Java™ Device Test Suite Tester's Guide

Version 2.4
Java ME Platform

Sun Microsystems, Inc.
www.sun.com

May 2009

Submit comments about this document by clicking the Feedback[+] link at: http://docs.sun.com

Please
Recycle

Contents

Figures

Tables

Preface

The *Java™ Device Test Suite Tester's Guide* provides the information needed to install and use the test harness including the Test Manager, the graphical user interface of the test harness for selecting, configuring, and running tests, and examining their results.

How This Book Is Organized

Chapter 1 provides an overview of the concepts and components of the Java Device Test Suite and the tester harness.

Chapter 2 describes how to install and launch the tester harness.

Chapter 3 describes how to update the tester harness.

Chapter 4 describes how to transfer test bundles from the harness to test devices and how to transmit test results from test devices to the harness. This chapter also covers test device requirements.

Chapter 5 describes how to start a test run by walking you through two examples, running an automated runtime test and running an interactive runtime test.

Chapter 6 describes the benchmark test results displayed in the Test Manager.

Chapter 7 describes how to use the readiness test pack to verify that your test device has the core capabilities required to run tests and that they are operational.

Appendix A documents the IP ports that tests and the harness use.

Appendix B introduces the issue of subjectivity in interpreting the results of quality tests.

Appendix C describes how to uninstall a harness.

Appendix D describes how to filter tests with JavaTest harness exclude lists.

Appendix E describes how to modify HTTP response headers sent from the Relay to the device to prevent test bundle caching.

Appendix F gives the specification and version numbers of technologies supported by the Java Device Test Suite.

Using Operating System Commands

This document does not contain information on basic Solaris™ operating system or Windows commands and procedures such as shutting down the system, booting the system, and configuring devices. Refer to the following for this information:

- Software documentation that you received with your system
- Solaris operating system documentation, which is at `http://docs.sun.com`

Shell Prompts

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

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*.

Related Documentation

The Java Device Test Suite documentation is divided among manuals and online help. For subjects that relate to graphical user interface menus, displays, and controls, consult the online help first. The manuals cover only subjects that are not related to graphical user interface features.

Application	Title
Test Development	*Java Device Test Suite Developer's Guide*
Test Execution	Online help (test harness edition), *Java Device Test Suite Test Notes*
Administration	Online help (administrator harness edition), *Java Device Test Suite Administration Guide*
Command Line (Batch) Operations	Online help, *JavaTest Command Line Interface Guide*

Accessing Sun Documentation Online

The Source for Java Developers web site enables you to access Java platform technical documentation on the web at
`http://java.sun.com/reference/docs/index.html`

Third-Party Web Sites

Sun is not responsible for the availability of third-party web sites mentioned in this document. Sun does not endorse and is not responsible or liable for any content, advertising, products, or other materials that are available on or through such sites or resources. Sun will not be responsible or liable for any actual or alleged damage or loss caused by or in connection with the use of or reliance on any such content, goods, or services that are available on or through such sites or resources.

Sun Welcomes Your Comments

Sun is interested in improving its documentation and welcomes your comments and suggestions. You can submit your comments on the web at
http://java.sun.com/docs/forms/sendusmail.html

CHAPTER 1

Overview

This chapter describes the main Java™ Device Test Suite concepts and components. Additional overview topics can be found in the *Java Device Test Suite Administrator's Guide* and the Java Device Test Suite online help.

Readers who do not want to read this background material can skip this chapter. The chapter covers the following subjects:

The Java Device Test Suite

The Java Device Test Suite helps wireless network operators and device manufacturers maximize product quality and minimize time to market. The Java Device Test Suite is an extensible set of test packs, a shared management facility, and a distributed test execution harness that assess the quality of any device that implements a compatible combination of the following Java Platform, Micro Edition (Java ME platform) technologies:

- Advanced Multimedia Supplements (AMMS)
- Connected Limited Device Configuration (CLDC)
- Contactless Communication API
- Content Handler API (CHAPI)
- Java APIs for Object Exchange (OBEX) and Bluetooth
- Java Technology for the Wireless Industry (JTWI)
- Location API Optional Package
- Mobile 3D Graphics API

- Mobile Information Device Profile (MIDP)
- Mobile Internationalization API
- Mobile Media API (MMAPI)
- Mobile Sensor API (MSAPI)
- Mobile Service Architecture (MSA) security tests
- OpenGL Embedded Subset (ES)
- Payment API
- Personal Digital Assistant (PDA) optional packages
- Scalable Vector Graphics (SVG)
- Security and Trust Services API (SATSA)
- Session Initiation Protocol (SIP)
- Web Services (JAXP and JAX-RPC Subset)
- Wireless Messaging API (WMA)
- XML API

For a detailed list of supported specifications and versions, see Appendix F.

The Java Device Test Suite also includes tests that do not relate to a particular specification:

- Benchmark (performance) tests that compare the performance of a device to a reference standard.
- Readiness tests that assess a device's ability to run tests and discover the application programming interfaces (APIs) that a device supports.
- Sample tests for the tutorial in Chapter 5.
- Tests for multitasking implementations.

The product's total of about 11,000 tests can be extended with new tests written by Sun or by others, including Java Device Test Suite users.

Compatibility and Quality Testing

When considering a device that implements Java technology, it is useful to distinguish between *compatibility testing* and *quality testing*. Compatibility testing exercises individual application programming interfaces (APIs) defined by a specification, such as the MIDP 2.0 specification.

When passed valid arguments, a device method call returns a valid result, the invoking test reports that the method implementation conforms to the specification. Compatibility testing is an important first step in testing a product. Sun Technology Compatibility Kits (TCKs) perform compatibility testing, and, ideally, are used in conjunction with the Java Device Test Suite.

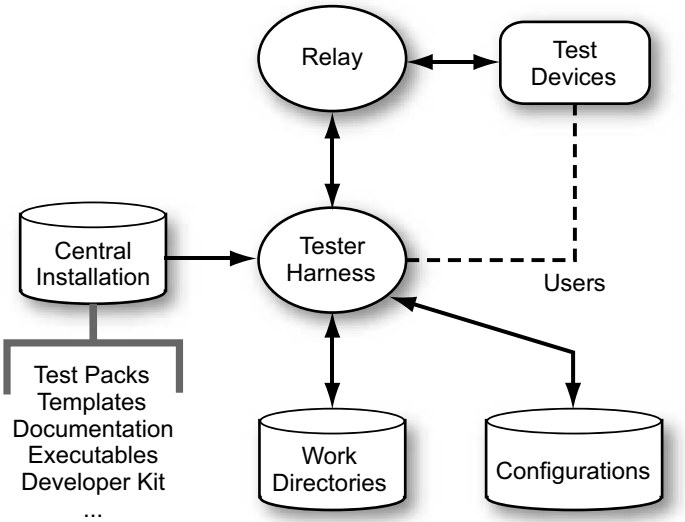
A quality product requires additional tests before it can be confidently released to run real applications in real-world conditions. The Java Device Test Suite complements Sun TCKs by performing quality testing, which can be divided into three main areas:

- *Runtime testing* imitates real applications and their interaction with external components such as message senders and receivers. Runtime tests exercise multiple APIs in realistic scenarios that include error conditions. Some runtime tests, called negative tests, verify that an implementation correctly handles invalid inputs, states, and usage. Other runtime tests, called robustness tests, exercise multiple APIs in parallel to expose implementation weaknesses. To summarize, devices that pass TCK tests have demonstrated that their implementations comply with a specification.
- *Benchmark testing* can be used to compare a device's actual performance with performance goals. It measures factors such as system load and frame rate.
- *Over-the-air (OTA) provisioning testing* verifies the ability of a device to correctly obtain and install applications over the air, to enforce security policies, and to communicate properly with a provisioning server.

Architecture: Tester's View

The Java Device Test Suite software consists of the components shown in FIGURE 1-1.

FIGURE 1-1 Java Device Test Suite Architecture: Tester's View



- The Central Installation is a set of directories that holds the key Java Device Test Suite components. It is a sharable repository for resources including test packs, templates, executables, and documentation. The Central Installation is not a server. It is an ordinary directory. An administrator creates and maintains the Central Installation as described in the *Java Device Test Suite Administrator's Guide*.
- The Relay is a web application that coordinates with test devices as tests are run. It supplies bundles of tests to devices, accepts test results from devices, emulates an application provisioning server for over-the-air (OTA) tests, and manages the server side of network tests. Tester input for interactive tests are also sent through the Relay. Once installed and launched, the Relay manages itself.
- Tester harnesses are functional subsets of the administrator harness. Testers run tests and create reports. Tester harnesses use Central Installation resources but cannot create or update them. Testers use harnesses to create configurations, which represent devices, and work directories, which store test results. The tester harness online help and this document describe how to use the tester harness.

Tester Harness and Central Installation

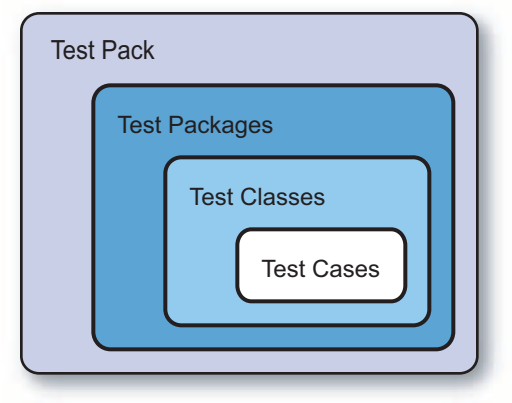
Testers interact with the Central Installation indirectly. When you launch a harness, the executable comes from the Central Installation. When you direct a harness to run tests, use a keystore, or other shared resources, the harness obtains them from the Central Installation.

Test Packs, Packages, Classes, and Cases

A *test pack* is a collection of tests that are functionally related and have common setup requirements. Functionally related means that the tests exercise a cluster of test device functions, for example, the MIDP 2.0 runtime APIs. Common setup requirements means that all the tests in the pack either require the same setup (such as starting a partner device that cooperates with the test device) or the same user interaction (such as inspecting test device behavior), or both. Developers create test packs as described in the *Java Device Test Suite Developer's Guide*.

A test pack has the hierarchical structure shown in FIGURE 1-2:

FIGURE 1-2 Test Pack Hierarchy



A test pack contains at least one test package. A test package can contain a nested test package or a test class. A test class contains at least one test case.

FIGURE 1-3 shows how the harness displays an example test pack in which the test cases in one test class are expanded (exposed to view). The harness does not visually distinguish packs, packages, classes, and cases, except by their position in the hierarchy. Test packs have no ancestors. Test cases have no descendents. The parents of test cases are test classes. Everything else is a test package.

FIGURE 1-3 Sample Test Pack Hierarchy

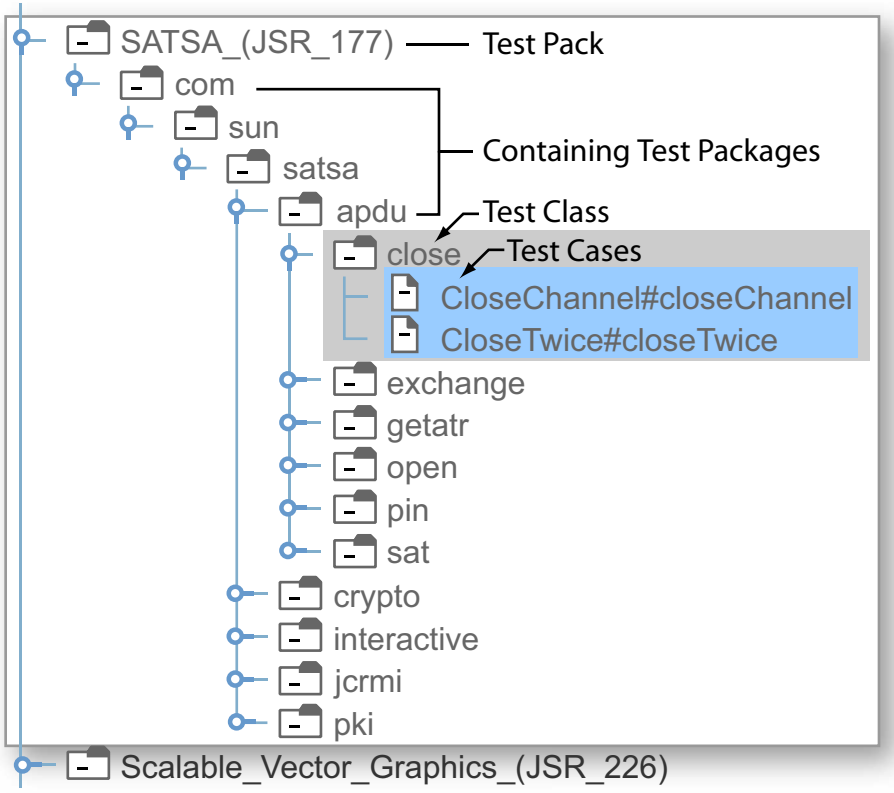


FIGURE 1-3 shows the SATSA_(JSR_177) test pack. The test pack contains the apdu test package, which contains the close test class, which contains the test cases CloseChannel#closeChannel and CloseTwice#closeTwice.

Test packages subdivide a test packs name space. Test packages can be nested, and they often are. For example, there are six packages visibly nested in the com.sun.satsa package in FIGURE 1-3.

A test class is the unit of code that test developers create. A test class contains one or more test cases.

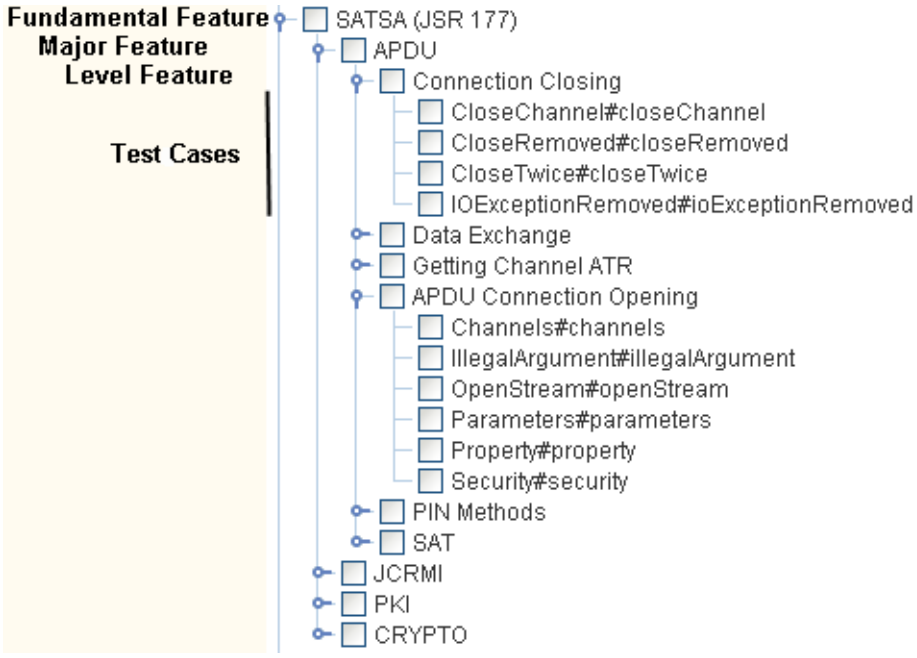
A test case, informally called a test, is the code in a test class that exercises one or more test device functions and passes, fails, or returns benchmark metrics. The *Java Device Test Suite Developer's Guide* describes test classes and test cases in detail.

Your interaction with the Java Device Test Suite software, and the interaction of the software components themselves, varies according to the kinds of tests that you run.

Device Features

FIGURE 1-4 shows some of the same SATSA tests shown in FIGURE 1-3. Instead of a test pack hierarchy, the tests are arranged in a device feature hierarchy. This hierarchy emphasizes capabilities that might be supported by a given test device. For some users, particularly those who work for mobile network operators, this hierarchy is easier to use because it maps better to the description of a device provided by a manufacturer. Java Device Test Suite testers and administrators can use either hierarchy to select tests to run and to view results in a report. The main harness test tree (shown on the left in FIGURE 1-8) always displays the package hierarchy. In either view, the test cases that run are the same. The two hierarchies organize the tests differently.

FIGURE 1-4 Sample Feature Hierarchy



The tutorial in Chapter 5 covers both methods of test selection.

In addition to optionally selecting tests by device feature, after a test run, you can create a report that is organized by device feature. FIGURE 1-5 is an example. The online help describes how to create a feature-based report.

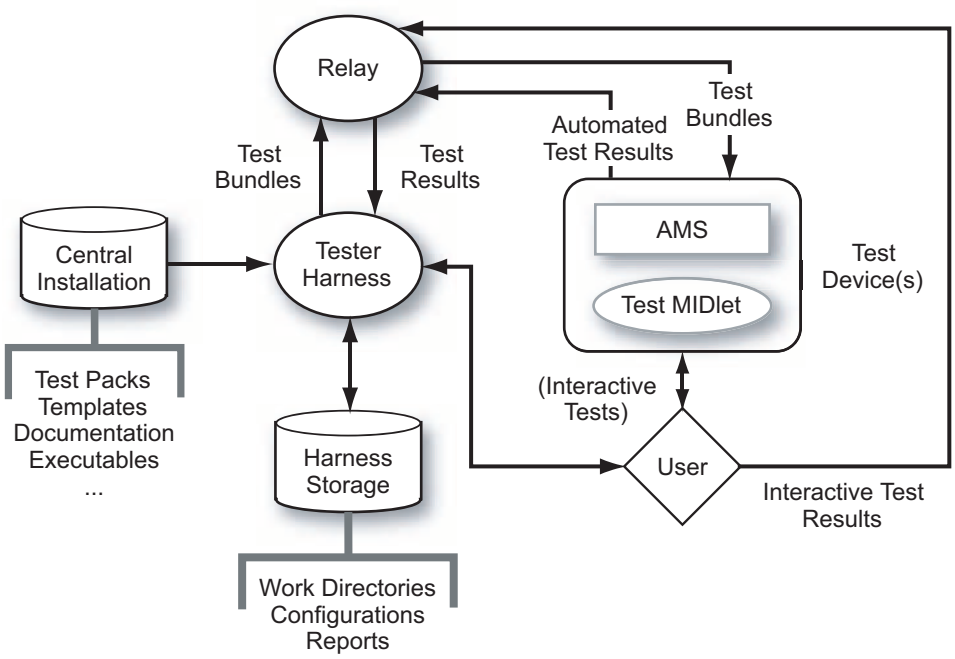
FIGURE 1-5 Feature Report Example

Runtime Test Architecture

Runtime tests execute on the test device under the control of a Java Device Test Suite component called an *agent*. The agent is usually included in the test bundles that users download and install in devices being tested. It is also possible to install an agent in some test devices' read-only memory. A *test bundle* is a collection of tests packaged as a MIDlet suite, a unit that a test device's application management system (AMS) must be able to download, install, and execute. Specifications, such as MIDP and MMAPI, define the device-resident APIs that runtime tests test.

FIGURE 1-6 summarizes the user and component interactions for runtime tests, and "Tester Harness" on page 11 provides details. Runtime tests that involve networking use additional Java Device Test Suite components to send and receive test messages. These are not shown in FIGURE 1-6:

FIGURE 1-6 Java Device Test Suite Runtime Architecture



From the user's point of view, there are two kinds of runtime tests, automated and interactive. An automated runtime test determines its result (passed or failed) and returns the result to the agent, which returns it to the harness via the Relay. An interactive runtime test relies on you to determine if the test passed or failed. When an interactive test begins to run, the harness displays instructions for you to operate the device and inspect the device's response (for example, what it displays). The instruction window includes Passed and Failed buttons. If the device behaves as the test instructions say it should, you click the Passed button. Otherwise, you click the Failed button.

Because interactive tests require your presence and automated tests do not, it is often convenient to run these tests separately. The tester harness has a selector that lets you do just that. You can choose to run only automated, only interactive, or both kinds of tests in a given test run. You can merge the results produced in multiple runs into a single report. You can view a report in a web browser or export its data to a spreadsheet or database application for analysis and presentation.

To reduce the time required to execute large numbers of tests, the architecture supports two styles of parallel testing, which are not shown in FIGURE 1-6.

- One user can simultaneously run different test subsets on multiple instances on one device, for example, on three instances of a hypothetical model 52 Parallel testing by one user produces an integrated set of results, as if it had been run on a single device.
- Multiple users can test different devices or multiple instances of the harness. You can merge the results produced by multiple testing instances.

The Java Device Test Suite offers a variant of the architecture shown in FIGURE 1-6, called *offline mode*. The intent of offline mode is to allow some testing to be done on devices that are not developed enough to support full testing. For most devices, the agent uses HTTP to return automated test results to the harness. In offline mode, the agent writes test results to the device display when the last test in a bundle has been completed. Tests that rely on the presence of Java methods that implement HTTP cannot be run in offline mode.

Benchmark Test Architecture

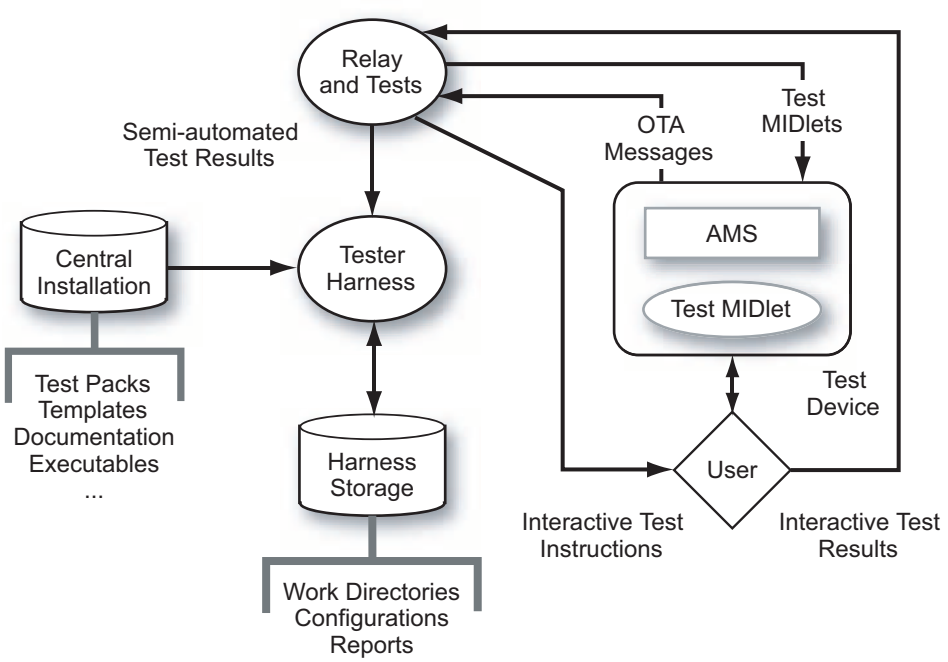
From a tester's perspective, *benchmark tests* are nearly identical to runtime tests. Benchmark tests return performance measurements called *metrics*, that the tests compute and the harness displays. Benchmark tests can also pass or fail. The pass or fail decision is made by comparing the test device's performance with the performance of a reference device on the same test. The reference device's performance constitutes a *threshold* that the test device must meet or exceed for the test to pass.

