editor to verify that the harness is correctly configured to use the agent on the test system.
  - c) Run the agent using the `-trace` option to verify that the paths in the stream messages for the test are correct. If the paths are not correct for the test system, create a map file [p 103] for the agent to use in translating host specific values into values that the agent can use.
  - d) If a map file was used to run the test, use the Test Run Messages pane to verify that the `-mapArgs` command is present in the stream messages. If the `-mapArgs` command is not present verify that both the agent *and* the harness are configured to use the map file. Use the Configuration Editor to verify that the harness has been configured to use the agent map file.

## 12.5. Installing Agent Classes on a Test System

Agent classes must be located on your test system before you can run the JavaTest agent.

There are two methods you can use to load agent classes on your test system. The method that you use is determined by the available space on your test system:

- If space is available on your test system, you can copy the `javatest.jar` file directly to it. The `javatest.jar` file contains all of the required JavaTest agent classes.
- If space is limited on your test system, you can extract and then install the minimum set of classes for the agent user interface that you are using from the `javatest.jar` file:

Agent User Interface	Required Classes
Agent GUI	Minimum set of classes required to run agents using a GUI [p 100] .
Command Line	Minimum set of classes required to run agents from the command line [p 102] .
Applets	Minimum set of classes required to run agents as applets [p 102] .

## 12.5.1. Classes Required to Run Agents Using a GUI

The following list contains the minimum set of classes required to run an agent by using a GUI on your test system. You may require additional classes for some tests run in the same VM as the agent.

```
com.sun.javatest.Command
com.sun.javatest.JavaTestSecurityManager
com.sun.javatest.NewJavaTestSecurityManager
com.sun.javatest.ProductInfo
com.sun.javatest.Status
com.sun.javatest.Test
com.sun.javatest.agent.ActiveConnectionFactory
com.sun.javatest.agent.ActiveModeOptions
com.sun.javatest.agent.Agent
com.sun.javatest.agent.Agent$1
com.sun.javatest.agent.Agent$Notifier
com.sun.javatest.agent.Agent$Observer
com.sun.javatest.agent.Agent$Task
com.sun.javatest.agent.AgentClassLoader
com.sun.javatest.agent.AgentFrame
com.sun.javatest.agent.AgentFrame$1
com.sun.javatest.agent.AgentFrame$2
com.sun.javatest.agent.AgentFrame$Listener
com.sun.javatest.agent.AgentPanel
com.sun.javatest.agent.AgentPanel$1
com.sun.javatest.agent.AgentPanel$AgentObserver
com.sun.javatest.agent.AgentPanel$ButtonPanel
com.sun.javatest.agent.AgentPanel$ErrorPanel
com.sun.javatest.agent.AgentPanel$HelpPanel
com.sun.javatest.agent.AgentPanel$HistoryList
com.sun.javatest.agent.AgentPanel$MapReader
com.sun.javatest.agent.AgentPanel$ParamPanel
com.sun.javatest.agent.AgentPanel$StatsPanel
com.sun.javatest.agent.AgentPanel$TaskPanel
com.sun.javatest.agent.AgentPanel$TaskState
com.sun.javatest.agent.AgentWriter
com.sun.javatest.agent.BadValue
com.sun.javatest.agent.Connection
com.sun.javatest.agent.ConnectionFactory
com.sun.javatest.agent.ConnectionFactory$Fault
com.sun.javatest.agent.Deck
com.sun.javatest.agent.Deprecated
com.sun.javatest.agent.Folder
com.sun.javatest.agent.Folder$1
com.sun.javatest.agent.Folder$Entry
com.sun.javatest.agent.Folder$Layout
com.sun.javatest.agent.Icon
com.sun.javatest.agent.Map
com.sun.javatest.agent.ModeOptions
com.sun.javatest.agent.PassiveConnectionFactory
com.sun.javatest.agent.PassiveModeOptions
com.sun.javatest.agent.Proxy
com.sun.javatest.agent.SerialPortModeOptions
com.sun.javatest.agent.SocketConnection
com.sun.javatest.agent.SocketConnection$1
com.sun.javatest.util.DynamicArray
com.sun.javatest.util.ExitCount
com.sun.javatest.util.MainFrame
com.sun.javatest.util.StringArray
com.sun.javatest.util.Timer
```

```
com.sun.javatest.util.Timer$1
com.sun.javatest.util.Timer$Entry
com.sun.javatest.util.Timer$Timeable
com.sun.javatest.util.WriterStream
```

### 12.5.2. Classes Required to Run Agents from the Command Line

The following list contains the minimum set of classes required to run an agent from a command line on your test system. You may require additional classes for some tests run in the same VM as the agent.

```
com.sun.javatest.Command
com.sun.javatest.JavaTestSecurityManager
com.sun.javatest.NewJavaTestSecurityManager
com.sun.javatest.Status
com.sun.javatest.Test
com.sun.javatest.agent.ActiveConnectionFactory
com.sun.javatest.agent.Agent
com.sun.javatest.agent.Agent$1
com.sun.javatest.agent.Agent$Notifier
com.sun.javatest.agent.Agent$Observer
com.sun.javatest.agent.Agent$Task
com.sun.javatest.agent.AgentClassLoader
com.sun.javatest.agent.AgentMain
com.sun.javatest.agent.AgentMain$1
com.sun.javatest.agent.AgentMain$BadArgs
com.sun.javatest.agent.AgentMain$ErrorObserver
com.sun.javatest.agent.AgentMain$Fault
com.sun.javatest.agent.AgentWriter
com.sun.javatest.agent.Connection
com.sun.javatest.agent.ConnectionFactory
com.sun.javatest.agent.ConnectionFactory$Fault
com.sun.javatest.agent.Deprecated
com.sun.javatest.agent.Map
com.sun.javatest.agent.PassiveConnectionFactory
com.sun.javatest.agent.SocketConnection
com.sun.javatest.agent.SocketConnection$1
com.sun.javatest.util.DynamicArray
com.sun.javatest.util.StringArray
com.sun.javatest.util.Timer
com.sun.javatest.util.Timer$1
com.sun.javatest.util.Timer$Entry
com.sun.javatest.util.Timer$Timeable
com.sun.javatest.util.WriterStream
```

### 12.5.3. Classes Required to Run Agents as Applets

The following list contains the minimum set of classes required to run an agent as an applet on your test system. You may require additional classes for some tests run in the same VM as the agent.

```
com.sun.javatest.Command
com.sun.javatest.ProductInfo
com.sun.javatest.Status
com.sun.javatest.Test
com.sun.javatest.agent.ActiveConnectionFactory
com.sun.javatest.agent.ActiveModeOptions
com.sun.javatest.agent.Agent
com.sun.javatest.agent.Agent$1
com.sun.javatest.agent.Agent$Notifier
com.sun.javatest.agent.Agent$Observer
com.sun.javatest.agent.Agent$Task
```

```

com.sun.javatest.agent.AgentApplet
com.sun.javatest.agent.AgentApplet$1
com.sun.javatest.agent.AgentClassLoader
com.sun.javatest.agent.AgentPanel
com.sun.javatest.agent.AgentPanel$1
com.sun.javatest.agent.AgentPanel$AgentObserver
com.sun.javatest.agent.AgentPanel$ButtonPanel
com.sun.javatest.agent.AgentPanel$ErrorPanel
com.sun.javatest.agent.AgentPanel$HelpPanel
com.sun.javatest.agent.AgentPanel$HistoryList
com.sun.javatest.agent.AgentPanel$MapReader
com.sun.javatest.agent.AgentPanel$ParamPanel
com.sun.javatest.agent.AgentPanel$StatsPanel
com.sun.javatest.agent.AgentPanel$TaskPanel
com.sun.javatest.agent.AgentPanel$TaskState
com.sun.javatest.agent.AgentWriter
com.sun.javatest.agent.BadValue
com.sun.javatest.agent.Connection
com.sun.javatest.agent.ConnectionFactory
com.sun.javatest.agent.ConnectionFactory$Fault
com.sun.javatest.agent.Deck
com.sun.javatest.agent.Deprecated
com.sun.javatest.agent.Folder
com.sun.javatest.agent.Folder$1
com.sun.javatest.agent.Folder$Entry
com.sun.javatest.agent.Folder$Layout
com.sun.javatest.agent.Icon
com.sun.javatest.agent.Map
com.sun.javatest.agent.ModeOptions
com.sun.javatest.agent.PassiveConnectionFactory
com.sun.javatest.agent.PassiveModeOptions
com.sun.javatest.agent.Proxy
com.sun.javatest.agent.SerialPortModeOptions
com.sun.javatest.agent.SocketConnection
com.sun.javatest.agent.SocketConnection$1
com.sun.javatest.util.DynamicArray
com.sun.javatest.util.MainFrame
com.sun.javatest.util.StringArray
com.sun.javatest.util.Timer
com.sun.javatest.util.Timer$1
com.sun.javatest.util.Timer$Entry
com.sun.javatest.util.Timer$Timeable
com.sun.javatest.util.WriterStream

```

## 12.6. Creating a Map File

Some tests require contextual information, such as the host name on which they are executed, before they can run. Because network file systems may be mounted differently on different systems, the path names used by the JavaTest harness may not be the same for the agent. The agent uses a map file to translate these strings into values it can use to run tests.

1. Use a text editor to open a simple ASCII file and enter the following types of lines:

- Comment line - begins with the # symbol and provides information that is not processed by the agent. Comment lines are optional.

Example:

```
#Replace all /home/jjg with /jjg
```

- Translation line - contains the target and substitution strings. Enter the string that is to be replaced followed by one or more spaces and the replacement string. The agent replaces all occurrences of the first string with the second.

Example:

```
/home/jjg /jjg
```

Because the agent uses the map file to perform global string substitution on *all* matching values received from the JavaTest harness, you should be as specific as possible when specifying strings in a translation line.

Refer to Troubleshooting JavaTest Agents [p 99] for additional information about determining the substitution strings required in a map file.

2. Save the map file in the test suite root directory. You can use any name and extension. If you are unable to use the root directory, you can use any directory on the test system that you can write to. When starting an agent you must specify which map file, if any, to use.

