JavaTest™ Harness User's Guide

Command-Line Interface

JavaTest Harness, 3.2.2_01

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, California 95054
U.S.A. 1-650-960-1300

August 2005

Please
Recycle

Contents

Preface

This manual describes how to use the JavaTest harness command-line interface to run tests of the test suite, write reports, and audit test results. This User's Guide is a PDF version of the JavaTest command-line interface online help. It is provided in PDF format so that users can conveniently view and print the contents of the command-line interface online help without starting the JavaTest harness.

There are minor structural differences between the online help and the PDF document although the basic contents are the same. For example, the contents of the online help have been resequenced to be more useful in book format; in the PDF format, page references embedded in the text are hypertext links in the online help; and, extensive online help navigation links have been removed from the PDF format.

The *JavaTest User's Guide: Command-Line Interface* is one of two User's Guides that the JavaTest harness provides for users, the *JavaTest User's Guide: Graphical User Interface and the JavaTest User's Guide: Command-Line Interface.*

If your test suite provides the JavaTest agent for use in running tests, the *JavaTest Agent Users' Guide* may also be included.

Before You Read This Book

In order to fully use the information in this document, you must have thorough knowledge of the topics discussed in your TCK documentation.

How This Book Is Organized

Chapter 1 describes the features of the JavaTest command-line interface provided by the JavaTest harness.

Chapter 2 describes the basic topics that the user should be familiar with before using the JavaTest command-line interface.

Chapter 3 provides a description of the types of commands and command formats used in the JavaTest command-line interface.

Chapter 4 describes the commands used to setup and modify a configuration used by the JavaTest harness.

Chapter 5 describes the commands used to perform tests from the command line.

Chapter 6 describes commands used to specify the properties of the JavaTest GUI.

Chapter 7 describes the information commands used display JavaTest online information without starting the JavaTest GUI.

Chapter 8 describes the legacy commands that JavaTest supports.

Chapter 9 describes the various special utilites provided by the JavaTest harness.

Chapter 10 provides a basic troubleshooting guide.

Using UNIX® Commands

This document may not contain information on basic UNIX® commands and procedures such as shutting down the system, booting the system, and configuring devices.

See one or more of the following for this information:

- *Solaris Handbook for Sun Peripherals*
- AnswerBook2™ online documentation for the Solaris™ operating environment
- Other software documentation that you received with your system

Typographic Conventions

This User's Guide uses the following typographic conventions:

Typeface	Meaning	Examples
`AaBbCc123`	The names of commands, files, and directories; on-screen computer output	Edit your `.login` file. Use `ls -a` to list all files. `% You have mail.`
`AaBbCc123`	What you type, when contrasted with on-screen computer output	`% su` `Password:`
AaBbCc123	Book titles, new words or terms, words to be emphasized	Read Chapter 6 in the *User's Guide*. These are called *class* options. You *must* be superuser to do this.
	Command-line variable; replace with a real name or value	To delete a file, type `rm` *filename*.

Shell Prompts

Examples in this User's Guide might contain the following shell prompts:

Shell	Prompt
C shell	*machine_name*`%`
C shell superuser	*machine_name*`#`
Bourne shell and Korn shell	`$`
Bourne shell and Korn shell superuser	`#`

Related Documentation

The following documentation provides additional detailed information about the JavaTest harness:

Application	Title
JavaTest GUI	*JavaTest Harness User's Guide; Graphical User Interface*
JavaTest Agent (optional)	*JavaTest Agent User's Guide*

Accessing Sun Documentation Online

The Java Developer Connection™ program web site enables you to access Java platform technical documentation at `http://java.sun.com/`.

Sun Welcomes Your Comments

We are interested in improving our documentation and welcome your comments and suggestions. Provide feedback to Sun at `http://java.sun.com/docs/forms/sendusmail.html`.

1

What is the JavaTest Harness Command-Line Interface?

The JavaTest harness is a powerful test harness that provides two interfaces, a GUI and a command-line interface. The command-line interface provides test harness functionality for configuring and executing tests as well as creating reports without using the GUI. This allows you to use the JavaTest harness to run tests in build scripts and other automated processes. See the *JavaTest Harness User's Guide: Graphical User Interface* for a description of the JavaTest harness GUI.

JavaTest Harness Command-Line Interface Features

Features of the command-line interface include the following capabilities:

- Enables configurable testing - Runs 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 tests, compiler tests, and regression tests).
- Runs tests on small systems - The JavaTest harness supports the use of 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.
- Generates HTML reports - Summarize test runs. HTML reports for the test run can be generated from the command line
- Audits test runs - Can be performed from the command line
- Provides Web server to monitor test runs in batch mode - Use to monitor and control test progress while tests are running from the command line.
- Provides extensive 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 from the command line without starting the JavaTest harness GUI.

For the New JavaTest Harness User

The JavaTest harness is designed to run test programs on a wide-variety of Java platforms. To run the tests, the JavaTest harness uses a configuration file that contains the required information about how the tests are run on a particular test platform. The JavaTest harness uses a specific work directory to contain all of the results from running the tests of a testsuite. See the Glossary for detailed descriptions of the terms **test suite**, **work directory**, and **configuration file**.

Providing Configuration Information

The JavaTest harness uses the GUI Configuration Editor to collect configuration information (both test environment and parameter values) in a single configuration file. While it is possible for existing test suites to use environment files (`.jte`) in the Configuration Editor, parameter files (`.jtp`) are no longer used. The JavaTest harness uses the Configuration Editor to include the parameter values with the values of the `.jte` file in a single configuration file (`.jti`).

For backwards compatibility, older test suites can continue to use environment and parameter files in the command line. See the Glossary for detailed descriptions of the terms `.jte` file, `.jtp` file, and `.jti` file.

Online Documentation and Context Sensitive Help

The JavaTest harness provides extensive online documentation that is available from the command line. To see the available command-line options, type the following at a system prompt:

```
java -jar test_suite/lib/JavaTest harness.jar -help
```

Note – Include the path of the directory where the `javatest.jar` file is installed (represented as `[test_suite/]` in the example). The `javatest.jar` file is usually installed in the test suite `lib` directory when the JavaTest harness is bundled with a test suite.

See Information Commands for detailed information about searching for and displaying command-line information without starting the JavaTest harness GUI.

2

Before Starting the JavaTest Harness

Before you start the JavaTest harness on a test system and run tests, you must have a valid test suite and Java Development Kit 1.4 or later installed on your test system. See your test suite documentation for information about installing the test suite and the JavaTest harness on your test system. Refer to `http://java.sun.com/products` for information about installing the current Java Development Kit on your test system.

You must also understand how your test group uses or intends to use the JavaTest harness in its test system. For example consider the following questions about your test groups use of the JavaTest harness:

- Does your test group use standard configuration files and templates from a central location, or does it use individual configuration files customized for each test run?
- Does your test group use the JavaTest harness and one or more agents to run distributed tests?
- Does your test group run the JavaTest harness from a central location or from local installations in the test system?

If you use the JavaTest harness agent to run tests, you must also install the JavaTest harness agent on the platform being tested. See *JavaTest harness Agent User's Guide* for detailed information about installing the JavaTest harness agent on a test platform.

3

Command-Line Summary

You can use commands in the command line or as a part of a product build process to configure the harness, run tests, write test reports, audit test results, and start the GUI using specific configuration values.

The JavaTest harness executes the commands from left to right in the sequence that they appear in the command string. Include commands in the command string as though you were writing a script. The JavaTest harness does not restrict either the number of commands or the groups of commands that you can use in a command string.

> javatest [*Setup Commands*] [*Task Commands*] [*Desktop Commands*] [*Information Commands*]

The commands are included as a formatted set in the following sequence:

1. Setup Commands - required by task commands to set values used for the test run and to set specific values used when performing other tasks. Set-up commands must precede the task or desktop commands. Setup commands can be used to set specific values (without a task command) when starting the GUI.
2. Task Commands - required to run tests, to write reports, and to audit tests. Task commands require one or more preceding set-up commands.
3. Desktop Commands - use in place of the task commands to start the GUI with a new desktop or to specify status colors used in the GUI. Set-up commands are optional when using Desktop commands.
4. Information Commands - use information commands to display command-line help, online help, or version information without starting the harness. Information commands do not require any other commands on the command line.

For additional information about using the command-line interface, see the following topics:

- About Command-Line Examples
- Formatting a Command
- Using Command Files
- Index of Available Commands

About the Command-Line Examples

This section provides many examples of command-line operations in the following basic sequence:

> *javatest* [*Set-up Commands*] [*Task Commands*]

In the examples, the following spresentations are used:

- > represents the command prompt. For Unix systems the command prompt may be either a shell prompt, such as %, or a user defined value. For win32 systems, the command prompt may be **c:** or another appropriate drive identifier.
- *javatest* represents the command or commands that your test suite would use to start the JavaTest harness. You should start the JavaTest harness from the root directory of the test suite.

See Setup Commands for commands and examples used to set up or change specific values in a configuration.

See Task Commands for commands and examples used to perform tasks from the command line.

Formatting a Command

You can use any one of the following formats to include commands on the command line:

- Command Options Format
- Single String Arguments Format
- Command File Format

All formats are used to accomplish the same tasks. Use the format that you prefer or that is easier to use. See Index of Available Commands for a complete listing of available commands.

Command Options Format

In the command options format, commands are preceded by "-," act as options, and do not use command terminators. Enclose complex command arguments in quotes. This format is recommended when long lists of commands are included in a command line.

Example:

```
> JavaTest harness -open default.jti -runtests
```

Single String Arguments Format

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

Example:

> *javatest* `"open` *default*`.jti; runtests"`

Command File Format

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

In the command file format, a file containing a series of commands and their arguments is included in the command line by preceding the file name with the "@" symbol.

Example:

> *javatest* `@`*mycmd*`.jtb -runtests`

Refer to Using Command Files for detailed information about using and creating command files.

Using Command Files

A command file is a text file that contains one or more commands used by the JavaTest harness from the command line or as a part of a product build process. You can place combinations of configuration settings and commands in the command file and use it to repeatedly perform the following actions:

- Perform test runs
- Write test reports
- Audit test results

The advantage of using the command file format is that it is easy to use a complex, persistent, repeatable set of commands in a command line.