OTA Provisioning Test Architecture

OTA provisioning tests verify the quality of a device's over-the-air application provisioning implementation. This includes obtaining, installing, and removing applications (MIDlet suites), and enforcing security requirements.

Unlike runtime and benchmark tests, OTA provisioning tests do not run on the test device. They run on an emulated provisioning server that is implemented as a servlet. Each OTA test has an associated application (a MIDlet suite) that you download from the provisioning server and install and launch on the test device. FIGURE 1-7 summarizes the roles of the user and the provisioning server in OTA provisioning testing. For simplicity, FIGURE 1-7 shows a single test device and a single harness. In reality, multiple instances of each can run in parallel, sharing one provisioning server.

FIGURE 1-7 Java Device Test Suite Architecture: OTA Provisioning Tests

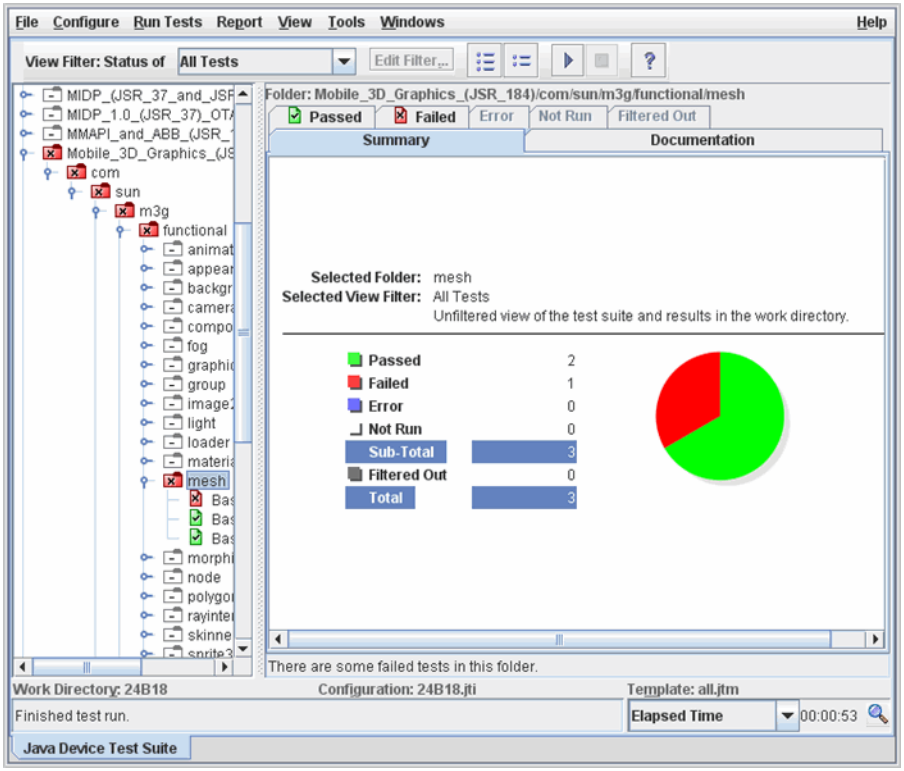


There are two kinds of OTA provisioning tests: semi-automated and interactive. For both kinds of tests, the harness displays instructions for you to download the associated application to the test device. A *semi-automated* OTA provisioning test decides if it passes or fails based on the content of status information returned to it by the test device. An *interactive* OTA provisioning test's instructions have Passed and Failed buttons and instructions for determining if the downloaded application has performed correctly. You compare the device's behavior to the expected outcome described in the instructions and click the Passed or Failed button as appropriate.

Tester Harness

Testers primarily interact with the Java Device Test Suite harness. FIGURE 1-8 shows the harness's graphical interface, the Test Manager. The harness can also be run in batch (command line) mode.

FIGURE 1-8 Test Manager User Interface



In a typical testing session, the first step is to open or create a work directory. A *work directory* is associated with a specific template. It contains a configuration file that is based on the template. When a test is run, the harness creates test result files that store information about the test run. The harness then stores these files in the work directory.

When you create a work directory, you select a *template*. Administrators create templates with the Java Device Test Suite Administrator harness. A template is the archetype of a *configuration*. Both specify tests to be run and configuration values. A template typically corresponds to a test device. A configuration typically corresponds to a subset of tests for a device. For example, two testers might divide the testing of a device by creating two configurations from the same template, each configuration specifying half of the tests to be run on the device. However, there are no limitations on how you use templates and configurations. The more that an administrator preconfigures a template, the less information that you need to supply in a configuration that is based on that template. A work directory's template is permanent, but you can switch a work directory's current configuration as you wish, for example, to run a different subset of tests. When you open a work directory, its configuration, if it has one, is loaded at the same time.

When you have a work directory and a configuration that represents the tests you want to run, you can start the test device and click the Start button on the harness to begin testing.

- For automated runtime and benchmark tests, depending on how your test device communicates with the harness, you might do nothing until the test run completes, or you might manually direct the downloading of test bundles to the test device. Once downloaded, automated runtime and benchmark tests can run without user intervention if the test device is set to automatically grant permission for tests to use protected interfaces.
- For interactive runtime tests, test instructions appear in a harness window. You compare the test device's actual test response to the expected response described in the instructions, and click the instruction window's Passed or Failed button.
- For OTA provisioning tests, test instructions appear in a harness window. They name a MIDlet (application) to download to the test device from the provisioning server and then launch. Some OTA provisioning tests ask you to observe the device's behavior and indicate if the test has passed or failed.

As a test run proceeds, the harness updates its summary results, which show the numbers of passed and failed tests. While a run is in progress or after it ends, you can inspect individual test results and diagnostic logs. The harness records each test's most recent pass or fail status. You can elect to rerun only the tests that failed in the previous run.

Testing a device might require multiple runs, for example, if there are configuration problems or the test device software is updated. These runs might be separated by several days. With a work directory, you can load the same configuration file and run the tests again if needed or edit the configuration to change certain test values or to run only those tests that previously failed.

You can save information about a test run by creating a report. Also, because running all of the tests for a device can be quite time consuming, it is common practice to divide the tests among multiple users, who run their tests in separate sessions. You can merge reports produced in multiple testing sessions run by a single tester or by multiple testers into a single report that gives an integrated view of the device's test status.

Documentation

The tester and administrator harnesses have extensive online help, which you should consult first when you have a question about using the tester harness. The Central Installation holds the following additional documentation, accessible from *InstallDir*`/index.html`:

- This guide and corresponding guides for administrators and developers.

- HTML documentation that describes each test pack, package, and class. You can view the online documentation with the harness or a web browser.
- A guide for using the command line interface (also available in the help).

Executables

The Central Installation holds the Java Device Test Suite executables, which the administrator and tester launch scripts share. The Relay executable is normally installed on a host located in the demilitarized zone between the organization's inner and outer firewalls.

CHAPTER 2

Installing and Launching the Tester Harness

This chapter describes how to install and launch the tester harness. This chapter has the following sections:

Installation Requirements

Before you install the tester harness, have the following software already installed:

- One of the following operating systems:
  - Solaris 10 operating system for the SPARC® processor
  - Windows XP with Service Pack 2
- A Java Runtime Environment (JRE™) software version 6 Update 3 (also known as 1.6.0_03) or greater

  Sun recommends using the latest JRE software. JRE version 6 Update 3 is the officially supported version.

  You can download the JRE software at
  `http://java.sun.com/javase/downloads/index.jsp`
- (Recommended) Sun Java Wireless Toolkit 2.5 for CLDC

  You can download the Wireless Toolkit at
  `http://java.sun.com/products/sjwtoolkit/download-2_5.html`

- Solaris operating system hosts: The POSIX version of the `df` command.

  In a default Solaris operating system installation, the command is `/usr/xpg4/bin/df`, and the installer looks for it there. If this directory does not exist or does not contain `df`, you must prepend the POSIX `df` command's location to your `PATH` so the installer finds it before any other version of `df`. If necessary, you can obtain the POSIX df command from the Solaris installation CD or DVD.

Use the following guidelines for hardware requirements:

- Disk space - The tester harness installation uses about 20 megabytes of disk space. Reports, especially XML reports, are the major variable consumer of disk space. 100 megabytes per report is good for estimating, assuming you are running all tests. You need less space to run test subsets.
- Memory - Allow about 500 megabytes for the harness, another 100 megabytes for the Sun Java Wireless Toolkit for CLDC.

Installing the Tester Harness

These instructions assume that the Relay and the Central Installation are already installed and that the Java Application Server is running. See the *Java Device Test Suite Administration Guide* for information on installing the application server.

Note – Testers must install their own copy of the harness. Testers executing the harness must have write permission for the installed files.

You can install the tester harness graphically or with a command line script.

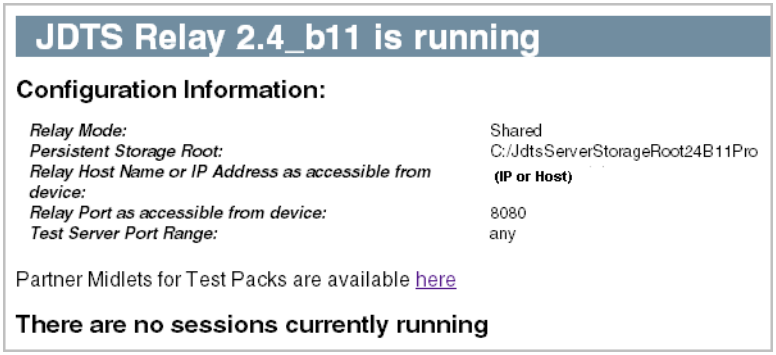
▼ Running the Graphical Installer

To interactively install the tester harness with the graphical user interface, follow these steps:

1. **Ensure that the Relay is running by entering** `http://`*relayHost*`:`*relayPort*`/`*appContext* **in a web browser.**

   An administrator can give you the values of *relayHost*, *relayPort*, and *appContext*. If the Relay is running, the browser displays a page similar to FIGURE 2-1. The version number and other information might be different.

FIGURE 2-1 Relay is Running Web Page



2. **Launch the Java Device Test Suite (tester) installer.**
   - Windows operating system hosts: Double click `jdts_tester-`*version*`-rr-`*build*`_windows-`*date*`.exe`.
   - Solaris operating system hosts:

     i. **Make the installer file executable.**

     The installer file is `jdts_tester-`*version*`-rr-`*build*`_solaris-`*date*`.bin`.

     ii. **Run this command:**

     `%` **`sh jdts_tester-`***version*`-rr-`***build*`_solaris-`***date*`.bin`**

The value of *version* depends on the version of the installer. The value of *build* and *date* depend on when the software was built.

3. **Click Next.**
4. **Accept the license agreement, then click Next.**
5. **Specify** *jdts_installDir* **as the Central Repository, then click Next.**
6. **Specify the installation directory of the Java Runtime Environment software, then click Next.**
7. **Specify the directory where you want to install the tester harness, then click Next.**

   For example, *yourHomeDir*`/Test`.

   Do not specify any directory whose name contains a space (" ") character.
8. **Select the location where you want to install the product icons (shortcuts), then click Next.**

   This screen only appears in the Windows installer. If you install two instances of the same version of the tester harness, the second instance's shortcut replaces the first.

9. **Review the installation summary information, then click Next to install the tester harness.**

   If any field contains the wrong value, click Previous to go back to the page and provide the correct information.

10. **When the installation has finished, click Done to exit the installer.**

After installation, open the harness's `index.html` file (example: *yourHomeDir*`/Test/index.html`) in a web browser to see a linked overview of the Java Device Test Suite documentation.

Note – Graphical user interface commands are described in the Java Device Test Suite online help. To learn about these commands, launch the harness as described in "Launching in Graphical Mode" on page 20, then choose Help > Online Help.

▼ Running the Command Line Installer

You can automate the installation of a tester harness by writing a script that invokes the installer. Invocation examples (ignore line breaks):

- Windows environment:

```
> jdts_tester-version-rr-build_windows-date.exe -f tester.properties
```

- Solaris operating system:

```
% sh jdts_tester-version-rr-build_solaris-date.bin -f
tester.properties
```

The value of *version* depends on the version of the installer. The value of *build* and *date* depend on when the software was built.

Note – Command line installation does no error checking. A mistake can produce an aborted or incorrect installation. To minimize the chance for error, you can install with the graphical interface first (see "Running the Graphical Installer" on page 16), write down the property values that you enter, and verify the installation.

Before installing, ensure that the Relay is running by entering `http://`*relayHost*`:`*relayPort*`/`*appContext* in a web browser. An administrator can give you the values of *relayHost*, *relayPort*, and *appContext*. If the Relay is running, the browser displays a page similar to FIGURE 2-1.

You create the text file `tester.properties`, which specifies where to install the harness, and so on. The file name can be *anything*`.properties`. This file has the general syntax of a Java programming language properties file. For a technical

description of this syntax, see http://java.sun.com/j2se/1.5.0/docs/api/java/util/Properties.html#load(java.io.InputStream). Be sure that no line has a trailing space (blank) character. TABLE 2-1 defines the required properties.

TABLE 2-1 Tester Harness Command Line Installer Properties

Property	Value
INSTALLER_UI	Must be `silent`.
CENTRAL_REP	Absolute path to the Central Installation, the shared directory created by an administrator that contains `ReleaseNotes.html` and other files.
JDK_FOLDER	Absolute path to a Java Runtime Environment directory. See "Installation Requirements" on page 15 for the required version.
USER_INSTALL_DIR	Absolute path to the directory that is to contain the tester harness.
INSTALLER_JAVA_DOT_HOME	Same as `JDK_FOLDER`.

Here is an example `tester.properties` file for the Windows environment:

```
INSTALLER_UI=silent
CENTRAL_REP=c\:\\JDTS-CI
JDK_FOLDER=C\:\\Java\\jre1.6.0_03
USER_INSTALL_DIR=c\:\\JDTS_Tester
INSTALL_DRIVE_ROOT=c\:\\
INSTALLER_JAVA_DOT_HOME=C\:\\Java\\jre1.6.0_03
```

Here is an example `tester.properties` file for the Solaris operating system:

```
INSTALLER_UI=silent
CENTRAL_REP=/home/userName/JDTS-CI
JDK_FOLDER=/usr/jdk/1.6.0_03
USER_INSTALL_DIR=/home/userName/JDTS_Tester
INSTALL_DRIVE_ROOT=/
INSTALLER_JAVA_DOT_HOME=/home/userName/bin/Java/jre1.6.0_03
```

Note – If you uninstall a harness that was installed from the command line, the harness is uninstalled "silently," that is, without a graphical user interface.

Troubleshooting

A command line installation does not provide visual feedback, console messages, or an error code. You can monitor its progress with an operating system tool that shows the status of processes.

The installation is complete when the file *USER_INSTALL_DIR*`/JDTS`*version*`_ConsoleInstallLog.log` has been created. Open this file with a text editor and search for messages containing the string "FATAL". If there are no such messages, installation was successful. The "FATAL" messages describe the details of the problems encountered, if any.

Installing Multiple Harnesses

You can install as many tester harnesses as you need on the same workstation. Ensure that each harness is installed in a different directory. For each instance of the harness, follow the procedures in "Installing the Tester Harness" on page 16. If you want shortcuts to multiple harnesses, direct the installer to create them in different folders.

Launching the Tester Harness

You can launch the harness in graphical or batch mode. In the examples, assume that the tester harness is installed in *yourHomeDir*`/Test/`.

▼ Launching in Graphical Mode

To start the tester harness, follow these steps:

- **In a terminal window:**
  - For the Solaris operating system (C shell):

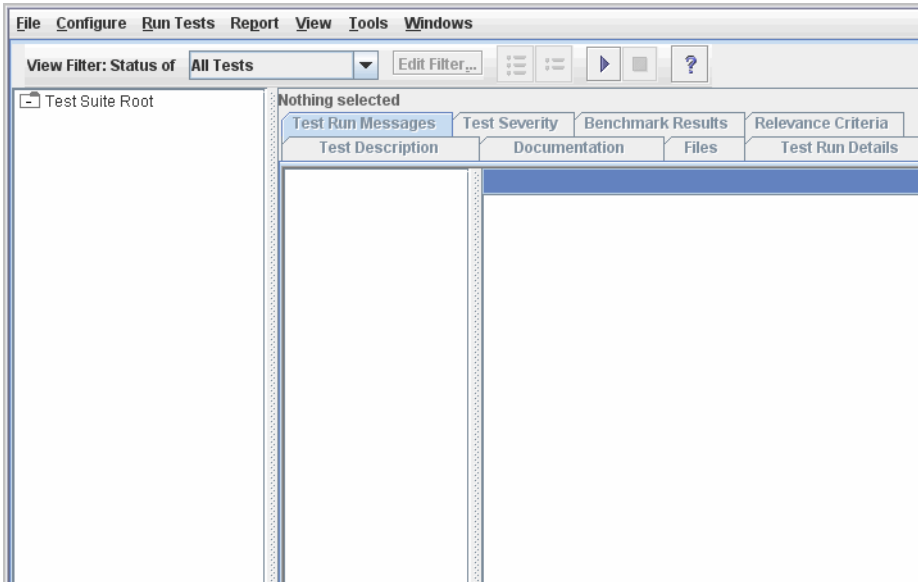
    `%` **`sh`** *yourHomeDir*`/Test/run.sh`
  - For Windows:

    `>` *yourHomeDir*`\Test\`**`run.bat`**

In Windows, you can alternatively double-click *yourHomeDir*`\Test\`**`run.bat`**. or use a shortcut you specified in the installer.

The Tester version of the harness graphical user interface appears:

FIGURE 2-2 Tester Harness Graphical User Interface



See the online help for a description of the graphical user interface.

▼ Launching in Batch Mode

To launch the tester harness in batch mode, follow one of these steps, depending on your operating system:

- **In a terminal window:**
  - For the Solaris operating system:

    `%` **`sh`** *yourHomeDir*`/Test/run.sh -batch` [*Set-up Commands*] [*Task Commands*]
  - For Windows:

    `>` *yourHomeDir*`/Test/run.bat -batch` [*Set-up Commands*] [*Task Commands*]

  Set-up Commands and *Task Commands* are described in the *JavaTest User's Guide: Command Line Interface*.

CHAPTER 3

Updating the Tester Harness

This chapter describes how to how to update the tester harness. The Central Installation must be updated before you update the tester harnesses. Check with your administrator to find out if the Central Installation has been updated.

The chapter contains these sections:

Obtaining the Software

Download the Java Device Test Suite updaters from the Java Device Test Suite licensee web site at `https://javapartner.sun.com`. Download one or both sets of updater files:

- Solaris operating system:

  `jdts_tester-`*version*`-update-`*build*`_solaris-`*date*`.bin` (tester harness updater)

- Windows operating system:

  `jdts_tester-`*version*`-update-`*build*`_windows-`*date*`.exe` (tester harness updater)

The value of *version* is the version of the Java Device Test Suite that running the updater creates. The value of *build* and *date* depend on when the software was built.

System Requirements

The updaters have the same resource requirements as the installers, described in "Installation Requirements" on page 15.

Updating the Installation

You can update a tester harness installation interactively with a graphical user interface or with a command line script.

▼ Running the Graphical Updater

1. **Ensure that the central installation is updated before you begin the tester harness update.**

   Check with your administrator to be sure the central installation is updated.

2. **Ensure that the Relay is running.**

   Verify by entering this URL in a web browser:
   `http://`*relayHost*`:`*relayPort*`/`*appContext*

   For example: `http://localhost:8080/JdtsServer`. An administrator can give you the values for *relayHost*, *relayPort*, and *appContext*.

   The Relay displays a page similar to FIGURE 2-1 if it is running.

3. **Ensure that *no* tester harness is running.**

   You must arrange for all harnesses to be shut down while the tester updater runs.

4. **Launch the tester updater:**

   - Windows operating system hosts:

     Double click `jdts_tester-`*version*`-update-`*build*`_windows-`*date*`.exe`.

   - Solaris operating system hosts:

     Enter the following command at the prompt:

     `%` **`sh jdts_tester-`***version*`-update-`***build**`_solaris-`***date**`.bin`**

   Depending on processor, disk, and network speed, the updater can take a few minutes to initialize before it displays its first screen.

5. **Accept the license agreement and click Next to continue.**

6. **Specify the current installation location for the tester harness, and click Next to continue.**

   Click Choose to navigate to the installation directory.

7. **Specify a directory containing a Java Runtime Environment software version.**

   You can choose a JRE software version that is newer than the version you have been using. "Installation Requirements" on page 15 describes the supported versions.

8. **Review the Update Summary information and click Install.**

9. **Click Finish when the update process completes.**

▼ Running the Command Line Updater

You can automate the update of a tester harness by writing a script that invokes the updater. Invocation examples (ignore line breaks):

- Windows environment:

  > **jdts_tester-***version***-update-***build***_windows-***date***.exe -f tester.properties**

- Solaris operating system:

  % **sh jdts_tester-***version***-update-***build***_solaris-***date***.bin -f tester.properties**

The value of *version* depends on the version of the updater. The values of *build* and *date* depend on when the updater was built.

`tester.properties` is a text file you create. It can be named *anything*`.properties`. However, the full path name of the `.properties` file must *not* contain a space.