Example of a map file:

```
#This is a sample map file
#Replace all /home/jjg with /jjg

/home/jjg /jjg

#Replace all /home/kasmith/javatest with /kas/javatest
/home/kasmith/javatest /kas/javatest
```



## 13. Using the JavaTest Command-Line

The JavaTest harness provides a command-line interface that you can use to:

- Perform batch mode tasks, such as configure and run tests, without using the GUI. Refer to [Using Batch Mode \[p 105\]](#) for detailed information.
- Start the JavaTest harness using specific settings. Refer to [Specifying Additional Options \[p 112\]](#) for detailed information.
- Display JavaTest information, such as online help, without starting the harness. Refer to [Displaying JavaTest Information \[p 115\]](#) for detailed information.

### 13.1. Using Batch Mode

You can use the `-batch` mode option and its commands to configure and run one or more tests or branches of tests, write test reports, and audit test results either from the command line or as a part of a product build process. The commands used with `-batch` are a formatted set of commands, executed in the sequence that they appear in the command string. See [Formatting Batch Commands \[p 105\]](#) for a description of the formats you can use.

You should use the commands in the command string much as you would if you were writing a script:

1. Initialize a configuration. See [Initializing a Configuration \[p 107\]](#) for detailed information.
2. Modify the current configuration (if required). See [Setting the Standard Values \[p 109\]](#) and [Setting Other Configuration Values \[p 110\]](#) for detailed information.
3. Specify the task performed. See [Running Tests \[p 111\]](#) , [Writing Reports \[p 111\]](#) , and [Auditing Tests \[p 111\]](#) for detailed information.

See [Specifying Additional Options \[p 112\]](#) for additional settings that can be used to start the JavaTest harness. If used, these options are included before the `-batch` option.

The topics in this section are:

- ▶ [Formatting Batch Commands \[p 105\]](#)
- ▶ [Initializing a Configuration \[p 107\]](#)
- ▶ [Setting the Standard Values \[p 109\]](#)
- ▶ [Setting Other Configuration Values \[p 110\]](#)
- ▶ [Running Tests \[p 111\]](#)
- ▶ [Writing Reports \[p 111\]](#)
- ▶ [Auditing Tests \[p 111\]](#)
- ▶ [Index of Available Batch Commands \[p 112\]](#)

### 13.1.1. Formatting Batch Commands

Commands are used with the `-batch` mode option in any one of three formats:

- Batch Options
- Single String Arguments
- Batch Command Files

Start the JavaTest harness from a writable directory where you intend to create files and store test results. Include the path of the directory `[jt_dir]` where the `javatest.jar` file is installed. The `javatest.jar` file is usually installed in the TCK lib directory when the JavaTest harness bundled with a TCK.

See Available Batch Commands [p 112] for a description of the commands that can be used in batch mode.

#### Batch Options

If you are setting a limited number of batch options you can use the batch options format. In the batch options format, the batch commands are preceded by `"-"`, act as options, and do not use command terminators. Enclose complex batch command arguments in quotes.

Example:

```
java -jar [jt_dir]/javatest.jar [harness-options] -batch -open default.jti -set host  
mymachine -runtests
```

#### Single String Arguments

If you are setting several batch options, you can use the single string arguments format. In the single string arguments format, one or more batch commands and their arguments can be enclosed in quotes as a single string argument to the `-batch` option. Multiple batch commands and arguments in the string are separated by semicolons.

Example:

```
java -jar [jt_dir]/javatest.jar [harness-options] -batch "open default.jti; set host  
mymachine; runtests"
```

#### Batch Command Files

If you are setting a series batch commands and options, you can use the batch command file format. Using the batch command file allows you to easily reuse the same configuration.

In the batch command file format, a file containing a series of batch commands and their arguments is included in the command line by preceding the file name with the `"@"` symbol. Refer to Creating a Batch Command File below for detailed information about creating a batch command file.

Example:

```
java -jar [jt_dir]/javatest.jar [harness-options] -batch -open default.jti @batchcmd.jtb
```

#### Creating a Batch Command File

You can put a lengthy series of batch commands and their arguments into an ASCII file and then include it in the command line. Using a batch command file allows you to repeatedly use a batch configuration without retyping the commands each time a test run is performed.

To help you identify the function of each batch file, it is recommended that you use a descriptive name and the extension `.jtb` when naming individual batch files.

Batch files can contain blank lines and comments as well as batch commands and their arguments:

File Contents	Description
Comments	Comments are started by the # symbol and stop at the end of the line.  Example: <code>#File contains batch commands</code>
Batch Commands	Are executed in the sequence that they appear in the batch file. Use commands listed in Available Batch Commands [p 112] . Commands used in the batch file must be separated by a semicolon (;) or a new line (#). The # symbol acts as a new line character and can terminate a command.  Examples: <code>open default.jti #opens file</code>  Example: <code>open default.jti ; -set host mymachine</code>
Command Arguments	Arguments that contain white space must be placed inside quotes. Use "\"" to escape special characters such as quotes (") and backslashes (\).

### ***Example Use of a Batch File***

An example of a typical application of a batch file:

Use the configuration editor to create a template .jti file and then use it in conjunction with -set commands in a batch file to override specific questions for unique test configurations.

By using a batch file in this manner, it is possible to develop multiple variations of a configuration without having to use the configuration editor each time a minor change is required.

## **13.1.2. Initializing the Configuration**

To use batch mode you must initialize the configuration by opening at least *one* of the following:

- An existing configuration (.jti) file
- A test suite and empty work directory
- An existing work directory

See Shortcuts for Initializing the Current Configuration [p 109] for additional information about specifying test suite and work directory in the current configuration.

The following lists the batch commands that you can use:

- `open name` [p 107]
- `testSuite testsuite` [p 108]
- `workDirectory work-directory` [p 108]

After initializing the configuration, you can then modify the current configuration for your specific requirements. See Setting Standard Values [p 109] and Setting Other Configuration Values [p 110] for the commands used to modify the current configuration.

### ***open *name****

Opens a test suite, work directory, parameter .jtp file, or a configuration .jti file. The .jtp files are supported for older test suites that do not use a configuration editor and .jti file.

Example:

```
..;open test_workdir.wd;..
```

### **testSuite *testsuite***

Specifies the test suite that is run.

Example:

```
..;testSuite sampleTestSuite;..
```

### **workDirectory *work-directory***

Each work directory is associated with a test suite and stores its test result files in a cache. You can use the `workDirectory` command to open an existing work directory, create a new work directory, or replace an existing work directory with a new work directory.

#### ***Open an Existing Work Directory***

To open an existing work directory for the test run, use the `workDirectory` command.

```
workDirectory work-directory
```

Example:

```
..;workDirectory sampletest;..
```

#### ***Create a New Work Directory***

To create a new work directory for the test run, use the `-create` command option. The new work directory must not previously exist.

```
workDirectory -create work-directory
```

Example:

```
..;workDirectory -create sampletest;..
```

#### ***Replace an Existing Work Directory with a New Work Directory***

When you replace an existing work directory with a new work directory, the JavaTest harness:

1. Deletes the existing work directory and its contents.
2. Creates the new work directory using the same name (if the old directory was successfully deleted).

To replace an existing work directory with a new work directory, use the `-overwrite` command option. The `-create` command option is optional when the `-overwrite` command is used.

```
workDirectory -create -overwrite work-directory
```

Example:

Replace the existing `sampletest` work directory with a new `sampletest` work directory.

```
..;workDirectory -overwrite sampletest;..
```

or

```
..;workDirectory -create -overwrite sampletest;..
```

## Shortcuts Used to Initialize the Current Configuration

The following apply to the test suite, work directory, and .jti file specified in a batch command:

- If you specify an existing work directory, you are not required to specify a test suite.
- If you specify an existing .jti file, you are not required to specify a test suite.
- If you specify an existing .jti file, you are not required to specify a work directory unless you want to use a work directory different from that specified in the .jti file.

You can include batch commands as any combination of options, single string arguments, or files on the command line. However, because commands are executed in the sequence that they appear in the command string, *if specified*:

- Test suites must precede the work directory or .jti file.
- The work directory and .jti file must match the test suite.
- The work directory must precede any standard values.
- The .jti file must precede any changes to the current configuration.

### 13.1.3. Setting the Standard Values

The following lists the batch commands used to set the Standard Values in the current configuration:

- concurrency *number* [p 109]
- env *environment* [p 109]
- envFile *environment-file* [p 109]
- excludeList *exclude-list-file* [p 110]
- keywords *keyword-expr* [p 110]
- params *parameter-arguments* [p 110]
- priorStatus *status-arguments* [p 110]
- tests *test-name* [p 110]
- timeoutFactor *number* [p 110]

#### **concurrency *number***

Specifies the number of tests run concurrently. If you are running the tests on a multi-processor computer, concurrency can speed up your test runs.

Example:

```
..;concurrency 2;..
```

#### **env *environment***

Specifies a test environment in an environment file. This is only used with envFiles of test suites that use parameter (.jtp) and environment (.jte) files.

Example:

```
..;env testsystem;..
```

#### **envFile *environment-file***

Specifies an environment file (.jte) containing test environments that the JavaTest harness uses to run older test suites. The environment file is not used by newer test suites that use a configuration (.jti) file.

Example:

```
..;envFile test.jte;..
```

### **excludeList** *exclude-list-file*

Exclude list files contain a list of tests that are not to be run. Exclude list files conventionally use the extension .jtx and are normally supplied with a test suite.

Example:

```
..;excludeList sample.jtx;..
```

### **keywords** *keyword-expr*

Restricts the set of tests to be run based on keywords associated with tests in the test suite

Example:

```
..;keywords interactive;..
```

### **params** *parameter-arguments*

These commands are deprecated. However, for backwards compatibility, if you are running a test suite that uses a parameter file (.jtp) you can continue to use the `params` command and its arguments to set parameter values when starting the JavaTest harness. Refer to Parameter Commands [p 114] for detailed information about using the `params` command.

### **priorStatus** *status-arguments*

Selects the tests included in a test run based on their outcome on a prior test run. The *status-arguments* that can be used are "pass," "fail," "error," and "notRun." If you use more than one argument, each argument must be separated by a comma.

Example:

```
..;priorStatus fail,error;..
```

### **tests** *test-name*

Create a list of test directories and/or tests to run. The JavaTest harness walks the test tree starting with the sub-branches and/or tests you specify and executes all tests that it finds (unless they are filtered out).

Example:

```
..;tests api;..
```

### **timeoutFactor** *number*

Increases the amount of time that the JavaTest harness waits for a test to complete before moving on to the next test. Each test's timeout limit is multiplied by the time factor value. For example, if you specify a value of "2", the timeout limit for tests with a 10 basic time limit becomes 20 minutes.