The commands used in a command file are a formatted set of commands, executed in the sequence that they appear in the command string. Use the commands in the command file as you would if you were writing a script. See Formatting a Command for a description of the formats you can use.

Creating a Command File

Use the single string arguments format style to write commands in a text file. See Formatting a Command for detailed information.

Command files can contain blank lines and comments as well as lines with commands and their arguments. The following table describes the contents of a command file.

TABLE 1 Command File Contents

File Contents	Description
Comments	Comments may begin anywhere on a line, are started by the # symbol, and stop at the end of the line. Example: `#File contains commands`
Commands	Commands are executed in the sequence that they appear in the file (for example, set-up commands must precede task commands). Commands used in the file must be separated by a semicolon (;) or a new line symbol (#). The # symbol acts as a new line character and can terminate a command. Examples: `open` *default*`.jti; #opens file` `-set host` *mymachine*
Command Arguments	Arguments that contain white space must be placed inside quotes. Use \ to escape special characters such as quotes (" ") and backslashes (\).

After writing the commands, save the text file by using a descriptive name and the extension `.jtb`. The file name should help you identify the function of each command file.

Examples of Using Command Files

In the following examples, a command file (*mycommandfile*`.jtb`) is used to override the `localHostName`value and the tests specified in the existing configuration.

The following three examples are provided:

- Example Command File Contents
- Command Line Using the Example Command File
- Changing Values After the Example Command File is Set

Note – If you attempt to run these examples, you must replace *mytestsuite*`.ts`, *myworkdir*`.wd`, and *myconfig*`.jti` with test suite, work directory, and `.jti` names that exist on your system. You must also modify the contents of the example command file for your configuration file and test suite. Win32 users must change `/` file separators to `\` to run these examples.

Example Command File Contents

The contents of the example command file, *mycommandfile*`.jtb`, are:

```
#File sets localHostName and tests
set jck.env.runtime.net.localHostName mymachine;
tests api/javax_swing api/java_awt
```

Note – The `-set` and `-tests`command forms are not used in the command file. Command files only use the "Single String Arguments Format."

See Setting Specific Configuration Values for additional examples of using the `set`command. See Specifying Tests to Run for additional examples of using the `tests`command.

Note – See About the Command-Line Examples for a description of the use of > *javatest* in the following example. See Command-Line Overview for a description of the command line structure. See Formatting a Command for descriptions and examples of the following command formats.

Command Line Using the Example Command File

In the following examples, a test suite (*mytestsuite*`.ts`), work directory (*myworkdir*`.wd`), and configuration file (*myconfig*`.jti`) are opened, and the command file (*mycommandfile*`.jtb`) is read and executed before running tests.

Command Options Format Example

> *javatest* `-open` *myconfig*`.jti @`*mycommandfile*`.jtb -runtests`

Single String Arguments Format Example

> *javatest* `"open` *myconfig*`.jti; @`*mycommandfile*`.jtb; runtests"`

Changing Values After the Example Command File is Set

You can also change values after the command file is set:

Command Options Format Example

> *javatest* `-open` *myconfig*`.jti @`*mycommandfile*`.jtb -exclude` *myexcludelist*`.jtx -runtests`

Single String Arguments Format Example

> *javatest* `"open` *myconfig*`.jti; @`*mycommandfile*`.jtb; exclude` *myexcludelist*`.jtx; runtests"`

Formating Configuration Values for `editJTI` or `-set`

The following table indentifies the types of questions supported by the JavaTest harness configuration interview and a description of the format of required to set the value in the command line.

TABLE 2 Types and Values of jti Questions

Question Type	Description and Example of Format
Integer questions	Example: `set` *mytck*`.port 5000` Note: Localized values can be used in the value. For example, 5,000 is acceptable in a US locale.
Floating point questions	Example: `set` *mytck*`.delay 5.0` Note: The value is evaluated using the current locale (e.g. European locales should enter 5,0).
String questions	Example: `set` *mytck*`.url http://`*machine/item*
String ListQuestion	Newline-separated list of values.
File Question	Example: `set` *mytck.file1 /tmp/bundle*`.jar`

TABLE 2 Types and Values of jti Questions

File List Question	If none of the filenames have embedded spaces, you can give a space-separated list of filenames. If any of the filenames in the list have embedded spaces, use a newline character to terminate or separate all of the filenames.
StringListQuestion	See String ListQuestion above.
Choice Question	Example: `set` *mytck*`.cipher 3DES` Note: The value supplied is case sensetive. This type of question appears in the Configuration Editor as a set of radio buttons or single-selection list of choices.
ChoiceArrayQuestion	Note: This type of question is rendered in the Configuration Editor as either a list that accepts multiple selections or a series of checkboxes.
InetAddressQuestion	The standard textual representation of the IP Address, as defined by Internet Engineering Task Force (IETF).

YesNoQuestion	Examples:
	`set` *mytck*`.needStatus Yes`
	`set` *mytck*`.needStatus No`
	
	Note: The values are case sensitive, no is not acceptable.

Using Newlines Inside Strings

When setting values of configuration questions in the command line, The internal command parser will accept newlines inside strings if they are preceded by a backslash.

Depending on the shell you use, this may or may not be possible to enter directly on the command line.

If you want to set values with embedded newlines, create a JavaTest harness batch command file, and put the set commands (and any other commands) in that file. In the batch file, you can enter strings with embedded escaped newlines, as in the following example:

```
# switch on verbose mode for commands
verbose:commands
# open a jti file
open /home/user1/tmp/idemo.11mar04.jti
# set a list of files
```

```
set demo.file.simpleFileList /tmp/aaa\
/tmp/bbb\
/tmp/ccc
# set a list of strings
set demo.stringList 111\
2222\
3333
```

On Solaris, using the Korn shell, you can simply put newline characters into strings and the right thing will happen.

Example:

```
$JAVA \
-jar image/lib/javatest.jar \
-verbose:commands \
-open /home/user1/tmp/idemo.11mar04.jti \
-set demo.file.simpleFileList /tmp/aaa
/tmp/bbb
/tmp/ccc
```

Extended Command-Line Examples

This section provides extended examples of command-line operations.

To use the following examples on your system, you must use class paths and directory names appropriate for your system.

Example 1

```
java -jar lib/javatest.jar -verbose -testSuite /tmp/myts \
-workdir -create /tmp/myworkdir -config /tmp/my.jti \
-runtests -writereport /tmp/report
```

This combination of commands does the following, in this order:

1. Tells the harness to be verbose during test execution
2. Opens the test suite `/tmp/`*myts*
3. Creates a work directory named `/tmp/`*myworkdir*
4. Uses *my*`.jti` as the configuration settings
5. Executes the tests (as specified by the configuration)
6. Writes a report to `/tmp/report/` after test execution

Example 2

```
java -jar lib/javatest.jar -startHttp -testsuite /tmp/myts \
-workdirectory /tmp/myworkdir -config /tmp/my.jti \
-runtests -writereport /tmp/report -set tck.foo.bar 4096 \
-runtests -writereport /tmp/report1
```

This combination of commands does the following, in this order:

1. Tells the harness to start the internal HTTP server
2. Opens the test suite `/tmp/`*myts*
3. Uses a work directory named `/tmp/`*myworkdir*
4. Uses *my*`.jti` as the configuration settings
5. Executes the tests (as specified by the configuration)
6. Writes a report to `/tmp/report/` after test execution
7. Changes a configuration value (not written to JTI file)
8. Runs tests again
9. Writes a new report in `/tmp/`*report1*

Example 3

```
java -cp lib/javatest.jar:lib/comm.jar \
com.sun.javatest.tool.Main \
-Especial.tck.value=lib/special.txt \
-agentPoolPort 1944 -startAgentPool "testsuite /tmp/myts ; \
workdir /tmp/myworkdir ; config myconfig.jti ; runtests"
```

This combination mixes two styles of command line arguments (quoted and dash-style). It invokes the harness by class name, rather than executing the JAR (`-jar`) file. An extra item is added to the JVM's classpath. The following commands are given to the harness:

1. Sets a particular value in the testing environment (TCK-specific)
2. Specifies the agent pool port and starts the agent pool
3. Loads the test suite `/tmp/`*myts*
4. Opens the work directory `/tmp/`*myworkdir*
5. Uses the configuration in *myconfig*`.jti`
6. Runs the tests

Example 4

```
java -jar lib/javatest.jar -config foo.jti -runtests
```

This relies on information in the JTI file to do the run. Specifically, it tries to use the work directory and test suite locations specified in the JTI file. If either of those are invalid or missing, the harness reports an error. Otherwise, if the configuration (in the JTI) is complete, the tests are run.

Example 5

```
java -jar lib/javatest.jar -config foo.jti -verbose \
-set tck.val1 2002 -runtests
```

This is the same as Example 4 with the exception that it turns on verbose mode and changes the answer of one of the questions in the configuration.

Example 6

```
java -jar lib/javatest.jar -config foo.jti \
-priorStatus fail,error -timeoutFactor 0.1 \
-set tck.needColor Yes \
-set tck.color1 orange -tests api/java_util1 -runtests
```

This example extends Example 4 by setting various Standard Values and the answer to particular configuration questions.

Example 7

Example for starting the GUI:

```
java -jar lib/javatest.jar -testsuite /tmp/foo.jti myts \
-workdirectory /tmp/mywd -config /tmp/myconfig.jti
```

This combination of commands does the following, in this order:

1. Opens the specified test suite
2. Opens the work directory given (assuming it is a work directory)
3. Opens the configuration file given
4. Starts the GUI since no execution action is given

Index of Available Commands

The following table describes all of the commands that can be used in command mode.

TABLE 3 Index of Available Commands

Command	Description
`concurrency`	Specifies the number of tests that are run concurrently.
`env`	Specifies a test environment in an environment file.
`envFile` or `envFiles`	Specifies an environment file (.jte) containing test environments.
`excludeList`	Specifies an exclude list file.
`keywords`	Restricts the set of tests to be run based on keywords.
`open`	Opens a test suite, work directory, or a configuration `.jti` file.
`params`	This command is deprecated.
`priorStatus`	Selects the tests included in a test run based on their outcome on a prior test run.
`set`	Overrides a specified value in a configuration (`.jti`) file.
`tests`	Creates a list of test directories and/or tests to run.
`testSuite`	Specifies the test suite.
`timeoutFactor`	Increases the amount of time the JavaTest harness waits for a test to complete
`workDir` or `workDirectory`	Opens an exiting work directory, creates a new work directory, or replaces an existing work directory with a new work directory.
`runTests`	Runs tests in command mode.
`audit`	Audits test results in command mode.
`writeReport`	Writes test reports in command mode.

