

OPTIMIZE

MERCURY QUICKTEST PROFESSIONAL™

JAVA™ ADD-IN

VERSION 9.1

Guide

MERCURY™

BUSINESS TECHNOLOGY OPTIMIZATION

Mercury QuickTest Professional Java™ Add-in

Guide

Version 9.1

Document Release Date: October 1, 2006

MERCURY™

Mercury QuickTest Professional Java Add-in Guide, Version 9.1

This manual, and the accompanying software and other documentation, is protected by U.S. and international copyright laws, and may be used only in accordance with the accompanying license agreement. Features of the software, and of other products and services of Mercury Interactive Corporation, may be covered by one or more of the following patents: United States: 5,511,185; 5,657,438; 5,701,139; 5,870,559; 5,958,008; 5,974,572; 6,137,782; 6,138,157; 6,144,962; 6,205,122; 6,237,006; 6,341,310; 6,360,332; 6,449,739; 6,470,383; 6,477,483; 6,549,944; 6,560,564; 6,564,342; 6,587,969; 6,631,408; 6,631,411; 6,633,912; 6,694,288; 6,738,813; 6,738,933; 6,754,701; 6,792,460 and 6,810,494. Australia: 763468 and 762554. Other patents pending. All rights reserved.

Mercury, Mercury Interactive, the Mercury logo, the Mercury Interactive logo, LoadRunner, WinRunner, SiteScope and TestDirector are trademarks of Mercury Interactive Corporation and may be registered in certain jurisdictions. The absence of a trademark from this list does not constitute a waiver of Mercury's intellectual property rights concerning that trademark.

All other company, brand and product names may be trademarks or registered trademarks of their respective holders. Mercury disclaims any responsibility for specifying which marks are owned by which companies or which organizations.

Mercury provides links to external third-party Web sites to help you find supplemental information. Site content and availability may change without notice. Mercury makes no representations or warranties whatsoever as to site content or availability.

Mercury Interactive Corporation
379 North Whisman Road
Mountain View, CA 94043
Tel: (650) 603-5200
Toll Free: (800) TEST-911
Customer Support: (877) TEST-HLP
Fax: (650) 603-5300

© 1992 - 2006 Mercury Interactive Corporation, All rights reserved

If you have any comments or suggestions regarding this document, please send them by e-mail to documentation@mercury.com.

Table of Contents

Welcome to This Guide	v
How This Guide Is Organized	vi
Who Should Read This Guide	vii
QuickTest Professional Online Documentation	viii
Additional Online Resources.....	x
Documentation Updates	xi
Typographical Conventions.....	xii

PART I: SETTING UP JAVA ADD-IN SUPPORT

Chapter 1: Installing the Java Add-in	3
Before You Install	4
Understanding Java Add-in Dependencies and Conflicts	4
Running the Setup Program	5
Chapter 2: Activating Java Add-in Support	15
Loading QuickTest with Java Add-in Support.....	16
Loading QuickTest without Java Add-in Support	18
Repairing and Uninstalling the Java Add-in	19

PART II: WORKING WITH THE JAVA ADD-IN

Chapter 3: Creating and Running Steps on Java Objects	27
About Creating and Running Steps on Java Objects	28
Understanding the Java Test Object Model	29
Defining Java Testing Options	32
Defining Java Settings for Individual Tests and Components.....	39
Defining Active Screen Capture Options for Tests	43
Defining Record and Run Options for Tests	45
Recording Tests and Components on Java Objects	49
Running Tests and Components on Java Applications and Applets	56

Chapter 4: Enhancing Your Java Test	57
About Enhancing Your Java Test	58
Viewing the Full Object Hierarchy.....	59
Checking Java Objects and Outputting Values.....	61
Using Java Objects, Methods, and Properties to Enhance Your Test	63
Chapter 5: Troubleshooting Testing Java Applets and Applications	71
Identifying and Solving Common Problems and Solutions.....	72
Checking Java Environment Variables Settings.....	74
Locating the Java Console.....	75
Running an Application or Applet with the Same Settings.....	78
Running the Java Add-in on Java 2, Java 5, and Java 6 Environments	78
Disabling Dynamic Transformation Support (Advanced)	80
Index.....	83

Welcome to This Guide

Welcome to the QuickTest Professional Java Add-in.

The QuickTest Professional Java Add-in enables you to create and run tests on Java applets and applications. You can create and run steps on Java objects in Internet Explorer, Netscape Browser, Mozilla Firefox, Java Web Start, Applet Viewer, and in stand-alone Java applications.

The Java Add-in records user operations on applets and applications, and on the standard Java objects within them. For information on supported Java toolkits and versions, refer to the *Java Add-in Readme* file.

QuickTest provides customized Java test objects, methods, and properties that make tests and components simple to read, maintain, enhance, and parameterize, enabling both advanced and novice users to create sophisticated tests and components on Java applets and applications.

This chapter describes:	On page:
How This Guide Is Organized	vi
Who Should Read This Guide	vii
QuickTest Professional Online Documentation	viii
Additional Online Resources	x
Documentation Updates	xi
Typographical Conventions	xii

How This Guide Is Organized

This guide explains everything you need to know to install the QuickTest Professional Java Add-in and to use QuickTest Professional to successfully test Java applets and applications.

This guide should be used in conjunction with the *QuickTest Professional User's Guide* and the *QuickTest Professional Object Model Reference*. In addition, the *QuickTest Professional Java Add-in Extensibility Developer's Guide* explains how to create support for custom Java controls. All of these guides can be accessed online by choosing **Help > QuickTest Professional Help** from the QuickTest main window.

This guide contains:

Part I Setting Up Java Add-in Support

Details the process of setting up the QuickTest Professional Java Add-in, including:

- Installing the Java Add-in
- Activating Java Add-in Support

Part II Working with the Java Add-in

Explains how to use the QuickTest Professional Java Add-in to test Java objects, including:

- Creating and Running Steps on Java Objects
- Enhancing Your Java Test
- Troubleshooting Testing Java Applets and Applications

Note: The information, examples, and screen captures in this guide focus specifically on working with QuickTest tests. However, much of the information applies equally to components.

Business components and scripted components are part of Mercury Business Process Testing, which utilizes a keyword-driven methodology for testing applications. For more information, refer to the *QuickTest Professional User's Guide* and the *QuickTest Professional for Business Process Testing User's Guide*.

Who Should Read This Guide

This guide is intended for QuickTest Professional users at all levels who want to use the QuickTest Professional Java Add-in to test Java applications.

Readers should already have some understanding of functional testing concepts and processes, and know which aspects of their application they want to test.

Note: The Java Add-in takes advantage of commonly used QuickTest features such as the object repository, Keyword View, and checkpoints and output value steps to enable you to test your applet or application. You should have at least a basic understanding of these concepts before you begin working with the QuickTest Professional Java Add-in.

QuickTest Professional Online Documentation

QuickTest Professional includes the following online documentation:

Readme provides the latest news and information about QuickTest. Choose **Start > Programs > QuickTest Professional > Readme**.

QuickTest Professional Installation Guide explains how to install and set up QuickTest. Choose **Help > Printer-Friendly Documentation > Mercury QuickTest Professional Installation Guide**.

QuickTest Professional Tutorial teaches you basic QuickTest skills and shows you how to design tests for your applications. Choose **Help > QuickTest Professional Tutorial**.

Product Feature Movies provide an overview and step-by-step instructions describing how to use selected QuickTest features. Choose **Help > Product Feature Movies**.

Printer-Friendly Documentation displays the complete documentation set in Adobe portable document format (PDF). Online books can be viewed and printed using Adobe Reader, which can be downloaded from the Adobe Web site (<http://www.adobe.com>). Choose **Help > Printer-Friendly Documentation**.

QuickTest Professional Help includes:

- ▶ **What's New in QuickTest** describes the newest features, enhancements, and supported environments in the latest version of QuickTest.
- ▶ **QuickTest User's Guide** describes how to use QuickTest to test your application.
- ▶ **QuickTest for Business Process Testing User's Guide** provides step-by-step instructions for using QuickTest to create and manage assets for use with Business Process Testing.
- ▶ **QuickTest Object Model** describes QuickTest test objects, lists the methods and properties associated with each object, and provides syntax information and examples for each method and property.

- **QuickTest Advanced References** contains documentation for the following QuickTest COM and XML references:
 - **QuickTest Automation** provides syntax, descriptive information, and examples for the automation objects, methods, and properties. It also contains a detailed overview to help you get started writing QuickTest automation scripts. The automation object model assists you in automating test management, by providing objects, methods and properties that enable you to control virtually every QuickTest feature and capability.
 - **QuickTest Test Results Schema** documents the XML schema that enables you to customize your test results.
 - **QuickTest Test Object Schema** documents the XML schema that enables you to extend test object support in different environments.
 - **QuickTest Object Repository Automation** documents the Object Repository automation object model that enables you to manipulate QuickTest object repositories and their contents from outside of QuickTest.
- **VBScript Reference** contains Microsoft VBScript documentation, including VBScript, Script Runtime, and Windows Script Host.

Choose **Help > QuickTest Professional Help**. Online Help is also available from specific QuickTest windows and dialog boxes by clicking in the window and pressing F1. You can also view a description, syntax, and examples for a QuickTest test object, method, or property by placing the cursor on it and pressing F1.

Note: Your QuickTest Help may contain additional items relevant to any QuickTest add-ins you have installed. For more information, refer to the relevant add-in documentation.

Additional Online Resources

Knowledge Base uses your default Web browser to open the Mercury Customer Support Web Site directly to the Knowledge Base landing page. Choose **Help > Knowledge Base**. The URL for this Web site is <http://support.mercury.com/cgi-bin/portal/CSO/kbBrowse.jsp>.

Customer Support Web Site uses your default Web browser to open the Mercury Customer Support Web site. This site enables you to browse the Mercury Support Knowledge Base and add your own articles. You can also post to and search user discussion forums, submit support requests, download patches and updated documentation, and more. Choose **Help > Customer Support Web Site**. The URL for this Web site is <http://support.mercury.com>.

Send Feedback enables you to send online feedback about QuickTest to the product team. Choose **Help > Send Feedback**.

Mercury Home Page uses your default Web browser to access Mercury's Web site. This site provides you with the most up-to-date information on Mercury and its products. This includes new software releases, seminars and trade shows, customer support, educational services, and more. Choose **Help > Mercury Home Page**. The URL for this Web site is <http://www.mercury.com>.

Mercury Best Practices contain guidelines for planning, creating, deploying, and managing a world-class IT environment. Mercury provides three types of best practices: Process Best Practices, Product Best Practices, and People Best Practices. Licensed customers of Mercury software can read and use the Mercury Best Practices available from the Customer Support site, <http://support.mercury.com>.

Documentation Updates

Mercury is continually updating its product documentation with new information. You can download the latest version of this document from the Customer Support Web site (<http://support.mercury.com>).

To download updated documentation:

- 1** In the Customer Support Web site, click the **Documentation** link.
- 2** Under **Please Select Product**, select **QuickTest Professional**.

Note that if the required product does not appear in the list, you must add it to your customer profile. Click **My Account** to update your profile.