The `.properties` file must contain the following lines:

```
INSTALLER_UI=silent
CONSOLE_DIR=pathToInstallDir
CHOSEN_INSTALL_FEATURE_NUM=1
CHOSEN_INSTALL_FEATURE_LIST=Update
CHOSEN_INSTALL_SET=Update
CHOSEN_INSTALL_BUNDLE_LIST=Update
CHOSEN_INSTALL_FEATURE_1=Update
```

`MY_JAVA_HOME2=`*pathToJRE*

pathToInstallDir is the absolute path to the directory containing the tester harness that is to be updated.

pathToJRE is the absolute path to a directory containing a Java Runtime Environment version described in "Installation Requirements" on page 15. You can specify a JRE software version that is newer than the version you specified when you installed the tester harness.

- Windows environment example:

```
CONSOLE_DIR=c\:\\JDTS_Tester
```

  Notice that the path characters ":" and "\" must be preceded by "\".

- Solaris operating system example:

```
CONSOLE_DIR=/home/userName/JDTS_Tester
```

Before updating, verify that the Relay is running by entering this URL in a web browser: `http://`*relayHost*`:`*relayPort*`/`*appContext*

For example: `http://localhost:8080/JdtsServer`. An administrator can give you the values for *relayHost*, *relayPort*, and *appContext*.

A command line update does not provide visual feedback, console messages, or an error code. You can monitor its progress with an operating system tool that shows the status of processes.

The update is complete when the file *CONSOLE_DIR*`/JDTS`*version*`_ConsoleInstallLog.log` has been created. Open this file with a text editor and search for messages containing the string "FATAL". If there are no such messages, installation was successful. The "FATAL" messages describe the details of the problems encountered, if any.

Verifying the Update

You can verify the update by launching the tester harness and choosing Help > About the Java Device Test Suite. The display shows Version *version* RR, where *version* is equivalent to the version in the updater file name (see "Obtaining the Software" on page 23). For example (your version number might be different):

- Updater file name version: 2_4
- Help > About the Java Device Test Suite: JDTS Version 2.4

CHAPTER 4

Connecting Test Devices

For runtime and benchmark tests, there are several ways to connect the test device to the harness so that tests can flow to the device and test results can flow to the harness. This chapter describes the options and the details of test device-harness communication in the following sections:

- Test Device Requirements
- Test Device Connection Options
- Specifying the Transmission of Bundles and Results

The material in this chapter does not apply to over-the-air provisioning (OTA) tests. Those tests are structured very differently and have no options for communication. In particular, the means of bundle transfer and result disposition described in this chapter has no effect on the execution of OTA tests. See "OTA Provisioning Test Architecture" on page 10 for information.

Test Device Requirements

Test devices must meet these requirements:

- Heap space. At least 128KB of heap space is required. Some tests require more space.
- MIDlet suite size: A test device must be able to download and install MIDlet suites (test bundles) of at least 80Kbytes in size. Most tests are larger but can fit in 128Kbyte bundles. A few tests are as large as 1,300Kbytes.

Test Device Connection Options

In addition to the device requirements, a test device must support at least one of the connection options described in this section. The connection options support the transfer of tests and results:

- Tests must be transferred from the Relay to a test device so they can run on the device.
- Test results must be transferred from the test device to the Relay so they can be incorporated in reports and displayed in the harness. If the device cannot transfer results, they can be displayed on the device.

Test Bundle Transfer

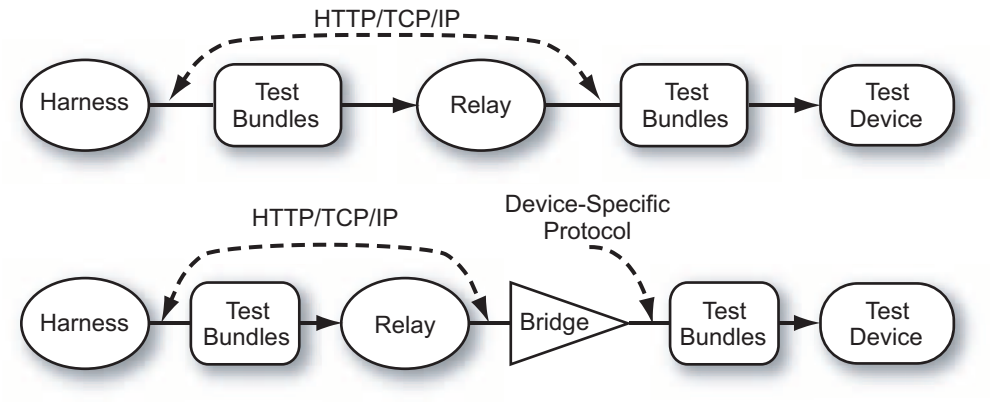
When you direct the harness to run tests, the harness packs groups of tests into MIDP MIDlet suites, which are called test bundles. To run the tests in a bundle, the bundle must be transferred to the test device, installed, and launched. The bundles can be transferred by HTTP or over a local link, such as a serial cable. You specify your choice in the Tests and Bundles section of a template or configuration. The operations for installing and launching a bundle are device dependent.

HTTP Bundle Transfer

When test bundles are transferred by HTTP, the harness repeatedly creates test bundles and sends them to the Relay. You then download a bundle from the Relay, install it and launch it, which causes the tests to run.

The Relay can transfer test bundles over a TCP/IP connection that supports HTTP version 1.0 or 1.1. The device itself does not need to support HTTP over TCP/IP if bridge hardware or software between the test device and the Relay can act as an HTTP client on behalf of the device. A WAP gateway is one example of such a bridge. FIGURE 4-1 illustrates HTTP test bundle downloading with and without a bridge.

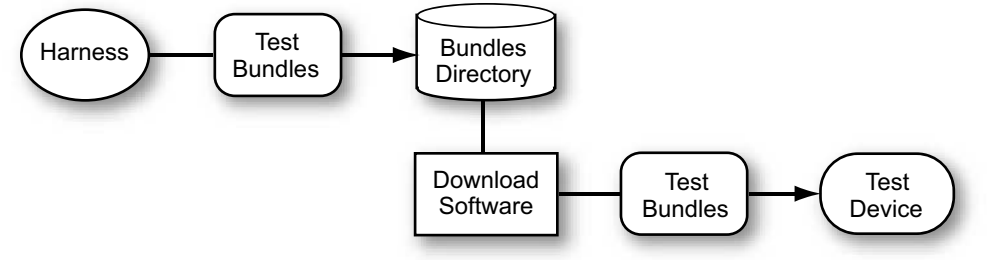
FIGURE 4-1 Test Bundle Transfer Options - HTTP



Local Link Bundle Transfer

The alternative to HTTP bundle transfer is to use a local link, such as serial, infrared, or Bluetooth. This option requires cooperating software on the harness host or another host. In the Configuration Editor, specify a directory into which the harness stores the test bundles. Direct the software to download the test bundles from the directory to the test device. FIGURE 4-2 illustrates test bundle transfer by a local link.

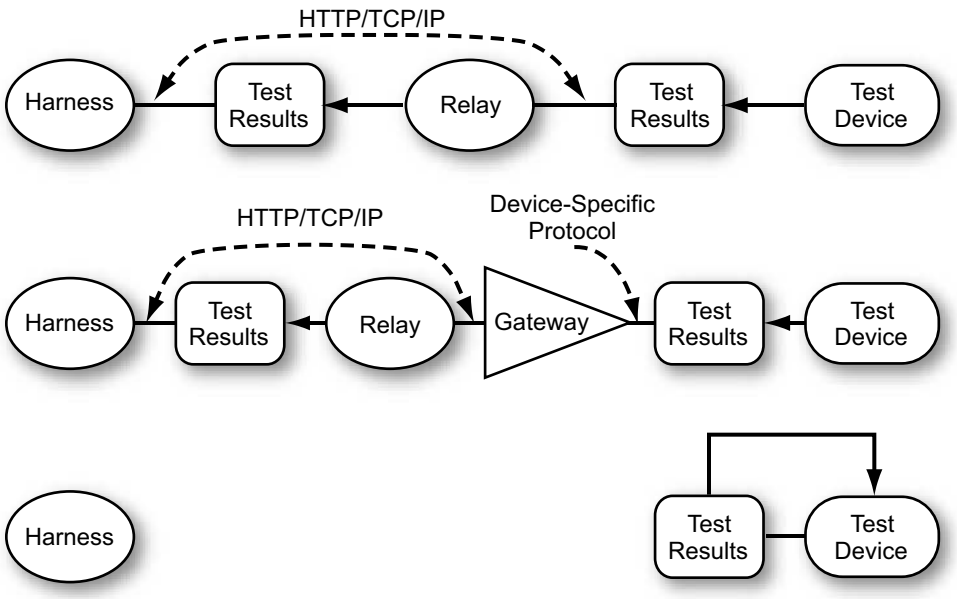
FIGURE 4-2 Test Bundle Transfer - Local Link



Test Result Disposition

For test results to be transferred from a test device to a harness, the test device's Java software or a gateway interposed between the device and the Relay, must support HTTP version 1.0 or 1.1 over TCP/IP. If the test device or gateway does not support HTTP, test results are not sent to the harness. Instead, test results can be displayed on the device. FIGURE 4-3 illustrates the possible test result disposition options:

FIGURE 4-3 Test Result Disposition Options



Specifying the Transmission of Bundles and Results

In the Configuration Editor, specify how tests are downloaded to the test device and how test results are handled. You can choose from the following test bundle transfer and result disposition options:

- **Option A.** Manually download each test bundle over an HTTP connection. Return the test results to the harness using HTTP.

  You can select this value if both of the following conditions are true:

  - **Bundle Download.** You can direct the test device to download, install, and launch a MIDlet suite (a test bundle) from a web server, possibly with assistance from an intermediating bridge, such as a WAP gateway.
  - **Result Disposition.** The test device has working Java methods for HTTP communication that the agent can use to send results to the harness. The results can also pass through a bridge.
- **Option B.** Manually download each test bundle over an HTTP connection. Hold and display the test results on the test device.

  This option is for test devices whose Java methods for HTTP communication are absent or do not work correctly. The device must still be able to download MIDlets using HTTP, which is possible if the device's native HTTP implementation is working.
- **Option C.** Manually download each test bundle from a directory on the harness, using a local link such as a serial line. Return the test results to the harness using HTTP.

  Select this value if both of the following conditions are true:

  - **Bundle Download.** You can direct the test device, or associated software, to download, install, and launch a MIDlet suite from a harness directory. The MIDlet file transfer can be conducted over a serial line, a Bluetooth connection, an infrared connection, or similar.
  - **Result Disposition.** The test device has working Java methods for HTTP communication that the agent can use to send results to the harness.
- **Option D.** Manually download each test bundle from a directory on the harness, using a local link such as a serial line. Hold and display the test results on the test device.

  This option is for devices whose Java methods for HTTP communication are absent or do not work correctly.
- **Option E.** Have the test device implement the harness's autotest protocol, in which the test device repeatedly and automatically downloads, installs, launches, and removes test bundles until all tests have been run. Test results are returned to the harness using HTTP. This option is only available if the device supports the autotest protocol.

If your test device supports more than one of the options consider these factors when making your choice:

- If the test device's HTTP connection is over the air, options C and D might transfer bundles faster. However, the reverse might be true if the HTTP connection is over a local network, such as Ethernet or Wi-Fi.

  When bundles are transferred using a local link and the last test run completes, you must press the Stop button so the harness is no longer waiting for more bundles to be transferred to the test device and is no longer in a run test mode.
- One option might be easier to use, and possibly faster, because it requires fewer test device interactions (such as key presses) to download, install, and launch a test bundle. For example, the local link download software might require fewer keystrokes than downloading over the air.
- If option D is used, you must manually stop the test run when test execution completes or if a problem occurs during test execution. Press the Stop button to stop the test run.
- The autotest protocol is easy to use because the device obtains, installs, launches, and removes test bundles. However, it is usually available only on emulators or specially manufactured versions of test devices.
- The options where results are held on the device are the most inconvenient to use. Select one only if none of the "Send" options work with the test device. The options where results are held on the device do not return results to the harness, require you to manually display interactive test instructions, and only support a subset of tests.

CHAPTER 5

Running a Test

This chapter walks you through an interactive mini-tutorial involving two types of tests, automated and interactive. The purpose of the tutorial is to give you an overview of running tests using the Java Device Test Suite.

Running the tests can take approximately a half-hour. Working through the sample tests, you select, configure, and run tests. Later, you view the test results and reports that you generate.

This chapter has the following sections:

Preparing for the Quick Tests

If are you are running the Solaris operating system, choose a test device that supports the HTTP protocol.

If you are running Windows, use the Wireless Toolkit emulator (if you choose to use a test device, be sure it supports the HTTP protocol). Use Wireless Toolkit version 2.5 to run these test sets. You can obtain the Wireless Toolkit emulators from `http://java.sun.com/products/j2mewtoolkit/index.html`.

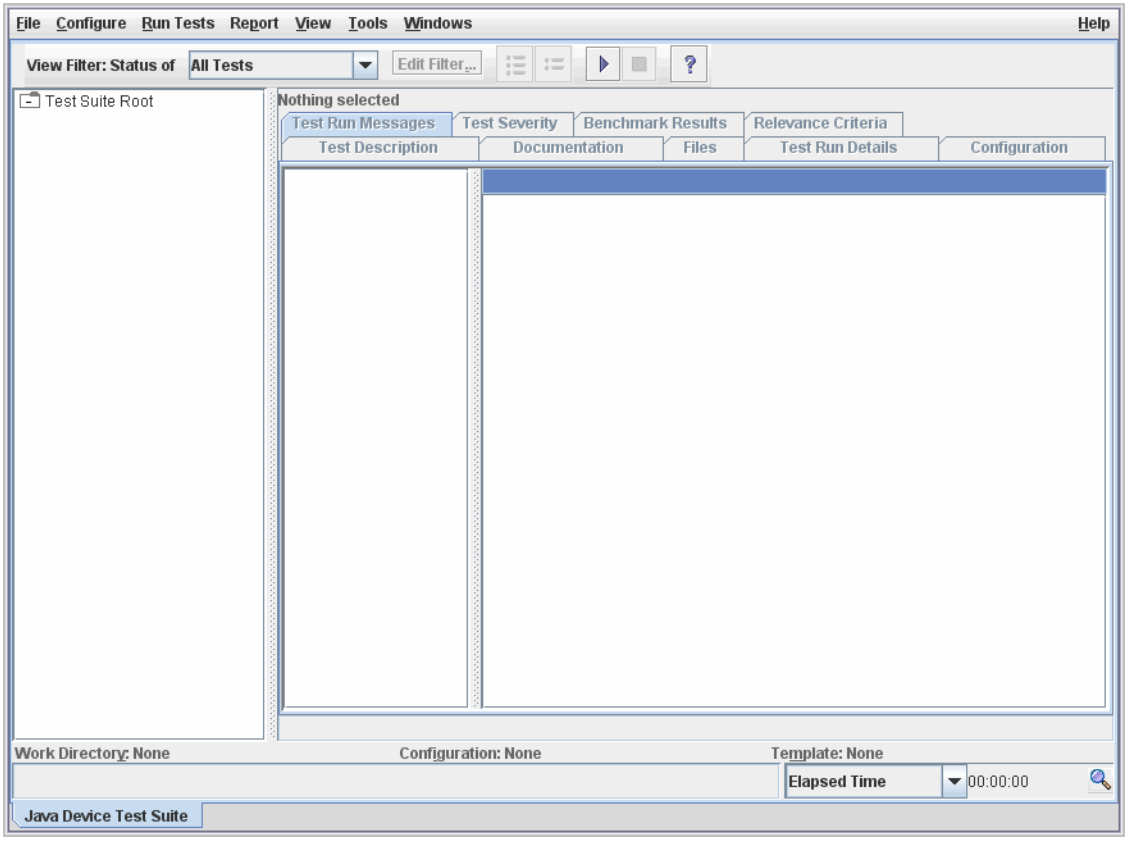
▼ Running the Test Harness and Setting Files

To be sure that your experience of the sample test runs matches the steps given in the following sections, follow these preliminary steps:

1. **Launch the harness to open the Test Manager window:**

   If you have run the harness before, the initial display might look different.

FIGURE 5-1 Test Manager Window



The left pane displays the test tree, which contains the Test Suite Root node and the right pane displays information relevant to the selected item in the test tree. See the online help for a description of the graphical user interface components.

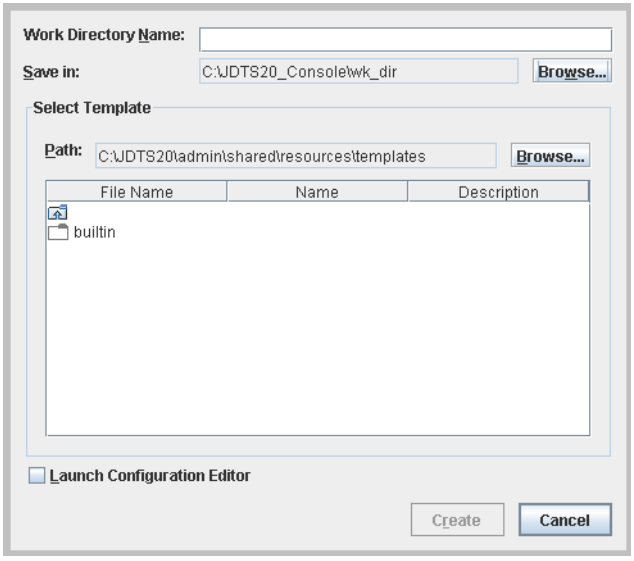
2. **Create a directory to contain the work directories and the configuration files that you use for the quick tour.**

   You can use the default location or you can create a work directory in a location that you prefer. For the purposes of this exercise, use the default location.

a. Choose File > Create Work Directory to create a work directory to use for the first sample test.

The Create Work Directory dialog box opens:

FIGURE 5-2 Create Work Directory Dialog Box



b. Enter a name for this work directory in the text field.

For instance, you can name it Sample_wd for this example.

c. Select a template.

The default location of the templates directory, *jdts_installDir*`/admin/shared/resources/templates` is already provided. If the templates are stored in another location for your situation, click Browse and use the File Chooser to select the templates directory.

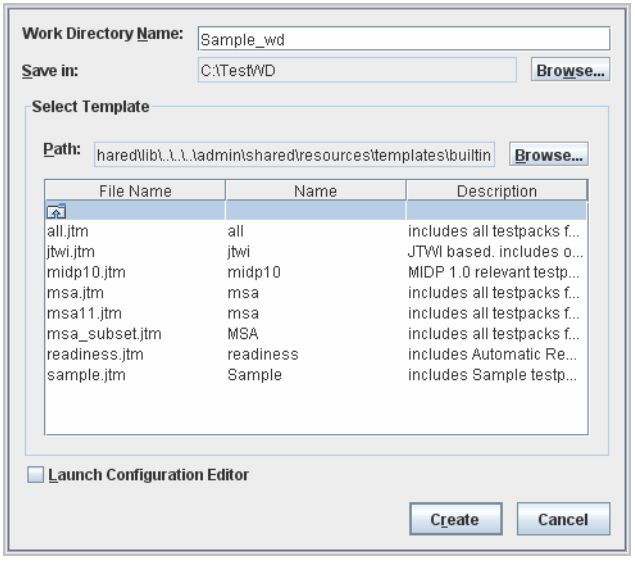
For the purposes of this tutorial, double click `builtin` in the `templates` directory.

When creating a work directory, you are asked to specify a template. The builtin templates provided with the Java Device Test Suite are *not* intended for use other than for demonstration tests. Do not use these templates when performing actual work. Updates to builtin templates do not propagate, which means that your configuration is not updated when its template is updated.

Your administrator is responsible for creating the templates you normally use for test runs. Updates to templates created by your administrator are propagated to configurations. For production work, use the templates provided by your administrator.

When you select builtin, the dialog box lists the available templates:

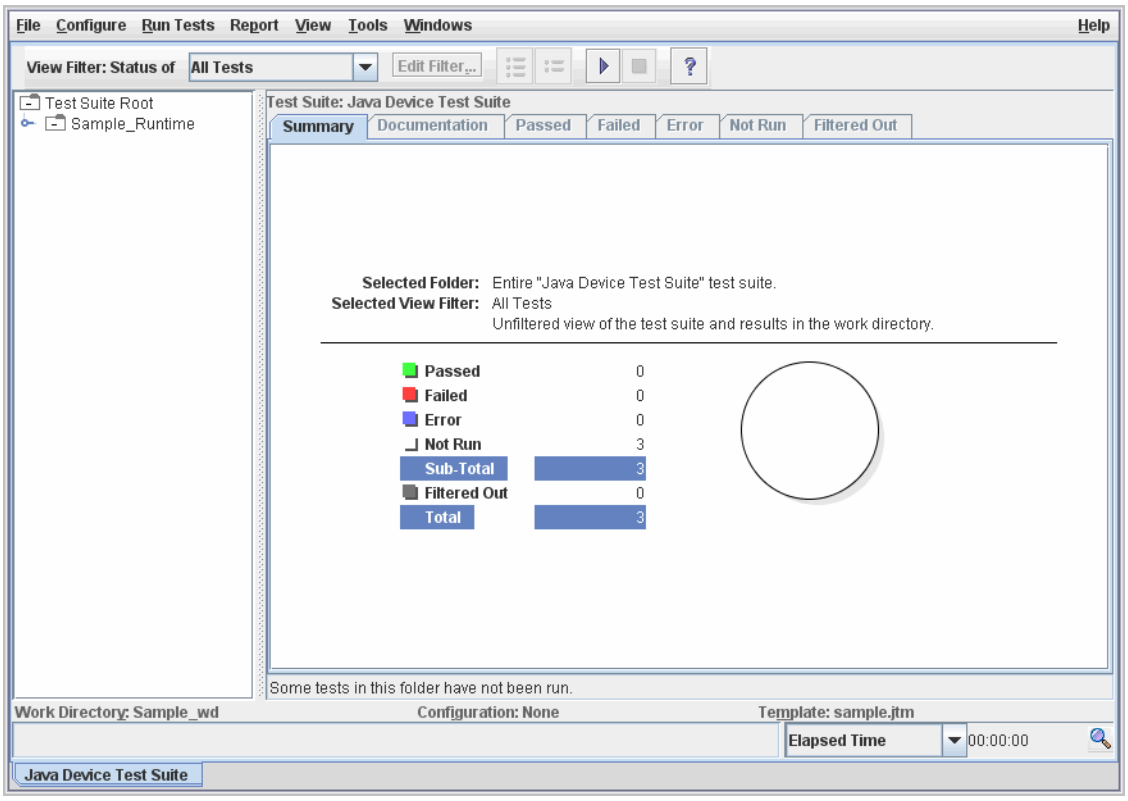
FIGURE 5-3 List of Templates



d. Select the `sample.jtm` template and click Create.