Example:

```
..;timeoutFactor 2;..
```

### 13.1.4. Setting Other Configuration Values

The following batch command is used to set test suite specific values in the current configuration:

```
set question-tag-name value [p 111]
```

#### **set *question-tag-name value***

Changes the response of a specific question in a configuration (.jti) file without having to open the JavaTest GUI. To display the *question-tag-name* of a question, open the configuration editor, select a block of text in the question panel, and press the "Alt" and "T" keys. The question panel's tag name is displayed at the bottom of the panel.

Example:

```
..; set jck.env.runtime.testExecute.cmdAsFile jdk_install_dir/bin/java ;..
```

### 13.1.5. Running Tests in Batch Mode

You can run one or more tests or branches of tests in batch mode as part of a series of batch commands (such as multiple test runs using different configuration values, write test reports, and audit test results) or as a single batch command.

Example:

```
...; runtests ;...
```

To configure and run tests in batch mode, format the `-batch` option, its commands, and the `-runtests` command as described in [Formatting Batch Commands \[p 105\]](#) .

See [Modifying Settings in a Configuration \[p 118\]](#) for an example of how to form the command used to run tests in batch mode.

### 13.1.6. Writing Reports in Batch Mode

You can write test reports in batch mode by using the `-writereport` batch command as part of a series of batch commands (such as run tests and audit test results) or as a separate write reports batch command.

Because the JavaTest harness executes batch commands in their command line sequence, identify the work directory before the `-writereport` command and providing the report directory as an option after the command.

Example:

```
..; -workdirectory work-directory -writereport report-directory ;..
```

To configure and write reports in batch mode, format the `-batch` option, its commands, and the `-writereport` command as described in [Formatting Batch Commands \[p 105\]](#) .

### 13.1.7. Auditing Tests

You can audit test results in batch mode by using the `-audit` batch command as part of a series of batch commands or as a separate batch command. The results of the audit are output to the terminal.

Because the JavaTest harness executes batch commands in their command line sequence, you must identify the work directory before the `-audit` command.

Example:

```
..; -workdirectory work-directory -audit ;..
```

To audit tests in batch mode, format the `-batch` option, its commands, and the `-audit` command as described in Formatting Batch Commands [p 105] .

### 13.1.8. Index of Available Batch Commands

The following lists all of the commands available in batch mode:

Command	Description
concurrency <i>number</i> [p 109]	Specifies the number of tests run concurrently.
env <i>environment</i> [p 109]	Specifies a test environment in an environment file.
envFile <i>environment-file</i> [p 109]	Specifies an environment file (.jte) containing test environments.
excludeList <i>exclude-list-file</i> [p 110]	Specifies an exclude list file.
keywords <i>keyword-expr</i> [p 110]	Restricts the set of tests to be run based on keywords.
open <i>name</i> [p 107]	Opens a test suite, work directory, parameter .jtp file, or a configuration .jti file.
params <i>parameter-arguments</i> [p 110]	These commands are deprecated.
priorStatus <i>status-arguments</i> [p 110]	Selects the tests included in a test run based on their outcome on a prior test run.
set <i>question-tag-name value</i> [p 111]	Changes the response of a specific question in a configuration (.jti) file.
testSuite <i>testsuite</i> [p 108]	Specifies the test suite that is run.
tests <i>test-name</i> [p 110]	Create a list of test directories and/or tests to run.
timeoutFactor <i>number</i> [p 110]	Increases the amount of time that the JavaTest harness waits for a test to complete
workDirectory <i>work-directory</i> [p 108]	Open an exiting work directory, create a new work directory, or replace an existing work directory with a new work directory.
runtests [p 111]	Run tests in batch mode.
audit [p 111]	Audit test results in batch mode.
writereport [p 111]	Write test reports in batch mode.



## 13.2. Specifying Additional Options

In most cases, command-line options perform functions that are also available through the GUI. However, there are several situations in which using command-line options to specify how the JavaTest harness starts is either uniquely useful or necessary.

When starting the JavaTest harness you can use options in the command line to:

- Include all system properties [p 114] in test execution environments
- Set an environment variable [p 114] that you want inherited in every test environment
- Set the agent pool port number [p 114]
- Set the agent pool timeout [p 114]
- Start the active agent pool [p 114]
- Specify parameters [p 114] if you are running a test suite that uses a parameter file
- Use a new desktop [p 114] when starting the JavaTest GUI

Start the JavaTest harness from a writeable directory where you intend to create files and store test results. You must include the path of the directory [*jt\_dir*] where the javatest.jar file is installed. The javatest.jar file is usually installed in the TCK lib directory when the JavaTest harness is bundled with a TCK.

Use the following example to start the JavaTest harness with command-line options:

```
java -jar [jt_dir]/javatest.jar [options]
```

If you are running the JavaTest harness in batch mode, refer to Using Batch Mode [p 105] for detailed information about using batch commands with the command-line options.



The JavaTest harness uses a new desktop when you include GUI options in the command line.

The following options can be used in the command line to specify how the Javatest harness starts:

Option	Function
<code>-EsysProps</code>	Includes all system properties in test execution environments.
<code>-Ename=value</code>	<p>Sets an environment variable that is inherited in every test environment created.</p> <p>The <code>-Ename=value</code> command line option tunnels in values from the external shell. The method used in previous versions of the JavaTest harness to tunnel in values from the external shell is now deprecated.</p>
<code>-agentPoolPort port</code>	<p>Set the Agent Pool Port Number</p> <p>Use this option only if you are configuring the JavaTest harness and the agent to use a port other than 1907.</p>
<code>-agentPoolTimeout #seconds</code>	<p>Set the Agent Pool Timeout</p> <p>Sets the number of seconds that the JavaTest harness waits between tests for an available agent before reporting the test result as an error. The default value of 180 seconds is usually sufficient. You can also set this value in the GUI if you are not running the JavaTest harness in batch mode.</p>
<code>-startAgentPool</code>	<p>Start the Active Agent Pool</p> <p>If you use an active agent and run the JavaTest harness in batch mode, you must add <code>-startAgentPool</code> to the command string to start the Agent Pool.</p>
<code>-params [commands] [initial-files]</code>	<p>Specify Parameters</p> <p>If you are running a test suite that uses a parameter file (<code>.jtp</code>), you can specify different parameter values when starting the JavaTest harness by including the <code>-params</code> option and the appropriate parameter command in the command line.</p> <p>If you are using the JavaTest GUI to run tests, refer to Using Parameter Commands [p 114] for detailed information about using the <code>-params</code> option and its commands.</p>
<code>-newDesktop</code>	<p>Add <code>-newDesktop</code> to the command string to start the JavaTest GUI without using a previous desktop. The JavaTest GUI will ignore any previous settings and open the Welcome to JavaTest dialog box. Refer to Welcome Dialog Boxes [p 5] for a description of the dialog box.</p> <p>A new desktop is automatically started and the old desktop is ignored when explicit GUI command line options are used.</p>

### 13.2.1. Using Parameter Commands

These commands are deprecated. You should use the `-batch` commands to perform these functions. If you are running a test suite that uses a parameter file (`.jtp`), for backwards compatibility you can continue to use the following commands with the `-param` option to change one or more parameter values used to start the JavaTest harness:

Command	Description
<code>-t testsuite</code> or <code>-testsuite</code> <i>testsuite</i>	Specifies the test suite that is run.
<code>-keywords</code> <i>keyword-expr</i>	Restricts the set of tests to be run based on keywords associated with tests in the test suite.
<code>-status status-expr</code>	Includes or excludes tests from a test run based on their status from a previous test run. Valid status expressions are error, failed, not run, and passed.
<code>-exclude</code> <i>exclude-list-file</i>	Specifies an exclude list file. Exclude list files contain a list of tests that are not to be run. Exclude list files conventionally use the <code>.jtx</code> extension and are normally supplied with a test suite.
<code>-envFile</code> <i>environment-file</i>	Specifies an environment file that contains information used by the JavaTest harness to run tests in your computing environment. You can specify an environment file for the JavaTest harness to use when running tests.
<code>-env environment</code>	Specifies a test environment from an environment file.
<code>-concurrency number</code>	Specifies the number of tests run concurrently. If you are running the tests on a multi-processor computer, concurrency can speed up your test runs.
<code>-timeoutFactor</code> <i>number</i>	<p>Increases the timeout limit by specifying a value in the time factor option.</p> <p>The timeout limit is the amount of time that the JavaTest harness waits for a test to complete before moving on to the next test. Each test's timeout limit is multiplied by the time factor value.</p> <p>For example, if you specify a value of "2", the timeout limit for tests with a 10 basic time limit becomes 20 minutes.</p>
<code>-r report-directory</code> or <code>-report</code> <i>report-directory</i>	Specifies the directory where the JavaTest harness writes test report files. If this path is not specified, the reports are written to a directory named report in the directory from which you started the JavaTest harness.
<code>-w work-directory</code> or <code>-workDir</code> <i>workDirectory</i>	Specifies a work directory for the test run. Each work directory is associated with a test suite and stores its test result files in a cache.

## 13.3. Displaying JavaTest Information

The following options are used at the end of the command line to display JavaTest information without starting the harness. You must include the path of the directory *[jt\_dir]* where the javatest.jar file is installed. The javatest.jar file is usually installed in the TCK lib directory when the JavaTest harness bundled with a TCK:

```
java -jar [jt_dir]/javatest.jar [information-options]
```

Function	Option
Display Command-Line Help	<code>-help</code> <code>-usage</code> or <code>-?</code> Displays command-line help without starting the JavaTest harness.
Display JavaTest Version Information	<code>-version</code> Displays the version, location, and build information for the installed copy of the JavaTest harness.
Display Online Help	<code>-onlineHelp</code> Displays JavaTest online help without starting the JavaTest harness.

## 13.4. Examples of Batch Commands

This section provides examples of different types of batch commands. The commands used with `-batch` are a formatted set of commands, executed in the sequence that they appear in the command string.

All batch command formats use combinations of commands in the following basic sequence:

Example:

```
java -jar [jt_dir]/javatest.jar [set harness options] -batch [initialize configuration] [set standard values] [set other configuration values] [perform task]
```

You can set configuration values by using a *tag-name* to *value* format. It is an error to set a configuration value in a batch command if the question *tag-name* is not found in the current interview path. To determine the current interview path, use the `-path` option.