- 3** Click **Retrieve**. The Documentation page opens and lists the documentation available for the current release and for previous releases. If a document was updated recently, **Updated** appears next to the document name.
- 4** Click a document link to download the documentation.

Typographical Conventions

This guide uses the following typographical conventions:

UI Elements	This style indicates the names of interface elements on which you perform actions, file names or paths, and other items that require emphasis. For example, “Click the Save button.”
<i>Arguments</i>	This style indicates method, property, or function arguments and book titles. For example, “Refer to the <i>Mercury User’s Guide</i> .”
<Replace Value>	Angle brackets enclose a part of a file path or URL address that should be replaced with an actual value. For example, <MyProduct installation folder>\bin .
Example	This style is used for examples and text that is to be typed literally. For example, “Type Hello in the edit box.”
CTRL+C	This style indicates keyboard keys. For example, “Press ENTER.”
Function_Name	This style indicates method or function names. For example, “The wait_window statement has the following parameters:”
[]	Square brackets enclose optional arguments.
{ }	Curly brackets indicate that one of the enclosed values must be assigned to the current argument.
...	In a line of syntax, an ellipsis indicates that more items of the same format may be included. In a programming example, an ellipsis is used to indicate lines of a program that were intentionally omitted.
	A vertical bar indicates that one of the options separated by the bar should be selected.

Part I

Setting Up Java Add-in Support

1

Installing the Java Add-in

This chapter lists the pre-installation requirements and explains how to install the Java Add-in.

This chapter describes:	On page:
Before You Install	4
Understanding Java Add-in Dependencies and Conflicts	4
Running the Setup Program	5

Before You Install

Before you begin to install the Java Add-in, review the system requirements listed below.

To work successfully with the Java Add-in, your system configuration should meet the requirements as specified for QuickTest Professional 9.1 (in the *QuickTest Professional 9.1 Readme*), plus the following add-in specific requirements:

Prerequisites: QuickTest Professional 9.1.
(The QuickTest Professional Web Add-in must also be installed if you plan to test Java applets in a Web browser.)

Hard Disk Space: 77 MB of free hard disk space is required on the disk on which you want to install the Java Add-in, in addition to the space required for the QuickTest Professional installation.

In addition, the disk on which Windows is installed (system disk) requires 115 MB of free hard disk space while the Java Add-in is being installed. After the installation is complete, 38 MB of this disk space is released (meaning that 77 MB of the system disk remains in use by QuickTest Professional).

Understanding Java Add-in Dependencies and Conflicts

The QuickTest Professional Java Add-in can be installed and run together with any other QuickTest Professional add-in (core or external). When testing Java applets in a Web browser, you must load the Web Add-in as well as the Java Add-in, and use the Web tab of the Record and Run Settings dialog box to specify your record and run preferences.

QuickTest Professional Java Add-in 9.1 is compatible with the WinRunner Java and/or Oracle Add-in 7.6 only. To work with the QuickTest Professional Java Add-in 9.1 and the WinRunner Java and/or Oracle Add-in 7.6 on the same computer, you must install the Java/Oracle patch for WinRunner 7.6 (**WR76DualAgentPatch.exe**, located in the **WR76DualAgentPatch** folder on the QuickTest Professional Java Add-in 9.1 installation CD-ROM).

When you install the QuickTest Professional Java Add-in 9.1, the Setup program checks whether the WinRunner Java and/or Oracle Add-in 7.6 are installed on the same computer. If either or both of these add-ins are installed, a message prompts you to install the Java/Oracle patch for WinRunner 7.6 after you finish the add-in installation. If you do not install this patch, you will not be able to use the WinRunner Java and/or Oracle Add-in on this computer.

After you install this patch, you can work with the QuickTest Professional Java Add-in 9.1 and the WinRunner Java and/or Oracle Add-in 7.6 on the same computer and also load them simultaneously.

Note: Do not install WinRunner Java or Oracle Add-in versions earlier than version 7.6 on a QuickTest Professional 9.1 computer.

Running the Setup Program

The Setup program installs add-in support in your QuickTest Professional installation folder for testing Java applications.

Notes:

To install the Java Add-in, you must be logged on with administrator privileges.

You must not run any other installation at the same time as you run the Java Add-in installation.

You must have QuickTest Professional 9.1 installed on your computer. Refer to the *QuickTest Professional Installation Guide* for information on installing QuickTest Professional.

To install the Java Add-in:

- 1** Close any instances of QuickTest Professional. It is also recommended to close all other open applications.
- 2** Insert the CD-ROM into the CD-ROM drive.
 - If the CD-ROM drive is on your local computer, the QuickTest Professional Java Add-in Setup window opens.
 - If you are installing from a network drive, browse to it and double-click **autorun.exe** in the root folder of the CD-ROM. The QuickTest Professional Java Add-in Setup window opens.

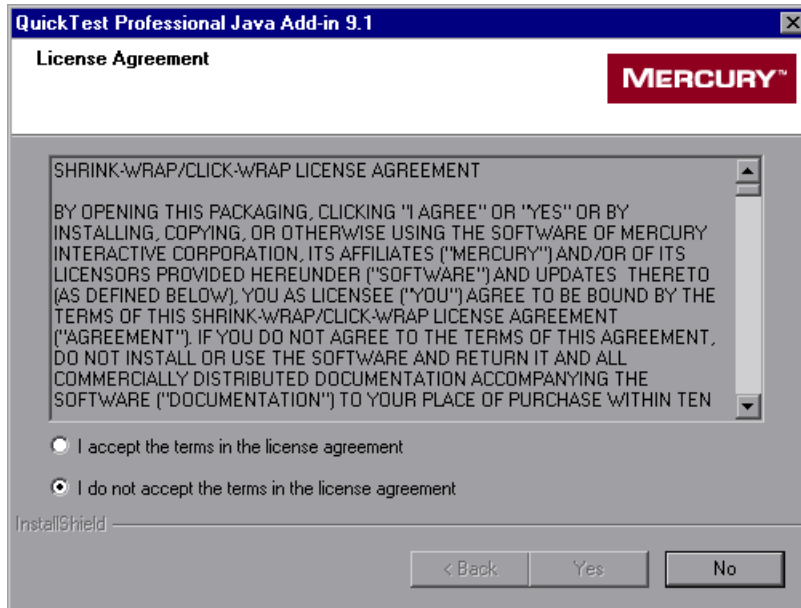


The QuickTest Professional Java Add-in Setup window contains the following options:

- **Add-in Setup.** Starts the Java Add-in installation program.
 - **Product Information.** Opens the product information site (<http://www.mercury.com/us/products/quality-center/functional-testing/quicktest-professional/>).
 - **Readme.** Opens the *QuickTest Professional Java Add-in Readme* file.
 - **Extensibility SDK Setup.** Starts the Java Add-in Extensibility SDK installation program. For more information, refer to the *QuickTest Professional Java Add-in Extensibility Developer's Guide*.
 - **Contact Mercury.** Opens the contact information page on the Mercury Interactive Web site (<http://www.mercury.com/us/company/corporate-info/contact-us/>).
 - **Support.** Opens the Mercury Interactive Customer Support Web site (<http://support.mercury.com>).
 - **Browse CD.** Displays the contents of the QuickTest Professional Java Add-in CD-ROM.
 - **Exit.** Exits the Setup program.
- 3** To start the Java Add-in Setup program, click **Add-in Setup**. The Java Add-in Setup program starts.

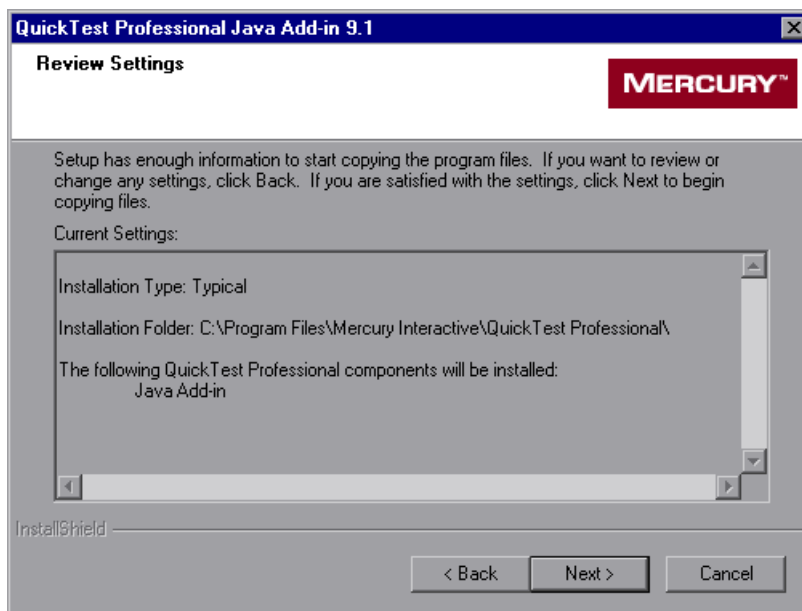
Note: The Setup program checks whether the WinRunner Java and/or Oracle Add-in 7.6 are installed on this computer. If either or both of these add-ins are installed, a message prompts you to install the Java/Oracle patch for WinRunner 7.6 (**WR76DualAgentPatch.exe**, located in the **WR76DualAgentPatch** folder on the QuickTest Professional Java Add-in 9.1 installation CD-ROM) after you finish the add-in installation. If you choose not to install the patch, you will not be able to use the WinRunner Java and/or Oracle Add-in on this computer.

- 4 The License Agreement screen opens. Read the agreement.



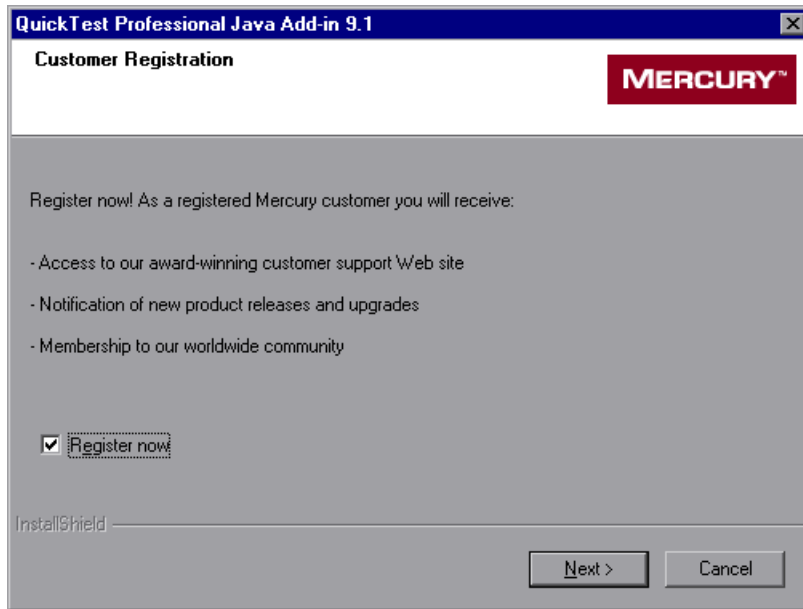
To install QuickTest Professional add-in support for testing Java applets and applications, you must accept the terms of the license agreement by selecting the **I accept the terms in the license agreement** option and clicking **Yes**. If you click **No**, the Setup program closes.