A list of test packs is displayed in the test tree:

FIGURE 5-4 New Instance of Test Manager



For templates that refer to a large number of test packs, it can take about a minute for the harness to load them.

The work directory is created. At this point you can proceed to the section, "Running Automated Tests" on page 37 or to the section, "Running an Interactive Test" on page 45 to configure and run the tests.

Running Automated Tests

Automated tests run without your intervention. They determine if the test device passed or failed. The harness captures results returned by the tests, summarizing them automatically.

1. **Create a configuration for the sample test run.**

a. **Choose Configure > New Configuration.**

   The Configuration Editor opens. The Configuration Editor presents a series of questions in an interview format. You only need to answer those questions that are relevant to your specific test.

b. **Click Next and enter a name to identify this configuration.**

   For example, you could call it `sample1`.

c. **Click Next and enter a brief description of the configuration.**

d. **Click Next until you get to the Specify Tests to Run question in the interview.**

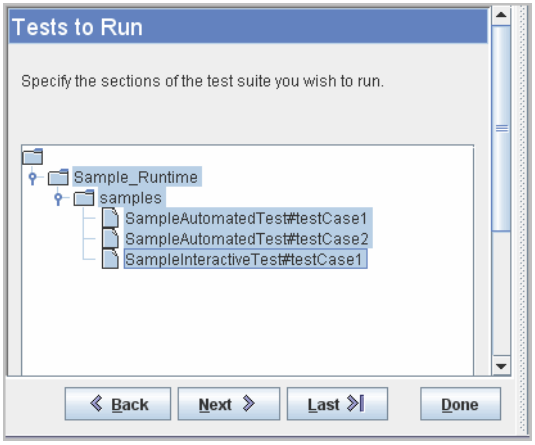
e. **Select Yes for Specify Tests to Run and click Next.**

f. **Select Directly by test or package name and click Next.**

   The Tests to Run panel of the interview is displayed.

g. **Expand the Sample_Runtime node in the Tests to Run panel.**

FIGURE 5-5 Tests to Run Panel of the Interview



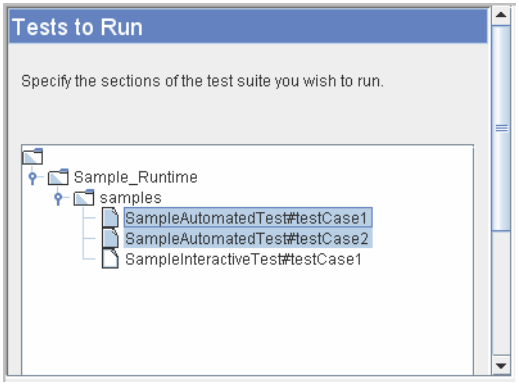
You see the following test cases:

- SampleAutomatedTest#testCase1
- SampleAutomatedTest#testCase2
- SampleInteractiveTest#testCase1

2. **Click SampleAutomatedTest#testCase1.**

3. **Shift + click to extend the selection to SampleAutomatedTest#testCase2 in the Tests to Run panel:**

FIGURE 5-6 Sample Automated Tests



If you select samples, all the test cases in samples are selected. Selecting a high-level node selects all items under that node.

You can use Ctrl + click to make discontiguous selections.

Note – Clicking a node unselects all previously selected nodes. Inadvertently clicking a node can undo a time-consuming selection task.

---

4. **Click Next until you get to the Autotest Support question.**
5. **Select No in the Autotest Support question and click Next.**

   Not all devices support the autotest protocol. In this example, you specify in the next series of questions how to send the test bundle to the device and how to send results back to the harness from the device.
6. **Select By HTTP for the means of transferring the test bundle from the harness to the device, then click Next.**
7. **For Next Bundle Auto-Request, select Yes if the device supports the MIDP 2.0 specification, then click Next.**

   This option automatically downloads test bundles.
8. **Select Yes to have test results sent by HTTP from the device to the harness.**
9. **Click Done.**

   A Save Configuration File dialog box appears.
10. **Enter a name for the configuration file, such as Sample, in the Save dialog box, then click Save File.**

    Configurations have a `.jti` extension, which is automatically added to the name you enter.

11. **Determine if you want to run tests on your device or on the Wireless Toolkit emulator.**

    To run tests on the emulator, open the Run Tests menu and make sure Run on Emulator is checked.

    If you want to run tests on your own device, uncheck Run on Emulator.

    Follow Step 12 through Step 17 if you are running tests on the emulators; otherwise, skip to step Step 18.

12. **(Emulator users only) Choose File > Preferences.**

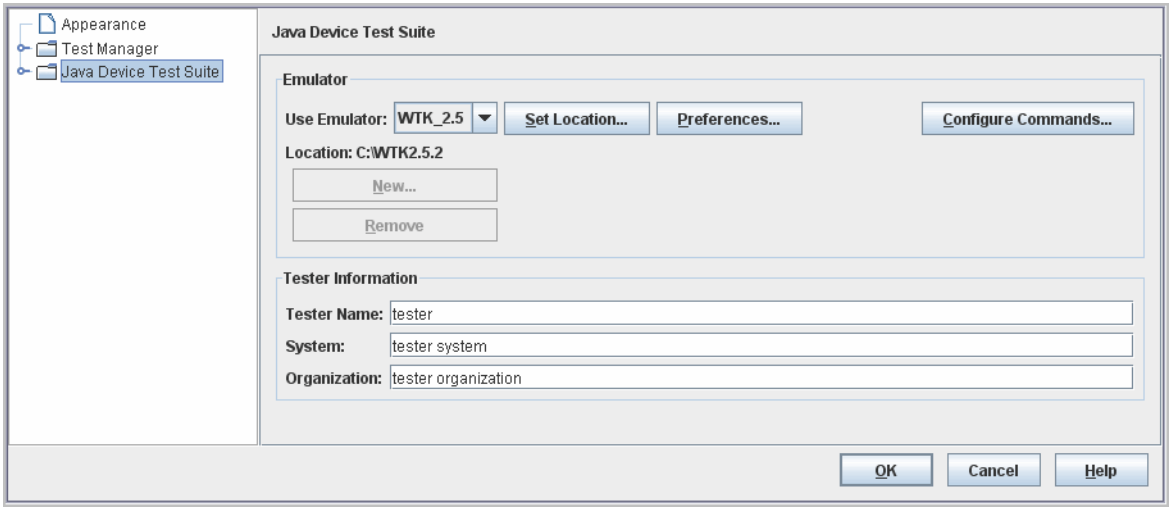
    The JavaTest Preferences dialog box opens.

    If you use the emulator, you must choose an emulator version and set its location.

13. **(Emulator users only) Select Java Device Test Suite.**

    From the drop-down list, choose WTK 2.5 for version 2.5 of the Wireless Toolkit emulator:

FIGURE 5-7 Java Device Test Suite Preferences



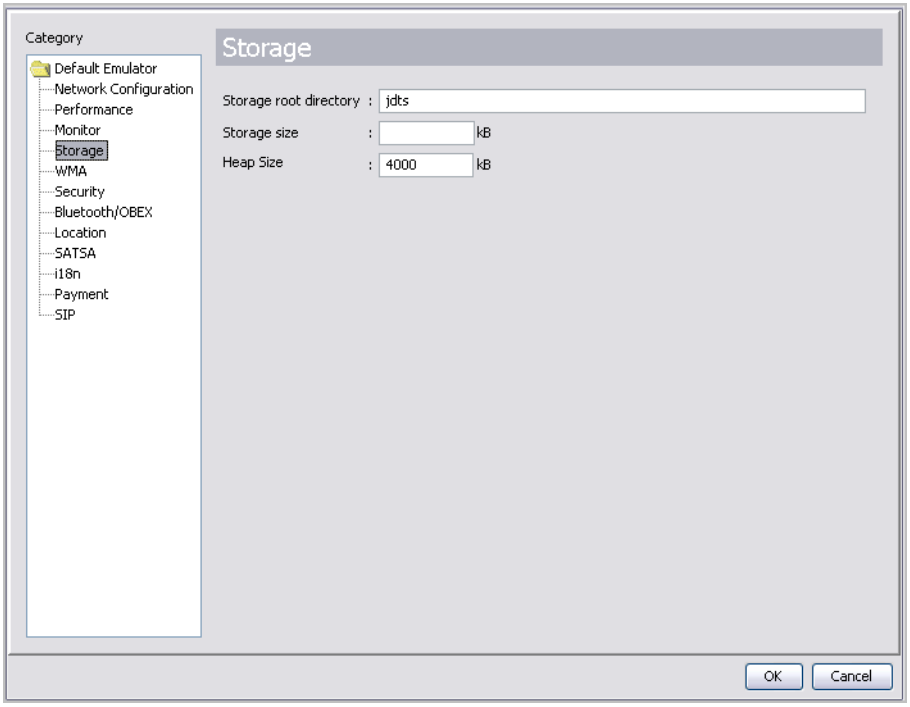
14. **(Emulator users only) Click Set Location and select the install directory for the Wireless Toolkit version.**

    Click Apply after the location is set.

15. **(Emulator users only) click Preferences, select Storage, and set the Storage Root Directory to jdts and the Heap Size to 4000:**

    Leave the Storage size field empty.

FIGURE 5-8 Emulator Preferences



16. **Click OK in the dialog box.**

17. **Click OK in the JavaTest Preferences dialog box to close it.**

18. **Choose View > Filter > Current Configuration.**

    Current Configuration enables you to see Summary and status information specifically on the tests selected for the current test run. The All Tests setting shows current totals and status icons for all the tests in the test pack, regardless of the configuration settings.

19. **Start the test device if you are not using the emulator.**

20. **Click the Start button**  **.**

    The Device Status window opens:

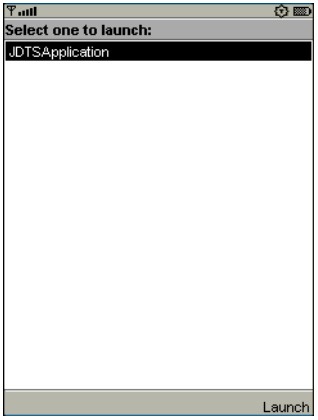
FIGURE 5-9 Device Status Window

The Launch Emulator button is disabled if Run on Emulator is not selected in the Run Tests menu.The bundle URL might be different for your installation.

21. **Click the Launch Emulator button in the Device Status window.**

In a second or two, the display screen on the device (or emulator) shows the application is loaded and ready to be launched:

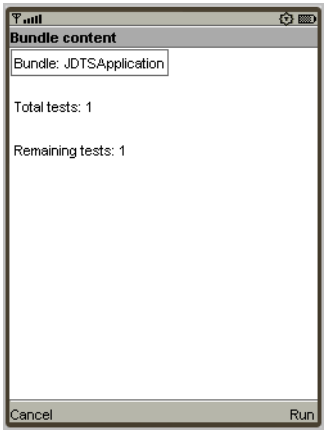
FIGURE 5-10 Application Transferred to Device



22. **Launch the application (this action is device-dependent).**

    If you are asked for permission to use air time, answer Yes. The device display screen shows that the test bundle contains one test.

FIGURE 5-11 Device Display Screen Showing Number of Tests



23. **Run the test (this action is device-dependent).**

24. **Choose Yes if you are asked if it is okay to use airtime.**

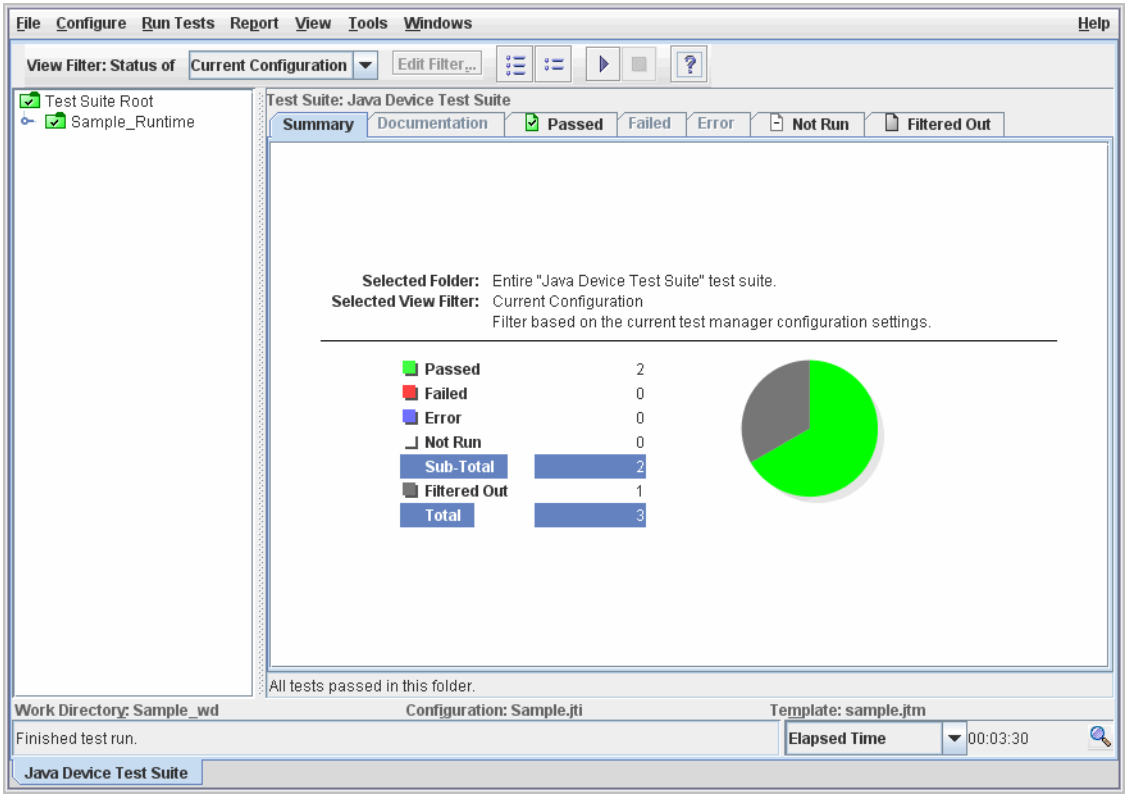
    For a moment, you can see information in the emulator's display screen that the test is running.

25. **When the test completes, close the emulator window.**

26. **Click the Launch Emulator button (**FIGURE 5-9**) again, and run the second test as you did the first.**

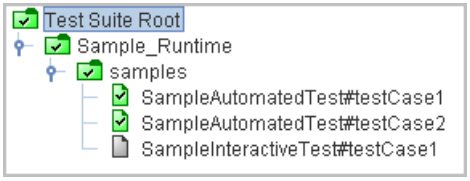
When the second test completes, the emulator exits and the harness Summary tabbed pane shows an overview of the test run:

FIGURE 5-12 Automated Test Results



Expand the Sample_Runtime node in the test tree to see that the two tests are now marked as passed:

FIGURE 5-13 Passed Test Notation in Test Tree



To see information for a specific test, select the test case in the test tree then click a tabbed pane. See the online help for a description of the information shown in the tabbed panes.

Running an Interactive Test

Interactive tests involve some action on the tester's part. A Test Evaluation window appears with instructions for you to perform. For this sample test, you simply decide whether the tests pass or fail. For more information on the Test Evaluation window, see the online help.

Note – If you have previously run tests using an emulator, clear the generated files in the Wireless Toolkit *install_dir*`/appdb/jdts` directory before proceeding with the test.

To run an interactive test, follow these steps:

1. **Create a configuration for the sample test run.**

   a. **Choose Configure > New Configuration.**

      The Configuration Editor opens. The Configuration Editor presents a series of questions in an interview format. You only need to answer those questions that are relevant to your specific test.

   b. **Click Next and enter a name to identify this configuration.**

      For example, you could call it sample1.

   c. **Click Next and enter a brief description of the configuration.**

   d. **Click Next until you get to the Specify Tests to Run question in the interview.**

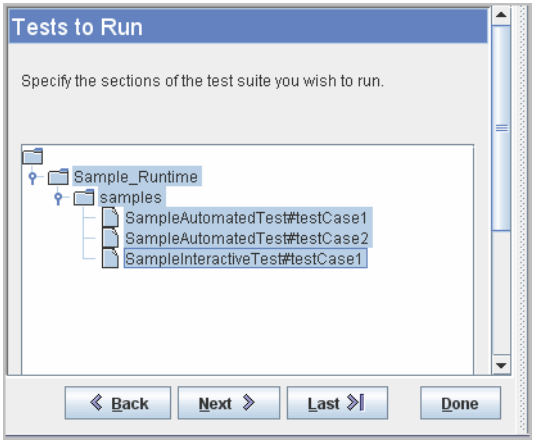
   e. **Select Yes for Specify Tests to Run and click Next.**

   f. **Select Directly for How to Specify Tests and click Next.**

      The Tests to Run panel of the interview is displayed.

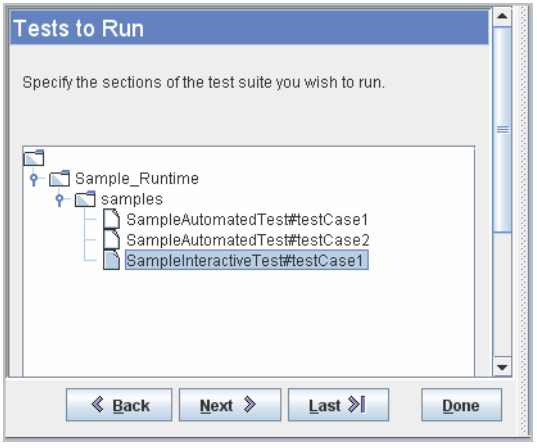
   g. **Expand the Sample_Runtime node in the Test to Run panel.**

FIGURE 5-14 Tests to Run Panel of the Interview



2. **In the test tree, expand the Sample_Runtime node until you see the following test cases:**
   - SampleAutomatedTest#testCase1
   - SampleAutomatedTest#testCase2
   - SampleInteractiveTest#testCase1
3. **Click SampleInteractiveTest#testCase1 to select it in the Tests to Run panel:**

FIGURE 5-15 Sample Interactive Test



If you select samples, all the test cases in samples are selected. Selecting a high-level node selects all items under that node.

Note – Clicking a node unselects all non-subordinate nodes. Inadvertently clicking a node can nullify a time-consuming selection.

4. **Click Next until you get to the Autotest Support question.**
5. **Select No in the Autotest Support question and click Next.**

   Not all devices support the autotest protocol. In this example, you specify in the next series of questions how to send the test bundle to the device and how to send results back to the harness from the device.
6. **Select By HTTP for the means of transferring the test bundle from the harness to the device and click Next.**
7. **Select Yes for Next Bundle Auto-Request if the device supports the MIDP 2.0 specification, then click Next.**

   This option automatically downloads test bundles.
8. **Select Yes to have test results sent by HTTP from the device to the harness and click Next.**
9. **Select One test per bundle.**

   The test is placed in a single test bundle.
10. **Click Done.**

    A Save Configuration File dialog box appears.
11. **Enter a name for the configuration file, such as Sample, in the Save dialog box, then click Save File.**
12. **Determine if you want to run tests on your device or on the Wireless Toolkit emulator.**

    To run tests on the emulator, open the Run Tests menu and make sure Run on Emulator is checked.

    If you want to run tests on your own device, click Run on Emulator to uncheck it.

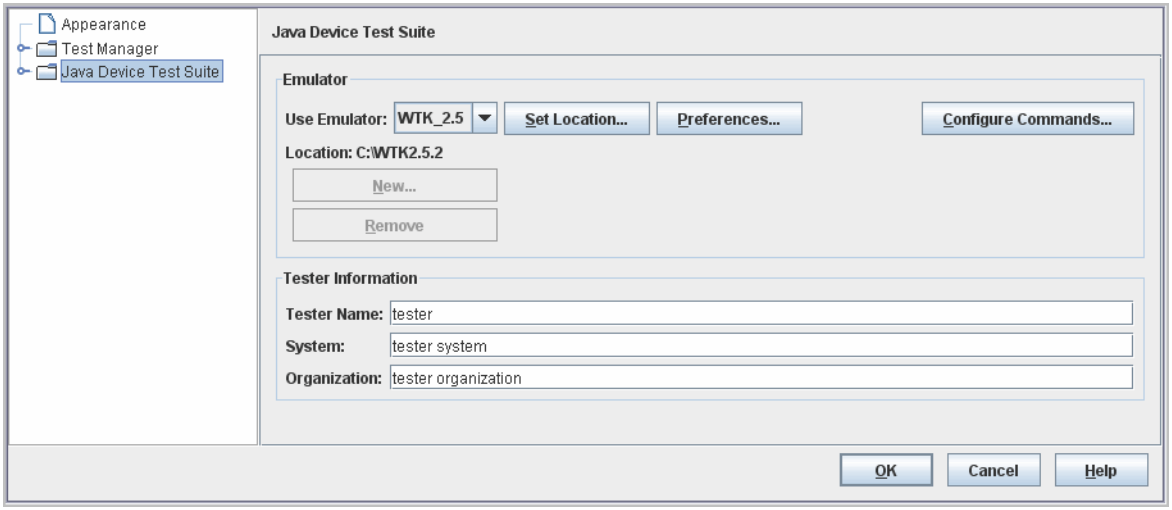
    Follow Step 12 through Step 18 if you are running tests on the emulators; otherwise, skip to Step 18.
13. **(Emulator users only) Choose File > Preferences.**

    The JavaTest Device Test Suite Preferences dialog box opens.

    If you use the emulator, you must choose an emulator version and set its location.
14. **(Emulator users only) Select Java Device Test Suite.**

    From the drop-down list, choose WTK 2.5 for version 2.5 of the Wireless Toolkit emulator.