See Obtaining the Question *tag-name* [p 116] below for detailed information about the *tag-name* for the question.

See Formatting Batch Commands [p 105] for a description of the different batch command formats that can be used.

The following sections provide different examples that you can use as patterns for creating batch commands:

- ▶ Editing in Batch Commands [p 117]
- ▶ Modifying Settings in a Configuration [p 118]
- ▶ Using a Batch File [p 119]

### 13.4.1. Obtaining the Question *tag-name*

There are two ways to obtain a configuration question *tag-name*:

- **ALT T** - Start the JavaTest harness, open the Configuration Editor and load the configuration file containing the value to be changed. Click in the Configuration Editor's question box and then press ALT and T. The Configuration Editor displays the *tag-name* for the question in the title bar. You can navigate through the interview until you locate the question whose value must be changed.
- **Question Log** - Start the JavaTest harness and load the configuration file that will be used to run tests. Choose Configure > Show Question Log in the Test Manager menu bar to view the Question Log of the current configuration. The Question Log displays the *tag-name* for each question in the interview

and its value.

## 13.5. Editing in Batch Commands

You can use the `-batch` mode option and its commands to edit a configuration either from the command line as part of a batch process or as a part of a product build process. The commands used with `-batch` are a formatted set of commands, executed in the sequence that they appear in the command string. See [Formatting Batch Commands \[p 105\]](#) for a description of the formats you can use.

The following are provided as examples of how editing can be accomplished in batch mode:

- Open a `.jti` file and change values before running tests [\[p 117\]](#)
- Create a new work directory [\[p 117\]](#)

Use the commands in the command string as you would if you were writing a script.

### 13.5.1. Open a `.jti` File and Change Values Before Running Tests

You can import a site specific `.jti` file (as a template) and then set specific configuration values before running tests.

In the following examples, a test suite and work directory are opened, a `.jti` file (`myconfig.jti`) is imported, and the host name in the `.jti` file is changed to "mymachine" before running tests. The test suite (`mytestsuite.ts`) and existing work directory (`myworkdir.wd`) must be compatible.



To run the following examples you must replace `mytestsuite.ts`, `myworkdir.wd`, and `myconfig.jti` with test suite, work directory, and `.jti` names that exist on your system. Win32 users must also change "/" file separators to "\" to run these examples.

You must also replace `jck.env.runtime.net.localHostName` and its value with a question *tag-name* and value in your current interview path. From the command line, you can only change values in your current interview path. See [Obtaining the Question tag-name \[p 116\]](#) for detailed information about the *tag-name* for the question.

Batch Options Example:

```
java -jar lib/javatest.jar -batch -testsuite mytestsuite.ts -workdir
myworkdir.wd -open myconfig.jti -set jck.env.runtime.net.localHostName
mymachine -runtests
```

Single String Arguments Example:

```
java -jar lib/javatest.jar -batch "testsuite mytestsuite.ts workdir
myworkdir.wd open myconfig.jti; set jck.env.runtime.net.localHostName
mymachine; runtests"
```

You can only change configuration values in the current interview path of the `.jti` file. If you change a value that is not in the current interview path the JavaTest harness displays a error message. You can view the current interview path in the Configuration Question Log or in the Configuration Editor All Values view.

### 13.5.2. Create a New Work Directory

You can open an existing work directory (as a template) and then use it to create a new work directory for the the test run before running tests.

In the following examples, a test suite (`mytestsuite.ts`) and work directory (`myworkdir.wd`) are opened, and a new work directory (`testrun.wd`) is created before running tests. The results of the test run are written to the new work directory.



To run the following examples you must replace `mytestsuite.ts`, `myworkdir.wd`, and `myconfig.jti` with test suite, work directory, and .jti names that exist on your system. Win32 users must also change "/" file separators to "\" to run these examples.

#### Batch Options Example:

```
java -jar lib/javatest.jar -batch -testsuite mytestsuite.ts -workdir
myworkdir.wd -create testrun.wd -open myconfig.jti -runtests
```

#### Single String Arguments Example:

```
java -jar lib/javatest.jar -batch "testsuite mytestsuite.ts workdir
myworkdir.wd; create testrun.wd; open myconfig.jti; runtests"
```

The JavaTest harness uses the work directory (`testrun.wd`) created by the command line when the tests are run, even if `myconfig.jti` was created using another work directory.

## 13.6. Modifying Settings in a Configuration

You can use the `-batch` mode option and its commands to temporarily modify the contents of a .jti file loaded in memory without changing the .jti file itself. This can be done when running tests, writing test reports, and auditing test results either from the command line or as a part of a product build process.

The commands used with `-batch` are a formatted set of commands, executed in the sequence that they appear in the command string. See *Formatting Batch Commands* [p 105] for a description of the formats you can use.

Use the commands in the command string as you would if you were writing a script.

### 13.6.1. Selecting Tests to Run

You can use an existing .jti file and then set the specific tests to be run. You can specify one or more individual tests or branches of tests to be run

In the following examples, a test suite (`mytestsuite.ts`) and work directory (`myworkdir.wd`) and a .jti file (`myconfig.jti`) are opened, and the tests to be run are set before running tests. In the following example, tests in branches `api/javax_swing` and `api/java_awt` will be run.



To run the following examples you must replace `mytestsuite.ts`, `myworkdir.wd`, and `myconfig.jti` with test suite, work directory, and .jti names that exist on your system. Win32 users must also change "/" file separators to "\" to run these examples.

#### Batch Options Example:

```
java -jar lib/javatest.jar -batch -testsuite mytestsuite.ts -workdir
myworkdir.wd -open myconfig.jti -tests api/javax_swing api/java_awt -runtests
```

#### Single String Arguments Example:

```
java -jar lib/javatest.jar -batch "testsuite mytestsuite.ts workdir
myworkdir.wd open myconfig.jti; tests api/javax_swing api/java_awt; runtests"
```

The path used for setting the test or folder of tests to be run is the same as that displayed in the tree folder. One or more tests can be specified.

## 13.6.2. Selecting an Exclude List

You can use an existing .jti file and then specify the exclude list to be used when running tests.

In the following examples, a test suite (mytestsuite.ts) and work directory (myworkdir.wd) and a .jti file (myconfig.jti) are opened, and the exclude list (myexcludelist.jtx) is set before running tests.



To run the following examples you must replace mytestsuite.ts, myworkdir.wd, myconfig.jti, and myexcludelist.jtx with test suite, work directory, .jti and .jtx names that exist on your system. Win32 users must also change "/" file separators to "\" to run these examples.

Batch Options Example:

```
java -jar lib/javatest.jar -batch -testsuite mytestsuite.ts -workdir  
myworkdir.wd -open myconfig.jti -exclude myexcludelist.jtx -runtests
```

Single String Arguments Example:

```
java -jar lib/javatest.jar -batch "testsuite mytestsuite.ts workdir  
myworkdir.wd open myconfig.jti; exclude myexcludelist.jtx; runtests"
```

## 13.6.3. Setting Configuration Values

You can use an existing .jti file and then specify the values to be used when running tests.

In the following examples, a test suite (mytestsuite.ts), work directory (myworkdir.wd), and a .jti file (myconfig.jti) are opened, then one or more name=values are specified for running tests.



To run the following examples you must replace mytestsuite.ts, myworkdir.wd, and myconfig.jti with test suite, work directory, and .jti names that exist on your system. Win32 users must also change "/" file separators to "\" to run these examples.

To change a value on the command line it must be in your current interview path. If your current interview path does not include jckdate.gmtOffset you must either add it to the interview path or replace it with a value that is in the path. To view the current interview path, open your .jti file in the Configuration Editor. See Obtaining the Question tag-name [p 116] for detailed information about the *tag-name* for the question.

Batch Options Example:

```
java -jar lib/javatest.jar -batch -testsuite mytestsuite.ts -workdir  
myworkdir.wd -open myconfig.jti -set jckdate.gmtOffset=8 -runtests
```

Single String Arguments Example:

```
java -jar lib/javatest.jar -batch "testsuite mytestsuite.ts workdir  
myworkdir.wd open myconfig.jti; set jckdate.gmtOffset=8; runtests"
```

## 13.7. Using a Batch File

You can place routinely used configuration settings in a "batch file" and then use it in additional or repeated test runs.

When you use the -batch mode option and its commands to configure and run tests, write test reports, and audit test results (either from the command line or as a part of a product build process), you can use a .jti file as a standard template and a batch file to modify specific configuration values before running tests.

The commands used in a batch file with `-batch` are a formatted set of commands, executed in the sequence that they appear in the command string. Use the commands in the batch file as you would if you were writing a script. See [Formatting Batch Commands \[p 105\]](#) for a description of the formats you can use.

### 13.7.1. Example of Using a Batch File

In a text file named "mybatchfile.jtb" is the following (`-set` is only used on the command line):

```
set jck.env.runtime.net.localHostName mymachine; tests api/javax_swing
api/java_awt
```

To change a value by using the `set` option, the question *tag-name* name must be in your current interview path. To view the current interview path, open your .jti file in the Configuration Editor. If your current interview path does not include `jck.env.runtime.net.localHostName` you must either add it to the interview path or replace it in the batch file with a *tag-name* and value that is in the path. See [Obtaining the Question tag-name \[p 116\]](#) for detailed information about the *tag-name* for the question.

In the following examples, a test suite (`mytestsuite.ts`), work directory (`myworkdir.wd`), and .jti file (`myconfig.jti`) are opened, and the batch file (`mybatchfile.jtb`) is read and executed before running tests.



To run the following examples you must replace `mytestsuite.ts`, `myworkdir.wd`, and `myconfig.jti` with test suite, work directory, and .jti names that exist on your system. Win32 users must also change "/" file separators to "\" to run these examples.