- 5 In the Review Settings screen, review the installation settings you selected.



To change your settings, click **Back**. To confirm the settings, click **Next**. The installation process begins.

- 6 In the Customer Registration screen, you can specify whether to register your copy of the QuickTest Professional Java Add-in.



If you register, you receive:

- access to the Mercury Interactive award-winning Customer Support Web site.
- notification of new product releases and upgrades.
- membership to the Mercury Interactive worldwide community of testers.

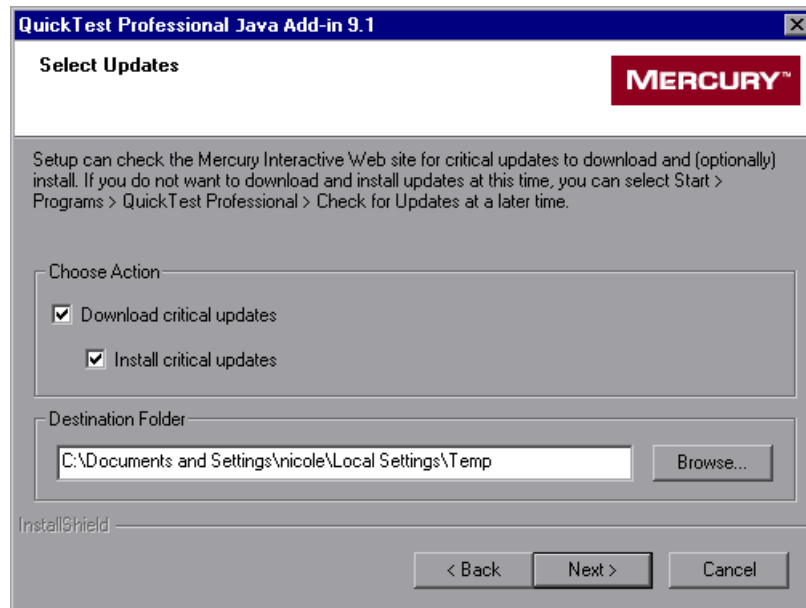
To register now, confirm that the **Register now** check box is selected. Click **Next** to proceed. Your browser opens to the Mercury Interactive Customer Support site: <http://support.mercury.com>. You can also choose to register at a later time by going directly to the Mercury Interactive Customer Support site.

Click **Next** to proceed.

- 7 During the installation process, the Setup program checks whether there are any critical updates to be installed for the version of the Java Add-in you are installing. It also checks whether there are any critical updates to be installed for your version of QuickTest Professional and any other external add-ins that are installed on your computer.

If no critical updates are found, the installation continues with step 9.

If the Setup program finds any critical updates to be installed, the Select Updates screen opens.



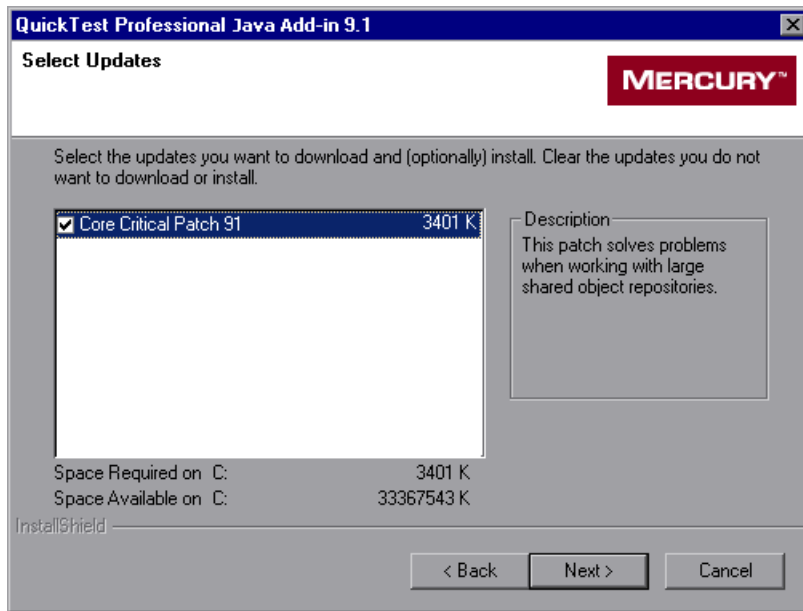
Choose one of the following options:

- **Download critical updates.** Downloads any critical updates to the specified destination folder. This option is selected by default.
- **Install critical updates.** Installs any downloaded critical updates. This option is selected by default.

Note: If the Setup program finds any critical updates to be installed, it is highly recommended to install them immediately. (You can also check for updates at a later time by choosing **Start > Programs > QuickTest Professional > Check for Updates.**)

A default download folder is displayed in the **Destination Folder** box. To select a different location to which to download the file, click **Browse**, choose a folder, and click **OK**.

Click **Next** to proceed. If you chose to download and (optionally) install critical updates, a list of all available critical updates opens. Otherwise, continue with step 9.



- 8 Select the updates that you want to download and (optionally) install. Clear the updates that you do not want to download or install.

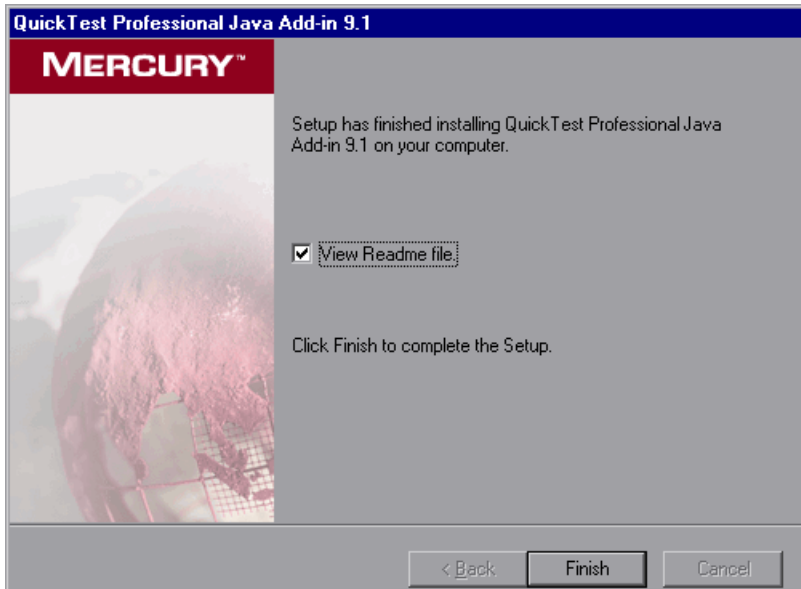
Click **Next** to proceed. QuickTest downloads the specified critical updates, according to the option you selected in step 7. If you chose to download and install the updates, they are installed at the end of the add-in installation process, after you restart your computer. (Note that you are always prompted to restart your computer if you choose to download and install critical updates.)

- 9 If the screen prompts you to restart your computer, you can choose to restart your computer at a later time, but you must restart your computer before you use QuickTest Professional.

Note: If you are prompted to restart, it is strongly recommended that you do so as soon as possible. Delaying the system restart could result in unexpected system behavior.

If you are not prompted to restart, proceed to step 10. Otherwise, step 10 occurs after the restart.

- 10 The Setup Complete screen opens. If you want to open the *QuickTest Professional Java Add-in Readme* file at the end of the Setup process, select the **View Readme file** check box.



Click **Finish** to complete the Setup program.

In the Readme file, you can view the latest technical and troubleshooting information for the Java Add-in. To open the Readme file at another time, choose **Start > Programs > QuickTest Professional > Add-ins > Java Add-in Readme**.

2

Activating Java Add-in Support

Before you can work with the QuickTest Professional Java Add-in, you must make sure that the Java Add-in is loaded. You can load QuickTest without add-in support for Java if you do not want to test Java applets and applications. You can also repair or uninstall your Java Add-in installation.

This chapter describes:	On page:
Loading QuickTest with Java Add-in Support	16
Loading QuickTest without Java Add-in Support	18
Repairing and Uninstalling the Java Add-in	19

Loading QuickTest with Java Add-in Support

You use the Add-in Manager to load support for testing Java applets and applications.

Notes:

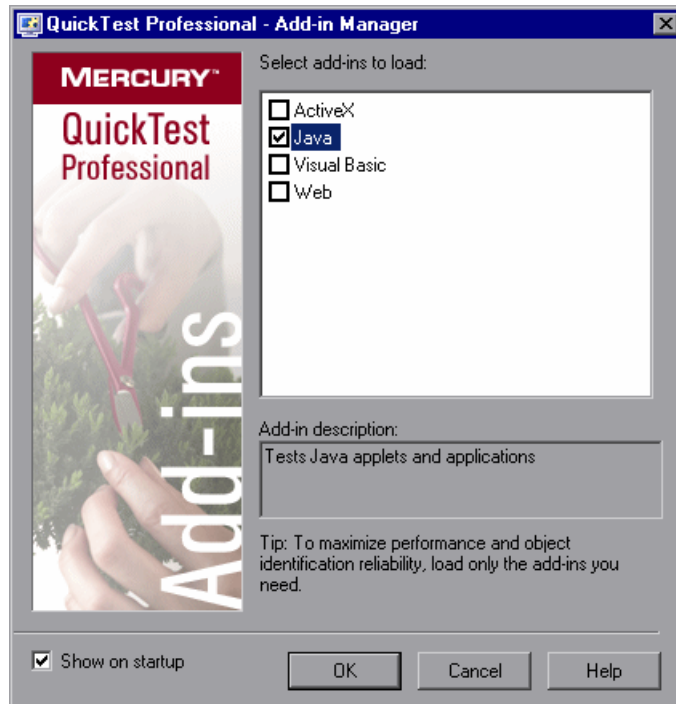
For optimal performance when testing Java applications, it is strongly recommended that you work with *only* the Java Add-in and Web Add-in loaded. Note that the QuickTest Professional Web Add-in does not need to be loaded when using the Java Add-in; it is required only if you are testing Java applets in a Web browser.

When testing applications that do not contain Java objects, it is strongly recommended that you do not load the Java Add-in.

To start QuickTest with add-in support for testing Java applications:

- 1** Choose **Start > Programs > QuickTest Professional > QuickTest Professional**. The QuickTest Professional - Add-in Manager dialog box opens.
(If the Add-in Manager dialog box does not open, see “Displaying the Add-in Manager Dialog Box” on page 18.)

- In the add-in list, select **Java** (or select **Java** and **Web** if you plan to test Java applets in a Web browser).



- Click **OK**.

For more information on the Add-in Manager dialog box, refer to the *QuickTest Professional User's Guide*.

Displaying the Add-in Manager Dialog Box

You can set an option in QuickTest to determine whether the Add-in Manager opens when you open QuickTest, or whether it automatically loads the same add-ins that were loaded in the previous QuickTest session.

To instruct the Add-in Manager dialog box to open when you open QuickTest:

- 1** Choose **Start > Programs > QuickTest Professional > QuickTest Professional** to start QuickTest.
- 2** From the QuickTest menu, choose **Tools > Options** and click the **General** tab.
- 3** Select **Display Add-in Manager on startup**.
- 4** Click **OK** to close the Options dialog box.
- 5** Close and reopen QuickTest.