FIGURE 5-16 Java Device Test Suite Preferences Dialog Box



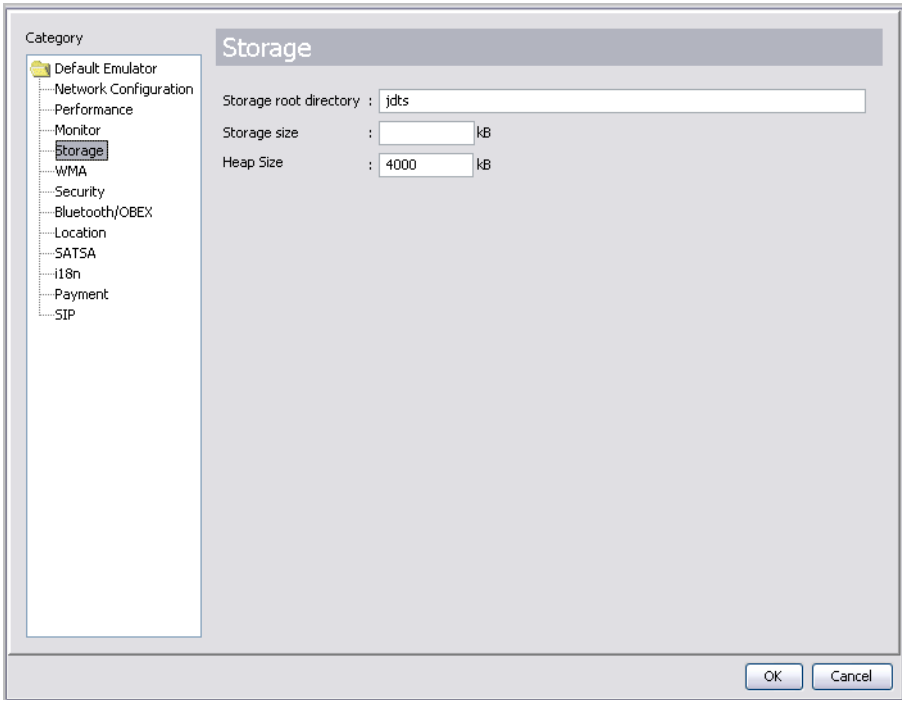
15. **(Emulator users only) Click Set Location and select the install directory for the Wireless Toolkit version.**

    Click Apply after the location is set.

16. **(Emulator users only) click Preferences and set the Storage Root Directory to jdts and the Heap Size to 4000:**

    Leave the Storage size field empty..

FIGURE 5-17 Emulator Preferences



17. **Click OK in the dialog box.**

18. **Click OK in the JavaTest Preferences dialog box to close it.**

19. **Choose View > Filter > Current Configuration.**

    Current Configuration enables you to see Summary and status information specifically on the tests selected for the current test run. The All Tests setting shows current totals and status icons for all the tests in the test pack, regardless of the actual configuration settings.

20. **Start the test device if you are not using the emulator.**

21. **Click the Start button .**

    The Device Status window opens:

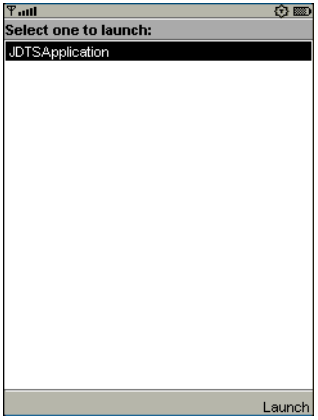
FIGURE 5-18 Device Status Window

The Run Emulator button is disabled if Run on Emulator is not selected in the Run Tests menu.The bundle URL might be different for your installation.

22. **(Emulator users only) Click Run Emulator in the Device Status window:**

    In a second or two, the display screen on the device (or emulator) shows the application is loaded and ready to be launched:

FIGURE 5-19 Application Transferred to Device



23. **Launch the application (this action is device-dependent).**

    If you are asked for permission to use air time, grant it. The device display screen shows that one bundle is loaded and a test is ready to run.

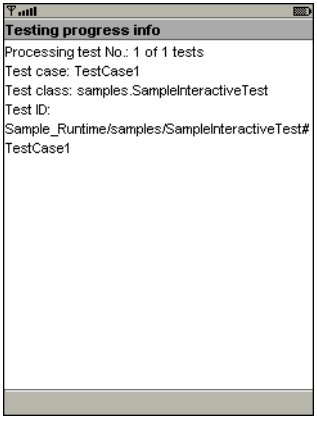
FIGURE 5-20 Device Display Screen Showing Number of Tests

24. **Run the tests (this action is device-dependent).**

25. **Choose Yes if asked if it is okay to use airtime.**

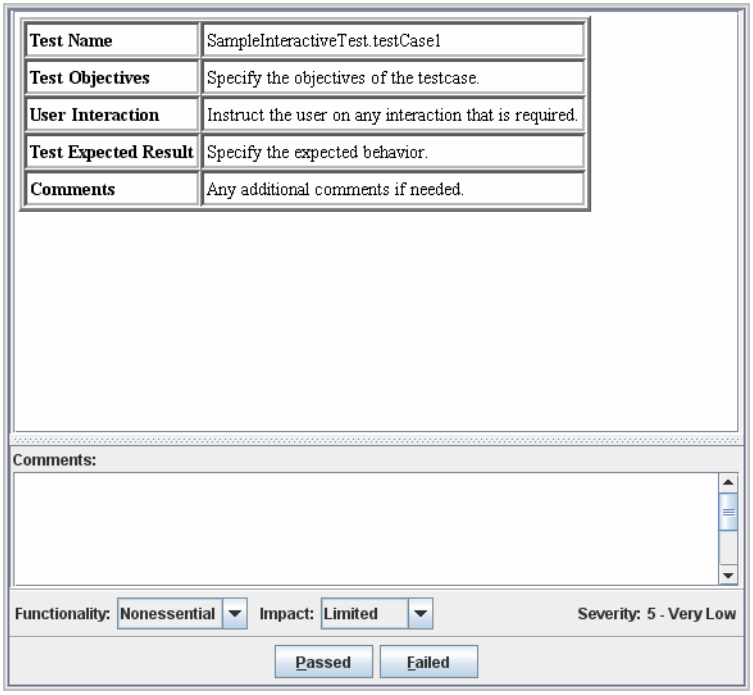
    The device screen display shows one test has run:

FIGURE 5-21 Device Display Screen Showing One Test Running



A test evaluation window similar to FIGURE 5-22 soon appears. In a real interactive test, this window instructs you to interact with or inspect the test device for some appearance or behavior, and then click the Passed button if the device behaves correctly or the Failed button if it does not. You can also record a comment typically to note the reason for the failure. testCase1, however, does nothing to the test device, and, therefore, has placeholder instructions.

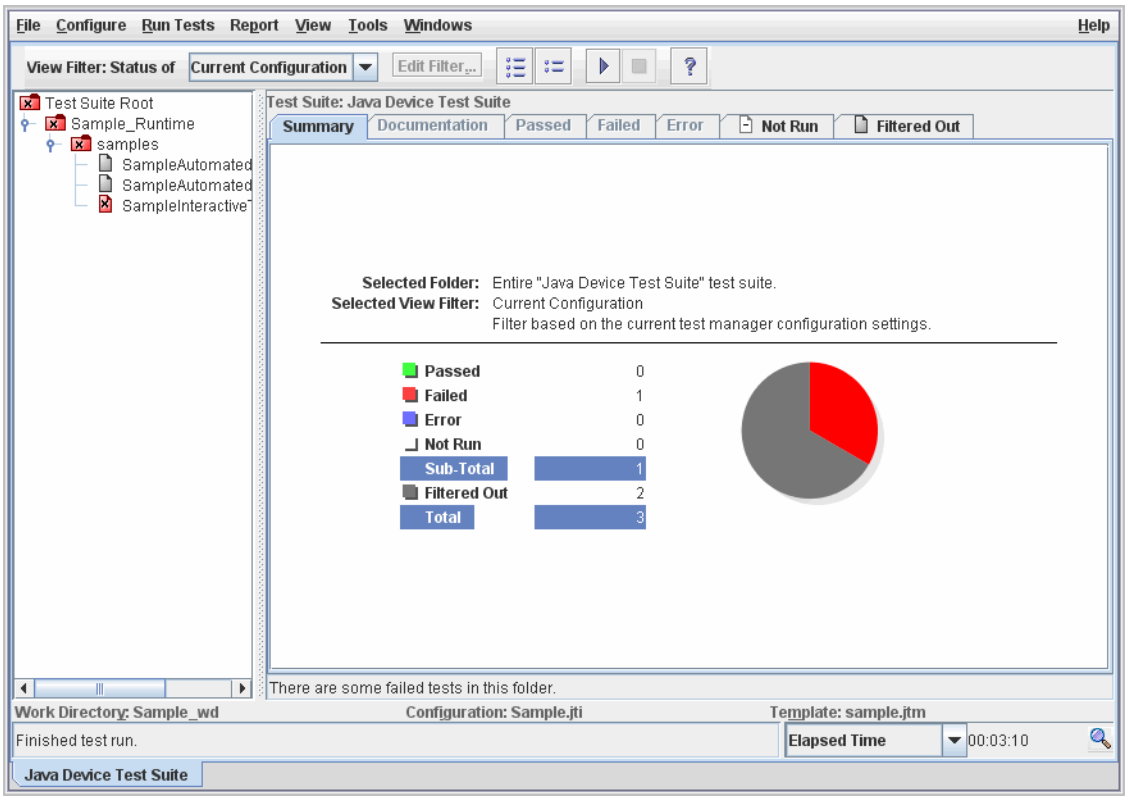
FIGURE 5-22 Test Evaluation Window



26. In the Comments pane, enter "Failed for demonstration", then click Failed.

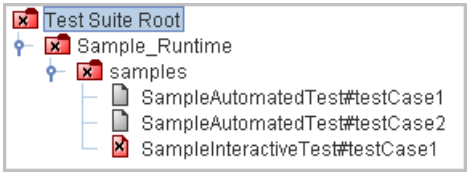
The test run ends and the number of failed tests is shown in the Summary tabbed pane:

FIGURE 5-23 Failed Test Results



Expand the Sample_Runtime node in the test tree, if necessary, to see that the interactive test is now marked as failed:

FIGURE 5-24 Failed Test Notation in Test Tree



The automated tests are shown as Filtered Out because they are not selected in the configuration's Tests To Run question.

You have completed the walk through of running basic tests in the Java Device Test Suite. To see information about the test runs, click on the tabs in the test information display pane.

Selecting Tests by Device Feature and Severity

In the Java Device Test Suite, there are several ways to specify the tests that run. You can use them separately or in combination. In the previous exercises, you selected tests directly by name. In this exercise you select them by a combination of device feature and test failure severity. For a description of device features, refer to "Device Features" on page 7. For a description of test failure severity, refer to "Test Failure Severity" on page 83.

1. **In the harness test tree, select Test Suite Root, right-click, and choose Clear Results.**

   This operation sets the status of all tests to Not Run.

2. **In the harness, choose Configure > Edit Configuration.**

   The Configuration Editor appears.

3. **Select the Test Selection: How to Specify Tests question.**

   If you do not see the question, select Test Selection: Specify Tests to Run, and click Yes.

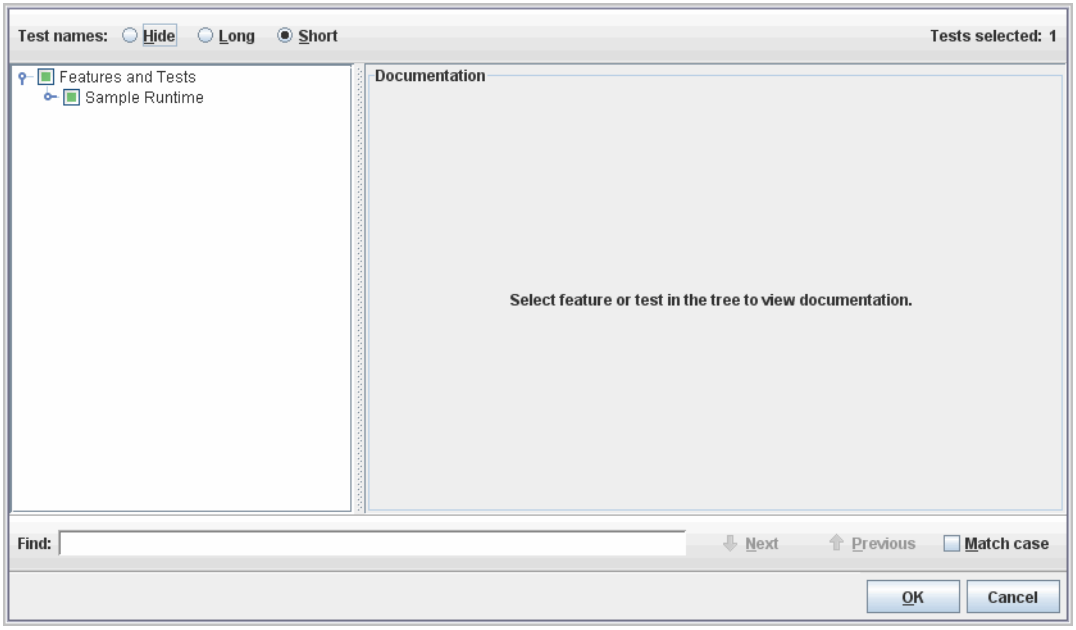
4. **Select "Directly by feature name", then click Next >.**

   The Features to Run question appears.

5. **Click Feature Tree.**

   The feature tree appears, similar to FIGURE 5-25.

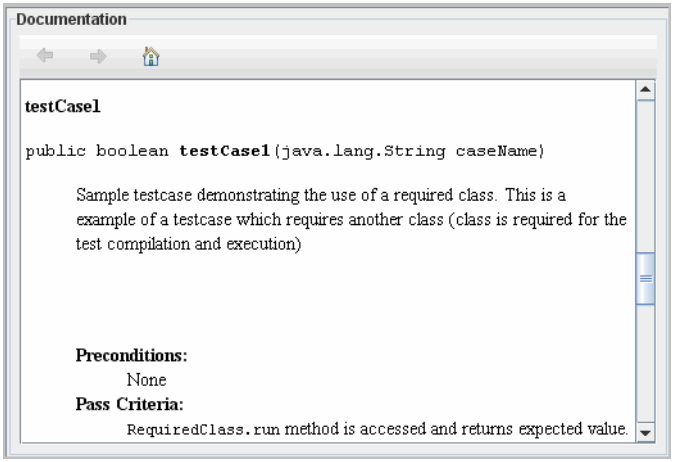
FIGURE 5-25 Feature Tree



6. **Select Sample Runtime.**

   A description of the Sample Runtime feature appears on the right. When you select a test case, its documentation appears on the right. FIGURE 5-26 shows an example.

FIGURE 5-26 Test Case Documentation Example

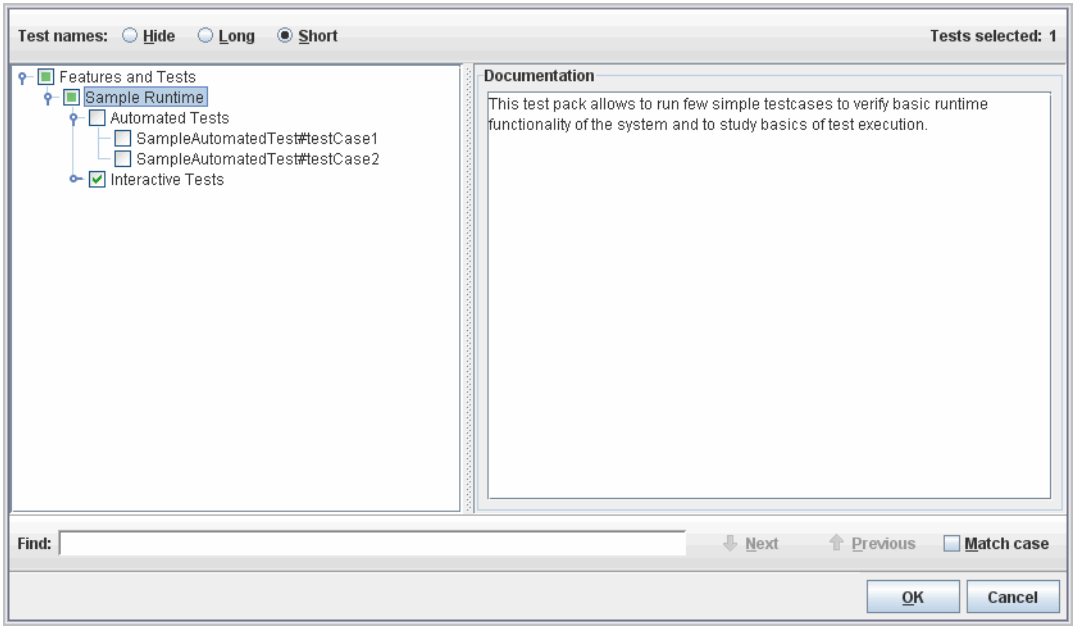


7. **Click the turner next to Sample Runtime to expose its features Automated Tests and Interactive Tests.**

   Because these are artificial sample tests, the feature names are not representative.

8. **Click the turner next to Automated Tests to expose its features or test cases.**

FIGURE 5-27 Feature Tree with Tests



Notice that the test case names (such as SampleAutomatedTest#testCase1) are the same as those in the Tests to Run tree (FIGURE 5-14). You can select tests in either tree or both.

9. **Uncheck the box next to Automated Tests, then check the box next to SampleAutomatedTest#testCase2.**

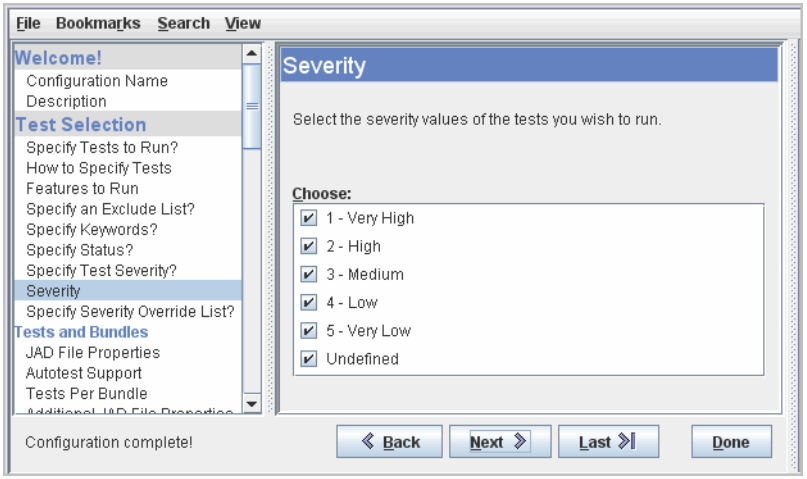
   Feature selection is "sticky". There is no need to use the Ctrl or Shift key to select multiple features. Selecting a feature selects its sub-features and tests. The total number of tests you have selected is shown in the upper right.

10. **Click OK to close the feature tree.**

11. **In the configuration editor, select the Specify Test Severity question, select Yes, then click Next >.**

    The Severity question appears, similar to FIGURE 5-28.

FIGURE 5-28 Severity Question



12. **Uncheck all boxes except 5 - Very Low.**

    The sample tests have this severity value. For more information on severities, see "Test Failure Severity" on page 83. You can see a test case's severity by selecting it in the harness test tree and clicking the Test Severity tab.

Note – The answers to the questions in the Test Selection section act as a series of filters that are AND-ed together. A test runs only if it passes all filters. For example, suppose a test in the Interactive Tests feature you left selected in Step 9 had a severity of 1 - Very High. This test will *not* run because it does not pass the Severity filter as it is currently configured (to pass only tests whose severity is 5 - Very Low). Similarly, no test in the Automated Tests feature will run because that feature was unselected in Step 9.

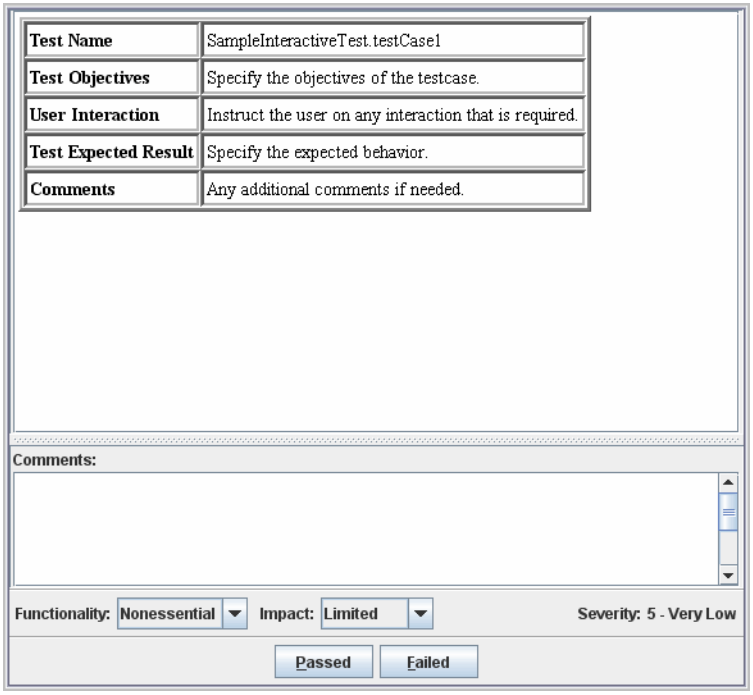
---

13. **Click Done.**

14. **Start the test run as you did in** "Running Automated Tests" on page 37 **or** "Running an Interactive Test" on page 45**.**

    When the interactive test begins to run, the evaluation window appears, similar to FIGURE 5-29.

FIGURE 5-29 Test Evaluation Window



15. **Choose Secondary in the Functionality drop-down.**

    Severity changes from 5 - Very Low to 4 - Low. You can change the severity of interactive and tests as they run. The change you have made indicates that you rank this test's functionality as more important than the test designer did. Accordingly, its failure severity rises.