4

Setup Commands

Before you can perform a task from the command line, you must first use setup commands to specify a configuration.

After setting up a configuration, you can then modify the values in the configuration for your specific requirements. These changed values override but do not change values in the configuration file. You can use configuration templates from a central resource to run tests on different test platforms and configurations.

Note – See About the Command-Line Examples for a description of use of > *javatest* in the following example.

The setup commands are used in the following sequence in the command line:

> *javatest* [*Initial Setup Commands*] [*Set Specific Values*] [*Additional Setup Commands*] [*Task Commands*]

Setting specific values and additional setup commands are optional.

The task command at the end of the example is also optional. If a task command is not included, the JavaTest harness uses the specified configuration and any changes set on the command line to open the GUI.

For additional information about using setup commands see the following topics:

- Initial Setup Commands
- Setting Specific Values
- Additional Setup Commands

Initial Set-up Commands

Before you can perform tasks from the command line, you must first set up a configuration for the JavaTest harness to use. You can set up a configuration by performing *at least one* of the following:

1. Specify an existing configuration (`.jti`) file. You are not required to specify either a test suite or a work directory.
2. Specify an existing work directory and a configuration file. You are not required to specify a test suite.
3. Open a test suite, create an empty work directory, and specify a configuration file.

After setting up a configuration, you can then change specific values for your specific requirements. See Setting Specific Values for the commands used to modify the values in the configuration.

You can include commands in any combination on the command line provided the intital set-up commands are specified before any other commands in the command line.

You can use any of the following commands to set up a configuration for the JavaTest harness to use when performing tasks:

- `config` - Used to specify an existing configuration file. See Specifying a Configuration for a detailed description of this command.
- `workDirectory` or `workDir` - Used to specify an existing work directory or to create a new work directory. See Specifying a Work Directory for a detailed description of this command.
- `testSuite` - Used to specify a test suite. See Specifying a Test Suite for a detailed description of this command.
- `open` - Used to specify a test suite, work directory, configuration file, or parameter file. See Specifying a Test Suite, Work Directory or Configuration for a detailed description of this command.

Specifying a Test Suite (`testsuite`)

To specify the test suite, use the `testsuite` command:

> *javatest* ... `-testsuite` *path/filename* [*work directory command*] [*configuration command*] ... [*task command*] ...

See About the Command-Line Examples for a description of the use of > *javatest*.

When you want to specify a test suite, include the commands in the following sequence:

1. Specify the test suite (`testsuite` *path/filename*)
2. Set up a configuration
3. Include a Task Command such as `runtests` (optional).

See Command-Line Overview for a description of the command line structure.

Detailed Example of testsuite Command

In the following example, *mytestsuite*, *myworkdir*`.wd`, and *myconfig*`.jti` represent file names that might exist on your system.

Command Options Format Example:

> *javatest* `-testsuite` *mytestsuite* `-workdir` *myworkdir*`.wd -config` *myconfig*`.jti -runtests`

See Formatting a Command for descriptions and examples of other command formats that you can use.

Specifying a Work Directory (`workdir` or `workdirectory`)

Each work directory is associated with a test suite and stores its test result files in a cache. You can use the work directory command to:

- Use an Existing Work Directory
- Create a New Work Directory
- Replace an Existing Work Directory

See Shortcuts to Initialize a Configuration for information about specifying a work directory in the command line.

Use an Existing Work Directory

To use an existing work directory for the test run, include either the `workdir` or `workdirectory` command in the command line:

> *javatest* ... `-workdir` *path/filename* ... [*task command*] ...

See About the Command-Line Examples for a description of the use of > *javatest* in the example.

See Command-Line Overview for a description of the command line structure.

See Formatting a Command for descriptions and examples of the following command formats.

Create a New Work Directory

To create a new work directory for the test run, use the `-create` command option:

> *javatest* ... `-workdir -create` *path/filename* [*configuration command*] ... [*task command*] ...

See About the Command-Line Examples for a description of use of > *javatest* in the example.

The new work directory must not previously exist. You can also use an existing work directory as a template to create a new work directory for the test run. To use an existing work directory as a template, put the template in the command line before the `create` command.

When creating the command string, include the commands in the following sequence:

1. Specify the test suite (optional)
2. Specify an existing work directory (optional)
3. Include the `workdir` or `workdirectory -create` *path/filename* command
4. Specify a configuration file
5. Set specific values (optional)
6. Include the `runtests` command (optional). The results of the test run are written to the new work directory.

See Command-Line Overview for a description of the command line structure.

Detailed Example of Creating a New Work Directory

In the following example, *myworkdir*`.wd` and *myconfig*`.jti` represent file names that might exist on your system.

Command Options Format Example:

> *javatest* `-workdir` *myworkdir*`.wd -create` *testrun*`.wd -config` *myconfig*`.jti -runtests`

When the tests are run, the JavaTest harness uses the work directory (*testrun*`.wd`) created by the command line, even if the configuration file (*myconfig*`.jti`) was created using another work directory.

See Formatting a Command for descriptions and examples of other command formats that you can use.

Replace an Existing 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.

> *javatest* ... `-workdir -overwrite` *path/filename* ... [*task command*] ...

or

> *javatest* ... `-workdir -create -overwrite` *path/filename* ... [*task command*] ...

The `-create` command option is optional when the `-overwrite` command is used.

See Command-Line Overview for a description of the command line structure.

Detailed Example of Replacing an Existing Work Directory

In the following example, *myconfig*`.jti` represents a configuration file name that might exist on your system.

Command Options Format Example:

> *javatest* `-workdir -overwrite` *testrun*`.wd -config` *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.

See Formatting a Command for descriptions and examples of other command formats that you can use.

Specifying a Configuration File (`config`)

To specify the configuration file the the JavaTest harness uses to run tests, use the `config`command.

> *javatest* ... `-config` *path/filename* ... [*task command*] ...

See About the Command-Line Examples for a description of > *javatest* in the example.

The configuration file may contain default values for the test suite (which contains the tests to be run) and the work directory (which is where to put the results).

Test suite and work directory values in the configuration file can be overridden with the `testsuite` and `workdirectory` commands. If the configuration file is a template and does not contain default values for the test suite and work directory, those values must be specified explicitly with the testsuite and workdirectory commands.

See Command-Line Overview for a description of the command line structure.

Detailed Example of config Command

In the following example, *myconfig*`.jti` represents a configuration file name that might exist on your system.

Command Options Format Example:

> *javatest* `-config` *myconfig*`.jti -runtests`

See Formatting a Command for descriptions and examples of other command formats that you can use.

Specifying a Test Suite, Work Directory or Configuration (`open`)

To specify a test suite, work directory, or a configuration `.jti` file, use the `open` command:

... `open` *path/filename* ...

Note – See About the Command-Line Examples for a description of the use of > *javatest* in the following example. See Command-Line Overview for a description of the command line structure. See Formatting a Command for descriptions and examples of the following command formats.

Command Options Example:

> *javatest* ... `-open` *path/filename* ... [*task command*] ...

Single String Arguments Example:

> *javatest* ... ; `open` *path/filename* ; ... [*task command*] ...

Command File Example:

> *javatest* `@`*mycmd*`.jtb` ... [*task command*] ...

In addition to any other commands, for this example the *mycmd*`.jtb` command file must contain the command:
"`open` *path/filename;*"

Refer to Using Command Files for detailed information about creating and using command files.

Setting Specific Values

After you set a configuration (see Initial Set-up Commands) you can specify individual values for a test run that override those in the configuration file.

Note – Values that you specify in the command string override but do not change the values specified in the configuration file.

You can use the following commands to specify individual values for a test run:

- `set` - used to set any value in a configuration file. See Setting Test Suite Specific Values for a detailed description of this command.
- `concurrency` - used to change the concurrency value set in the configuration file. See Setting Concurrency (concurrency) for a detailed description of this command.
- `envFile` - used to specify or change the environment file name set in the configuration file. This option only applies to legacy test suites that use environment (.jte) files. See Specifying an Environment File (envFile or envFiles) for a detailed description of this command.
- `envFiles` - used to specify or change the environment file names set in the configuration file. This option only applies to legacy test suites that use environment (.jte) files. See Specifying an Environment File (envFile or envFiles) for a detailed description of this command.
- `env` - used to specify or change the environment name set in the configuration file. This option only applies to legacy test suites that use environment (.jte) files. See Specifying a Test Environment (env) for a detailed description of this command.
- `excludeList` - used to specify or change the exclude list set in the configuration file. See Specifying Exclude List Files (excludeList) for a detailed description of this command.

- `keywords` - used to specify or change the keyword values set in the configuration file. See Using Keywords (keywords) for a detailed description of this command.
- `priorStatus` - used to specify or change prior status values set in the configuration file. See Selecting Tests Based on Previous Results (priorStatus) for a detailed description of this command.
- `tests` - used to specify or change the tests specified in the configuration file. See Specifying Tests or Directories to Run (tests) for a detailed description of this command.
- `timeoutFactor` - used to specify or change the test timeout value specified in the configuration file. See Increasing the Timeout (timeoutFactor) for a detailed description of this command.

Obtaining the Question Tag-Name

The following three ways can be used to obtain a configuration question *tag-name*:

- **Question Tag** - Start the JavaTest harness GUI, open the configuration editor, and load the configuration file used to run tests. Choose View **->** Question Tag in the configuration editor menu bar. The configuration editor displays the *tag-name* at the bottom of the question pane. Navigate through the configuration until you locate the question whose value must be changed. Use the question *tag-name* in the command line.
- **Control and T** - Start the JavaTest harness GUI, open the configuration editor, and load the configuration file used to run tests. Click on text in question pane and then press the Control and T keys. The configuration editor displays the *tag-name* at the bottom of the pane. Navigate through the configuration until you locate the question whose value must be changed. Use the *tag-name* in the command line.
- **Question Log** - Start the JavaTest harness GUI 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 configuration and its value.
- **Report Question Log** - Start the JavaTest harness GUI and choose Create Report in the Test Manager menu bar. Check the Question Log option to generate a Question Log of the current configuration. View the report and click the Confioguration link. The Question Log displays the *tag-name* for each question in the configuration and its value.