Loading QuickTest without Java Add-in Support

If you want to work with QuickTest without support for Java, you can load QuickTest without the Java Add-in.

To load QuickTest without add-in support for Java:

- 1** Choose **Start > Programs > QuickTest Professional > QuickTest Professional**. The QuickTest Professional Add-in Manager dialog box opens. (If the Add-in Manager dialog box does not open, see “Displaying the Add-in Manager Dialog Box” above.)
- 2** Clear the **Java** check box and click **OK**. QuickTest opens without add-in support for Java.

Repairing and Uninstalling the Java Add-in

You can repair a QuickTest Professional Java Add-in 9.1 installation that has become corrupted. In addition, you can uninstall the QuickTest Professional Java Add-in 9.1 without uninstalling QuickTest or any other add-ins.

Note: You can also uninstall the QuickTest Professional application entirely. If you uninstall QuickTest Professional, the uninstall program also removes all installed features, including any external add-ins that are installed. For more information on uninstalling QuickTest Professional, refer to the *QuickTest Professional Installation Guide*.

Repairing the Java Add-in Installation

Your QuickTest Professional Java Add-in 9.1 CD-ROM enables you to repair an existing Java Add-in 9.1 installation, by replacing any missing or damaged files from your previous Java Add-in installation.

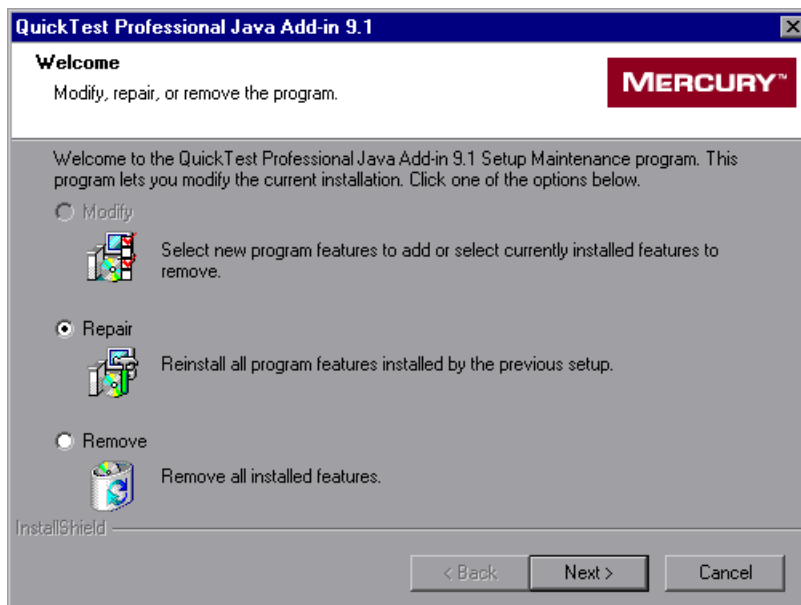
To repair your Java Add-in installation:

- 1 Insert the QuickTest Professional Java Add-in CD-ROM into the CD-ROM drive. If the CD-ROM drive is on your local computer, the QuickTest Professional Java Add-in Setup window opens.

If the CD-ROM is in a network drive, double-click **autorun.exe** in the root folder of the CD-ROM. The QuickTest Professional Java Add-in Setup window opens.

Note: You must use the CD-ROM with the exact same version of the Java Add-in that you used for the original installation.

2 Click **Add-in Setup**. The Welcome screen opens.



3 In the Welcome screen, select **Repair** and click **Next**.

The maintenance program repairs your QuickTest Professional Java Add-in installation.

- 4 The Maintenance Complete screen may prompt you to restart your computer. If it does, you can choose to restart your computer at a later time, but you must restart your computer before you use QuickTest Professional.

Note: You can save any open files, but you should restart your computer as soon as possible. Delaying the system restart could result in unexpected system behavior.

- 5 Click **Finish** to complete the repair process.

Uninstalling the Java Add-in

You can uninstall the Java Add-in using either the **Add/Remove Programs** option in the Windows Control Panel, or using the QuickTest Professional Java Add-in CD-ROM.

To uninstall the Java Add-in using the Add/Remove Programs option:

- 1 Make sure that QuickTest is closed.
- 2 Choose **Start > Settings > Control Panel**.
- 3 Double-click the **Add/Remove Programs** option.
- 4 In the Add/Remove Programs dialog box, select **QuickTest Professional Java Add-in** and then click **Change/Remove**.
- 5 A message prompts you to confirm your decision to uninstall the QuickTest Professional Java Add-in. Click **Yes** to uninstall the Java Add-in. The uninstall program removes the Java Add-in from your computer. QuickTest Professional and any other installed add-ins remain on your computer.

Note: Clicking **No** keeps the Java Add-in installed on your computer.

- 6 The Maintenance Complete screen may prompt you to restart your computer. If it does, you can choose to restart your computer at a later time, but you must restart your computer before you use QuickTest Professional.

Note: You can save any open files, but you should restart your computer as soon as possible. Delaying the system restart could result in unexpected system behavior.

- 7 Click **Finish** to complete the uninstall process.

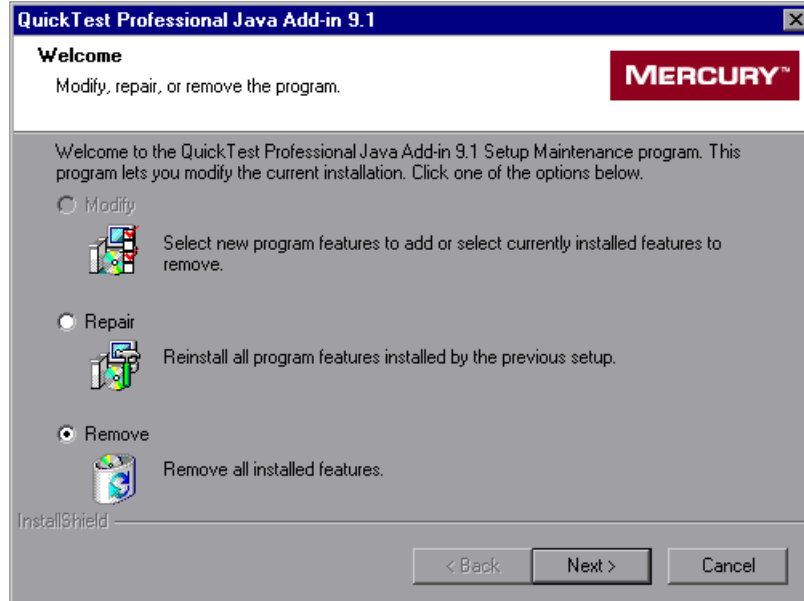
To uninstall the Java Add-in using the QuickTest Professional Java Add-in CD-ROM:

- 1 Insert the QuickTest Professional Java Add-in CD-ROM into the CD-ROM drive. If the CD-ROM drive is on your local computer, the QuickTest Professional Java Add-in Setup window opens.

If the CD-ROM is in a network drive, double-click **autorun.exe** in the root folder of the CD-ROM. The QuickTest Professional Java Add-in Setup window opens.

Note: You must use the CD-ROM with the exact same version of the Java Add-in that you used for the original installation.

2 Click **Add-in Setup**. The Welcome screen opens.



3 In the Welcome screen, select **Remove** and click **Next**.

4 A message prompts you to confirm your decision to uninstall the QuickTest Professional Java Add-in. Click **OK** to uninstall the Java Add-in. The uninstall program removes the Java Add-in from your computer. QuickTest Professional and any other installed add-ins remain on your computer.

Note: Clicking **Cancel** keeps the Java Add-in installed on your computer and returns to the previous screen.

- 5 The Maintenance Complete screen may prompt you to restart your computer. If it does, you can choose to restart your computer at a later time, but you must restart your computer before you use QuickTest Professional.

Note: You can save any open files, but you should restart your computer as soon as possible. Delaying the system restart could result in unexpected system behavior.

- 6 Click **Finish** to complete the uninstall process.

Part II

Working with the Java Add-in

3

Creating and Running Steps on Java Objects

This chapter explains how to use QuickTest to set testing preferences and to record and run steps on Java applets and applications. The chapter assumes basic knowledge of QuickTest features and capabilities. For more information about working with QuickTest, refer to the *QuickTest Professional User's Guide*.

This chapter describes:	On page:
About Creating and Running Steps on Java Objects	28
Understanding the Java Test Object Model	29
Defining Java Testing Options	32
Defining Java Settings for Individual Tests and Components	39
Defining Active Screen Capture Options for Tests	43
Defining Record and Run Options for Tests	45
Recording Tests and Components on Java Objects	49
Running Tests and Components on Java Applications and Applets	56

About Creating and Running Steps on Java Objects

After installing the Java Add-in, Microsoft Internet Explorer, Netscape Browser, Mozilla Firefox, and other Java applications will always open with Mercury Java support active. You can confirm that your Java environment has opened properly by checking the Java console for a message similar to the following confirmation message: "Loading Mercury Interactive QuickTest Professional Java Support (version x.x.x.x) (<App> version x.x.x.x)." (where <App> is IE, IBM, or SUN).

Before creating or running steps on Java objects, you can set Java-specific preferences in the Java tab of the Options dialog box and the Java tab of the Test Settings or Business Component Settings dialog box for the specific test or component, or in the Application Area Settings dialog box for all associated components.

When working with tests, you can also use the Active Screen tab of the Options dialog box to set additional preferences that control the way QuickTest captures information for Java objects in the Active Screen.

Each time you begin recording or running a test, you can use the Java tab of the Record and Run Settings dialog box to instruct QuickTest to activate a specified Java application or applet. Alternatively, you can instruct QuickTest to record and run on any open Java application or applet. If you are running a test on an applet that uses Internet Explorer, Netscape Browser, or Mozilla Firefox, you can also use the Web tab of the Record and Run Settings dialog box to instruct QuickTest to open the relevant browser at the beginning of the run session.

When working with components, the Record and Run Settings dialog box is not available. Therefore, when you record or run a component on a Java application or applet, you must either open and connect to it manually, or create a step using the **OpenApp** operation or a user-defined function that instructs QuickTest to open and connect to the Java application or applet (using the **SystemUtil** utility object). For more information on working with components and user-defined functions, refer to the *QuickTest Professional for Business Process Testing User's Guide*.

As you record on a Java application or applet, QuickTest inserts steps into your test or component that represent the operations you perform. The QuickTest Professional Java Add-in recognizes Java objects such as tree, edit, and table objects. It records these objects in relation to the data selected or entered, and to the object within its parent object.

Instead of recording, you can also instruct QuickTest to learn the objects in your application or applet. You do this by creating a repository of all of the relevant test objects. (You can use the Object Repository Manager's **Navigate and Learn** or **Add Objects** options to create the object repository.) Then, you create each step by choosing the required object and operation (keyword), and then specifying the arguments (values) that apply to that step. This process is known as keyword-driven testing. If you are working with the SWT toolkit, you must use keyword-driven testing to create your test or component. Recording on SWT-based Java objects is not supported. For more information on QuickTest functionality, refer to the *QuickTest Professional User's Guide*.