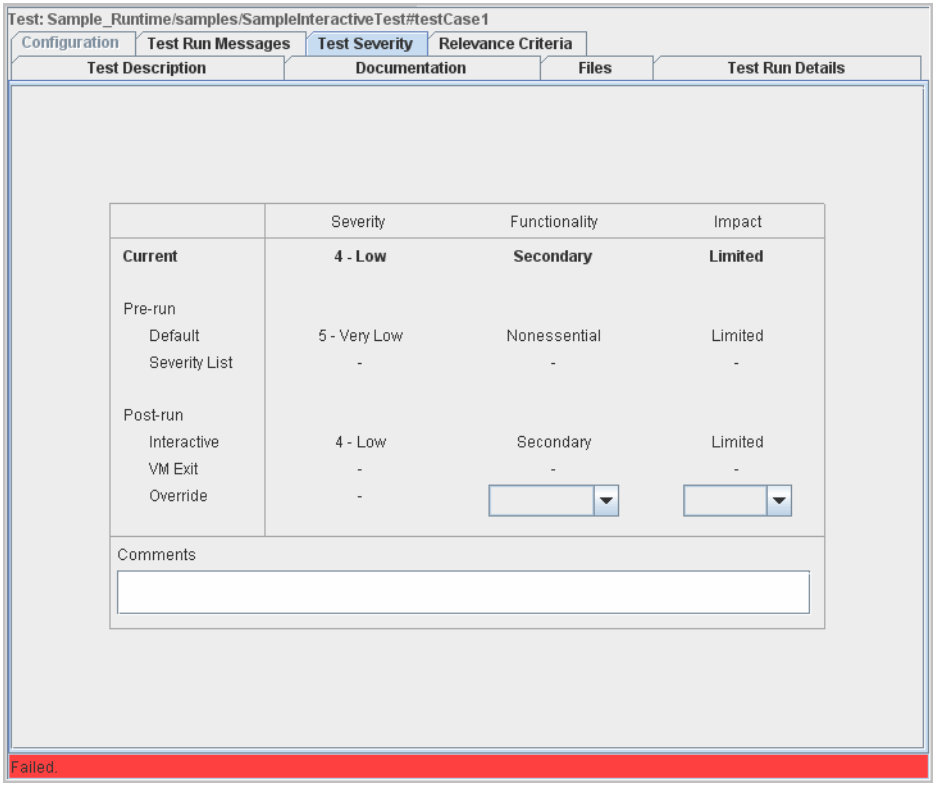
16. **Click Failed.**

    The test run ends.

17. **In the harness test tree, select SampleInteractiveTest#testCase1 and click the Test Severity tab.**

    FIGURE 5-30 shows that your action changed the test severity from 5 to 4.

FIGURE 5-30 Test Severity Tab



You can change the severity of any test after it has run by selecting new values in the Functionality or Impact drop-downs. Post-run severity changes are erased if the test is run again.

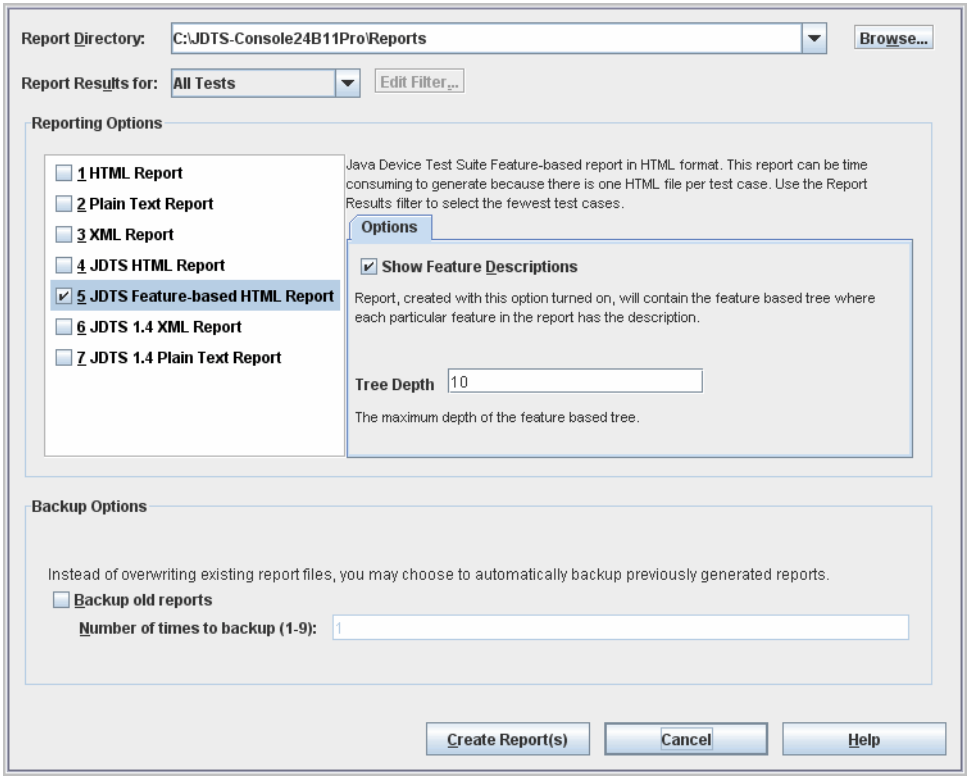
Creating Feature and Severity Reports

After a test run, the Java Device Test Suite can generate several different kinds of reports. In this section, you create a feature report and a severity report. To create a feature report, follow these steps.

1. **In the harness, choose Report > Create Report.**

   The Create a New Report dialog box appears, similar to FIGURE 5-31.

FIGURE 5-31 Create a New Report Dialog Box



2. **In the Report Directory area, browse to or type the name of a directory to contain report files, for example,** `C:\JDTSSampleReports`**.**

3. **In the Report Results for drop-down, select All Tests.**

   You can use the Report Results filter to specify a subset of features, such as those selected by the current configuration.

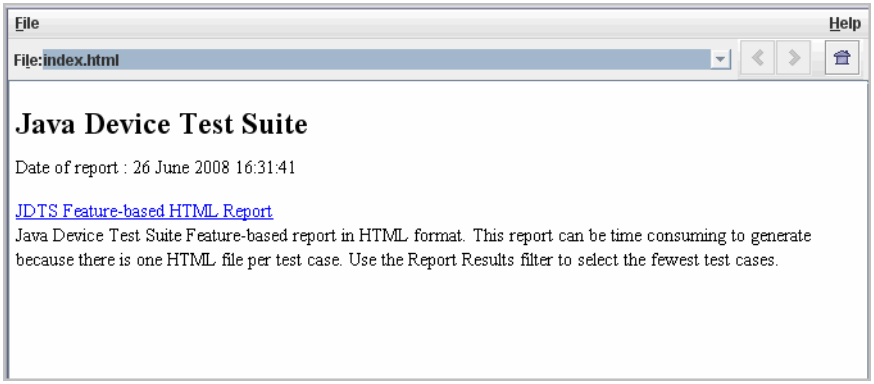
4. **Click the box next to 5 JDTS Feature-based HTML Report.**

5. **Click Create Report(s).**

6. **When the View Report dialog box appears, click Yes.**

   You can also view reports with a web browser. The report browser appears, similar to FIGURE 5-32.

FIGURE 5-32 Report Browser First Page



7. **Click JDTS Feature-based HTML Report.**

   The report browser displays the report summary page similar to FIGURE 5-33. The features in the Sample Runtime test pack do not have typical names.

FIGURE 5-33 Sample Feature Report

A feature report is organized by device feature. Graphical bars visually indicate the percentage of passed, failed, and not run tests in each feature. To minimize the size of the report, features that have no Passed or Failed tests are not displayed, and their Not Run totals are added to their parent feature. To see report details, click an underscored number.

8. **Close the report browser.**

To create a severity report, follow these steps:

1. **In the harness, choose Report > Create Report.**

   The Create a New Report dialog box, similar to FIGURE 5-31, appears.

2. **Check JDTS HTML Report.**

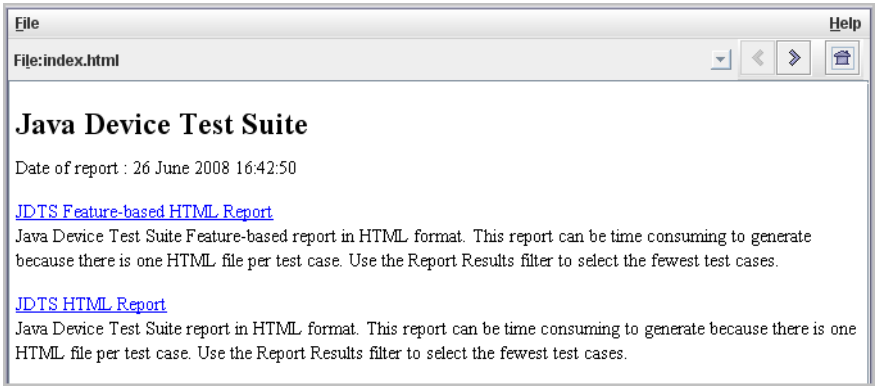
3. **In the Options tab, check Consider Tests Severities.**

4. **Click Create Report(s).**

5. **When the View Report dialog box appears, click Yes.**

   The first page of the report appears, similar to FIGURE 5-34.

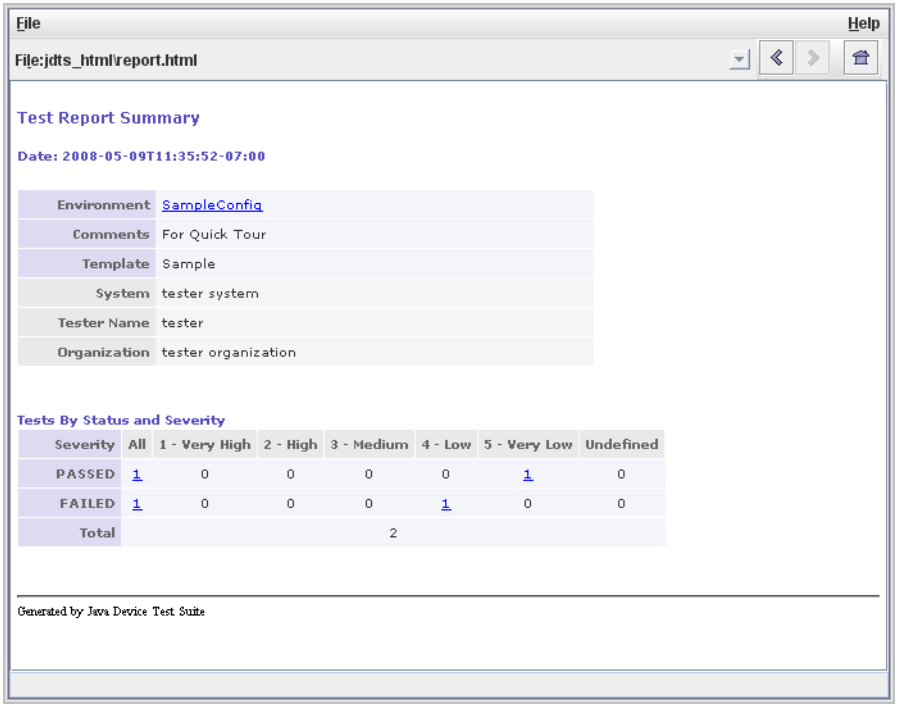
FIGURE 5-34 First Page of Multiple Reports



6. **Click JDTS HTML Report.**

   The report organized by severity appears, similar to FIGURE 5-35.

FIGURE 5-35 Report by Severity



To explore the report details, click an underscored number.

CHAPTER 6

Interpreting Benchmark Statistics

The test manager information pane displays benchmark information in the Benchmark Results tabbed pane. Different types of tests return different statistics. Test runs that have corresponding threshold values show the values produced by both the test device and the reference device (as possibly edited by an administrator).

This chapter describes benchmark results in the following sections:

- Unit Rate Test Statistics
- System Load Test Statistics

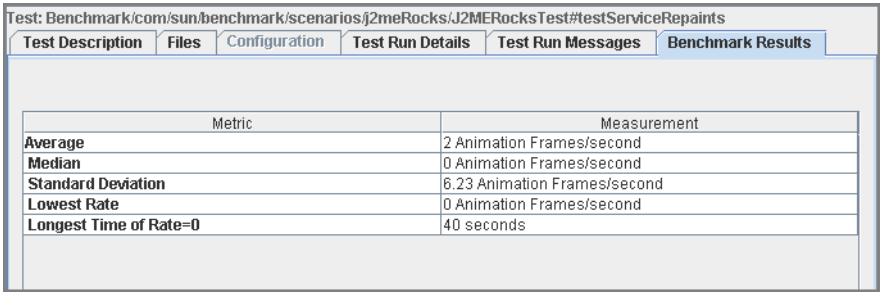
For information on running benchmark tests and the Benchmark Results tabbed pane, see the online help.

Unit Rate Test Statistics

Most benchmark tests measure Unit Rate. A Unit Rate test measures the rate at which an important and ongoing unit of work, such as displaying a frame of animation, is completed. A Unit Rate test runs for one minute. Every 100 milliseconds, it records the number of operations completed in the last 100 milliseconds. The result is an array of 600 samples, one for each of the 100-millisecond intervals in one minute.

FIGURE 6-1 shows an example of Unit Rate results when a test is run in a session whose profile has no corresponding threshold value. In this example, the Unit Rate is animation frames per second.

FIGURE 6-1 Unit Rate Performance Statistics in Benchmark Results Tab

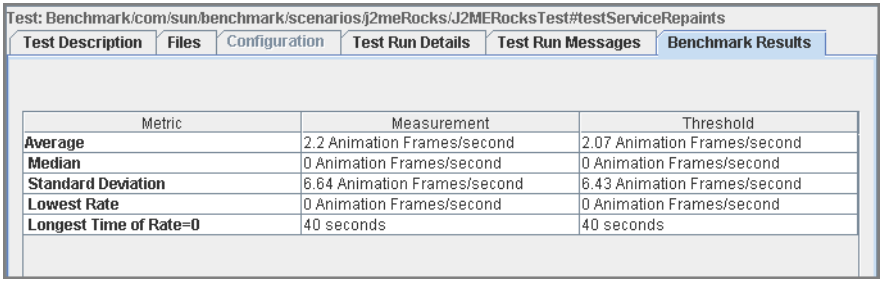


The test returns the array to the harness. Ignoring the first 150 samples, which are subject to warm-up effects such as optimization and class loading, the harness computes and displays the following for the remaining 450 samples:

- Average
- Median
- Standard Deviation
- Lowest Rate, which is the average work done in the half second with the least work completed
- Longest Time of Rate=0, which is the longest number of seconds in which the test did not complete at least one unit of work

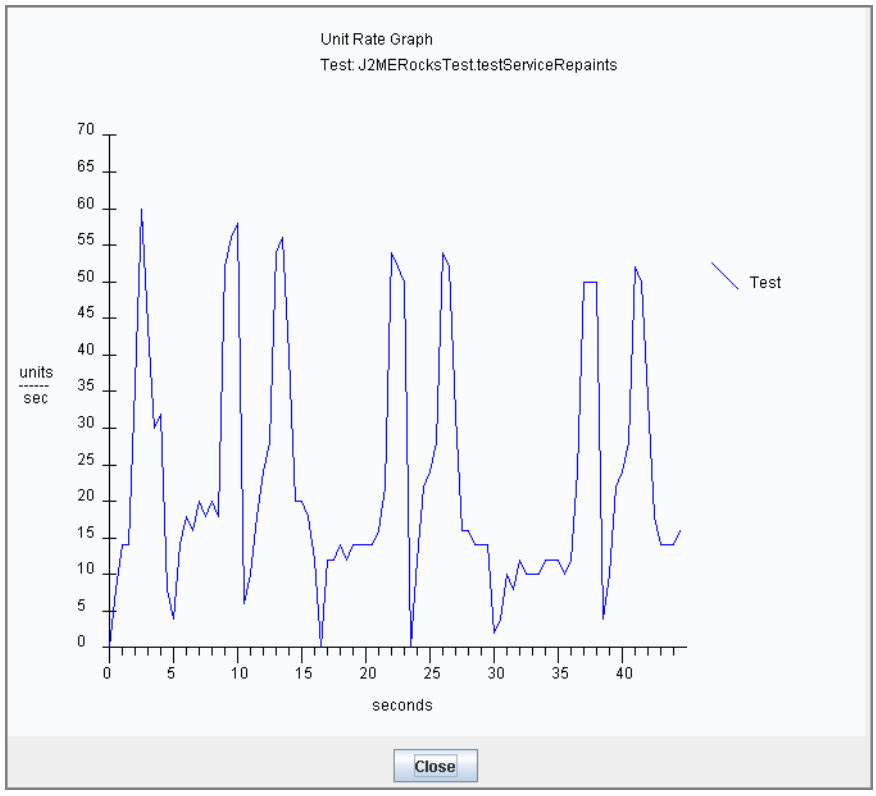
When a threshold file exists for the same test, the Benchmark Results tab shows both the test result and the threshold value (the result produced by the reference device, possibly as edited by an administrator). FIGURE 6-2 shows an example.

FIGURE 6-2 Example Benchmark Results Tab with Threshold



Click the View Graph button below the statistics table to display the test's second-by-second performance. FIGURE 6-3 shows an example.

FIGURE 6-3 Example Unit Rate Performance Graph



The graphs of Unit Rate tests for an implementation that passes or fails (based on existing threshold values) compare the performance of the test device to the threshold. FIGURE 6-4 is an example of a passing test's performance graph. The graph does not explicitly show why the implementation passed, but gives insight into its second-by-second performance compared to the performance of the reference device (as possibly adjusted by an administrator) on which the threshold is based. You can see that the test device's performance (Test line) is generally above the threshold. For an explanation of the pass or fail calculation, see "Pass or Fail Calculation" on page 70.

FIGURE 6-4 Example Passing Performance Graph

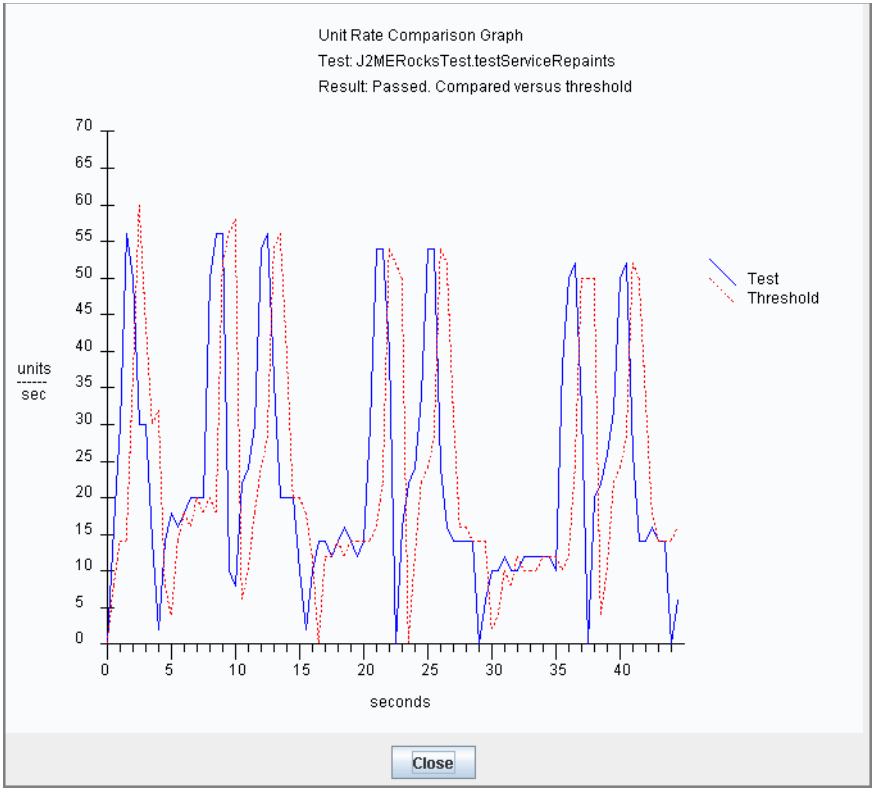
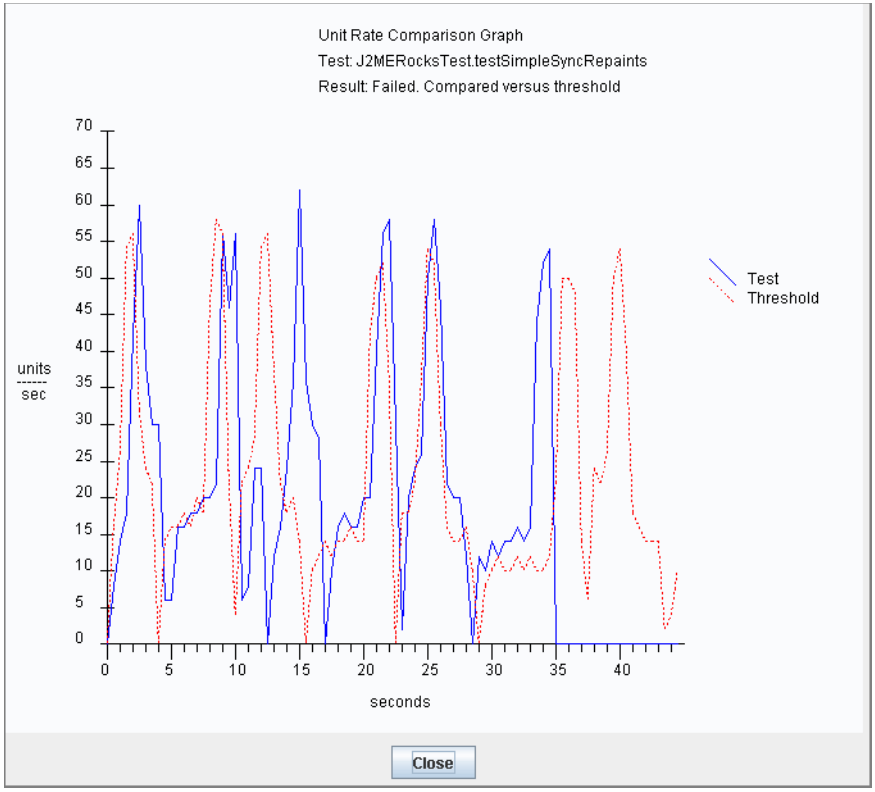


FIGURE 6-5 shows an example graph from an implementation that failed. Notice that the test device's performance (Test line) is generally lower than the threshold line.