Setting Specific Configuration Values

You can use the `set`command to override a specific value in the current configuration (`.jti`) file:

> *javatest ...* [*initial set-up commands*] *...* `-set` *question-tag-name ...* [*task command*] *...*

See About the Command-Line Examples for a description of use of > *javatest* in the example.

You can also use `-set -file` *input-file-name* or `-set -f` *input-file-name* to import a Java properties file containing the values of multiple configuration questions. A hand edited configuration file can be used as an input file.

> *javatest* ... [*initial set-up commands*] ... `-set -file` *input-file-name* ... [*task command*] ...

The JavaTest harness uses the values in the input file to override the values in the configuration. Any values in the input file that are not used in the configuration are ignored.

Values changed by the `set`command are only used for the session and override but do not change the configuration file. To change a configuration file, use the Configuration Editor window provided by the JavaTest harness GUI.

When creating a command string to set specific values in a configuration, include the commands in the following sequence:

1. Set up a configuration
2. Specify configuration values (`set` *question-tag-name value*)
3. Include the `runtests`command (optional).

See Command-Line Overview for a description of the command line structure.

To use the `set`command, you must identify the *question-tag-name* associated with the *value* in the configuration file that you are changing. In the command line, following the `set`command, enter the *question-tag-name* and its new *value*:

A value can only be changed if its *tag-name* exists in the initialized configuration file. If the configuration does not include the *tag-name* you must use the Configuration Editor in the JavaTest harness GUI to include the question and value in the configuration file.

See Obtaining the Question tag-name for detailed information about the *tag-name* for the question. See Formating Configuration Values for editJTI or -set for detailed information about formatting the values. See Detailed Examples for examples of using the `set` command and the *tag-name*.

Detailed Example of Setting Test Suite Specific Values

In the following example, *myconfig*`.jti` represents a file name that might exist on your system.

Command Options Example:

> *javatest* `-config` *myconfig*`.jti -set jckdate.gmtOffset 8 -runtests`

See Formatting a Command for descriptions and examples of other command formats.

Setting Concurrency (`concurrency`)

If you are running the tests on a multi-processor computer, you can use concurrency to speed up your test runs. Use the `concurrency`command to specify the number of tests to run concurrently:

> *javatest* ... [*initial set-up commands*] ... `-concurrency` *number* ... [*task command*] ...

See About the Command-Line Examples for a description of the use of > *javatest* in the example.

Unless your test suite restricts concurrency, the maximum *number* of threads specified by the `concurrency` command is 50. See your test suite documentation for additional information about using concurrency values greater than 1.

When creating a command string to specify the number of tests to run concurrently, include the commands in the following sequence:

1. Set up a configuration
2. Specify the concurrency value (`concurrency` *number*)
3. Include the `runtests` command (optional).

See Command-Line Overview for a description of the command line structure.

Detailed Example of concurrency Command

In the following example, *myconfig*`.jti` represents a file name that might exist on your system and value represents a numeric value from 1 to 50 that you might use.

Command Options Format Example:

> *javatest* `-config` *myconfig*`.jti -concurrency` *value* `-runtests`

See Formatting a Command for descriptions and examples of other command formats.

Specifying an Environment File [deprecated]

This command only applies to older test suites that use environment files (`.jte`) instead of configuration (`.jti`) files. Use the `envFile` command to specify an environment file (`.jte`) containing test environments that the JavaTest harness must use to run the test suite:

> *javatest* ... [*initial set-up commands*] ... `-envFile` *path/filename* ... [*task command*] ...

See About the Command-Line Examples for a description of the use of > *javatest* in the following example.

When creating a command string to specify an environment file, include the commands in the following sequence:

1. Set up a configuration
2. Specify an environment file (`envFile` *path/filename*)
3. Include the `runtests` command (optional).

See Command-Line Overview for a description of the command line structure.

Detailed Example of envFile Command

In the following example, *myenvFile*`.jte` represents a file name that might exist on your system.

Command Options Format Example:

> *javatest* `-envFile` *myenvFile*`.jte -runtests`

See Formatting a Command for descriptions and examples of other command formats that you can use.

Specifying a Test Environment [deprecated]

This command is only used for test suites with environment (`.jte`) files containing multiple environments. You can use the `env` command with the `envFiles` command to specify a specific test environment contained in an environment file:

> *javatest* ... [*initial set-up commands*] ... `-envFile` *path/filename* `-env` *environment-name* ... [*task command*] ...

See About the Command-Line Examples for a description of the use of > *javatest* in the following example.

When creating a command string to specify a specific test environment, include the commands in the following sequence:

1. Set up a configuration
2. Specify a test environment (`env` *environment-name*)
3. Include the `runtests` command (optional).

See Command-Line Overview for a description of the command line structure.

Detailed Example of envFile Command

In the following example, *path/filename* represents a file name that might exist on your system and *environment-name* represents an environment name that exists in the environment file.

Command Options Format Example:

> *javatest* `-envFile` *path/filename* `-env` *environment-name* `-runtests`

See Formatting a Command for descriptions and examples of other command formats that you can use.

Specifying Exclude List Files (`excludeList`)

Test suites normally supply exclude list files which contain the list of tests that the harness is not required to run. Exclude list files conventionally use a `.jtx` extension. Once you have set up a configuration, you can use an `excludeList` command to specify the exclude list for your test run:

> *javatest* ... [*initial set-up commands*] ... `-excludeList` *path/filename* ... [*task command*] ...

The `-excludeList` *path/filename* command can be used multiple times to specify multiple exclude lists for a test run.

See About the Command-Line Examples for a description of the use of > *javatest*.

The exclude list that you specify in the command line overrides any exclude list specified in the configuration file without changing the configuration file.

When you want to specify an exclude list, include the commands in the following sequence:

1. Set up a configuration
2. Specify the exclude list (`excludeList` *path/filename*)
3. Include a Task Command such as `runtests`(optional).

See Command-Line Overview for a detailed description of the command line structure.

Detailed Example of excludeList Command

In the following example, *myconfig*`.jti` and *myexcludelist*`.jtx` represent file names that might exist on your system.

Command Options Format Example:

> *javatest* `-config` *myconfig*`.jti -exclude` *myexcludelist*`.jtx -runtests`

See Formatting a Command for descriptions and examples of other command formats that you can use.

Using Keywords (`keywords`)

The test suite may provide keywords that you can use on the command line to restrict the set of tests to be run. Use the `keyword` command to specify the keywords used to filter the tests that are run.

> *javatest* ... [*initial set-up commands*] ... `-keywords` *expression* ... [*task command*] ...

See About the Command-Line Examples for a description of the use of > *javatest* in the following example.

Refer to the test suite documentation for a list of supported keywords.

When creating a command string that specifies keywords, include the commands in the following sequence:

1. Set up a configuration
2. Specify keywords used (`keywords` *expression*)
3. Include a Task Command such as `runtests` (optional).

See Command-Line Overview for a detailed description of the command line structure.

Detailed Example of keywords Command

In the following example, *myconfig*`.jti` and *myexcludelist*`.jtx` represent file names that might exist on your system.

Command Options Format Example:

> *javatest* `-config` *myconfig*`.jti -keywords interactive -runtests`

See Formatting a Command for descriptions and examples of other command formats that you can use.

Selecting Tests Based on Previous Results (`priorStatus`)

Tests can be selected for a test run based on their prior test status. Use the `priorStatus` command to run tests based on their results from a previous test run:

> *javatest* ... [*initial set-up commands*] ... `-priorStatus fail,error` ... [*task command*] ...

See About the Command-Line Examples for a description of the use of > *javatest* in the example.

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.

When creating a command string to specify the prior test status, include the commands in the following sequence:

1. Set up a configuration
2. Specify the prior test status (`priorStatus` *status-arguments*)
3. Include a Task Command such as `runtests` (optional).

See Command-Line Overview for a detailed description of the command line structure.

Detailed Example of priorStatus Command

In the following example, *myconfig*`.jti` represents a configuration file name that might exist on your system.

Command Options Format Example:

> *javatest* `-config` *myconfig*`.jti -priorStatus fail,error -runtests`

See Formatting a Command for descriptions and examples of other command formats that you can use.

Specifying Tests or Directories to Run (`tests`)

You can specify one or more individual tests or directories 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 then executes all tests that it finds (unless they are filtered out).

You can use the `tests` command to specify one or more individual tests or directories of tests:

`tests` *path/filename*

When creating a command string, include the commands in the following sequence:

1. Set up a configuration
2. Specify tests or directories of tests (`tests` *path/filename*)
3. Include the `runtests` command (optional).

See About the Command-Line Examples for a description of the use of > *javatest*.

Example of tests Command

In the following example, *path/filename* represents a file name that might exist on your system.

Command Options Format Example:

> *javatest* ... [*initial set-up commands*] ... `-tests` *path/filename* ... [*task command*] ...

See Formatting a Command for descriptions and examples of other command formats that you can use.

Increasing the Timeout (`timeoutFactor`)

Each test in a test suite has a timeout limit. The JavaTest harness waits for a test to complete for the duration of that limit before moving on to the next test. You can use the `timeoutFactor` command to change the timeout limit:

> *javatest* ... [*initial set-up commands*] ... `-timeoutFactor` *number* ... [*task command*] ...

See About the Command-Line Examples for a description of the use of > *javatest* in the example.

Each test's timeout limit is multiplied by the time factor value. For example, if you specify a value of 2.0, the timeout limit for tests with a 10 basic time limit becomes 20 minutes. See Formatting a Command for descriptions of the command formats. Note that the format of the value input for the timeout factor is dependant on the locale.

When creating a command string to change the timeout limit, include the commands in the following sequence:

1. Set up a configuration
2. Specify the timeout limit (`timeoutFactor` *number*)
3. Include the `runtests` command (optional).

See Command-Line Overview for a detailed description of the command-line structure.

Detailed Example of timeFactor Command

In the following example, *myconfig*`.jti` and *myexcludelist*`.jtx` represent file names that might exist on your system.