You run tests or components on Java applications or applets just as you would with any other QuickTest test or component. The test results tree displays the same icons for Java objects as those used in the Keyword View, and if you choose to save screen captures to the test results (**Tools > Options > Run** tab), the bottom right pane of the Test Results window displays the page captured during the run session.

Understanding the Java Test Object Model

The **test object model** is the set of object types, or classes, that QuickTest uses to represent the objects in your application. Each test object class has a list of properties that can uniquely identify objects of that class and a set of relevant methods that QuickTest can perform during a run session.

A **test object** is an object that QuickTest creates in the test or component to represent the actual object in your application and to store information about that object. This information, stored in the object repository, helps QuickTest identify and check the object during a run session.

A **run-time object** is the actual object in your Java applet or application on which methods are performed during a run session.

When you perform an operation on your Java applet or application while recording a test or component, QuickTest:

- identifies the run-time Java object on which you performed the operation (method) and creates a corresponding test object.
- reads the current property values of the object in your application and stores them in the object repository as the test object's identification property values. In addition, for tests, QuickTest stores other properties in the Active Screen data (depending on the capture level).
- chooses a unique name for the object, generally using the value of one of its prominent identification properties.
- records the operation (method) that you performed on the object, and displays the operation as a step in the Keyword View (for tests and components) and the Expert View (for tests only).

For example, suppose you select a radio button next to a specific flight class type in a dialog box in a Flight Reservation Java applet. This radio button has the text **First Class** attached to it.

QuickTest identifies the field as a `JavaRadioButton` object. It creates a `JavaRadioButton` test object with the name **First Class** and records the following property and value as the description for the **First Class** `JavaRadioButton`.

Type	Property	Value
JRC	attached text	First Class

It also records that you performed a **Set** method on the `JavaRadioButton` object.

QuickTest displays your step in the Keyword View like this (last row):

FlightApplet			
First Class	Set		Select the "First Class" radio button.

QuickTest displays your step in the Expert View like this:

```
Window("Microsoft Internet Explorer").JavaApplet("FlightApplet").
JavaRadioButton("First Class").Set
```

Note: When you select a step in the Keyword View or the Expert View in a test, the corresponding object is highlighted in the Active Screen.

When you run a test or component, QuickTest identifies each object in your application by its test object class and its description—the set of identification properties and values used to uniquely identify the object.

In the previous example, during the run session, QuickTest searches in the object repository for the `JavaRadioButton` object with the name `First Class` to look up its description. Based on the description it finds (**attached text** = `First Class`), QuickTest then looks in the application for a `JavaRadioButton` object with the **attached text** `First Class`. When it finds the object, it performs the `Set` method to change the field value to `ON` (selects the radio button).

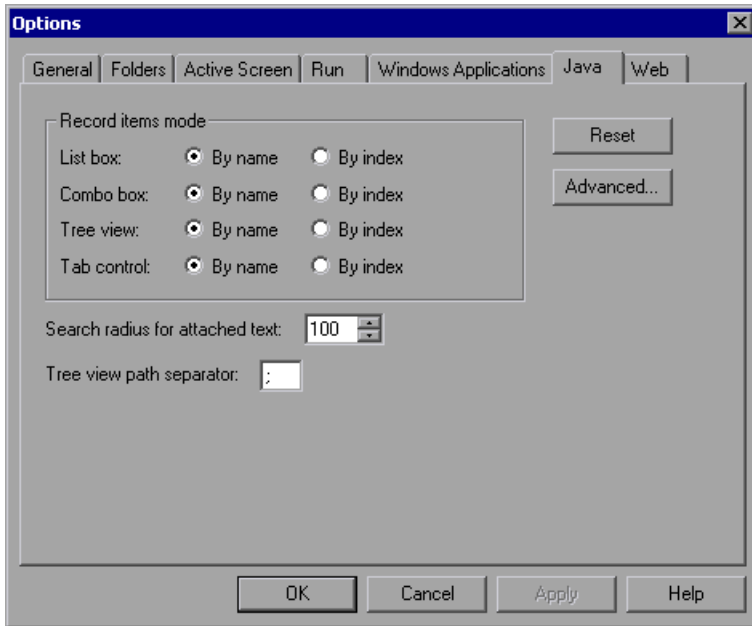
Tip: You can view all the properties and methods of any object in your Java applet or application using the Object Spy (except for SWT-based `JavaMenu` objects). For more information on the Object Spy, refer to the *QuickTest Professional User's Guide*.

For general information on the test object model and the object repository, refer to the relevant chapters in the *QuickTest Professional User's Guide*.

For more information on Java test objects and methods, refer to the Java section in the *QuickTest Professional Object Model Reference*.

Defining Java Testing Options

The Java tab of the Options dialog box (**Tools > Options > Java tab**) enables you to configure how QuickTest records and runs tests on Java applets or applications.



The Java tab is available only when the Java or Oracle Add-ins are installed and loaded. If you are using the Oracle Add-in, and you add steps to your test for Java objects within your Oracle application (or your Oracle test was created using version 6.5 of the Oracle Add-in), the options in this tab/dialog box are relevant for the Java steps in your test.

For more information on the Options dialog box, refer to the *QuickTest Professional User's Guide*.

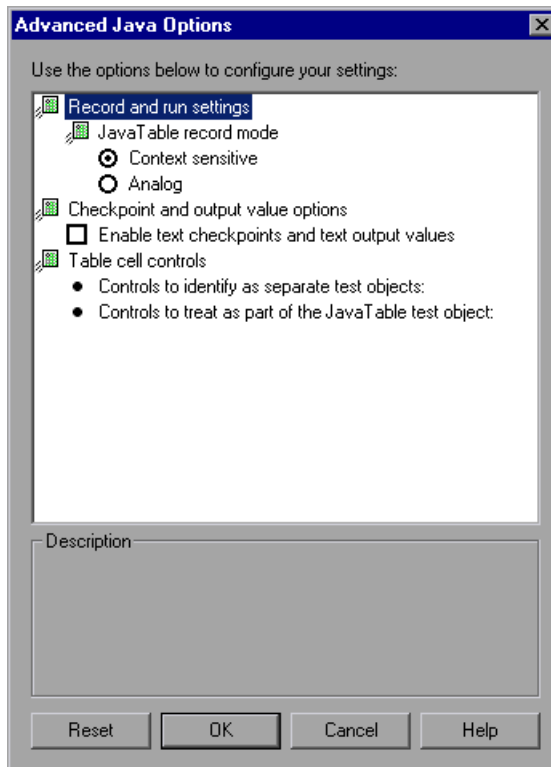
The Java tab includes the following options:

Option	Description
Record items mode	<p>Determines how QuickTest records operations on items in List box, Combo box, Tree view, and Tab control objects. Select one of the following options for each object:</p> <ul style="list-style-type: none"> • By name. Records operations on an item within the object (for example, selected list item or tab) according to the item's name. • By index. Records operations on an item within the object (for example, selected list item or tab) according to the item's position within the Java object. <p>Default value: By name</p> <p>Note: If you choose the By index option for Tree view, do not specify "#" as the default separator in the Tree view path separator option below.</p>
Search radius for attached text	<p>Sets the maximum distance in pixels to search for attached text.</p> <p>Default value: 100</p> <p>Note: This option is relevant only when the label_attr run-time property is unavailable.</p>
Tree view path separator	<p>Specifies the default separator used to separate entries in a path to a node of a Tree view control.</p> <p>Default value: ;</p> <p>Possible value: Any single character</p> <p>Notes: If you specify more than one character, QuickTest treats each of the characters as a separator (but not both of them in sequence). For example, if you specify %\$ for this option and a particular path contains MyNode\$%MySubNode, then QuickTest treats the \$ character (one of the two characters you specified) as the separator, but treats the % character as the first character of the second node.</p> <p>If you choose the By index option for Tree View in the Record Items mode area above, do not specify "#" as the default separator.</p>

Option	Description
Reset	Resets the Java test settings to their default values.
Advanced	Opens the Advanced Java Options dialog box. For more information, see “Understanding the Advanced Java Options Dialog Box”, below.

Understanding the Advanced Java Options Dialog Box

The Advanced Java Options dialog box (**Tools > Options > Java tab > Advanced** button) enables you to specify additional Java options. You can configure table record mode preferences, enable support for text checkpoints and text output values, and specify lists of controls.



Note: If you are using the Oracle Add-in, and you add steps to your test for Java objects within your Oracle application (or your Oracle test was created using version 6.5 of the Oracle Add-in), the options in this tab/dialog box are relevant for the Java steps in your test.

The Advanced Java Options dialog box includes the following options:

JavaTable record mode

Sets the record mode for table objects. Choose one of the following modes:

- **Context Sensitive.** Records operations on table objects in context-sensitive mode.
- **Analog.** Records only low-level (analog) table methods: **ClickCell**, **DoubleClickCell**, and **Drag**.

Default value: **Context sensitive**

Note: This option corresponds to the `Setting.Java("table_record_mode")` variable.

Checkpoint and output value options

Sets preferences for checkpoint and output value steps on Java objects. It contains the following option:

- **Enable text checkpoints and text output values**

Enables you use the QuickTest text recognition mechanism to check or retrieve text values displayed in Java objects.

Default value: **Disabled**

Note: Because the text recognition mechanism is supported only for Java objects that meet very specific criteria, this option is disabled by default. For more information, see “Considerations for Using Text Checkpoints and Text Output Value Steps with Java Objects” on page 38.

Table cell controls

Sets preferences for the way that QuickTest identifies controls inside table cells. It includes the following options:

► **Controls to identify as separate test objects**

Specifies the list of controls that you want QuickTest to identify as separate test objects and not as part of a `JavaTable` object. Use this option to access methods that are specific to the object type or to otherwise improve the functionality of steps that QuickTest would normally record and run as operations on a `JavaTable` object.

Notes:

- This option is relevant for `JTable` Swing toolkit tables.
 - Specify control class names separated by a space, tab, newline, or return character. Values are case sensitive.
 - This option corresponds to the `Setting.Java("table_internal_editors_list")` variable.
-

For information on changing the settings for this option, see “Modifying Table Cell Controls Options” on page 37.

► **Controls to treat as part of the `JavaTable` test object**

Specifies the list of controls for which you want QuickTest to record and run `JavaTable` operations. Use this option to record and run `JavaTable` operations (such as `SetCellData` and `Select`) on controls that QuickTest would normally treat as separate test objects.

Notes:

- ▶ This option is relevant for JTable Swing toolkit tables.
 - ▶ Specify editor class names separated by a space, tab, newline, or return character. Values are case sensitive.
 - ▶ This option corresponds to the `Setting.Java("table_external_editors_list")` variable.
-

For information on changing the settings for this option, see “Modifying Table Cell Controls Options” on page 37.

Modifying Table Cell Controls Options

In the Advanced Java Options dialog box, you can specify a list of table cell controls that you want QuickTest to identify as separate test objects. You can also specify a list of table cell controls for which you want QuickTest to record and run JavaTable operations.

To modify a Table Cell Controls option:

- 1** In the Advanced Options dialog box, click the relevant option once to highlight it.
 - 2** Click the option again or press F2 to open an edit box in which you can add or modify a list of controls.
 - 3** Change the value as necessary.
-

Note: Specify editor class names separated by a space, tab, newline, or return character. Values are case sensitive.