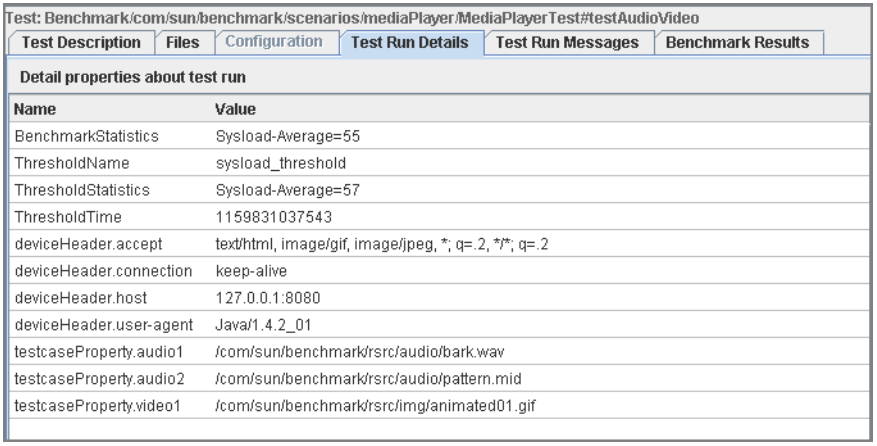
FIGURE 6-5 Example Failing Performance Graph



System Load Test Statistics

System Load tests measure an artificial representation of the load they place on the test device. Lower values are better. If a System Load test has a threshold value, the Test Run Details tab shows both the value returned by the test device and the value returned by the reference device (as possibly edited by an administrator). FIGURE 6-6 shows an example measurement. System Load tests do not have performance graphs.

FIGURE 6-6 Example System Load Information in Test Run Details Tab



A System Load test measures how much of the device's ability to do concurrent work remains while the test is running. A System Load test compares the work accomplished by a test-independent thread first running by itself and then in competition with the test. The object is to determine how much the test degrades the performance of the independent thread. The test performs time-constant operations such as playing a video file. Because the video frames must be displayed at a protocol-determined rate, the test performs the same number of operations per unit of time whether it runs on a fast device or a slow one. System Load tests automatically run multiple times and return the average System Load. The harness displays this average.

Pass or Fail Calculation

If a benchmark test has a corresponding threshold (in the work directory), the harness uses it to determine if the test passes or fails. The calculation is different for System Load and Unit Rate tests.

Tests that Measure System Load

If the test's System Load value is less than or equal to the threshold value plus a small buffer, the test passes. Otherwise it fails.

Tests that Measure Unit Rate

The Unit Rate pass or fail calculation compares the sample arrays produced by the test device and the reference device (as possibly modified by an administrator). The calculation has two phases. In both phases, the first 150 (of 600) samples in the arrays are ignored because they are subject to warm-up effects such as optimization.

In the first phase, the harness attempts to determine if the test device is in fact the reference device. Running a reference device against itself should produce a passing result, even if the arrays are not identical. This phase compares the average and variance of the two arrays. If they are both close, the test passes.

The second phase is only executed for a test device that the harness judges is not the reference device. In this phase, the harness uses a statistical technique called the Sign Test to determine if the test sample is significantly worse (in the statistical sense) than the threshold array. If the test sample is significantly worse, the test fails. Otherwise, it passes.

CHAPTER 7

Readiness Tests

If you have trouble running tests, use the readiness tests to verify that the device has the core capabilities required to run tests and that they are operational. Readiness tests take a "lowest common denominator" approach to downloading test bundles, concentrating on over-the-air (OTA) transfer, which all test devices must be able to perform. Some devices might also be able to download tests over a local link, such as a serial cable, or TCP/IP. These options, which can be faster or easier to use than OTA, are described in Chapter 4.

Preparing

Before you can run the readiness tests, you must know how to download (OTA), install, and remove MIDlet suites, and how to launch MIDlets on the test device. A test bundle is a MIDlet suite. For OTA tests, the MIDlet suite contains only one test. The agent that manages the test is the MIDlet. In particular, you must know if the test device downloads MIDlet suites over the air in one of the following ways:

- By a link to the MIDlet suite's Java Application Descriptor (JAD) file in an HTML page that you display with the device's web browser
- By a link to the MIDlet suite's JAD file in a WML web page that you display with the device's web browser
- By the JAD file name you enter into the device

To learn more about the concepts and terms of the Java Device Test Suite, see the online help in addition to the rest of this guide.

Note – Run the tests one at a time as described in the instructions.

▼ Verifying Bundle Capacity

A test device that can download MIDlet suites (test bundles) that are up to 128 kilobytes in size can run most tests. The smallest tests fit in bundles of about 75KB. A few MMAPI tests require bundles as large as 1300KB. 128KB is a good base and enables multiple small tests to be packed into a single bundle, which reduces traffic between the test device and the Relay. The test128K test verifies that a test device can download and install a 128KB bundle.

1. **Determine if the device supports CLDC version 1.0 or 1.1 and MIDP version 1.0 or 2.0.**
2. **Launch the harness.**
3. **Close the current test suite.**
4. **Choose File > Open > Open Test Suite and select a test suite from the browser dialog box.**
5. **Choose File > Create Work Directory.**

   Provide a name for the work directory and set the path of the directory.
6. **For the purposes of this test, select the** *jdts_installDir*`/admin/shared/resources/templates/builtin/readiness.jtm` **template.**

   When creating a work directory, you are asked to specify a template. The builtin templates provided with the Java Device Test Suite are not intended for use other than for demonstration tests. Do not use these templates when performing actual work. Updates to builtin templates do not propagate, which means that your configuration is not updated when its template is updated.

   Your administrator is responsible for creating the templates you normally use for test runs. Updates to templates created by your administrator are propagated to configurations. For production work, use the templates provided by your administrator.
7. **Choose Configure > New Configuration.**

   Enter a name and a brief description of the configuration and click Next to proceed to the Tests Selection section.
8. **Specify Yes to run only selected tests in the Specify Tests to Run question in the Configuration Editor.**
9. **Select Directly to specify the test from the test tree.**
10. **Expand the Readiness test suite in the test tree to see the test cases under the readiness test node.**

11. **Select Size#test128K.**

    Continue answering the questions in the Configuration Editor as needed. Skip the individual test pack questions until you get to the Readiness section (be sure you go to the Readiness panel, not the Automatic Readiness section).

12. **In the General panel of the Readiness section, specify the number of clients that download the JAD file simultaneously.**

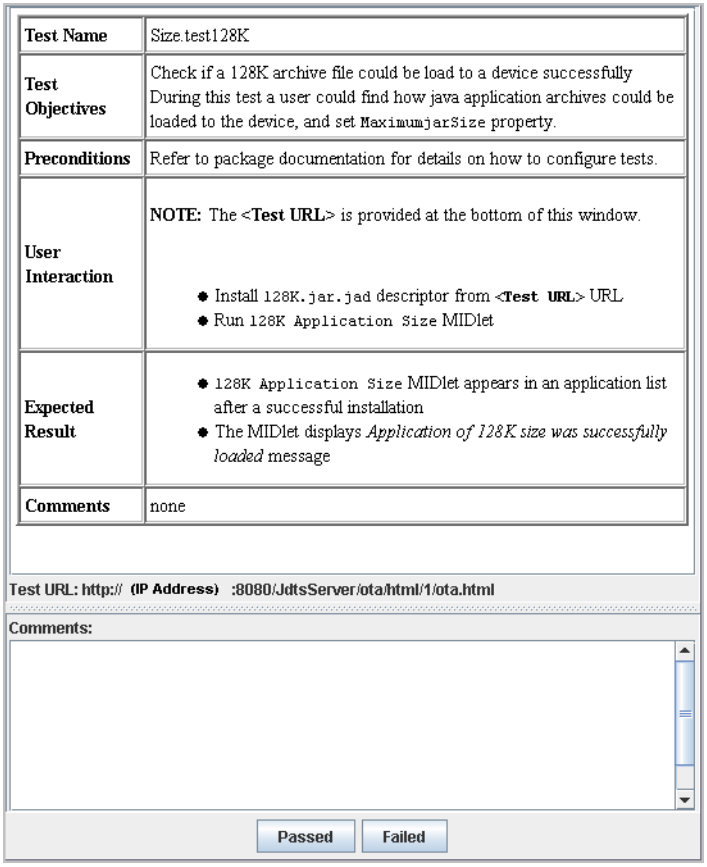
    The default value is two. Three clients is an average number. Do not specify any less than two. Click Next to proceed to the Headers panel.

13. **In the Headers panel, specify which versions of CLDC and MIDP the device supports and click Done.**

14. **Click the Start button to begin testing.**

    Instructions for downloading and running the test, similar to FIGURE 7-1, appear.

FIGURE 7-1 Size#test128K Instructions



15. **Launch the test device.**

16. **If the test device supports bookmarks, create a bookmark for the URL shown in the Test URL field.**

    The Test URL you see is probably different from the one shown in FIGURE 7-1.

17. **Download the URL shown in the Test URL field.**

    How you download URLs is device dependent. The readiness tests use the same URL so you can use one bookmark to download every test.

18. **Install `128K.jar.jad` file.**

    The test device asks for confirmation to install the MIDlet suite. Install it. How you install MIDlet suites is device dependent.

    The device downloads `128K.jar.jad` if it can. If the device indicates that it cannot download `128K.jar.jad`, there is no point to continuing with the Readiness tests.

19. **If the test device asks if you want to continue the installation even though the application is not signed, click Continue.**

20. **Launch 64K Application Size test.**

    How you launch MIDlets is device dependent.

    The device displays: `Application of a 128K size was successfully loaded`.

21. **Exit the MIDlet and remove 128K Application Size test.**

    How you exit and remove a MIDlet is device dependent.

22. **Click either the Passed or Failed button at the bottom of the test instructions.**

    It does not matter if you click the Passed or Failed button. The test run terminates.

▼ Verifying Essential Facilities

For tests to run, the following facilities must be operational on the test device:

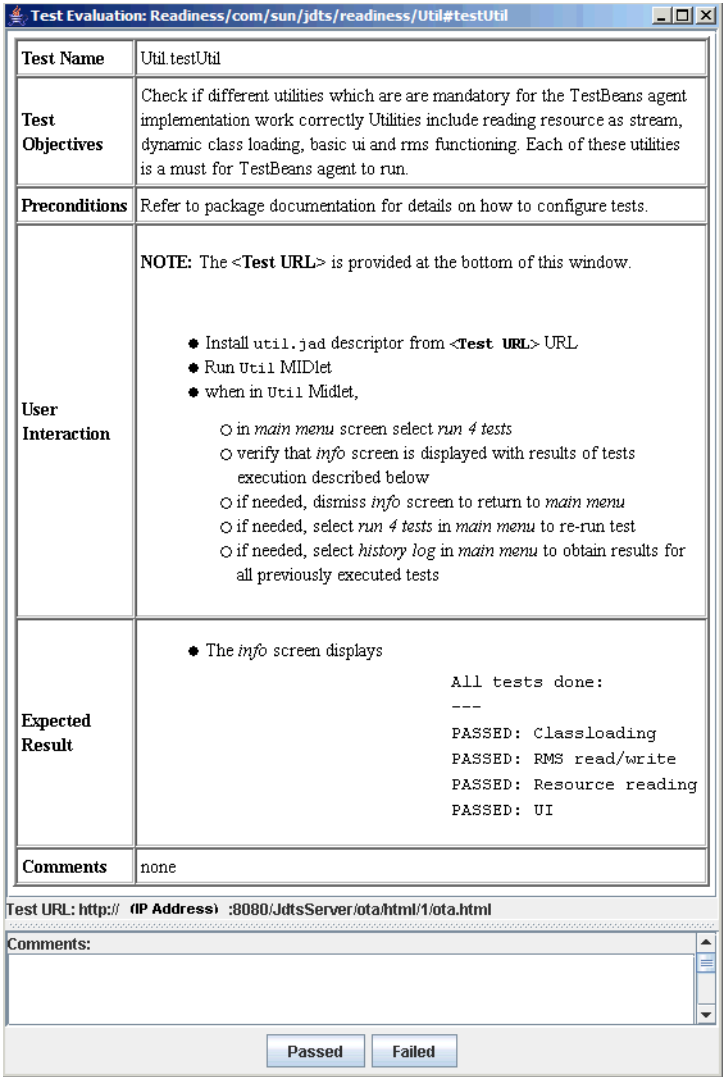
- Loading classes
- User interface
- Reading a resource as a stream
- Reading and writing the record management store (RMS)

The testUtil test verifies these facilities.

To run testUtil, follow these steps (which assume you have just finished successfully running Size#test64K test):

1. **Select Configure > Edit Configuration.**
2. **Click next until you get to the Tests to Run panel.**
3. **In the test tree, navigate to the Size#test64K test under the Readiness node.**
4. **Select the Util#testUtil test.**
   The Size#test64K test becomes unselected.
5. **Click Done in the Configuration Editor.**
6. **Click Start.**
   Instructions for downloading and running the test, similar to FIGURE 7-2, appear.

FIGURE 7-2 Util#testUtil Instructions



7. **Launch the test device.**

8. **Download the URL named in the Test URL field.**

   The URL you see is probably different from that shown in FIGURE 7-2. How you download URLs is device dependent.

9. **Install the `util.jad` file.**

   How you install MIDlet suites is device dependent.

10. **If the test device asks if you want to continue the installation even though the application is not signed, click Continue.**

    The device downloads Util.

11. **Launch Util.**

    How you launch MIDlets is device dependent.

    The device displays a menu containing:

    ```
    run 4 tests
    info
    history log
    ```

12. **Select** `run 4 tests`**.**

    The device should display:

    ```
    All tests done:
    PASSED: Classloading
    PASSED: RMS read/write
    PASSED: Resource reading
    PASSED: UI
    ```

    If the device does not display the above lines, there is no point continuing with the Readiness tests.

13. **Click the Done soft button or select the Done command in a device-dependent way.**

14. **Exit the MIDlet and remove Util.**

    How you exit and remove a MIDlet is device dependent.

15. **Click either the Passed or Failed button at the bottom of the test instructions.**

    It does not matter if you click the Passed or Failed button. The test run terminates.

▼ Verifying HTTP Communication

If you want the test device to return test results to the harness, and you want the harness to automatically display interactive test instructions, the test device must support HTTP communication with Java methods. Supporting HTTP at a lower level is sufficient for downloading tests, but it is not sufficient for returning test results. The testGet and testPost tests verify that the test device's Java methods for HTTP are working.

Run these tests as follows. The instructions assume that you have successfully run testUtil.

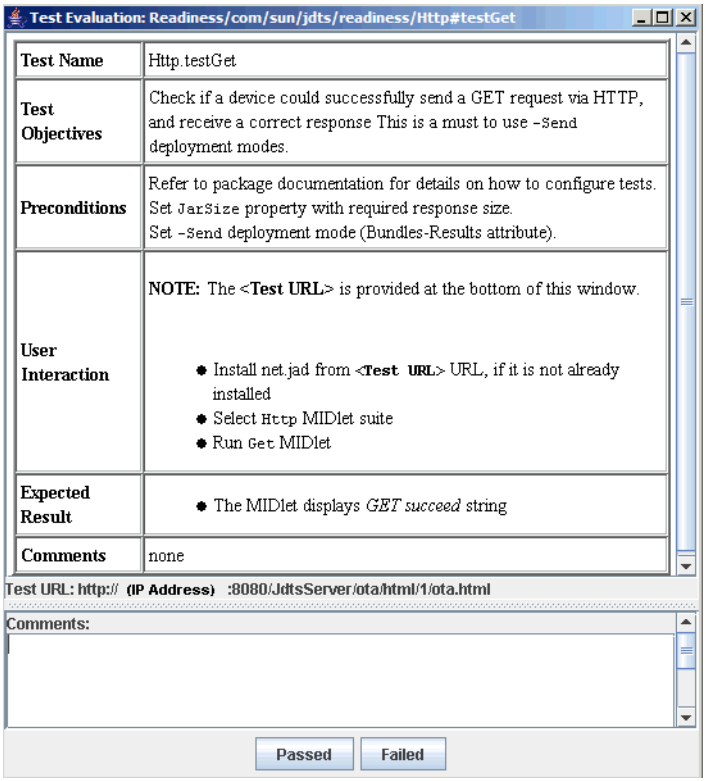
1. **Select Configure > Edit Configuration.**

2. **Click next until you get to the Tests to Run panel.**
3. **In the test tree, navigate to the Util#testUtil test under the Readiness node.**
4. **Select Http#testGet and Http#testPost.**

   The Size#test64K test becomes unselected.
5. **Click Done in the Configuration Editor.**
6. **Click Start.**

   Instructions for downloading and running the test, similar to FIGURE 7-3, appear.

FIGURE 7-3 testGet Instructions



7. **Launch the device.**
8. **Download the URL displayed next to Test URL.**

   The URL you see is probably different from the one shown in FIGURE 7-3. How you download URLs is device dependent.

9. **Install the `net.jad` file.**

   How you install MIDlet suites is device dependent.

10. **If the test device asks if you want to continue the installation even though the application is not signed, click Continue.**

    The device downloads the `net.jad` file.

11. **Launch the Http MIDlet.**

    How you launch MIDlets is device dependent.

12. **If the test device asks permission to use air time, grant it.**

    The device downloads the Http MIDlet.

13. **Launch the Get test.**

    How you launch applications is device dependent.

14. **If the test device asks permission to use air time, grant it.**

    If the test passes, the device displays:

    ```
    GET succeed
    ```

    If it does not pass, the device displays an error message.

15. **Exit the MIDlet.**

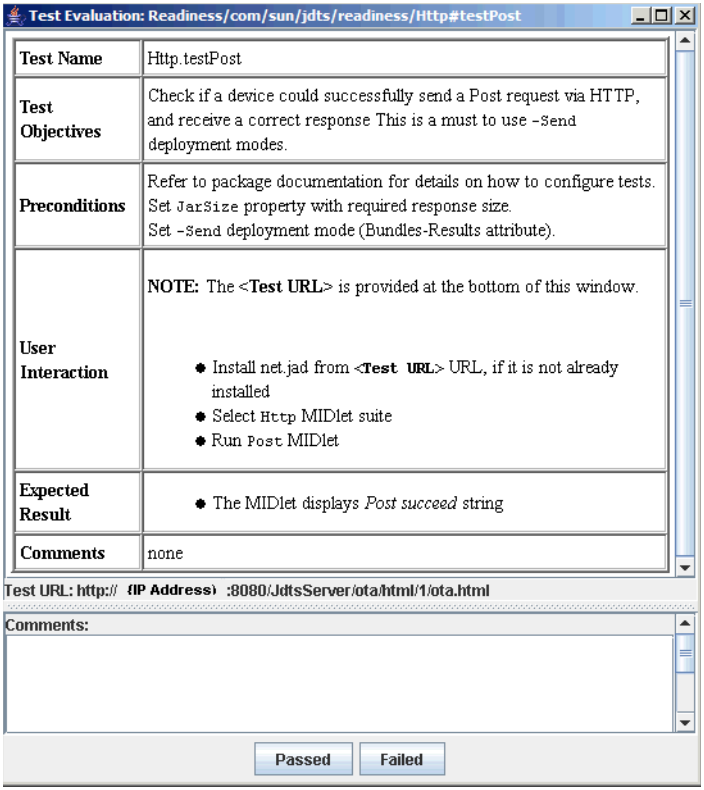
    How you exit an application is device dependent.

16. **Click either the Passed or Failed button at the bottom of the test instructions.**

    It does not matter if you click the Passed or Failed button.

    The harness displays the instructions for testPost, similar to FIGURE 7-4.

FIGURE 7-4 testPost Instructions



17. **Launch Http again.**

    How you launch an application is device dependent.

18. **Select and launch Post.**

    How you launch an application is device dependent.

19. **If the test device asks permission to use air time, grant it.**

    If the test succeeds, the device displays:

    ```
    POST succeed.
    ```

    If the test fails, the device displays an error message.

20. **Exit and remove Http.**

    How you exit and remove an application is device dependent.

21. **Click either the Passed or Failed button at the bottom of the test instructions.**

    It does not matter if you click the Passed or Failed button.

CHAPTER 8

Test Failure Severity

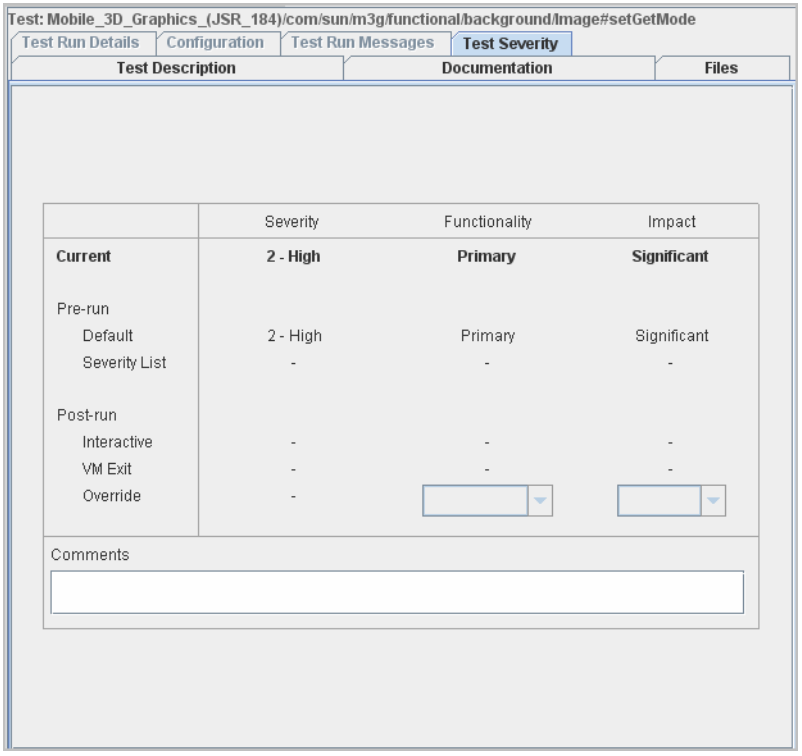
The failure of some tests is more important than others. For example, a test that probes an obscure feature is less important than one that tests a feature used by almost every application. Beginning in Java Device Test Suite version 2.2, all tests written by Sun are tagged with a severity value ranging from 1 (very high severity) through 5 (very low severity). Non-Sun test developers can tag their tests in the same way, as described in the *Java Device Test Suite Developer's Guide*.

You can select tests by severity (to save time by running the most important tests), you can view severity values in the harness, and you can organize reports by severity to focus on the most important failures. You can also override default severity values. This chapter describes test failure severity features in the following sections:

Viewing Test Failure Severity

To see a test's failure severity, select the test in the harness test tree and click the Test Severity tab. FIGURE 8-1 shows an example for a test that has not been run (current severity equals default severity). The harness computes severity from the values of functionality and impact. "How Severity is Calculated" on page 85 describes the calculation.