Command Options Format Example:

```
> javatest -config myconfig.jti  -timeoutFactor  2.0  -runtests
```

See Formatting a Command for descriptions and examples of other command formats that you can use.

Additional Setup Commands

In most cases, you use the command line to perform functions that are also available through the GUI. However, you can also use the command line to specify how the JavaTest harness starts.

When starting the JavaTest harness, you can include additional commands in the command line to:

- Include all system properties in test execution environments.
- Set an environment variable that you want inherited in every test environment.
- Set the agent pool port number.
- Set the agent pool timeout.
- Start the active agent pool.

The JavaTest harness uses a new desktop when you include GUI commands in the command line.

The following table describes the commands used in the command line to specify how the JavaTest harness starts.

TABLE 4 JavaTest Harness Commands

Command	Function
`-EsysProps`	Includes all system properties in test execution environments.
`-E`*name=value*	Sets an environment variable that is inherited in every test environment created. The `-E`*name=value* command 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.
`-agentPoolPort` *port*	Set the Agent Pool Port Number Use this command only when you are configuring the JavaTest harness and the agent to use a port other than 1907.
`-agentPoolTimeout` #*seconds*	Set the Agent Pool Timeout 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 from the command line.
`-startAgentPool`	Start the Active Agent Pool If you use an active agent and run the JavaTest harness from the command line, you must add `-startAgentPool`to the command string to start the Agent Pool.

5

Task Commands

In the command line, after setting up a configuration, you can include commands to perform tasks such as run tests, write reports, and audit tests. See Set-up Commands for detailed information about setting up a configuration.

Information about the following task commands can be found in the following topics:

- Running Tests
- Writing Reports
- Auditing Tests

Running Tests (`runtests`)

Use the `runtests` command to run the tests specified in the configuration.

> *javatest* [*Monitor option*] [*Setup commands*] ... `-runtests` ...

See About the Command-Line Examples for a description of the use of > *javatest* in the example.

You can also use the `runtests` command as part of a sophisticated command sequence that resembles and functions as a script. You can include command files and multiple commands in the same command string to programmatically perform repeated, multiple test runs of different configurations without starting the JavaTest harness GUI.

See Using Command Files for detailed information about creating and using command files.

A *Monitor option* can be set in the command line to display test progress information during the test run. See Monitor Test Progress Option for detailed information about setting this option.

When creating the command string to run one or more tests, include the commands in the following sequence:

1. Monitor Test Progress Option (optional) use the command line to monitor a test run.
2. Set up a configuration (required) set the specific values required to run the tests.
3. Include the `runtests` command.

See Command-Line Overview for a detailed description of the command-line structure.

Detailed Example of runtests Command

In the following example, *myconfig*`.jti` represents a configuration file name that might exist on your system.

Command Options format example:

> *javatest* `-config` *myconfig*`.jti -runtests`

See Formatting a Command for descriptions and examples of other command formats that you can use.

Monitor Test Progress (`verbose`)

Including the `verbose` command and optional monitoring options in a run command allows the user to monitor test progress from the command line. This command uses `stdout` to display the specified levels of monitoring test run progress. This monitoring function is not available in the GUI.

Monitoring Options

The monitoring options are specified in the command line as a comma-separated list following the `-verbose` option. A colon is used to separate the `-verbose` command from the options. Ordering and capitalization within the list are ignored. Whitespace within the list is prohibited.

If you do not specify a level, the `progress` option is automatically used.

> *javatest* `-verbose:` *Monitor option* [*Setup commands*] ... `-runtests` ...

See About the Command-Line Examples for a description of the use of > *javatest* in the example.

See Examples of Monitoring Output for detailed examples of the command line.

The following table describes monitoring options specified in the command line.

TABLE 5 Monitoring Options

Option	Description
`commands`	Traces the individual JavaTest harness commands as they are executed. This includes options given on the command line, commands given in command strings, and commands given in command files.
`no-date`	Do not prefix entries with the data and time stamp. Normally, each logical line of output will print the month, day, hour, minute and second.
`non-pass`	Print non-passing (error, fail) test names and their status string. The status string includes the status (error, fail) and the reason for the failure/error.
`max`	Output the maximum possible amount of output. This includes all the options which are individually available. If this option is present, only the `no-date`and `quiet`flags have any additional effect.
`quiet`	Suppress any output from the verbose system. It may be useful to temporarily deactivate monitoring while debugging, without removing other levels coded into a script. `-verbose:stop,progress,quiet` results in no output, as does `-verbose:quiet,stop,progress`. This option takes precedence over all other options. It does not suppress the pass/fail/error statistics printed at the end of the test run.
`start`	Prints the test name when it goes into the the harness' engine for execution. Note: On some test suites, this may only indicate that the test has been handed to the plug-in framework, not that it is actually executing.
`stop`	Prints the test name and status string (see `non-pass`) when a test result is received by the harness.
`progress`	Prints a progress summary, which indicates pass/fail/error/not-run numbers. If any of the `max, non-pass, stop,` or `stop` options were specified, each summary may be printed on it's own line. If not, the summary will be updated on the current line. The progress information is printed/updated each time a test result is reported to the harness.

Detailed Examples of Monitoring Commands

The following are examples of monitoring commands and their resulting command line output.

1. An example of the default monitoring output:

```
> java -jar lib/javatest.jar -verbose -open foo.jti -runtests
  14:21:31 Sept 14 - Harness starting test run with configuration
"foo".
  14:24:33 Sept 14 - Pass: 12  Fail: 0  Error: 1  Not-Run: 33
  14:24:30 Sept 14 - Finished executing all tests, wait for
```

```
cleanup...
  14:26:31 Sept 14 - Harness finished test run.
```

2. An example of the `start` monitoring output:

```
> java -jar lib/javatest.jar -verbose:start -open foo.jti -
runtests
  14:21:31 Sept 14 - Harness starting test run with configuration
"foo".
  14:24:39 Sept 14 - Running foo/bar/index#id1
  14:24:30 Sept 14 - Test run stopped, due to failures, errors,
user request. Wait for cleanup...
  14:26:31 Sept 14 - Harness finished test run.
```

3. An example of the `start` and `stop` monitoring output:

```
> java -jar lib/javatest.jar -verbose:start,stop -open foo.jti -
runtests
  14:21:31 Sept 14 - Harness starting test run with configuration
"foo".
  14:24:31 Sept 14 - Running foo/bar/index#id1
  14:24:32 Sept 14 - Finished foo/bar/index#id1 Fail.  Invalid
index did not throw exception.
  14:26:33 Sept 14 - Running foo/bar/index#id2
  14:27:34 Sept 14 - Finished foo/bar/index#id2 Pass.
  14:28:35 Sept 14 - Running foo/bar/index#id3
  14:29:36 Sept 14 - Finished foo/bar/index#id3 Error.  Cannot
invoke JVM.
  14:30:30 Sept 14 - Finished executing all tests, wait for
cleanup...
  14:30:31 Sept 14 - Harness finished test run.
```

4. An example of the `no-date`, `start`, and `stop` monitoring output:

```
> java -jar lib/javatest.jar -verbose:no-date,start,stop -open
foo.jti -runtests
  Harness starting test run with configuration "foo".
  Running foo/bar/index#id1
  Finished foo/bar/index#id1 Fail.  Invalid index did not throw
exception.
  Running foo/bar/index#id2
  Finished foo/bar/index#id2 Pass.
  Running foo/bar/index#id3
  Finished foo/bar/index#id3 Error.  Cannot invoke JVM.
  Test run stopped, due to failures, errors, user request. Wait
```

```
for cleanup...
  Harness finished test run.
```

5. An example of the `non-pass` monitoring output:

```
> java -jar lib/javatest.jar -verbose:non-pass -open foo.jti -
runtests
  Harness starting test run with configuration "foo".
  Running foo/bar/index#id1
  Finished foo/bar/index#id1 Fail.  Invalid index did not throw
exception.
  Running foo/bar/index#id2
  Finished foo/bar/index#id2 Pass.
  Test run stopped, due to failures, errors, user request. Wait
for cleanup...
  Harness finished test run.
```

6. An example of the `progress` and `non-pass` monitoring output:

```
> java -jar lib/javatest.jar -verbose:progress,non-pass -open
foo.jti -runtests
  14:23:39 Sept 14 - Harness starting test run with configuration
"foo".
  14:24:39 Sept 14 - Pass: 12  Fail: 0  Error: 0  Not-Run: 33
  14:25:32 Sept 14 - Finished foo/bar/index#id1 Fail.  Invalid
index did not throw exception.
  14:26:39 Sept 14 - Pass: 12  Fail: 1  Error: 0  Not-Run: 32
  14:27:39 Sept 14 - Pass: 12  Fail: 1  Error: 0  Not-Run: 32
  14:30:36 Sept 14 - Finished foo/bar/index#id3 Error.  Cannot
invoke JVM.
  14:32:39 Sept 14 - Pass: 12  Fail: 1  Error: 1  Not-Run: 31
  14:33:01 Sept 14 - Test run stopped, due to failures, errors,
user request. Wait for cleanup...
  14:33:10 Sept 14 - Harness finished test run.
```

7. An example of the `no-date` and `max` monitoring output:

```
> java -jar lib/javatest.jar -verbose:no-date,max -open foo.jti
-runtests
  Harness starting test run with configuration "foo".
  Running foo/bar/index#id1
  Finished foo/bar/index#id1 Fail.  Invalid index did not throw
exception.
  Pass: 0  Fail: 1  Error: 0  Not-Run: 33
  Running foo/bar/index#id2
  Finished foo/bar/index#id2 Pass.
  Pass: 1  Fail: 1  Error: 0  Not-Run: 32
  Test run stopped, due to failures, errors, user request. Wait
```

```
for cleanup...
  Harness finished test run.
```

Batch (`batch`)

The `batch` command is a legacy command that is used to run tests from the command line or as part of a build process. If a task command is not included in the command line, the JavaTest harness begins running tests automatically. The `batch` command has been superceeded by the `runTests` command.

The `batch` command is also used in the command line to close the JavaTest harness when all commands have been processed. If the `batch`command is used, the JavaTest harness GUI will not start unless explicitly started by another commands in the command string.