- 4** When you finish editing the value, click another location in the dialog box to set the value.
- 5** When you finish making all of the required changes in this dialog box, click **OK** and close the dialog box.

Notes:

- ▶ Your changes are not applied to the currently open test or component. To apply your changes, close your test or component and reopen it.
 - ▶ You can restore the default settings in the Advanced Java Options dialog box by clicking the **Reset** button.
-

Considerations for Using Text Checkpoints and Text Output Value Steps with Java Objects

When working with tests, you can use checkpoints or output values to check that text in your Java application or applet displays correctly. Similar to many other supported environments, it is recommended to retrieve and check text from your Java applet or application by inserting a standard checkpoint or output value for the object containing the desired text, and selecting to check or output its **text** (or similar) identification property (for example, **text**, **attached text**, or **label**).

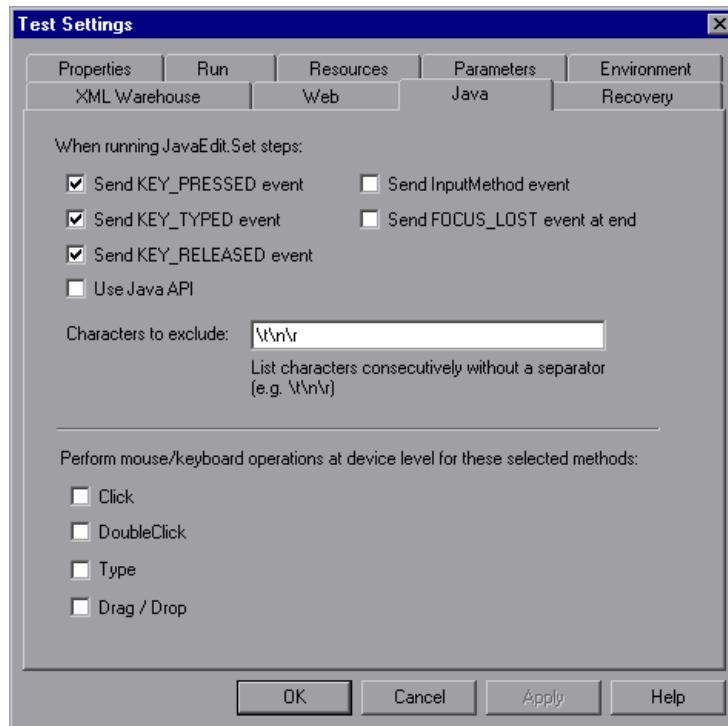
If the object you want to work with does not have an appropriate identification property, or, if for any other reason, the above recommendation does not answer your needs (for example, the text before or after the selected text is important), you can consider inserting a QuickTest text checkpoint or text output value step for a Java object if it meets the following criteria:

- ▶ Text checkpoints and text output values do not work for custom drawn controls.
- ▶ The object must draw text by overriding the **paint()** method and calling the **standard graphics.drawString()** method to draw text. For example, the object cannot use special drawing methods for writing text, such as using a method that can draw oval circles to draw the letter O.
- ▶ The object cannot use the **double (image) buffering** drawing technique.

Note: Because many Java objects do not answer these criteria, the text checkpoint and text output mechanism for Java objects is disabled by default.

Defining Java Settings for Individual Tests and Components

You set Java test or component variables using the Test Settings, Business Component Settings, or Application Area Settings dialog box. (By default, the Business Component Settings dialog box displays the options selected in the Application Area Settings dialog box. However, you can modify the options in the Business Component Settings dialog box to suit your component's needs.)



Note: The options shown in the Java tab of the Test Settings dialog box (above) are the same options that are available in the Application Area Settings dialog box and the Business Component Settings dialog box.

The Java tab is available only when the Java or Oracle Add-ins are installed and loaded. If you are using the Oracle Add-in, and you add steps to your test for Java objects within your Oracle application (or your Oracle test was created using version 6.5 of the Oracle Add-in), the options in this tab/dialog box are relevant for the Java steps in your test.

For more information on the Test Settings dialog box, refer to the *QuickTest Professional User's Guide*. For more information on the Business Component Settings and Application Area Settings dialog boxes, refer to the *QuickTest Professional for Business Process Testing User's Guide*.

The Java tab in the Test, Component, and Application Area Settings dialog boxes includes the following options:

When running JavaEdit.Set steps

Specifies how operations are performed on edit boxes during a test run. It is recommended not to modify these settings unless you fully understand Java key events and input methods, as well as the implications of sending or not sending these events. Note that **JavaEdit.Set** steps may fail during a run session if an incorrect value is used for these settings. You can set one or more of the following options:

- ▶ **Send KEY_PRESSED event.** Sends a KEY_PRESSED event to the object for every character from the input string. (Selected by default.)

This setting corresponds to the **P** value of the `Setting.Java("edit_replay_mode")` variable.

- ▶ **Send KEY_TYPED event.** Sends a KEY_TYPED event to the object for every character from the input string. (Selected by default.)

This setting corresponds to the **T** value of the `Setting.Java("edit_replay_mode")` variable.

- ▶ **Send KEY_RELEASED event.** Sends a KEY_RELEASED event to the object for every character from the input string. (Selected by default.)

This setting corresponds to the **R** value of the `Setting.Java("edit_replay_mode")` variable.

- ▶ **Use Java API.** Sends the `setValue()` method to set a value of the edit object.

This setting corresponds to the **S** value of the `Setting.Java("edit_replay_mode")` variable.

- ▶ **Send InputMethod event.** Sends an **InputMethod** event to the object for every character from the input string. This event is used with Unicode applications (for example, for some non-English applications).

This setting corresponds to the **I** value of the `Setting.Java("edit_replay_mode")` variable.

- ▶ **Send FOCUS_LOST event at end.** Generates a FOCUS_LOST event after running the step.

This setting corresponds to the **F** value of the `Setting.Java("edit_replay_mode")` variable.

For more information about AWT-based Java key events and input methods, refer to Java documentation at <http://java.sun.com>.

For more information about SWT-based Java key events, refer to Java documentation at <http://www.eclipse.org/SWT>.

Characters to exclude

Instructs QuickTest to ignore the specified characters during a run session. This option is relevant only if the **Use Java API** check box is selected in the upper section of this dialog box, or if the value of the `Setting.Java("edit_replay_mode")` variable is set to **S**. List characters consecutively, without a separator.

Default value: `\t\n\r`

This setting corresponds to the `Setting.Java("exclude_control_chars")` variable.

Perform mouse/keyboard operations at device level for these selected methods

By default, QuickTest performs mouse operations at the context-sensitive level. You can use this option to select specific operations to perform using device-level replay. Device-level replay simulates mouse or key operations exactly as if they occur on the mouse or keyboard drivers. When a mouse action is simulated on device replay, the mouse pointer moves on the screen to the point where the action is to be performed during the run session. You can select from the following mouse and keyboard methods:

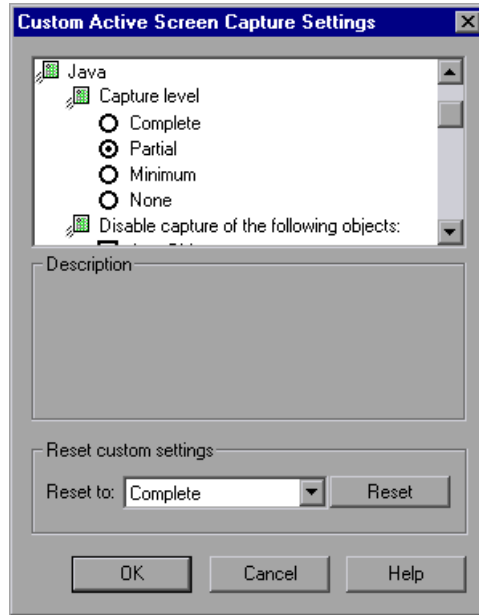
- **Click**
- **DoubleClick**
- **Type**
- **Drag / Drop**

Default value: All check boxes are cleared.

This option corresponds to the `Setting.Java("device_replay_mode")` variable.

Defining Active Screen Capture Options for Tests

When working with tests, the Active Screen tab of the Options dialog box (**Tools > Options > Active Screen**) enables you to specify active screen capture settings for all environments. The Custom Active Screen Capture Settings dialog box (**Tools > Options > Active Screen > Custom Level**) enables you to customize how QuickTest captures and saves Active Screen information for specific environments.



In addition to the core options described in the *QuickTest Professional User's Guide*, the Custom Active Screen Capture Settings dialog box contains the following Java-specific options:

- **Capture level.** You can specify for which Java objects identification properties are captured when a Java applet or application is captured for the Active Screen:
- **Complete.** Instructs QuickTest to save all identification properties of all objects in the application or applet's open window/dialog box in the Active Screen of each step.

- ▶ **Partial.** (Default) Instructs QuickTest to save all identification properties of all objects in the application or applet's open window/dialog box in the Active Screen of the first step performed in that window, plus all properties of the recorded object only, in subsequent steps in the same window.
- ▶ **Minimum.** Instructs QuickTest to save all identification properties for the recorded object plus all identification properties for the parent objects in the recording hierarchy.
- ▶ **None.** Disables capture of Active Screen files for Java applets or applications.

Depending on your testing requirements, you can choose between different levels of Active Screen capture. However, you should take into consideration that the less information captured for the Active Screen, the better the performance.

For example, if you choose **Complete**, you can add checkpoints to every test object that is displayed in any Active Screen capture, but it will take more time and use more disk space to record a single operation. Alternatively, if you choose **Minimum** or **Partial**, QuickTest can record faster and use less disk space, but there may be limitations on the operations you can perform from the Active Screen after recording.

- ▶ **Disable capture of the following objects.** Prevents QuickTest from capturing the data of steps performed on other objects for the selected test object types in the Active Screen. These objects will be visible in the Active Screen as images only. This setting is relevant only when the **Complete** or **Minimal** capture levels are selected.

Note: If you record on a specific test object, its identification properties will be captured even if the **Disable capture of the following objects** option is selected.

Default = **JavaObject** and **JavaMenu** selected (meaning that identification properties are not captured for these objects).

Tip: When you apply custom Active Screen settings, you override your previous capture-level settings with all of the settings in the Custom Active Screen Capture Settings dialog box. If you want to customize only specific settings, use the **Reset to** option to ensure that all other settings are using the capture-level setting you prefer and then modify the specific settings you need.

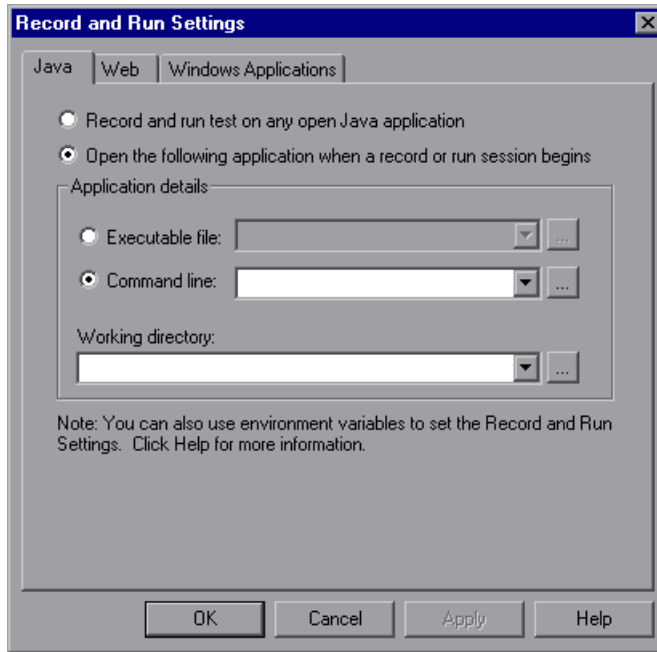
Defining Record and Run Options for Tests

You can use the Java tab of the Record and Run Settings dialog box (**Automation > Record and Run Settings > Java tab**) to instruct QuickTest to open your Java applet or application each time you begin a recording session, or to instruct QuickTest to record on any open Java application.