FIGURE 8-1 Default Test Severity Tab



A test can have a different severity after it has been run, as described in "How Severity is Calculated" on page 85. For an illustrative Test Severity tab, refer to FIGURE 5-30.

Selecting Tests by Severity

In a configuration or template (if you are an administrator), you can select tests to run by their pre-run failure severity (see "Pre-run Severity" on page 87). You can, for example, minimize the number of tests you run by filtering out low-severity tests. In the Test Selection section of the interview, answer these questions as follows:

- Specify Test Severity: Yes
- Severity: Check the boxes representing the severity values you want to run, for example, 1, 2, and 3. By default, all severity values are selected.

"Selecting Tests by Device Feature and Severity" on page 54 has an exercise for selecting tests by Severity.

Selection by severity is a filter, like selection by keyword. A test runs if it passes all filters. Therefore, you can use severity in combination with keywords, status, and so on. For example, you can select failed network tests that have a severity of 1 or 2.

Organizing a Report by Severity

When you select JDTS HTML Report in the Reports > Create Report dialog, you can select Consider Tests Severities. If you check (tick) this box, the report is organized by test severity, so you can easily focus on the failures that are most severe. FIGURE 5-35 shows a report with severity data. "Creating Feature and Severity Reports" on page 59 has an exercise for creating a report with severity data.

XML reports automatically include severity data. The online help describes report generation options.

How Severity is Calculated

The harness calculates a tests's severity from two factors, called *functionality* and *impact*. Each factor has a numeric value of 1-3, with 1 indicating the highest importance. TABLE 8-1 shows how the harness derives severity from each combination of functionality and impact.

TABLE 8-1 Severity Derivation from Functionality and Impact

Severity	Functionality	Impact
1 (very high)	1 (primary)	1 (critical)
2 (high)	2 (secondary)	1 (critical)
	1 (primary)	2 (significant)
3 (medium)	2 (secondary)	2 (significant)
	3 (non-essential)	1 (critical)
	1 (primary)	3 (limited)
4 (low)	3 (non-essential)	2 (significant)

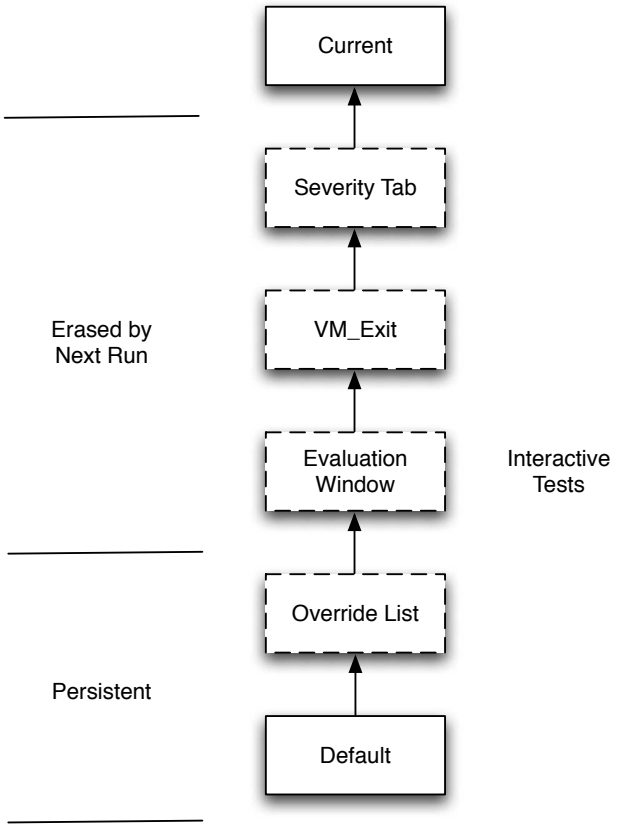
TABLE 8-1 Severity Derivation from Functionality and Impact *(Continued)*

Severity	Functionality	Impact
	2 (secondary)	3 (limited)
5 (very low)	3 (non-essential)	3 (limited)

To see the developer guidelines for assigning functionality and impact values, refer to the *Java Device Test Suite Developer's Guide*.

There are multiple sources of these factors, a default and several optional overrides (see FIGURE 8-2). The harness displays the final value, current severity, and the contributing sources, in the Test Severity tab.

FIGURE 8-2 Sources of Severity Factors



Pre-run Severity

Each test has a default severity, calculated from functionality and impact values specified by the test developer. The default values can optionally be overridden by an entry in a severity override list. Pre-run impact and functionality values persist across test runs.

Default Severity

Test developers assign functionality and impact values to each of their tests. These values determine a test's default severity.

Severity Override List

You can override a test's default severity in an override list file and then specifying the file in a template (if you are an administrator) or a configuration. Specify the fully qualified name of the override list file in the Test Selection > Severity List interview question.

A severity override list can contain blank lines, comment lines (# in first position), and test lines. A test line has the following format:

testName Impact Functionality

The line components are:

- *testName* is a test pack name plus optional package names, optional class name, and optional case name. To obtain fully qualified test case names, do one of the following:
  - Select the test names in the graphical interface's Passed, Failed, Error, or Not Run tabs, right-click, and choose Copy > Names as multiple lines. Then paste into a text editor.
  - Alternatively, after running the tests, create a plain text report and copy the fully qualified names from it.
- *Impact* is the impact of a failure, represented as a value in the set [1,2,3]. TABLE 8-1 gives the meaning of these values.
- *Functionality* is the importance of the function tested, represented as a value in the the set [1,2,3]. TABLE 8-1 gives the meaning of these values.

Here are some examples with explanatory comment lines:

```
# Override one test case
Bluetooth_(JSR_82)/com/sun/jsr082/bluetooth/functional/push/service
/UpdateTests#testServiceRegistrationException 1 2
```

```
#Override all cases in a class
MMAPI_and_ABB_(JSR_135)/com/sun/mmapi_10/functional/video/playback/
thirdgp/ThirdGenPlayerHTTP 2 3

# Override all tests in a package
MMAPI_and_ABB_(JSR_135)/com/sun/mmapi_10/functional/video 3 1

# Override all tests in a test pack
OpenGL_ES_(JSR_239) 3 2
```

Post-run Severity

A test's pre-run severity can be overridden during or after the test run in three ways.

- Interactive tests: Interactive test evaluation windows include drop-down controls that you can use to set a test's impact and functionality.
- Virtual machine exit: A test that returns a status of Error - VM_Exit automatically receives a severity value of 1. VM_Exit means that the Pass/Fail status of the test is indeterminate, as might happen if the tester restarts the test device.
- Test Severity tab. After a test has run, you can set its impact and function in the Test Severity tab. The values you specify override all others.

Post-run changes do not persist across test runs. The next time the test runs, it begins with its pre-run severity. It ends with that severity unless the test VM_Exits or you manually override impact or functionality in the test evaluation window or the Test Severity tab.

APPENDIX **A**

Test and Harness Ports

TABLE A-1 lists the IP ports that tests and the harness use.

TABLE A-1 Test and Harness IP Ports

Port	Used By	Property	Notes
4645-4695	Network test servers	MIDP - testPort	A network test server tries to open 4645. If that fails, it increments the port number by 1 and tries again. After 50 such failures, the server tries to open any port. If that fails, the test is failed and the Test Server Monitor displays a message that no ports are available.
4774-4824	Datagram connection servers	MIDP - datagramServerPort	The UDP ports used within datagram tests.
8080	harness	Console - ServerPort	The port on which the Java Device Test Suite server listens for connections between the tester harness and test devices.
9919	WMA tests	WMA Tests - UUTPortNumber	For sending and receiving sms/cbs messages

APPENDIX **B**

Subjectivity and Quality Testing

Subjectivity is unavoidable in quality testing. Unlike compatibility testing, there is usually no formal quality specification against which test device quality can be measured. Each testing organization needs to establish its own quality guidelines to minimize subjectivity, bearing in mind customer satisfaction, testing and support costs, competitive devices, and other factors. Although the development of such guidelines is beyond the scope of this guide, usability testing is one tool that deserves serious consideration.

The remainder of this appendix gives examples of subjectivity in different kinds of tests:

Interactive Tests

Some interactive test results require subjective judgment to designate them as passed or failed. Two major categories are differences in device display and differences in device responses to anomalous conditions.

Display Differences

Devices with different display capabilities can produce different results for the same test. For example, TABLE B-1 shows the reference image for com.sun.m3g.functional.image2d.Conversions.alphaAndRGBa and the displays of five devices that are identical except for the number of colors or grays they display. Notice the use of the word “probably” in the assessments of device performance on

this test.

TABLE B-1 Interpreting One Test on Different Devices

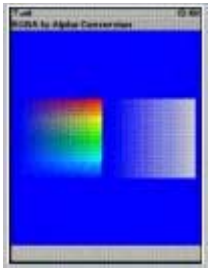
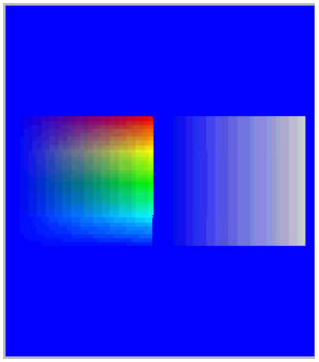
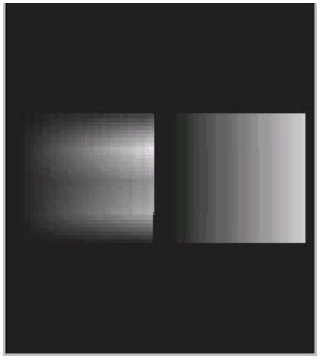
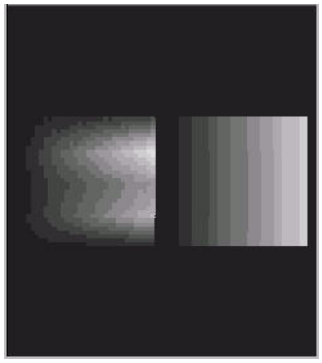
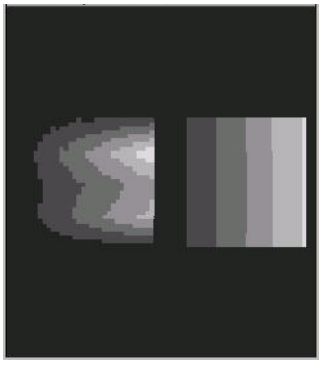
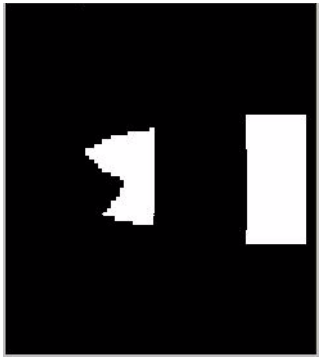
Test Device Display	Description
	Reference image. “Running Runtime Tests” in the online help describes test evaluation windows and reference images
	Test device has 4096 colors. This display is essentially identical to the reference image. It is obvious that this device passes the test.
	Test device has 4096 grays. This test could be considered to have passed result. Users would probably understand an image in an application that uses the tested feature.

TABLE B-1 Interpreting One Test on Different Devices *(Continued)*

Test Device Display	Description
	Test device has 16 grays. This test could be considered to have passed result. Users would probably understand an image in an application that uses the tested feature.
	Test device has eight grays. This test could be considered to have a failed result because it differs markedly from the reference image.
	Test device has two grays. This test could be considered to have a failed result. Users would most likely find a similar image displayed by an application as incomprehensible.

Anomalous Conditions

Test evaluation windows cannot describe how test devices behave in anomalous conditions because there are no specifications for such behavior. Nor can all tests indicate that they have started and finished. Displaying such information might interfere with the test. Therefore, when an interactive test does not behave as described in the test description window, you must use your knowledge of the device and your role as a potential device user to decide if the test result is a pass or fail.

For example, a test device might decide that a test is not authorized to use a protected API. Accordingly, it could refuse to launch the test or test MIDlet. Alternatively, it could launch the test or MIDlet but terminate it when it tries to use the API. It could inform you of the problem in several ways or it could not inform you at all.

Benchmark Tests

Subjectivity is an element in other tests as well. Benchmark tests automatically have pass or fail results, but they do so based on a comparison to a reference device's performance. That reference device is probably chosen subjectively. The only criterion for choosing it, as far as the Java Device Test Suite is concerned, is that the reference device consistently exhibits "acceptable" performance.

Robustness Tests

Robustness tests are another example of inherent quality test subjectivity. A test developer might decide that a robust implementation should perform 100 repetitions of a code sequence without error. The developer's decision is an informed one, but it is also ultimately a subjective evaluation.

APPENDIX C

Uninstalling

This appendix describes how to uninstall the tester harness in these sections:

- Uninstalling: Solaris Operating System
- Uninstalling: Windows Environment

For instructions on uninstalling the Central Installation, see the *Java Device Test Suite Administration Guide.*

Uninstalling: Solaris Operating System

To uninstall the tester harness, follow these steps:

1. **Change to the** *jdts_installDir*`/Uninstall JDTS`*version* `Console` **directory.**

   jdts_installDir/ is the directory in which you installed the tester harness (see "Installing the Tester Harness" on page 16). *version* is the installed harness version, such as 2.4.

2. **Run the uninstaller with this command:**

   % **`sh Uninstall_JDTS`***version*`_Console`**

Uninstalling: Windows Environment

To uninstall the tester harness, follow these steps:

1. **Choose Start > Control Panel > Add or Remove Programs**

2. **Select JDTS*version* Console in the Add or Remove Programs window.**
   version is the installed harness version, such as 2.4.
3. **Click the Change/Remove button.**
   Follow the steps in the Uninstall wizard to remove the test harness.

APPENDIX **D**

Filtering Tests with Exclude Lists

If you have one or more tests that should not be run on a device, you can filter them with an exclude list that you create with a text editor and associate with a template (if you are an administrator) or a configuration. An exclude list filter is like a keyword filter. It does not remove or hide tests but prevents them from being added to test bundles. If you set View Filter to Current Configuration, excluded tests are colored gray.

Note – This feature might be modified in the future.

This appendix covers these subjects:

Creating an Exclude List

An exclude list is a text file whose extension is `.jtx`. It must be located on a file system that is accessible to harnesses whose templates and configurations refer to it. Multiple templates or configurations can share an exclude list.

Note – Create exclude lists with care. Errors might not be reported or might cause unpredictable behavior.

An exclude list can contain blank lines, comment lines, and test lines.

Comment Lines

Use comment lines to identify an exclude list, and to group and document test lines. A comment line begins with a single "#" character.

Note – Do not write a comment beginning with "###". This sequence is reserved for possible future use.

Test Lines

Each test line in an exclude list file contains the fully qualified name of a package, class, or test case. Excluding a package or class is shorthand for excluding its contents (analogous to deleting a file system directory). Forward slashes are required in test names.

Here are three examples with comment annotations (ignore line breaks):

```
# Exclude a test case
CLDC_(JSR_30_and_JSR_139)/com/sun/cldc/cldc1_0/functional/cpack/c04_
1/c0410201/test0410201#testcase1

# Exclude all cases in a class
Scalable_Vector_Graphics_(JSR_226)/com/sun/jsr226/functional/animati
on/ControlAnimation

# Exclude all packages, classes, and cases in a package
Benchmark/com/sun/benchmark/scenarios
```

To obtain the fully qualified names of tests, do one of the following:

- Select the test names in the graphical interface's Passed, Failed, Error, or Not Run tabs, right-click, and choose Copy > Names as multiple lines. Then paste into a text editor.
- Alternatively, after running the tests, create a plain text report and copy the fully qualified names from it.

You can add up to three optional fields after the test name. Each field is separated by spaces and/or tabs. A line break terminates the entry. There is no way to indicate that an entry continues on the next line. The three optional fields are:

- BugIDs: A comma-separated (no spaces) list of bug identifiers associated with the excluded test. Your organization chooses the bug identifiers, typically from your bug tracking system. Letters, integers, dashes and underscore characters are valid in BugIDs.

- Keywords: A comma-separated (no spaces) list of keywords that can be used to classify exclusions. The keywords are organization-specific and are unrelated to the configuration and template keywords used to select tests. Keyword entries must start with a letter and can contain letters, numbers, and the underscore character. It is good practice to limit keyword names to 20 characters.
- Synopsis: The reason the test is excluded. It is good practice to limit synopses to 100 characters.

If you omit an optional field, you must also omit the fields to its right. For example, if you do not specify keywords, you must not specify a synopsis. If you do specify keywords, synopsis is optional, but BugIDs is required.

Example

CODE EXAMPLE D-1 shows a simple exclude list. Part of each test name (...) has been omitted to fit the lines into the available space.

CODE EXAMPLE D-1 Simple Exclude List

```
# Exclude list for handset Alpha, software version 2.0.2
# Revised Mon Jul 23 18:15:04 PDT 2007


# Test name only
AMMS.../camera/CameraControlTest#checkExposureModes

# Two BugIDs, Keyword
CLDC.../TableLookupSwitch/CaseWithReturn 22720,22333 notSupported

# BugID, Keyword, Synopsis
MIDP.../write 33655 spec Interpretation dispute
```

Associating an Exclude List with a Configuration or Template

Direct the harness to use an exclude list by associating it with a configuration (or if you are an administrator, with a template). A template or configuration can name multiple exclude lists, in which case their effect is additive.

▼ Specifying an Exclude List with the Configuration or Template Editor

1. **Set Specify an Exclude List = Yes**
2. **Specify Exclude List Files, click Add.**
3. **Select the exclude list file.**

You can remove an exclude list from a configuration or template by selecting it and clicking Remove. Remove does not delete the file. Ignore the Up and Down buttons.

▼ Specifying an Exclude List with the Quick Set Editor

1. **Choose Configure Edit Quick Set > Exclude List.**
2. **Select Other, click Add.**
3. **Select the exclude list file.**

You can remove an exclude list from a configuration or template by selecting it and clicking Remove. Remove does not delete the file. Ignore the Up and Down buttons.

APPENDIX **E**

Specifying HTTP Headers

Note – The information in this appendix is for advanced users who want to tune test bundle downloading to devices. To use these instructions successfully, you must understand HTTP headers. An error in the configuration files described here can have serious negative consequences.

For a tester, two files can influence the contents of the HTTP response headers generated by the Relay when a test device requests a bundle:

- *CentralInstallDir*`/admin/shared/conf/httpheaders.properties`: An administrator can modify this file to change the default HTTP headers for all harnesses.
- *TesterHarnessInstallDir*`/conf/httpheaders.properties`: A tester can create this optional file to specify HTTP headers for a particular harness. This file, if present, overrides *CentralInstallDir*`/admin/conf/httpheaders.properties`.

The default *CentralInstallDir*`/admin/shared/conf/httpheaders.properties` file contains these lines:

```
Cache-Control=no-cache
Pragma=no-cache
```

These lines disable caching in many networks and devices. The second line is for HTTP 1.0 devices. Disabling caching is important for downloading test bundles by a URL bookmark. For tester convenience, the Relay gives every bundle the same URL, namely
`http://`*ipAddressPort*`/`*appContext*`/jad/`*harnessID*`/getNextApp.jad`.
When a tester requests this URL with a bookmark, a caching facility in the network or the test device can repeatedly return a test bundle that has already been run. Disabling caching forces the device or network to obtain the correct bundle from the Relay.

You can create or modify a harness-local `httpheaders.properties` file to disable caching by your network or by a particular test device. You can also add lines to perform additional network or device tuning.

APPENDIX **F**

Supported Technologies

TABLE F-1 lists the technologies and versions whose implementations can be tested with the Java Device Test Suite.

TABLE F-1 Java Device Test Suite Supported Technologies

Technology	Versions and Specifications
Advanced Multimedia Supplements (AMMS)	1.0 - JSR 234
Connected Limited Device Configuration (CLDC)	1.0 - JSR 30 1.1, 1.1.1 - JSR 139
Contactless Communication API	1.0 - JSR 257
Content Handler API (CHAPI)	1.0 - JSR 211
Java APIs for Object Exchange (OBEX) and Bluetooth	1.0, 1.1, 1.1.1 - JSR 82
Java Technology for the Wireless Industry (JTWI)	1.0 - JSR 185
Location API Optional Package	1.0 - JSR 179
Mobile 3D Graphics API	1.0, 1.1 - JSR 184
Mobile Information Device Profile (MIDP)	1.0 - JSR 37 2.0, 2.1 - JSR 118
Mobile Internationalization API	1.0 - JSR 238
Mobile Media API (MMAPI)	1.0, 1.1 - JSR 135
Mobile Sensor API (MSAPI)	1.1 - JSR 256
Mobile Service Architecture (MSA) security tests	1.0 - JSR 248
OpenGL Embedded Subset (ES)	1.0 - JSR 239
Payment API	1.1 - JSR 229
Personal Digital Assistant (PDA) optional packages	1.0 - JSR 75
Scalable Vector Graphics (SVG)	1.0 - JSR 226

TABLE F-1 Java Device Test Suite Supported Technologies *(Continued)*

Technology	Versions and Specifications
Security and Trust Services API (SATSA)	1.0 - JSR 177
Session Initiation Protocol (SIP)	1.0, 1.1 - JSR 180
Web Services (JAXP and JAX-RPC Subset)	1.0 - JSR 172
Wireless Messaging API (WMA)	1.0, 1.1 - JSR 120 2.0 - JSR 205
XML API	1.0 - JSR 280

Note – The Java Device Test Suite supports respective clarifications for all JSRs that are in MSA JSR 248.

When specification versions are proper subsets-supersets, one test pack covers the versions with a different package for each version. The tests in the packages do not overlap. For example, the tests in com.sun.wma2_0 do not include the tests in com.sun.wma1_1. Therefore, to fully test a device that supports WMA 2.0, you must run the tests in com.sun.wma, com.sun.wma1_1, and com.sun.wma2_0.

When a newer specification is not a proper superset of an older one, the versions are represented by different test packs. The MIDP 1.0 OTA and MIDP OTA test packs are an example. The MIDP 1.0 OTA test pack contains all of the tests related to the MIDP 1.0 specification. The MIDP OTA test pack contains the MIDP 1.0 tests that are compatible with MIDP 2.0 and MIDP 2.1 devices, plus the MIDP 2.0 and 2.1 tests.

Index

R

S

T

U