If the GUI is started in batch mode (such as by using the `monitorAgent` command), the JavaTest harness displays a dialog after all commands are executed that allows the user to cancel the automatic shutdown and to use the JavaTest harness GUI in normal mode.

In its legacy format, the `batch` command was required to precede the other commands. In the present format, the `batch` command can be specified in any location on the command line.

> *javatest* ... `-batch` ... [*Setup commands*] ... [*task command*] ...

See About the Command-Line Examples for a description of use of > *JavaTest harness* in the example.

See Command-Line Overview for a description of the command-line structure.

Detailed Example of batch Command

In the following example, *myconfig*`.jti` represents a configuration file name that might exist on your system.

Command Options Format Example:

> *javatest* `-batch -config` *myconfig*`.jti`

See Formatting a Command for descriptions and examples of other command formats that you can use.

Observer (`observer`)

The `observer` command is an advanced command that allows you to register `com.sun.javatest.Harness.Observer` for monitoring a test test run. For example, an observer can monitor the progress of each test run and implement custom behavior such as sending email if a test in a test run fails. See the API documentation for details or contact the harness development team for help in using an observer.

Writing Reports (writereport)

Use the -writereport command in the command line as a separate command or as part of a series of task commands (such as run tests and audit test results). Use a web browser to view the reports.

Because the JavaTest harness executes commands in their command-line sequence, you must identify the work directory before the `-writereport` command and provide the report directory as an option after the command:

> *javatest* ... `-workdir` *mywork-directory* `-writereport` *myreport-directory*

See About the Command-Line Examples for a description of the use of > *javatest* in the example.

See Command-Line Overview for a detailed description of the command line structure.

Detailed Example of writereport Command

In the following example, *myworkdirectory* represents a work directory name that might exist on your system.

Command Options Format Example:

> *javatest* `-workdirectory` *myworkdirectory* `-writereport` *myreport-directory*

See Formatting a Command for descriptions and examples of other command formats that you can use.

Auditing Tests

You can audit test results by using `-audit` as a separate command or as part of a series of task commands (such as to run tests and write test reports). The results of the audit are sent to the terminal.

Because the JavaTest harness executes commands in their command-line sequence, you must identify the work directory before the `-audit` command.

> *javatest* ... `-workdirectory` *mywork-directory* `-audit`

See About the Command-Line Examples for a description of the use of > *javatest* in the example.

See Command-Line Overview for a description of the command line structure.

Detailed Example of audit Command

In the following example, *mywork-directory* represents a work directory name that might exist on your system.

Command Options Format Example:

> *javatest* `-workdirectory` *mywork-directory* `-audit`

See Formatting a Command for descriptions and examples of other command formats that you can use.

6

Desktop Commands

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 perform the following tasks:

- Use a new desktop when starting the JavaTest harness GUI
- Specify Status Colors if you prefer to use colors in the GUI that are different from the defaults

Using a New JavaTest harness Desktop

When starting the JavaTest harness, include `-newDesktop` in the command string to start the JavaTest harness GUI without using a previous desktop. The JavaTest harness ignores any previous desktop information and opens the JavaTest harness Quick Start wizard.

Note – The JavaTest harness uses a new desktop when you include GUI options in the command line. Using this option preserves any preferences set for the desktop. Use the following example to start the JavaTest harness with a new Desktop.

```
> javatest -newDesktop
```

See About the Command-Line Examples for a description of the use of > *javatest*.

Specifying Status Colors

The JavaTest harness allows you to specify the status colors used in the GUI. This property is set on the command line as a system property when starting the JavaTest harness GUI. Status colors set this way are added to the user preferences and restored in subsequent sessions.

The user can specify each status color by using system properties in the following format:

`-Djavatest.color.passed=`*color-value* ...

`-Djavatest.color.failed=`*color-value* ...

`-Djavatest.color.error=`*color-value* ...

`-Djavatest.color.notrun=`*color-value* ...

`-Djavatest.color.filter=`*color-value* ...

The *color-value* used must be an RGB value parsable by the `java.awt.Color` class (octal, decimal or hex).

The value portion of the color property must be explicitly defined. The value portion of the property accepts hex values, prefixed by either a pound character (`#`) or a zero-x (`0x`).

Values can also be specified in octal, in which case the value begins with a leading zero and must be two or more digits.

The following are possible formats for setting color integers:
`#ffaa66` (hex)
`0xffaa66` (hex)
`0111177` (octal)

Detailed Example of Specifying a Status Color

You might need to escape the pound character in hte following example for the command to work on your platform.

7

Information Commands

The JavaTest harness command-line interface allows you to display command-line help, online help, or version information without starting the harness.

> *javatest* [*Information Command*]

Including an information command at the end of the command line causes the JavaTest harness to display JavaTest harness information without starting the GUI.

The following topics provide detailed information about the commands that can be used to display JavaTest harness information:

- Displaying JavaTest Harness Command-Line Help
- Displaying JavaTest Harness Online Help
- Display JavaTest Harness Version Information

Command-Line Help

The JavaTest harness allows you to display the command-line interface in the following forms:

- All information
- Topic information
- Word search information

All Information

To display all of the information in command-line help, include `-help all` at the end of the command line.

> *javatest* `-help all`

See About the Command-Line Examples for a description of the use of > *javatest*.

Topic Information

To display only the command-line help for specific topics, include `-help` and the name of the topic at the end of the command line.

> *javatest* `-help` *topic name*

See About the Command-Line Examples for a description of the use of > *JavaTest harness*.

The following table lists the available command-line help topics.

TABLE 6 Options Used to Display Command-Line Information

Topic	Function
Desktop	Displays information about the command-line options for starting the JavaTest harness graphical user interface.
JavaTest harness Agent	Displays information about the command-line options for the JavaTest harness Agent.
Batch Mode	Displays information about the command-line options for running tests in batch mode.
Configuration	Displays information about the command-line options for setting up and changing a configuration.
Environment	Displays information about the command-line options for adding values into JavaTest harness environments.
HTTP server	Displays information about the command-line options for the JavaTest harness HTTP server.
Options	Displays information about the command-line options accepted on the command line.
Files	Displays information about the types of files accepted as command-line arguments.

Display the List of Available Topics

To display the list of help topics provided by command-line help, include `-help`, `-usage`, or `-?` at the end of the command line.

> *javatest* `-help`

See About the Command-Line Examples for a description of the use of > *JavaTest harness*.

Word Search Information

The JavaTest harness allows you to search the full command-line help for a specific word or phrase and then displays only the information containing that word or phrase.

To display command-line help information containing a specific word or phrase, include `-help` and the word or phrase at the end of the command line.

> *javatest* `-help` *word or phrase*

See About the Command-Line Examples for a description of the use of > *javatest*.

> *javatest* `-help audit test`

The console displays the following output:

```
 Audit    Options for auditing test results
     -audit [-showEnvValues] [-showMultipleEnvValues]
         Audit the test results defined in the current
configuration


 For complete details and examples, see the JavaTest
harness online help. You can access
 this directly from the command line with "-onlineHelp
...", or you can
 start the JavaTest harness and use the Help menu. The
online help is also
 available in PDF format in the JavaTest harness
documentation directory.


 Copyright (C) 2005 Sun Microsystems, Inc. All rights
reserved.
 Use is subject to license terms.
```

JavaTest harness Version Information

To display the version, location, and build information of the installed copy of the JavaTest harness, include the `-version` command at the end of the command line.

> *javatest* `-version`

See About the Command-Line Examples for a description of the use of > *javatest*.

Displaying JavaTest harness Online Help

Without opening the JavaTest harness GUI, the JavaTest harness allows you to display JavaTest harness online help. To display the JavaTest harness online help, include the `-onlinehelp` command at the end of the command line.

```
> javatest -onlinehelp
```

See About the Command-Line Examples for a description of the use of > *javatest*.

8

Legacy Commands

The JavaTest harness command-line interface supports the commands used in previous versions of the JavaTest harness. In most cases using the current commands is preferred, however, if you are running a test suite that uses a parameter file you can continue to use options in the command line to specify parameter values. These commands are deprecated and may be removed from future versions of the JavaTest harness.

See Using Parameter Commands for a detailed description of these Legacy Commands.

Using Parameter Commands (`params`) [deprecated]

If you are running a test suite that does not use a parameter file, use the current commands instead of the `-params` command and option.

If you are running a test suite that uses a parameter file (`.jtp`), you can specify different parameter values by including `-params` and the appropriate parameter command in the command line.

> *javatest* ... `-params` [*command*] [*value*] [*Task Commands*] ...

See About the Command-Line Examples for a description of the use of > *javatest*.

The following table describes the parameter commands.

TABLE 7 Parameter Commands

Command	Description
`-t` *testsuite* or `-testsuite` *testsuite*	Specifies the test suite.
`-keywords` *keyword-expr*	Restricts the set of tests to be run based on keywords associated with tests in the test suite.
`-status` *status-expr*	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.
`-exclude` *exclude-list-file*	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 `.jtx`extension and are normally supplied with a test suite.
`-envFile` *environment-file*	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.
`-env` *environment*	Specifies a test environment from an environment file.
`-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.
`-timeoutFactor` *number*	Increases the timeout limit by specifying a value in the time factor option. 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. For example, if you specify a value of 2, the timeout limit for tests with a 10 basic time limit becomes 20 minutes.
`-r` *report-directory* or `-report` *report-directory*	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.
`-w` *work-directory* or `-workDir` *workDirectory*	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.

9

Utilities

The JavaTest harness allows you to use additional utilities to remotely monitor and control a test run, browse result (`.jtr`) files without starting the harness, browse exclude list files without starting the harness, change responses in a configuration file without starting the harness, and move test reports.

Information about performing tasks by using additional utilities can be found in the following topics:

- Monitoring Results with HTTP Server
- Browsing Result (`.jtr`) Files
- Browsing Exclude List Files
- Changing Configuration Values with EditJTI
- Changing Configuration Values with a Text Editor
- Moving Test Reports

Monitoring Results with HTTP Server

The JavaTest harness provides a web server that you can use to remotely monitor and control a test run. The HTTP Server provides the following two types of output:

- HTML Formatted Output for users to remotely monitor batch mode test runs in a web browser.
- Plain Text Output for use by automated testing frameworks.