Note: Components do not require specific record and run settings to work with Java applets and applications. To record a component, you need to first open the Java applet or application manually. Alternatively, you can include steps in your component that connect to the Java applet or application, for example, you can include a step that contains the **OpenApp** operation.

When you begin recording a new component, the Applications dialog box opens (unless you previously specified a Windows environment in the Application Area Settings or Business Component Settings dialog box). Simply close the dialog box to begin recording. For more information on the Applications tab and Applications dialog box, refer to the *QuickTest Professional User's Guide*.

If you do not modify the record and run settings before you begin recording, the Record and Run Settings dialog box opens automatically when you begin recording a new test (by clicking **Record** or choosing **Automation > Record**). You can also open this dialog box by choosing **Automation > Record and Run Settings**.



Notes:

- ▶ When testing Java applets in a Web browser, you must load both the Web Add-in and the Java Add-in. In this case, you use the Web tab of the Record and Run Settings dialog box to specify your record and run preferences. For more information on the Web tab, refer to the *QuickTest Professional User's Guide*.
 - ▶ The Java and Web tabs shown in the image above are available only when the corresponding add-ins are installed and loaded. If other add-in(s) are loaded, the corresponding tabs (if any) are also displayed.
-

When you run the test, or if you begin a new recording session on an existing test, QuickTest automatically uses the existing record and run settings for the test and does not open the Record and Run Settings dialog box. However, it is important to confirm that the options in the Record and Run Settings Java tab are appropriate for the first step of your test before running it because you (or someone else) may have modified the Record and Run Settings dialog box manually in a prior record session.

The Java tab includes the following options:

Option	Description
Record and run test on any open Java application	Instructs QuickTest to record and run the test on any open Java application or applet.
Open the following application when a record or run session begins	Instructs QuickTest to open a new Java application or applet using the specified application details. Note: When working with a Java applet inside a browser, use the Web tab of the Record and Run dialog box to open the URL containing the applet.
Application details	Defines details of the Java application on which to run the test: <ul style="list-style-type: none"> • Executable file. Instructs QuickTest to open the specified executable or batch file. • Command line. Instructs QuickTest to open the application from the specified command line. • Working directory. Instructs QuickTest to run the specified executable file or command line from the specified directory. Make sure you specify the full directory path, for example, C:\Program Files\Java\jre1.6.0\bin. Note: If you define values for the EXEPATH_ENV, CMDLINE_ENV, and/or WORKDIR_ENV test environment variables, these values override the values in the Executable file , Command line , and Working Directory boxes of the Java tab during a run session. For more information, see “Defining Application Details Environment Variables for Tests” on page 48.

Defining Application Details Environment Variables for Tests

You can use application details environment variables to specify the applications you want to use for recording and running your test. If you define any of these application details environment variables, they override the values in the **Executable file**, **Command line**, and **Working directory** boxes in the Java tab of the Record and Run Settings dialog box. For more information, see “Defining Record and Run Options for Tests” on page 45.

Use the variable names listed in the table below to define Java application details:

Option	Variable Name	Description
Executable file	EXEPATH_ENV	The executable file or a batch file to open.
Command line	CMDLINE_ENV	The command line to use to open the file.
Working directory	WORKDIR_ENV	The folder to which the specified command line or executable file refers.

For more information on defining and working with environment variables, refer to the *QuickTest Professional User's Guide*.

Optimizing Settings for Other Record and Run Settings Dialog Box Tabs

In addition to setting the appropriate settings in the Java tab (or Web tab for applets in browsers), you should confirm that the other tabs in the dialog box have the appropriate settings for your test. The following settings are recommended:

- **Windows Applications tab.** Choose **Record and run on these applications (opened when a session begins)** and confirm that the list of Windows applications is empty.
- **Other tabs.** (If displayed.) Choose the option to record and run on any open application (upper radio button of each tab).

While these settings do not directly affect your record or run sessions when working with Java applets and applications, these settings prevent you from inadvertently recording operations performed on Windows applications (such as e-mail) during your recording session. These settings also prevent QuickTest from opening unnecessary applications when you record or run tests on Java applets and applications.

For more information on the Record and Run Settings dialog box, refer to the *QuickTest Professional User's Guide*.

Recording Tests and Components on Java Objects

When you record an operation on an applet, application, or Java object, QuickTest records the appropriate object icon next to the step in the Keyword View (for tests and components) and adds the relevant statement in the Expert View (for tests only).

If you try to record an operation on an unsupported or custom Java object, QuickTest records a generic **JavaObject.Click** statement that includes the coordinates of the click and the mouse button (that is, left or right) that was clicked. You can create support for your custom object using the QuickTest Professional Java Add-in Extensibility. For more information, refer to the *QuickTest Professional Java Add-in Extensibility Developer's Guide*.

Note: The way in which QuickTest records operations depends on the type of JTable cell editor in the table cell. For more information, see Recording on Table Objects.

The QuickTest recorded hierarchy is composed of two or three levels of Java test objects. The top level is represented by the `JavaApplet`, `JavaDialog`, or `JavaWindow` object, as appropriate. The actual object on which you performed an operation may be recorded as a second or third level object. If the object is located directly in the top level object, it is recorded as a second level object (for example, `JavaApplet.JavaButton`). If a `JavaDialog` or `JavaInternalFrame` exists at the second level, then the object on which you performed the operation is recorded as a third level object (for example, `JavaWindow.JavaDialog.JavaButton`).

When testing applets in a browser, the two- or three-level hierarchy is recorded within the standard Web object hierarchy (for example, `Browser.Page.JavaApplet.JavaTestObject.SubJavaTestObject`).

Even though the object on which you record may be embedded in several levels of objects, the recorded hierarchy does not include these objects. For example, if the `JavaList` object on which you record is actually contained in several `JPanel` objects, which are all contained in a `JavaWindow`, the recorded hierarchy is only `JavaWindow.JavaList`.

For example, in a test, if you record a click on a Java check box, the Keyword View may be displayed as follows:

▼ Action1			
▼ Microsoft Internet Explorer			
msctls_statusbar32	Click	776,2	Click the "msctls_statusbar32" status bar.
▼ Periodic			
Toggle	Set	"ON"	Set the state of the "Toggle" check box to "ON".

QuickTest records this step in the Expert View as:

```
Window("Microsoft Internet Explorer").JavaApplet("Periodic").
  JavaCheckBox("Toggle").Set "ON"
```

In a component, if you record a click on this same Java check box, the Keyword View would displayed as follows:

Toggle	Set	"ON"	Set the state of the "Toggle" check box to "ON".
--------	-----	------	--

You can view the recorded hierarchy of a test object in the object repository.

For information on accessing the full hierarchy of a test object, see “Viewing the Full Object Hierarchy” on page 59.

Recording on Table Objects

When you record an operation that changes the data in a cell of a Java table object, QuickTest generally records the end result of the data in the cell in the form of a `JavaTable.SetCellData` statement. (`JavaTable.SetCellData` is not used when the `JavaTable record mode` is set to `Analog`. For more information on `JavaTable` record mode, see “Understanding the Advanced Java Options Dialog Box” on page 34.)

Recording on Standard Cell Editors in Swing JTable Tables

The QuickTest Professional Java Add-in also provides built-in support for several standard Swing `JTable` cell editor types. This means that by default, QuickTest records operations on these standard cell editors in the same way as other table objects, using `SetCellData` statements.

Recording on Custom Cell Editors in Swing JTable Tables

When a `JTable` contains a custom (non-standard) cell editor, the default `SetCellData` statement cannot be recorded. For example, if a cell contains both a check box and a button that opens a dialog box, then a `SetCellData` statement may not always provide an accurate description of the operation(s) performed inside the cell.

If you record an operation on a custom cell editor, QuickTest records a statement that reflects the operation you performed on the object inside of the cell. For example, if the cell editor contains a custom check box, QuickTest might record the following statement:

```
Browser("Periodic").Page("Periodic").JavaWindow("CoolJava").JavaDialog("Set
Options").JavaCheckBox("MyCheckBox").Set "ON"
```

instead of:

```
Browser("Periodic").Page("Periodic").JavaWindow("CoolJava").JavaDialog("Set
Options").JavaTable("MyTable").SetCellData "ON"
```

Modifying the Default JTable Recording Behavior (Advanced)

In most cases, the default recording behavior for JTables (described in the preceding sections) works well and maximizes the readability of your test. However, if you are not satisfied with the value that QuickTest records for the **SetCellData** statement of a particular editor, you can set that editor to be recorded, like a custom cell editor, in terms of the operation performed on the object inside the cell.

To do this, use the **Table cell controls > Controls to identify as separate test objects** option in the Advanced Java Options dialog box and specify specific cell editor type(s) that should always be treated as separate objects, and not as part of a JavaTable object. Alternatively, use a **Setting.Java** ("table_internal_editors_list") statement. For more information, see "Understanding the Advanced Java Options Dialog Box" on page 34 and refer to the *QuickTest Professional Object Model Reference*.

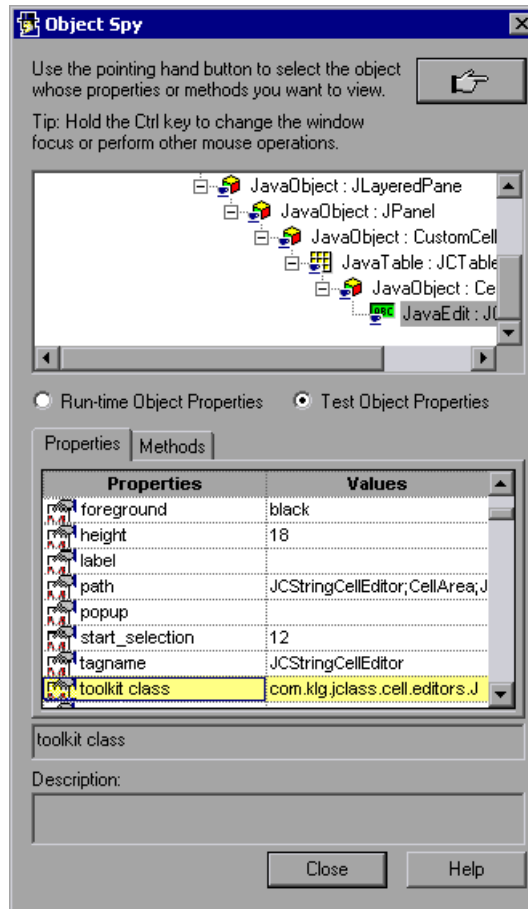
Finding the Toolkit Class of a JTable Cell Editor

If you do not know the value of the toolkit class for an editor for use with the **table_external_editors_list** variable, you can find it either by using the Object Spy (tests and components), by running a short test in QuickTest to retrieve the value (tests only), or by creating a user-defined function and inserting it as a step (tests and components).