#### Batch Options Example:

```
java -jar lib/javatest.jar -batch -testsuite mytestsuite.ts -workdir
myworkdir.wd -open myconfig.jti @mybatchfile.jtb -runtests
```

#### Single String Arguments Example:

```
java -jar lib/javatest.jar -batch "testsuite mytestsuite.ts workdir
myworkdir.wd open myconfig.jti; @mybatchfile.jtb; -runtests"
```

You can also change values after the batch file is set:

#### Batch Options Example:

```
java -jar lib/javatest.jar -batch -testsuite mytestsuite.ts -workdir
myworkdir.wd -open myconfig.jti @mybatchfile.jtb -exclude myexcludelist.jtx
-runtests
```

#### Single String Arguments Example:

```
java -jar lib/javatest.jar -batch "testsuite mytestsuite.ts workdir
myworkdir.wd open myconfig.jti; @mybatchfile.jtb; exclude myexcludelist.jtx;
runtests"
```



# 14. Using Additional JavaTest Utilities

The JavaTest harness provides additional utilities that you can use to:

- Monitor results with HTTP server. Refer to Monitoring Results with HTTP Server [p 121] for detailed information.
- Browse result (.jtr) files without starting the harness. Refer to Browsing Result (.jtr) Files [p 125] for detailed information.
- Browse exclude list files without starting the harness. Refer to Browsing Exclude List Files [p 125] for detailed information.
- Edit responses in a configuration file without starting the harness. Refer to Editing Responses in a Configuration File [p 125] for detailed information.
- Move test reports. Refer to Moving Test Reports [p 130] for detailed information.

## 14.1. Monitoring Results with HTTP Server

The JavaTest harness provides a small web server that you can use to remotely monitor and control a test run. The HTTP Server provides two types of output:

- HTML Formatted Output [p 121]
- Plain Text Output [p 123]

### 14.1.1. HTML Formatted Output

The HTML formatted output is provided as human readable pages (these pages are subject to change in future releases of the JavaTest harness), allowing users to remotely monitor batch mode test runs in a web browser and stop any test runs that are not executing as expected:

- Server Index Page [p 122]
- Server Harness Page [p 123]
- Server Test Result Index Page [p 122]
- Harness Environment Page [p 122]
- Harness Interview Page [p 123]
- Stop a Test Run [p 123]

### Accessing HTTP Server HTML Formatted Output

1. Use the following command on the command line to activate the web server. Include the path of the directory [*jt\_dir*] where the javatest.jar file is installed. The javatest.jar file is usually installed in the TCK lib directory when the JavaTest harness bundled with a TCK:

```
java -jar [jt_dir]/javatest.jar -startHttp -batch [options]
```

2. Copy the URL reported to the console:

Example:

```
JavaTest HTTPd - Success, active on port 1903  
JavaTest HTTPd server available at http://129.145.162.75:1903/
```

3. Launch a web browser and enter or paste the URL in the browser URL field:

Example:

`http://129.145.162.75:1903/`

## Viewing HTTP Server Index Page

The root of the web server provides an index page that only lists the handlers registered with the internal web server; not all available URLs on the server. You can also display the HTTP Server Index page by including `/index.html` at the end of the URL in the browser URL field:

Example:

`http://129.145.162.75:1903/index.html`

Each JavaTest harness has it's own handler, identified by a unique number as the second component of the URL.

## Viewing HTTP Server Harness Page

When the JavaTest harness is running tests, the harness page displays:

- Name and location of the current test suite
- Location of the work directory
- Link to view the environment information provided to the JavaTest harness and used in the current test run. Displays an HTML formatted view of the current environment.
- Link to view the configuration interview used by the JavaTest harness in the current test run. Displays a formatted view of the interview settings.
- Link to view the current test results. Displays the Test Result Index page.

In addition to the list of registered handlers, the page also prints the UTC/GMT date on which that page was generated (subject to the system clock on the machine which JavaTest is running) and provides the JavaTest version number and build date.

You can display the HTTP Server Harness page by choosing its link on the index page or by including `/harness` at the end of the URL in the browser URL field:

Example:

`http://129.145.162.75:1903/harness`

## Viewing HTTP Server Test Result Index Page

The Test Result Index page displays:

- Work directory
- The total number of tests in the test suite.

The total number of tests is also a link to view the current test results. The test results are displayed in a two column table, by test name and status message.

You can display the Test Result Index page by choosing its link on the harness page.

## Viewing the Harness Environment Page

The Harness Environment page displays the environment information provided to the JavaTest harness and used in the current test run. The environment information is displayed in an HTML table and provides a view of the current settings.

You can display the Harness Environment page by choosing its link on the harness page or by including `/harness/env` at the end of the URL in the browser URL field:

Example:

```
http://129.145.162.75:1903/harness/env
```

## Viewing the Harness Interview Page

The Harness Interview page displays the configuration interview provided to the JavaTest harness and used in the current test run.

You can display the Harness Interview page by choosing its link on the harness page or by including `/harness/interview` at the end of the URL in the browser URL field:

Example:

```
http://129.145.162.75:1903/harness/interview
```

## Using HTTP Server to Stop a Test Run

If you want to remotely terminate a test run for any reason, you can use the HTTP server. Include `/harness/stop` at the end of the URL in the browser URL field:

Example:

```
http://129.145.162.75:1903/harness/stop
```

To stop the test run, you must click the STOP button on the page displayed in the browser.

### 14.1.2. Plain Text Output

The HTTP server provides plain text output that can be used for automated monitoring of the JavaTest harness during test runs. The plain text output does not include HTTP headings or HTML formatting and is intended for use by automated testing frameworks, not for viewing in web browsers. Consequently, future releases of the JavaTest harness will attempt to maintain the content formatting and URLs of this output.

Two types of JavaTest information can be accessed by automated testing frameworks:

- Accessing Version Information [p 123]
- Accessing Harness Information [p 122]

## Accessing Version Information

The HTTP Server Version page displays version information about the JavaTest harness. You can display the HTTP Server Version page by choosing its link on the index page or by including `/version` at the end of the URL in the browser URL field:

Example:

```
http://129.145.162.75:1903/version
```

A dump of the version information is provided.

Example:

```
JavaTest 3.0.3 Built on 06 Feb 2002
```

## Accessing Harness Information

The following strings access specific information about the JavaTest harness:

- `/harness/text/config`

Currently provides, in `java.util.Properties` format, the Test Suite name location and work directory of the current harness configuration values.

**Example:**

```
testsuite.path=/export/scratch/sampleJCK-compiler-13a testsuite.name=J2SE
Sample Compiler 1.3a TCK (JCK) workdir=/export/scratch/wdsc13a
```

- /harness/text/tests

Provides in java.util.Properties format the initial tests used for the current test run.

**Example:**

```
url0=api/java_lang url1=api/java_util
```

- /harness/text/stats

Provides, in java.util.Properties format, the current count of test results in the each state (pass, fail, error, not run). Whitespace is not present in the output:

**Example:**

```
Passed.=0
Failed.=151
Error.=54
Not_run.=1
```

For performance reasons, the Not\_run number usually equals the concurrency setting in batch mode and matches the "not run" number shown in the GUI when in GUI mode (Current Configuration view filter).

- harness/text/results

Provides alternating lines of test name, test status.

**Example:**

```
lang/FP/fpl005/fpl00506m1/fpl00506m1.html
Error. context undefined for hardware.xFP_ExponentRanges
lang/FP/fpl005/fpl00506m2/fpl00506m2.html
Error. context undefined for hardware.xFP_ExponentRanges
vm/classfmt/atr/atrnew003/atrnew00301m1/atrnew00301m1.html
Failed. unexpected exit code: exit code 1
vm/classfmt/atr/atrnew003/atrnew00302m1/atrnew00302m1.html
Failed. unexpected exit code: exit code 1
vm/classfmt/atr/atrnew003/atrnew00303m1/atrnew00303m1.html
Failed. unexpected exit code: exit code 1
```

- /harness/text/state

Indicates whether JavaTest is currently running. It will return one of the following:

```
running=true
running=false
```

- /harness/text/env

Provides, in java.util.Properties format, the current environment settings for the test run.

**Example:**

```
command.testExecute=com.sun.jck.lib.ExecJCKTestOtherJVMCmd
/work/jdk1.3.1/bin/java -classpath $testSuiteRootDir/classes
-Djava.security.policy=$testSuiteRootDir/lib/jck.policy $testExecuteClass
$testExecuteArgs context.nativeCodeSupported=true description=bar
jniTestArgs=-loadLibraryAllowed nativeCodeSupported=true
platform.expectOutOfMemory=true
```

## 14.2. Browsing Result (.jtr) Files

Included in the javatest.jar file is a servlet that allows you to use a web browser to view .jtr files.

To view .jtr files in your web browser, you must configure your web server to use the JavaTest ResultBrowser servlet:

```
com.sun.javatest.servlets.ResultBrowser
```

Refer to your server documentation for information about configuring it to use the JavaTest ResultBrowser servlet. Typically, you configure the web server to direct .jtr files to the servlet for rendering.

## 14.3. Browsing Exclude List Files

Included in the javatest.jar file is a servlet that allows you to use a web browser to view .jtx files.

To view .jtx files in your web browser, you must configure your web server to use the JavaTest ExcludeBrowser servlet:

```
com.sun.javatest.servlets.ExcludeBrowser
```

Refer to your server documentation for information about configuring it to use the JavaTest ExcludeBrowser servlet. Typically, you configure the web server to direct .jtx files to the servlet for rendering.

## 14.4. Editing Responses in a Configuration File

The JavaTest harness provides an EditJTI utility that you can use from the command line to edit the responses in a configuration file without opening the JavaTest GUI. The EditJTI utility is the batch command equivalent of the JavaTest™ Configuration Editor.

### 14.4.1. Format of EditJTI Command

The EditJTI utility loads a configuration (.jti) file, and applies a series of specified edits. You can save the file back to the original file, or to another file. You can use the EditJTI utility to generate an HTML log of the questions and responses as well as write a quick summary of the questions and responses to the console. The EditJTI utility provides a preview mode; in addition, configuration files are normally backed up before being overwritten.

Example:

```
java -cp lib/javatest.jar com.sun.javatest.EditJTI [OPTIONS] [EDIT COMMANDS]  
original configuration file
```

While it is possible to use EditJTI to change the order of commands in a configuration file, because of the dependencies between values, it is recommended that major changes in the .jti file be accomplished by using the Configuration Editor.

#### OPTIONS

The various options are as follows:

- help
- usage
- /?

Display a summary of the command line options.