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 test runs that are not executing as expected. The following topics describe the HTML formatted output:

- Server Index Page
- Server Harness Page
- Server Test Result Index Page
- Harness Environment Page
- Harness Interview Page
- Stop a Test Run

Accessing HTTP Server HTML Formatted Output

1. Use the following command on the command line to activate the web server.

   > *javatest* `-startHttp -runTests` *[options]*

   See About the Command-Line Examples for a description of the use of > *javatest*.

2. Copy the URL reported to the console.

   Example:
   ```
   JavaTest harness HTTPd - Success, active on port 1903
   JavaTest harness 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/
   ```

Displaying the 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.

Displaying HTTP Server Harness Page

When the JavaTest harness is running tests, the harness page displays the following information:

- 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 harness is running) and provides the JavaTest harness 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
```

Displaying the HTTP Server Test Result Index Page

The Test Result Index page displays the following information:

- 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.

Displaying 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
```

Displaying 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, 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.

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.

The following two types of JavaTest harness information can be accessed by automated testing frameworks:

- Version Information
- Harness Information

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 harness 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 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 harness 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
```

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 harness `ResultBrowser` servlet:

```
com.sun.javatest.servlets.ResultBrowser
```

Refer to your server documentation for information about configuring it to use the JavaTest harness `ResultBrowser` servlet. Typically, you configure the web server to direct `.jtr` files to the servlet for rendering.

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 harness `ExcludeBrowser` servlet.

```
com.sun.javatest.servlets.ExcludeBrowser
```

Refer to your server documentation for information about configuring it to use the JavaTest harness `ExcludeBrowser` servlet. Typically, you configure the web server to direct `.jtx` files to the servlet for rendering.

Changing Configuration Values With EditJTI

The JavaTest harness provides the `EditJTI` utility for you to use in changing the values in a configuration file from the command line. You can also make changes in a configuration by using the appropriate set command (see Command-Line Summary for detailed information).

While you can use `EditJTI` to change the order of commands in a configuration file, the dependencies between questions can introduce errors into the the configuration. You should use the configuration editor window in the JavaTest harness GUI when making major changes in a configuration.

If your changes to a configuration introduce errors, you can use the JavaTest harness GUI configuration editor window to troubleshoot and repair the configuration.

EditJTI Command Format

The `EditJTI` command loads a configuration (`.jti`) file, and applies a series of changes specified on the command line. See Formating Configuration Values for editJTI or -set for detailed information about formatting the values.

You can save the changes in the original configuration file or save the changes in a new configuration file. You can also use `EditJTI` 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. 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*

OPTIONS

The following are the available options:

`-help`, `-usage` or `/?`

Displays a summary of the command line options.

`-classpath` *classpath* or `-cp` *classpath*

Overrides 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 is specified by this option.

`-log` *log-file* or `-l` *log-file*

Generates an HTML log containing the questions and responses from the configuration file. The log is generated after edits are applied.

`-out` *out-file* or `-o` *out-file*

Specifies where to write the configuration file after the edits (if any) are applied. The default setting is to overwrite the input file if the interview is edited.

`-path` or `-p`

Generates a summary to the console output stream of the sequence of questions and responses from the configuration file. The summary is generated after edits are applied.

`-preview` or `-n`

Does not write out any files, but instead, preview what would happen if this option were not specified.

`-testsuite` *test-suite* or `-ts` *test-suite*

Overrides 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` or `-v`

Verbose mode. As the edit commands are executed, details of the changes are written to the console output stream.

COMMANDS

Two different types of commands are supported.

tag-name=value

Sets the value for the question whose tag is *tag-name* to *value*. It is an error if the question is not 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*.

/search-string/replace-string/

Scans 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 use `/`in the search string, you use some other character (instead of `/`) as a delimiter.
For example:
|*search-string*|*replace-string*|
Regular expressions are not currently supported in *search-string*, but may be supported in a future release.

Depending on the shell in use, quote the commands to protect characters in them from interpretation by the shell.

RETURN CODE

The following table describes the return codes generated when a program exits.

TABLE 8 Exit Return Codes

Code	Description
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 configuration file. The default is 2. A value of 0 disables backups.

Changing Configuration Values

When using `EditJTI` to change the values in a configuration, you can use either of the following command formats:

- Use *tag=value* for direct replacement of values. You must know the *tag-name* for the question that sets the value.
- Use */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 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/|"
```

Note – To run the following examples of changing configuration values, you must replace *myoriginal*`.jti` with a `.jti` file name that exists on your system. Win32 users must also replace the `/` file separators with `\` file separators to run these examples.

Example:

`java -cp [`*jt_dir*`/lib/]javatest.jar com.sun.javatest.EditJTI -o` *mynew*`.jti "|/java/jdk/1.3/|/java/jck/1.4/|"` *myoriginal*`.jti`

Generating 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 [`*jt_dir* `/lib/] javatest.jar com.sun.javatest.EditJTI -o` *mynew*`.jti -l` *myeditlog*`.html "|/java/jdk/1.3/|/java/jck/1.4/|"` *myoriginal*`.jti`

Preview Without Change

You can use the -n option to preview but not perform updates to the `jti` file.

Example:
`java -cp [`*jt_dir*`/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 [`*jt_dir* `/lib/] javatest.jar com.sun.javatest.EditJTI -n -v -o `*mynewl*`.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 [`*jt_dir*`/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 [`*jt_dir* `/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, the interview will be invalid and the tests cannot be run.

Generally, you should not use `EditJTI` to change the interview path, but only the values on the existing path. If you are in doubt about the current interview path, open the configuration editor window in the JavaTest harness GUI and use it to change the values. The configuration editor window displays the current interview path for that question name/value pair.

Change the HTTP Port

You can use Edit JTI to change the HTTP port and either overwrite the original configuration file or create a new configuration file. Examples of both are provided below.

Note – To run the following examples, 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 for detailed information about the *tag-name* for the question.

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).

`java -cp [`*jt_dir*`/lib/] javatest.jar com.sun.javatest.EditJTI httpPort=8081` *myoriginal*`.jti`

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.

`java -cp [`*jt_dir*`/lib/] javatest.jar com.sun.javatest.EditJTI -o` *myoutput*`.jti httpPort=8081` *myoriginal*`.jti`

Doing Escapes in a Unix Shell

The following example uses the syntax for doing escapes in a Unix shell. Changes to the original configuration file (*myoriginal*`.jti` in this example) are written to a new configuration file (*my-newconfig*`.jti` in this example).

In the following example, *myoriginal*`.jti` represents a configuration file name that might exist on your system. Win32 users must also replace the `\` file separators with `/` 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 for detailed information about viewing the *tag-name* for the question.

```
java -cp [jt_dir/lib/] javatest.jar com.sun.javatest.EditJTI -o
my-newconfig.jti tck.serialport.midPort=/dev/term/a
tck.connection.httpsCert="\"CN=<Somebody>, OU=<People>,
O=<Organisation>, L=<Location>, ST=<State>,
C=US\""myoriginal.jti
```

10

Changing Configuration Values with a Text Editor

The JavaTest harness allows you to use a text editor from within a script (such as `sed`) to change responses in a configuration file and then launch the JavaTest harness to run tests.

The configuration file is a standard JavaTest harness properties file in which double backslashes and escaped new lines are required. If you edit this file in a text editor, you must also remove the checksum for JavaTest harness to accept it when running tests.

Checksums are used by the JavaTest harness to ensure that a configuration used to run tests is complete. By removing the checksum, you risk introducing errors in the configuration used to run tests.

We recommend that you test your changes in the configuration editor window before applying them in a text editor. The configuration editor checks the value and displays the correct set of related questions. See Configuring a Test Run in the *JavaTest Harness User's Guide: Graphical User Interface* for detailed information about the configuration editor window.

The relationship between the questions in a configuration depends on the test suite and the interdependance of the questions. A change in the value of one question may change subsequent, related configuration questions and values. If your response changes the set of required configuration values, the configuration editor window displays the incomplete configuration and provides you with a new set of required configuration questions.

After you have tested your changes and are satisfied with the results, you can use the text editor to apply them to the configuration. Remove the checksum from the configuration file before using the changed configuration to run tests.

Moving Test Reports

JavaTest harness reports contain relative and fixed links to files that break when the test reports are moved. To prevent this, the JavaTest harness provides an `EditLinks` command-line utility in the main JavaTest harness jar file, `javatest.jar`, for you to use when moving JavaTest harness reports.

The `EditLinks` utility checks all files with names ending in `.html` for HTML links beginning with file names you specified in the `EditLinks` command. These links are rewritten using the corresponding replacement name from the `EditLinks` command and are copied to the new location. `EditLinks` copies all other files to the new location without change.

Format of the EditLinks Command

Example:
`java -classpath` [*jt_dir*`/lib/`] `javatest.jar com.sun.javatest.EditLinks` *OPTIONS file...*

OPTIONS

The available *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. The edit is effectively transparent when the file is viewed in a browser. Multiple `-e`options can 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 following table describes the return codes that the program displays when it exits.

TABLE 9 Exit Return Codes

Code	Description
0	The copy was successful.
1	There was a problem with the command line arguments.
2	There was a problem with the command line arguments.
3	An error occurred while performing the copy.

Detailed Example of EditLinks Command

In the following example, *test12_dir.wd* and *myworkdir.wd* represent file names that might exist on your system. Win32 users must also replace the \ file separators with / to run these examples.

`java -cp [`*jt_dir*`/lib/] javatest.jar com.sun.javatest.EditLinks -e /work/ /java/jck-dev/scratch/12Jun00/jck-lab3/ -o test12_dir.wd` *myworkdir.wd*

11

Troubleshooting

The JavaTest harness provides information in the following topics that you can use to troubleshoot problems:

- JavaTest Harness Exit Codes
- Using the JavaTest Harness
- Running Tests
- Writing Reports
- Moving Reports

JavaTest Harness Exit Codes

When the JavaTest harness exits, it displays an exit code that you can use to determine the exit state. The following table contains a detailed description of the exit codes.

TABLE 10 JavaTest Harness Exit Codes

Exit Code	Description
0	If tests were executed, all tests passed.
1	One or more tests were executed and failed.
2	One or more tests were executed and had errors.
3	There was a problem with the command-line arguments.
4	JavaTest harness internal error.

Problems Using the JavaTest Harness

If the JavaTest harness fails, you can use the `harness.trace` file in your work directory to troubleshoot the problem. The `harness.trace` file is a plain-text file that contains a log of JavaTest harness activities during the test run. It is written in the work directory, is incrementally updated, and is intended primarily as a log of JavaTest harness activity.

Problems 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. See Troubleshooting a Test Run in the *JavaTest Harness User's Guide: Graphical User Interface* for information about the resources that the JavaTest harness provides for troubleshooting.

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 GUI Test tabbed panes and configuration editor window to help determine the change required in the configuration.

The following is an example of how the GUI Test Manager tabbed panes and the configuration editor window can be used to identify and correct a configuration error:

1. Use the test tree to identify the folder(s) containing test(s) that have errors.
2. Click the folder icon to open its Summary tab in the Test Manager window.
3. Click the Error tab to display the list of tests in the folder that has errors.
4. Double-click a test in the list to display it in the test tree and view its detailed test information.

5. Click the Test Run Messages tab to display detailed messages describing what happened during the running of each section of the test. The contents of each output section vary 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.
6. Click the Configuration tab to display a two column table of the name/value pairs that were derived from the configuration file and used to run the test. The names in the table identify test environment properties used by the JavaTest harness to run the test. The values displayed were used to run the test. Refer to your test suite documentation for detailed descriptions of the name/value pairs for your test.
7. Choose Configure > Show Question Log to view the Question Log of the current, saved configuration. Use the question log to identify the configuration value that is incorrect and its configuration question.
8. Press F2, choose Configure > Change Configuration from the menu bar, or click the button on the tool bar to open the configuration editor window.
9. Search the configuration file for the specific characters or character strings that must be changed.
10. Click the Done button to save your changes to the configuration file and rerun the tests.

Tests that Fail

Tests that fail are tests that were executed but had failing results. The test or the implementation may have errors.

The following is an example of how the GUI Test Manager tabbed panes can be used to identify and correct a test failure:

1. Use the test tree to identify the folder(s) containing test(s) that had errors.
2. Click the folder icon to open its Summary tab in the Test Manager window.
3. Click the Error tab to display the list of tests in the folder that had errors.
4. Double-click a test in the list to display it in the test tree and view its detailed test information.
5. Click the Test Run Messages tab to display detailed messages describing what happened during the running of each section of the test. The contents of each output section vary 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.

Problems Viewing Reports

The JavaTest harness does not automatically generate reports of test results after a test run. You must generate test reports either from the command line or from the JavaTest harness GUI.

Problems Writing Reports

You use filters to write test reports for a specific set of test criteria. See Creating Reports in the *JavaTest Harness User's Guide: Graphical User Interface*. Verify that you are using the appropriate filter to generate reports of test results.

Problems Moving Reports

Test reports contain relative and fixed links to other files that may be broken when you move reports to other directories.

You must update these links when moving reports to other directories. The JavaTest harness provides an EditLinks utility that updates the links in the reports for you when moving reports.

Glossary

`.jtb` Files

See command file.

`.jte` Files

See environment files.

`.jti` Files

See configuration file.

`.jtp` Files

See parameter files.

`.jtr` File

See test result files.

`.jtx` Files

See exclude list.

A

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 use the GUI or the command-line interface to audit a test run.

B

Batch Mode

You can use either the `-batch` mode option or the current `-run` command to run tests from the command line or as part of a build process. Unless you are running tests from the command line and are using the GUI to monitor the test run, the `-batch` mode option is no longer required.

C

Class

The prototype for an object in an object-oriented language. A class might also be considered a set of objects that share a common structure and behavior. The structure of a class is determined by the class variables that represent the state of an object of that class and the behavior is given by a set of methods associated with the class.

Command File

You can place routinely used configuration settings in a command file and include it in a command line. The command file is an ASCII file containing a lengthy series of commands and their arguments used in the command-line interface to modify specific configuration values before running tests.

You can use command files 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. Using a command file allows you to repeatedly use a 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 command file.

Configuration

Information about the computing environment required to execute a test suite.

In the GUI, you can use the Configuration Editor to collect or modify configuration information or to load an existing configuration. See Configuration Editor. The Configuration Editor collects the following two types of data in an configuration file:

- Test environment
- Standard Values

In the command-line interface, you can perform the following tasks:

- Use the `EditJTI` utility to modify configuration information (see `EditJTI`).

- Set specific configuration values in the command line when starting the JavaTest harness.

  Configuration File

  Contains all of the information collected by the configuration editor about the test platform.

  The JavaTest harness derives the configuration values required to execute the test suite from environment entries in a configuration file (`.jti`).

  Use the Configuration Editor or EditJTI to change configuration values in a `.jti` file.

  You can also set specific values in the command line.

  Configuration Value

  A value specified by the user for the purpose of configuring a test run.

  Configuration values are derived from environment entries in a configuration file and used by test suite specific plugin code to execute and run tests.

  For test suites prior to the JavaTest harness, version 3.0, the configuration value is read from an environment file (`.jte`).

  For JavaTest harness, version 3.0 (or later) test suites, the configuration value is derived from the configuration file (`.jti`).

  Use the Configuration Editor or `EditJTI` to change the configuration values in the `.jti` file.

  You can also set specific configuration values in the command line.

  Current Configuration

  The configuration containing the test environment and standard values currently loaded in the test manager or specified in the command line for use in running tests and displaying test status.

D

E

EditJTI

The JavaTest harness provides an EditJTI utility that you can use from the command line to edit the values entered in a configuration file without opening the JavaTest GUI. The EditJTI utility is the batch command equivalent of the JavaTest Configuration Editor.

Environment

See Test Environment.

Environment Entry

A name-value pair derived from a configuration file and used by test suite specific plugin code to execute and run tests. These name-value pairs provide information (configuration values) about how to run tests of a test suite on a particular platform.

For test suites prior to the JavaTest harness, version 3.0, the name-value pairs are read from an environment file (`.jte`).

For JavaTest harness, version 3.0 (or later) test suites, the name-value pairs are derived from the configuration file (`.jti`).

Environment Files

Contain one or more test environments used by test suites prior to the JavaTest harness version 3.0. Environment files are identified by the `.jte` extension in the file name.

Error

The test is not filtered out and the JavaTest harness could not execute it. There are no test results for tests having errors. Errors usually occur because the test environment is not properly configured.

In the GUI, the JavaTest harness displays error icons for tests with errors and for folders containing any tests with errors. Folders marked with error icons can also contain tests and folders that are Failed, Not Run, Passed, and Filtered out.

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.

In the GUI, use the configuration editor to add or remove exclude lists from a test run. In the command line, you can specify an exclude list in the command.

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.

F

Fail

Test results determined by the JavaTest harness that do not meet passing criteria.

In the GUI, the JavaTest harness displays Failed icons for tests that the test suite has determined have failing results and for folders containing any tests with fail results. Folders marked with Failed icons can also contain tests and folders that are Not Run, Passed, and Filtered out.

Filtered Out

Folders and their tests that are excluded from the test run by one or more test run filters.

In the GUI, 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. There are two types of filters in the JavaTest harness, view filters and run filters. View filters are set in the Test Manager to display the results for specific folders and tests and to create test reports. Run filters are set in the Configuration Editor or are specified as commands in the command-line to specify which tests are run.

G

H

HTTP Server

Software included in the JavaTest harness that services HTTP requests used to monitor a test run from a remote work station.

I

Interview File

See configuration file.

J

JTI

Standard file extension for a configuration file. See configuration file.

K

Keywords

Special values in a test description that describe how the test is executed.

Keywords are provided by the test suite for use in the Configuration Editor or command line as a filter to exclude or include tests in a test run.

L

M

Multiple Document Interface

A window style in which the JavaTest harness desktop is a single top-level window that contains all JavaTest harness windows opened to perform a task.

Use the Preferences dialog box to select the MDI window style. See JavaTest Harness Preferences.

N

O

Observer

An optional class instantiated from the command line to observe a test run. The class implements a specific observer interface.

P

Parameter Files

Files used prior to the JavaTest harness, version 3.0, to configure how the JavaTest harness runs the tests on your system. Parameter files have the file name extension of `.jtp`.

Use of parameter files is deprecated, however, the JavaTest harness provides support for those test suites that use parameter files.

Pass

Test results determined by the JavaTest harness to meet passing criteria.

The JavaTest harness displays Passed icons for tests that the test suite has determined have passing results and for folders containing only tests with passing results.

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 used to restrict the set of tests in a test run based on the last test result information stored in the test result files (`.jtr`) in the work directory.

Use the configuration editor or command line to enable the Prior Status filter for a test run.

Q

R

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.

S

Standard Values

The Quick Set mode of the Configuration Editor displays the standard values of a configuraion.

System Properties

Contains environment variables from your system that are required to run the tests of a test suite.

Because the JavaTest harness cannot directly access environment variables, you must use command-line options to copy them into the JavaTest harness system properties.

T

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.

Each test in a test suite has a corresponding test description that is typically contained in an HTML file.

Test Environment

A collection of configuration values derived from environment entries in the configuration file that provide information used by test suite specific plugin code about how to execute and run each test on a particular platform.

When a test in a test suite is run, the JavaTest harness gives the script a test environment containing environment entries from configuration data collected by the configuration editor. See configuration.

Prior to the JavaTest harness, version 3.0, the environment entries were read from an environment file. Use of environment files is deprecated. However, the JavaTest harness continues to provide support for those test suites that use environment files. See environment file.

Test Manager

The JavaTest harness 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 associated with the test suite.

You can view the test result files in a web browser configured to use the JavaTest harness `ResultBrowser` servlet.

Test Run Filters

Include or exclude tests in a test run. Tests are included or excluded from test runs by the following means:

- Exclude lists

- Keywords
- Prior status

Test run filters are set using the Configuration Editor or the command-line interface.

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 interpret the test description information returned to it by the test finder. The test script is a plug-in provided by the test suite. In the GUI, 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 before the JavaTest harness can run its tests.

U

V

W

Work Directory

A directory associated with a specific test suite 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.

X

Y

Z

Index

Symbols

A

B

C

E

F

H

I

J

K

M

O

P

R

S

T

U

V

W