- classpath *classpath*
- cp *classpath*

Override the default classpath used to load the classes for the configuration interview. The default is determined from the work directory and test suite specified in the configuration file. The new

- location will be as specified by this option.
- log *log-file*
  - l *log-file*  
Generate an HTML log containing the questions and responses from the configuration file. The log is generated after any edits have been applied.
  - out *out-file*
  - o *out-file*  
Specify where to write the configuration file after the edits (if any) have been applied. The default is to overwrite the input file if the interview is edited.
  - path
  - p  
Generate a summary to the console output stream of the sequence of questions and responses from the configuration file. The summary is generated after any edits have been applied.
  - preview
  - n  
Do not write out any files, but instead, preview what would happen if this option were not specified.
  - testsuite *test-suite*
  - ts *test-suite*  
Override the default location used to load the classes for the configuration interview. The default is determined from the work directory and test suite specified in the configuration file. The new location is determined from the specified test suite.
  - verbose
  - v  
Verbose mode. As the edit commands are executed, details of the changes are written to the console output stream.

## EDIT COMMANDS

Two different types of edit command are supported.

*tag-name=value*

Set the value for the question whose tag is *tag-name* to *value*. It is an error if no such question is found. The question must be on the current path of questions being asked by the interview. To determine the current path, use the `-path` option. See Obtaining the Question *tag-name* [p 127] below.

*/search-string/replace-string/*

Scan the path of questions being asked by the interview, looking for responses that match (contain) the search string. In such answers, replace *search-string* by *replace-string*. Note that changing the response to a question may change the subsequent questions that are asked. It is an error if no such questions are found.

If you wish to use `/` in the search string, you may use some other punctuation character as a delimiter, instead of `/`. For example, `|search-string|replace-string|`

Note: regular expressions are not currently supported in *search-string*, but may be supported in a future release.

Depending on the shell you are using, you may need to quote your edit commands, to protect any characters in them against interpretation by the shell.

## RETURN CODE

The program will exit with one of the following return codes:

- 0 the operations were successful; the configuration file is complete and ready to use.
- 1 the operations were successful, but the configuration file is incomplete and is not yet ready to use for a test run.
- 2 there was a problem with the command line arguments
- 3 an error occurred while trying to perform the copy

## SYSTEM PROPERTIES

Two system properties are recognized:

`EditJTI.maxIndent`

Used when generating the output for the `-path` option, this property specifies the maximum length of tag name after which the output will be line-wrapped before writing the corresponding value. The default value is 32.

`EditJTI.numBackups`

Specifies how many levels of backup to keep when overwriting a `.jti` file. The default is 2. A value of 0 disables backups.

### 14.4.2. Obtaining the Question *tag-name*

There are two ways to obtain the question *tag-name*:

- **ALT T** - Start the JavaTest harness, open the Configuration Editor and load the configuration file containing the value to be changed. Click in the Configuration Editor's question box and then press ALT and T. The Configuration Editor displays the *tag-name* for the question in the title bar. You can navigate through the interview until you locate the question whose value must be changed.
- **Question Log** - Start the JavaTest harness and load the configuration file that will be used to run tests. Choose Configure > Show Question Log in the Test Manager menu bar to view the Question Log of the current configuration. The Question Log displays the *tag-name* for each question in the interview and its value.

A detailed description of the utility is also available in your TCK at:

`doc/javatest/editJTI.html`

## 14.5. Examples of Using EditJTI

The following are examples of basic operations that can be performed by using EditJTI:

- Edit a Configuration File [p 127]
- Generate a Log of All Updates [p 128]
- Preview But Not Change [p 128]
- Echo Results of Your Edit [p 128]
- Show Paths for Debugging [p 128]
- Change Test Suites or Create a New Interview [p 128]
- Change the HTTP Port and Overwrite Original Configuration File [p 129]
- Change the HTTP Port and Create New Configuration File [p 129]
- Doing Escapes in a UNIX Shell [p 129]

See Editing Responses in a Configuration File [p 125] for a detailed list of options that can be used in the EditJTI command.

### 14.5.1. Edit a Configuration File

When using EditJTI to edit a configuration file you can use either one of two edit command formats:

1. *tag=value* for direct replacement of values. You must know the *tag-name* for the question that sets the value.
2. */old pattern/new pattern/* to replace all occurrences (strings) of an "old pattern" to a "new pattern." This format replaces all occurrences in the file.

When using the */old pattern/new pattern/* format, the separator used can be any character, however, it is recommended that the string be enclosed in quotes to avoid shell problems:

```
"|/java/jdk/1.3/|/java/jck/1.4/|"
```



To run the following examples of editing configuration files, you must replace `myoriginal.jti` with a `.jti` file name that exists on your system. Win32 users must also replace the `"\"` file separators with `"/` to run these examples.

Example:

```
java -cp lib/javatest.jar com.sun.javatest.EditJTI -o mynew.jti  
"|/java/jdk/1.3/|/java/jck/1.4/|" myoriginal.jti
```

## Generate a Log of All Updates

You can use the `-l` option to generate a log of all updates to the `jti` file which can be used later.

Example:

```
java -cp lib/javatest.jar com.sun.javatest.EditJTI -o mynew.jti -l  
myeditlog.html "|/java/jdk/1.3/|/java/jck/1.4/|" myoriginal.jti
```

## Preview But Not Change

You can use the `-n` option to preview but not perform updates to the `jti` file:

Example:

```
java -cp lib/javatest.jar com.sun.javatest.EditJTI -n -o mynew.jti -l  
myeditlog.html "|/java/jdk/1.3/|/java/jck/1.4/|" myoriginal.jti
```

## Echo Results of Your Edit

You can include the `-v` option to echo results of your edit.

Example:

```
java -cp lib/javatest.jar com.sun.javatest.EditJTI -n -v -o mynew.jti -l  
myeditlog.html "|/java/jdk/1.3/|/java/jck/1.4/|" myoriginal.jti
```

## Show Paths for Debugging

The `-p` option can be used to show the path during debugging. Using `-p` options in the command string displays how the path is changed by your edit.

Example:

```
java -cp lib/javatest.jar com.sun.javatest.EditJTI -n -o mynew.jti -l  
myeditlog.html -p "|/java/jdk/1.3/|/java/jck/1.4/|" myoriginal.jti
```

## Change Test Suites or Create a New Interview

The following example uses the `-ts` option to create an empty interview derived from the test suite (`mytestsuite.ts`). This is only recommended for very simple test suites.

Example:

```
java -cp lib/javatest.jar com.sun.javatest.EditJTI -o mynew.jti -l  
myeditlog.html -ts mytestsuite.ts "|/java/jdk/1.3/|/java/jck/1.4/|" myoriginal.jti
```

If a change is made that is not in the current interview path, then the interview will be invalid and the tests can not be run.



Generally, you should not use EditJTI to change the interview path, only the values on the existing path. If you are in doubt about the current interview path, open the Configuration Editor in the JavaTest harness and use it to change the values. The Configuration Editor will then display the current interview path for that question name/value.

### 14.5.2. Change the HTTP Port and Overwrite Original Configuration File

The following example changes the http port used when running tests and overwrites original configuration file (`myoriginal.jti` in this example).



To run the following example you must use a .jti file that exists on your system and include `httpPort` in your current interview path. If your current interview path does not include `httpPort` you will not be able to change its value from the command line. To view the current interview path, open your .jti file in the Configuration Editor. See Obtaining the Question tag-name [p 127] for detailed information about the *tag-name* for the question.

```
java -cp lib/javatest.jar com.sun.javatest.EditJTI httpPort=8081 myoriginal.jti
```

### 14.5.3. Change the HTTP Port and Create a New Configuration File

The following example changes the http port used when running tests and writes the changed configuration to a new configuration file (`myoutput.jti` in this example). The original configuration file (`myoriginal.jti` in this example) remains unchanged.



To run the following example you must use a .jti file that exists on your system and include `httpPort` in your current interview path. If your current interview path does not include `httpPort` you will not be able to change its value from the command line. To view the current interview path, open your .jti file in the Configuration Editor. See Obtaining the Question tag-name [p 127] for detailed information about the *tag-name* for the question.

```
java -cp lib/javatest.jar com.sun.javatest.EditJTI -o myoutput.jti  
httpPort=8081 myoriginal.jti
```

### 14.5.4. Doing Escapes in a UNIX Shell

The following uses the syntax for doing escapes in a UNIX shell. Changes to the original configuration file (`myconfig.jti` in this example) are written to a new configuration file (`my-newconfig.jti` in this example).



To run the following example you must replace `myoriginal.jti` with a .jti file name that exists on your system and only change values that are in your current interview path. To view the current interview path, open your .jti file in the Configuration Editor. Win32 users must also change "/" file separators to "\" to run these examples.

To change a value in the command line, use the *tag-name* for the question that sets the value. See Obtaining the Question tag-name [p 127] for detailed information about viewing the *tag-name* for the question.

```
java -cp lib/javatest.jar com.sun.javatest.EditJTI -o my-newconfig.jti  
tck.serialport.midPort=/dev/term/a tck.connection.httpsCert="\CN=Brian K,  
OU=JSW, O=Sun, L=SCA22, ST=CA, C=US\" myoriginal.jti
```

A detailed description of the utility is also available in your TCK at:

## 14.6. Moving Test Reports

Because JavaTest reports contain links to files, you must update the links when moving reports to other directories. The JavaTest harness provides an EditLinks utility for you to use when moving reports.

### 14.6.1. Format of EditLinks Command

*EditLinks* reads one or more files or directories and copies them to a new file or directory. Files whose names end with ".html" are checked to see if they contain any HTML links that begin with certain filenames. Any that do are rewritten with a corresponding different filename. All other files are copied without change.

Example:

```
java -classpath lib/javatest.jar com.sun.javatest.EditLinks OPTIONS file...
```

#### OPTIONS

The various options are as follows:

**-e** *oldPrefix newPrefix*

Any links that begin with the string *oldPrefix* are rewritten to begin with *newPrefix*. Note that only the target of the link is rewritten, and not the presentation text. Thus the edit is effectively transparent when the file is viewed in a browser. Multiple **-e** options may be given; when editing a file, the options are checked in the order they are given.

For example, if the argument

**-e** /work/ /java/jck-dev/scratch/12Jun00/jck-lab3/

is used on a file that contains the following segment:

```
<a
href="/work/api/java_lang/results.jtr">work/api/java_lang/results.jtr</a>
```

the text shown bold below will match:

```
<a
href="/work/api/java_lang/results.jtr">work/api/java_lang/results.jtr</a>
```

and the resulting new file will contain the following:

```
<a
href="/java/jck-dev/scratch/12Jun00/jck-lab3/api/java_lang/results.jtr">work/api/java_lang/results.jtr</a>
```

**-ignore** *file*

When scanning directories, ignore any entries named *file*. Multiple **-ignore** may be given.

For example, '**-ignore** SCCS' will cause any directories named SCCS to be ignored.

**-o** *file*

The output file or directory. The output may only be a file if the input is a single file; otherwise, the output should be a directory into which the edited copies of the input files will be placed.

*file...*

The input files to be edited. If any of the specified files are directories, they will be recursively copied to the output directory, and any HTML files within them updated.

#### RETURN CODE

The program will exit with one of the following return codes:

- 0    the copy was successful
- 1    there was a problem with the command line arguments
- 2    an error occurred while trying to perform the copy

## 14.6.2. Example of EditLinks Command



To run the following example you must replace `myworkdir.wd` with a work directory name that exists on your system. Win32 users must also replace the "\" file separators with "/" to run these examples.

```
java -cp lib/javatest.jar com.sun.javatest.EditLinks -e /work/  
/java/jck-dev/scratch/12Jun00/jck-lab3/ -o test12_dir.wd myworkdir.wd
```

A detailed description of the utility is also available in your TCK at:

`doc/javatest/editlinks.html`

# 15. Troubleshooting



The JavaTest harness provides information you can use in troubleshooting problems in running tests and in using JavaTest Agents.

## 15.1. Problems in Running Tests

The goal of a test run is for all tests in the test suite that are not filtered out to have passing results.

If the root test suite folder contains tests with errors or failing results, you must troubleshoot and correct the cause to satisfactorily complete the test run.

There are two types of problems that can occur when running tests:

Problem	Description
 Tests with errors	Tests with errors are tests that could not be executed by the JavaTest harness. These errors usually occur because the test environment is not properly configured. Use the Test tabbed panes and the Configuration Editor to help determine the change required in the configuration.
 Tests that fail	Tests that fail are tests that were executed but had failing results. The test or the implementation may have errors. Use the Test tabbed panes

See Troubleshooting a Test Run [p 41] for information about the resources that the JavaTest harness provides for troubleshooting.

## 15.2. Problems Using Agents

See Troubleshooting JavaTest Agents [p 99] for information about the tools that the JavaTest harness provides for troubleshooting problems using JavaTest Agents.

## 16. Glossary

### Active Agent

An agent [p 133] that initiates a connection to the JavaTest harness.

Active agents allow you to run tests in parallel using many agents at once and to specify the test machines at the time you run the tests. Use the Agent Monitor [p 133] window to view the list of registered active agents and synchronize active agents with the JavaTest harness before running tests.

### Agent

A lightweight Java application that receives tests from the test harness, runs them on the implementation being tested, and reports the results back to the test harness. Normally test agents are only used when the TCK and implementation being tested are running on different platforms. When running tests on a platform other than the one running the JavaTest harness, you must use an agent. The JavaTest harness uses three types of agents:

- Active agents [p 133]
- Passive agents [p 136]
- Serial agents [p 137]

### Agent Monitor

The JavaTest window used to synchronize active agents and to monitor agent activity. The Agent Monitor window displays the agent pool [p 133] and the agents [p 133] currently in use. To open the Agent Monitor window, choose Tasks > Monitor Agent Activity from the menu bar.

### Agent Pool

A list in the Agent Monitor [p 133] of the active agents [p 133] that are connected with the JavaTest harness and available to run tests. Agents are removed from the agent pool when they are running tests.

### All Tests

A test tree view filter [p 139] that displays all folders and tests in the test suite [p 138] .

### Audit

The JavaTest harness includes an audit tool that you can use to analyze the test results in a work directory. The audit tool verifies that all tests in a test suite ran correctly and identifies any audit categories of a test run that had errors.

You can audit a test run in GUI mode or in batch mode.

### Batch Mode

The mode in which the JavaTest harness is run from the command line without using a GUI. Running in batch mode allows you to include the JavaTest harness as part of a product build process.

### Configuration

Information about the computing environment, required to execute a test suite. The JavaTest harness uses the configuration editor to collect two types of data in an interview file [p 135] :

- Test environment [p 137]
- Standard Values [p 137]

Use the configuration editor to collect or modify configuration information or to load an existing configuration. See Configuration Editor [p 134] .

### **Configuration Editor**

The JavaTest Configuration Editor provides two views, the All Values View and the Standard Values view. Use the configuration editor's All Values view to create a configuration file for the test run and to search interview titles, questions, and answers for character strings. Use the configuration editor's Standard Values view to modify specific runtime values.

To open the Configuration Editor, choose Configure > Configuration Editor from the Test Manager menu bar.

### **Current Configuration Filter**

The configuration containing the test environment and standard values currently loaded in the test manager or specified in batch mode for use in running tests and displaying test status.

### **Custom Filter**

A filter [p 139] that can be edited in the Filter Editor to include or exclude the results of specific tests or folders either in views of the test tree [p 138] or in test reports.

### **Desktop**

The configuration and layout of the windows used by the JavaTest harness.

The desktop is saved when you exit from the harness and is automatically restored in your next session.

The JavaTest desktop is displayed in three user selectable styles:

- SDI [p 137]
- MDI [p 136]
- Tabbed [p 137]

### **Environment**

See Test Environment [p 137] .

### **Environment Files**

Contain one or more test environments used by test suites prior to the JavaTest 3.0 harness. Environment files are identified by the .jte extension in the file name.

### **Environment Variables**

Name/value pairs used by a test environment [p 137] to provide information about how to run tests of a test suite on a particular platform.

- For test suites prior to the JavaTest 3.0 harness, the environment variables are read from an environment file [p 134] (.jte).
- For JavaTest 3.0 (or later) test suites, the environment variables are derived from an interview file [p 135] (.jti).

### **Exclude List**



Exclude list files (\*.jtx), supply a list of invalid tests to be filtered out of a test run by the test harness. The exclude list provides a level playing field for all implementors by ensuring that when a test is determined to be invalid, then no implementation is required to pass it. Exclude lists are maintained by the technology specification Maintenance Lead and are made available to all technology licensees.

Use the configuration editor to add or remove exclude lists from a test run.

To view the contents of an exclude list, choose Configure > Show Exclude List from the Test Manager menu bar. Exclude lists can only be edited or modified by the test suite Maintenance Lead.

### **Filtered Out**

Folders and their tests that have been excluded from the test run by one or more test run filters [p 138] .

Filtered out folders and tests are identified in the test tree by grey  folder and  test icons.

### **Filters**

A facility in the JavaTest harness that accepts or rejects tests based on a set of criteria. Used to configure the Test Manager view [p 139] and to create test reports. Use the Configuration Editor [p 134] or the filter editor to configure the filters.

### **HTTP Server**

Software included in the JavaTest harness that services HTTP requests used to monitor a test run from a remote work station.

### **Interview Files**

Contains all of the information collected by the configuration editor [p 133] about the test platform.

The JavaTest harness uses the interview file (.jti) to derive the environment variables [p 134] required execute the test suite.

When changes to the environment variables are required, use the configuration editor to modify the .jti file.

### **.jtb Files**

An ASCII file containing a lengthy series of batch commands and their arguments used in the batch mode command line. Using a batch command file allows you to repeatedly use a batch configuration without retyping the commands each time a test run is performed.

It is recommended that a descriptive name and the extension .jtb are used to help identify the function of each batch file.

### **.jte Files**

See Environment Files [p 134] .

### **.jti Files**

See Interview Files [p 135] .

### **.jtr Files**

See Test Result Files [p 138] .

### **.jtx Files**

See Exclude List [p 134] .

## **Keywords**

Special values in a test description [p 137] that describe how the test is executed.

Keywords are provided by the test suite for use in the Configuration Editor as a filter to exclude or include tests in a test run.

## **MDI**

See Multiple Document Interface [p 136] .

## **Multiple Document Interface**

A window style in which the JavaTest desktop [p 134] is a single top-level window that contains all JavaTest windows opened to perform a task.

Use the JavaTest Preferences dialog box to select the MDI window style. See JavaTest Preferences.

## **Observer**

An optional class instantiated from the command line to observe a test run. The class implements a specific observer interface.

## **Parameters**

Values used to configure an agent. The agent parameters can be set at the time the agent is started or, if the agent GUI is used, from the Parameters tab after the agent GUI has started.

## **Passive Agent**

Agents that must wait for a request from the JavaTest harness before they can run tests.

The JavaTest harness initiates connections to passive agents as needed. Passive agents are simpler, but less flexible than active agents [p 133] because you must specify the test machine as part of the test configuration [p 133] , not at the time you run the tests. Passive agents do not allow you to run tests in parallel.

## **Port Number**

A number assigned to the JavaTest harness that is used to link specific incoming data to an appropriate service.

## **Prior Status**


A filter [p 138] used to restrict the set of tests in a test run based on the last test result information stored in the test result [p 138] files (.jtr).

Use the configuration editor to enable the Prior Status filter for a test run.

## **Progress Monitor**

A dialog box that displays detailed information about the current configuration of a test run. Information displayed in the Progress Monitor is not altered by view filter settings.

To display the Progress Monitor, do one of the following:

- Choose View > Progress Monitor from the Test Manager menu bar
- Click the  icon in the test status display



## **Report Directory**

The directory in which the JavaTest harness writes test reports.

The location of the report directory is set in the GUI or from the command line by the user when generating test reports.

## **SDI**

See Single Document Interface [p 137] .

## **Serial Agent**

Use serial mode (serial agent) when you want the agent to use an RS-232 serial connection. Serial agents wait for the JavaTest harness to initiate the connection. Infrared, parallel, USB, and firewire connections can also be added through the JavaTest API by modeling the existing serial system.

## **Single Document Interface**

A window style in which the JavaTest harness opens a console window and individual tool windows as separate top-level windows on an unbounded desktop. Use the JavaTest Preferences dialog box to select the SDI window style. See JavaTest Preferences.

## **Standard Values**

Values in a configuration that govern how the JavaTest harness runs the tests of a test suite. Standard values are a part of the configuration interview but can change from test run to test run. You can use the Standard Values view of the Configuration Editor to edit the standard values in the current configuration. See Current Configuration [p 134] .

## **System Properties**

Contains environment variables [p 134] from your system that are required to run the tests of a test suite [p 138] .

Because the JavaTest harness cannot directly access environment variables, you must use command-line options to copy them into the JavaTest harness system properties.

## **Tabbed**

A window style in which the JavaTest desktop [p 134] is a single top-level window that displays all JavaTest windows as tabbed panes.

Use the JavaTest Preferences dialog box to select the Tabbed window style. See JavaTest Preferences.

## **Test Description**

Machine readable information that describes a test to the JavaTest harness so that it can correctly process and run the related test. The actual form and type of test description depends on the attributes of the test suite. When using the JavaTest harness, the test description is a set of test-suite-specific name/values pairs in either HTML tables or Javadoc-style tags

Each test in a test suite [p 138] has a corresponding test description that is typically contained in an HTML file.

## **Test Environment**

A collection of values that provide information about how to run tests on a particular platform.

When a test in a test suite [p 138] is run, the JavaTest harness gives the script a test environment containing environment variables [p 134] that are derived from configuration data collected by the configuration editor. See configuration [p 133] .

Prior to the JavaTest 3.0 harness, the environment variables were read from an environment file. Use of environment files is deprecated, however, the JavaTest harness provides support for those test suites that use environment files. See environment file [p 134] .

## **Test Manager**

The JavaTest window used to configure, run, monitor, and manage tests from its panels, menus, and controls.

The Test Manager window is divided into two panes. It displays the folders and tests of a test suite in the tree pane on the left and provides information about the selected test or folder in the information panes on the right. A new Test Manager window is used for each test suite that is opened.

## **Test Result Files**

Contains all of the information gathered by the JavaTest harness during a test run.

The test result files (.jtr) are stored in a cache in the work directory [p 139] associated with the test suite.

You can view the test result files in a web browser configured to use the JavaTest ResultBrowser servlet.

## **Test Run Filters**

Include or exclude tests in a test run:

- Exclude lists [p 134]
- Keywords [p 136]
- Prior status [p 136]

Test run filters are set using the configuration editor or the parameter editor.

## **Test Script**

A script used by the JavaTest harness, responsible for running the tests and returning the status (pass, fail, error) to the harness. The test script must understand how to interpret the test description information returned to it by the test finder. The test script is a plug-in provided by the test suite. The Test Manager Properties dialog box lists the plug-ins that are provided by the test suite.

## **Test Suite**

A collection of tests, used in conjunction with the JavaTest harness to verify compliance of the licensee's implementation of the technology specifications.

A test suite must be associated with a work directory [p 139] before the JavaTest harness can run its tests.

## **Test Tree**

The hierarchical representation of the folders and tests in a test suite [p 138] .

The test tree is displayed in the Test Manager [p 138] window and uses colored status icons to indicate the test status of the folders and tests. Use view filters [p 139] to specify the folders and tests whose test status are displayed in the test tree.

## **Tests to Run**

Test folders and/or tests specified as a starting point in the test tree [p 138] for running tests.

Initial files are used to limit the test run to a specific branch of the test tree or to a specific test. The JavaTest harness walks the test tree starting at the folder and/or test and runs all tests (unless otherwise filtered out [p 135] ) that it finds.

Use the configuration editor or the parameter editor to modify the initial files listed in the test run configuration.

### **View Filters**

Include or exclude the results of tests in views of the test tree [p 138] . The following filters are provided in the combo box at the bottom of the test tree:

- Current Configuration [p 134]
- All Tests [p 133]
- Custom [p 134]

View filter settings do not include or exclude tests from a test run. Use the test run filter [p 138] settings to include or exclude tests from a test run.

### **Work Directory**

A directory associated with a specific test suite [p 138] and used by the JavaTest harness to store files containing information about the test suite and its tests.

Until a test suite is associated with a work directory, the JavaTest harness cannot run tests.

# 17. Index

---

## A

- active agents, troubleshooting [p 99]
- active host (active agent), setting [p 89]
- Agent Monitor [p 94]
- agent pool [p 95]
- agent pool [p 95]
- all tests (filter) [p 58]
- arrows, folder icons [p 45]
- arrows, test icons [p 46]
- audit tests, batch mode [p 111]
- audit tests, GUI [p 67]
- auto-start the agent [p 93]

## B

- batch commands, specifying [p 105]
- batch file [p 105]
- batch mode, running tests in [p 111]
- browse exclude list files [p 125]
- browse result files [p 125]
- browsing exclude list contents [p 32]
- browsing test environments [p 32]
- browsing test information [p 43]
- browsing test reports [p 64]
- button bar (same as toolbar) [p 10]

## C

- checklist, viewing [p 31]
- clearing test results, GUI [p 47]

- clearing test results, batch mode [p 108]
- colors, folder icons [p 45]
- colors, test icons [p 46]
- command-line interface [p 105]
- command-line options, using [p 112]
- concurrency [p 29]
- concurrency values (agents), setting [p 92]
- configuration editor, All Values [p 18]
- configuration editor, Standard Values [p 23]
- configuration editor, using the [p 17]
- Configuration pane [p 54]
- Configure menu [p 11]
- configuring test information [p 17]
- context sensitive help [p 15]
- creating a map file (agents) [p 103]
- creating a work directory [p 16]
- current configuration (filter) [p 58]
- custom filter [p 59]

## **D**

- desktop, JavaTest harness [p 71]
- dialog box, Exclude List [p 33]
- dialog box, Filter Editor [p 57]
- dialog box, New Report [p 63]
- dialog box, Test Environment [p 32]
- dialog box, Preferences [p 75]
- dialog box, Progress Monitor [p 38]
- dialog boxes, Welcome [p 5]
- display command-line help [p 115]
- displaying command-line help (agents) [p 92]

## **E**

- editJti [p 125]
- editLinks [p 130]
- editor, configuration [p 17]
- editor, filter [p 57]
- environment files [p 28]
- Error status pane [p 52]
- errors, testsuite [p 62]
- exclude list contents, browsing [p 32]
- exclude list files, browse [p 125]
- exclude list, initial [p 24]
- exclude list, latest [p 24]
- exclude list, other [p 24]
- exclude lists, specifying [p 24]

## **F**

- Failed status pane [p 50]
- File menu [p 73]
- file, specifying a map [p 92]
- files, batch [p 105]
- files, creating map [p 103]
- files, .jte [p 114]
- files, .jti [p 18]
- files, .jtp [p 114]
- files, .jtr [p 28]
- files, .jtx [p 125]
- Files pane [p 51]
- filter editor [p 59]
- folder icons [p 45]
- folder tabbed pane [p 50]

## **G**

- getting started [p 9]

## **H**

help, agent command-line [p 92]

help, context sensitive [p 15]

help, JavaTest online [p 7]

Help menu [p 75]

HTTP server, monitor results with the [p 121]

## **I**

icons, folder [p 45]

icons, test [p 46]

initial exclude list [p 24]

initial files [p 24]

interview, editing in the command-line [p 125]

interview, editing in the GUI [p 17]

interview, editing in batch mode [p 109]

## **J**

.jte files [p 114]

.jti files [p 18]

.jtp files [p 114]

.jtr files [p 28]

.jtx files [p 125]

## **K**

keywords [p 26]

## **L**

latest exclude list [p 24]

log, viewing the question [p 33]

## **M**

map files, specifying [p 92]

map file, creating a (agents) [p 103]

MDI window style [p 77]

memory (Test Monitor) [p 38]

- menu, Configure [p 11]
- menu, File [p 73]
- menu, Help [p 75]
- menu, Report [p 63]
- menu, Run Tests [p 12]
- menu, Tasks [p 74]
- menu, View [p 13]
- menu, Window [p 75]
- menus, Test Manager [p 10]
- messages, test run [p 55]
- Monitor, Agent [p 94]
- monitor results with the HTTP server [p 121]
- monitoring a test run [p 37]
- monitoring JavaTest Agents [p 94]
- Move report files [p 130]
- multiple configurations, working with [p 31]

## **N**

- Not Run tab [p 52]

## **O**

- Test Run Messages pane [p 55]
- opening a jti file, batch mode [p 107]
- opening a report [p 63]
- opening a work directory [p 15]
- other exclude list [p 24]

## **P**

- Passed status pane [p 51]
- passive agents, troubleshooting [p 99]
- plug-ins [p 61]
- pool, agent [p 94]
- popup menu, test tree [p 47]



port number (active agents), setting [p 89]

Preferences dialog box [p 75]

prior status [p 27]

progress meter [p 38]

Progress Monitor dialog box [p 38]

properties, Test Manager [p 61]

## **Q**

question log, viewing the [p 33]

quick pick test execution (test tree popup menu) [p 47]

## **R**

refresh tests (test tree popup menu) [p 47]

report files, move [p 130]

reports, browsing test [p 64]

resetting tests (test tree popup menu) [p 47]

Run Tests menu [p 35]

running tests [p 35]

running tests in batch mode [p 105]

## **S**

SDI window style [p 78]

set tracing (agents) [p 93]

set (batch mode) [p 110]

setting active host (active agents) [p 89]

setting concurrency (agents) [p 92]

setting tasks in history (agents) [p 93]

specifying a map file (agents) [p 92]

specifying batch commands [p 105]

starting a test run [p 36]

starting JavaTest [p 4]

status panes [p 51]

stopping a test run [p 40]

sub-total (Test Manager window) [p 50]

Summary pane (Test Manager window) [p 50]

## **T**

tabbed pane, folder [p 50]

tabbed pane, test [p 53]

tabbed window style [p 77]

tasks in history, setting (agents) [p 93]

Tasks menu [p 74]

Test Description pane [p 54]

test icons [p 46]

Test Manager [p 10]

Test Progress Display [p 38]

Test Run Details pane [p 55]

test suite, choosing [p 15]

test tabbed pane [p 53]

test tree pane [p 43]

tests, running [p 35]

time factor [p 29]

tool tips [p 76]

tracing agent activity, set [p 93]

tree, test [p 43]

Troubleshooting [p 132]

troubleshooting a test run [p 41]

troubleshooting active agents [p 99]

troubleshooting JavaTest Agents [p 99]

troubleshooting passive agents [p 100]

## **U**

user interfaces, JavaTest [p 71]

using the Configuration Editor [p 17]

## **V**

- view filter (Test Manager window) [p 57]
- view, folder (Test Manager window) [p 50]
- View menu [p 13]
- view, test (Test Manager window) [p 53]
- viewing test reports [p 64]

## **W**

- Welcome dialog boxes [p 5]
- Window menu [p 75]
- window styles [p 76]
- work directory, creating [p 16]
- work directory, opening [p 15]
- working with multiple configurations [p 31]
- writereport (batch mode) [p 111]

## **X,Y,Z**