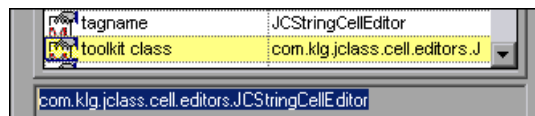
To find the toolkit class of a JTable cell editor using the Object Spy:

- 1** Open the table and activate a cell in the cell editor column. For example, make sure the cursor is blinking inside an edit field or display the drop-down list of a combo box.
- 2** With the appropriate cell activated, use the Object Spy to point to the active cell. For information on using the Object Spy, refer to the *QuickTest Professional User's Guide*.

- 3 Make sure the Properties tab of the Object Spy is displayed and select the **Test Object Properties** radio button.



- 4 In the **Properties** column, scroll to **toolkit class**.
- 5 In the **Values** column, select the value of the **toolkit class**. The value is displayed in the box below the Properties tab.



- 6 Copy and paste the value from the Object Spy to the **Table cell controls > Controls to identify as separate test objects** option or your Setting.Java ("table_internal_editors_list") statement.

Finding the Toolkit Class of a JTable Editor by Running a QuickTest Script

For some cell editors, it is difficult or impossible to capture an activated cell with the Object Spy because the cell does not stay activated for a long enough period of time. For example, after a check box is selected or cleared, the cell is no longer active. If you need to find the toolkit class value to use for these types of cell editors, you can run a short test in QuickTest to retrieve the value. If you are working with components, you can create a user-defined function and insert it as a step.

When working with Java 2 JVMs, insert steps similar to the following example:

```
' Sample test for Java 2 JVMs to retrieve the toolkit class of a table cell editor  
' that cannot be made continuously active
```

```
Set table = JavaWindow("TableDemo").JavaTable("Left table").Object  
Set jTableCS = table.mic_get_supp_class()  
Set comp = jTableCS.getComponentAt(table, 0, 6) 'row 0, col 6  
MsgBox comp.getClass().getName()
```

```
' Set the value of TABLE_EXTERNAL_EDITORS_LIST  
Setting.Java("TABLE_EXTERNAL_EDITORS_LIST") =  
comp.getClass().getName()
```

When working with Java 1 JVMs (JDK 1.1.x-based), insert steps similar to the following example:

```
' Sample test for Java 1.1.x JVMs to retrieve the toolkit class of a table cell editor
' that cannot be made continuously active
```

```
Set table = JavaWindow("TableDemo").JavaTable("Left table").Object
```

```
Set editor = table.getCellEditor(0, 6)
```

```
Set comp = editor.getTableCellEditorComponent(table,
editor.getCellEditorValue(), 0, 0, 6)
```

```
MsgBox comp.getClass().getName()
```

```
' Set the value of TABLE_EXTERNAL_EDITORS_LIST
```

```
Setting.Java("TABLE_EXTERNAL_EDITORS_LIST") =
comp.getClass().getName()
```

Recording on SWT-based Java Objects

If you record on a context menu in an SWT-based Java application, QuickTest identifies and records the object as a Standard Windows WinMenu (instead of a Java object), for example:

```
JavaWindow("Java - CustomClassReplayMethod").WinMenu("ContextMenu")
.Select "MyProject"
```

In general, recording is not supported for SWT-based Java objects. If you want to include an SWT-based Java object in a step, you must first add it to the object repository. You can then choose the required object and operation for the step, and specify the values for that step. For more information, refer to the *QuickTest Professional User's Guide*.

Running Tests and Components on Java Applications and Applets

You run tests and components containing Java test objects in the same way you run any other test or component. For tests, you can use the Record and Run Settings dialog box to instruct QuickTest to open your Java application or applet each time you begin running your test, or you can instruct QuickTest to run on open applications or applets. For components, you can open and connect to your Java application or applet manually, or you can insert a step using the **OpenApp** operation (or a user-defined function) to open the Java application or applet.

You can view the details of your run session in the Test Results window. The test results tree displays the same Java test object icons as those used in the Keyword View. If you choose to save screen captures to the test results (**Tools > Options > Run** tab), the bottom right pane of the Test Results window displays the page captured during the run session. The corresponding Java object is highlighted in the captured screen as you move through each step in the test results tree.

For more information on running tests and components, and analyzing the results, refer to the *QuickTest Professional User's Guide*.

4

Enhancing Your Java Test

After you create your test, you can enhance it by adding checkpoints and outputting values, parameterizing values, and inserting Java objects, methods, and properties into your test.

This chapter describes:	On page:
About Enhancing Your Java Test	58
Viewing the Full Object Hierarchy	59
Checking Java Objects and Outputting Values	61
Using Java Objects, Methods, and Properties to Enhance Your Test	63

Note: Most of the features described in this chapter are relevant only for tests (and scripted components). For information on the features that are available when working with business components, refer to the *QuickTest Professional for Business Process Testing User's Guide*.

About Enhancing Your Java Test

After you create a test, you can use a variety of options to enhance it. This section describes some of the options you can use to create comprehensive tests for your Java application or applet.

Note: Checkpoints and output values are not available for business components.

- You can add checkpoints to your test to check that the actual Java objects in your application match expectations. A **checkpoint** is a step in your test that compares actual values during a run session with expected values, which are stored in your test. This enables you to identify whether or not your Java application or applet is functioning correctly.

For example, you can create a checkpoint on a `JavaButton` object to make sure the button is not enabled before a particular step (such as selecting a check box that enables a button) and another checkpoint to confirm that the same button is enabled after that step is performed. For more information about checkpoints, see “Checking Java Objects and Outputting Values” on page 61.

The results of the checkpoint can be viewed in the QuickTest Test Results window. For more information on the Test Results window, refer to the *QuickTest Professional User's Guide*.

- You can retrieve values from your application during the test run and store them as output values. You can subsequently use these values as input parameters in your test. For more information about output values, see “Checking Java Objects and Outputting Values” on page 61.
- You can parameterize steps to replace fixed values with values from an external source during your run session. The values can come from a data table, environment variables that you define, or values that QuickTest generates during the run session. For more information about parameterizing tests, refer to the *QuickTest Professional User's Guide*.

- ▶ You can specify test or action parameters to pass values into and from your test, and between actions in your test. For more information, refer to the *QuickTest Professional User's Guide*.

Note: You can pass data between components using component parameters. You can pass data within the steps of the current component using local parameters. For more information, refer to the *QuickTest Professional for Business Process Testing User's Guide*.

- ▶ You can insert statements containing Java test object methods and properties in the Keyword View, in the Expert View, or using the Step Generator (**Insert > Step > Step Generator**). This enables you to include other operations that may not have been added while recording, such as retrieving property values, checking that objects exist, and checking other elements of your application. For more information about Java test object methods and properties, see “Using Java Objects, Methods, and Properties to Enhance Your Test” on page 63 and the **Java** section of the *QuickTest Professional Object Model Reference*.

Viewing the Full Object Hierarchy

The Java Add-in enables you to view the full object hierarchy of each of the objects in your application in the Object Spy and Object Selection dialog boxes. In contrast to the recorded object hierarchy, the full object hierarchy shows you all of the parent objects associated with the clicked locations and, in some cases, the child objects of the clicked object.

The full object hierarchy enables you to view associated methods and properties of non-recorded objects in the Object Spy. When working with tests, you can also access non-recorded objects from the Object Selection dialog box that opens when using the Step Generator or when inserting a checkpoint or output value step during a recording session.

The Object Spy and Object Selection dialog boxes enable you to insert statements or perform operations even for elements of an object (class components) that are not recorded, such as **java.awt.Component**. For example, you can access the edit box, drop-down list, and button elements of a combo box.

Notes:



















- ▶ The bottommost object in the Object Selection dialog box hierarchy may be a child of the object you selected using the pointing hand mechanism. Therefore, when you insert a checkpoint, output value, or method on an object while recording, make sure that you select the required object in the Object Selection dialog box.
- ▶ The Active Screen captures only the recorded object hierarchy. Therefore you cannot view the full object hierarchy in the Object Selection dialog box when inserting steps from the Active Screen.

For more information on the Object Spy and Object Selection dialog boxes, refer to the *QuickTest Professional User's Guide*.

Note: You cannot add SWT-based JavaMenu objects directly to an object repository using the **Add Objects to Local** button in the Object Repository window or the **Add Objects** button in the Object Repository Manager. If you want to add an SWT-based JavaMenu object to the object repository, you can use the **Add Objects** or **Add Objects to Local** button to add its parent object and then select to add the parent object together with its descendants.

Identifying Java Objects in the Object Spy or Object Selection Dialog Box

The Object Spy and Object Selection dialog boxes display the name of each object in the hierarchy with an icon that shows the object class. The following is a list of Java-specific test object classes and icons:

Icon	Test Object Class	Icon	Test Object Class
	JavaApplet		JavaRadioButton
	JavaButton		JavaSlider
	JavaCheckBox		JavaSpin
	JavaDialog		JavaStaticText
	JavaEdit		JavaTab
	JavaInternalFrame		JavaTable
	JavaList		JavaToolbar
	JavaMenu		JavaTree
	JavaObject		JavaWindow

Note: The Object Spy is not supported for SWT-based JavaMenu objects.

Checking Java Objects and Outputting Values

After you record a test, you can use a variety of options to enhance it. The Java Add-in provides a variety of checkpoint verifications to ensure that your Java applet or application works as expected. You can check object property values, table cell content, and the external appearance of any object or area in your applet or application. You can also output property or text values from the objects in your Java application or applet for use later in your test.

You use checkpoints and output values for Java objects similar to the way you check or output the values for any other object. The following checkpoints and output values are supported when testing Java objects:

- ▶ **Standard** checkpoints check the property values of an object in your Java application or applet. You can also insert a standard output value step to retrieve object property values.
- ▶ **Table** checkpoints check information within a table in your Java application or applet. You can also create a table output value from the contents of a table cell.
- ▶ **Bitmap** checkpoints check an area of your Java application or applet as a bitmap.
- ▶ **Text** checkpoints check that a text string is displayed in the appropriate place in your Java applet or application. You can also create a text output value from a text string.

Note: Because text checkpoints and text output values are supported only for Java objects that meet certain criteria, these features are disabled for Java objects by default. For more information, see “Considerations for Using Text Checkpoints and Text Output Value Steps with Java Objects” on page 38.

For more information about checkpoints and output values, refer to the *QuickTest Professional User’s Guide*.

Using Java Objects, Methods, and Properties to Enhance Your Test

A test consists of statements coded in Microsoft VBScript. These statements are composed of objects, methods, and/or properties that instruct QuickTest to perform operations or retrieve information. When you record, these statements are generated automatically in response to input to the application. You can add non-recordable functionality and otherwise enhance your test by adding and modifying statements manually in the Keyword View, Expert View, or using the Step Generator. You can mix recorded and programmed statements in the same test.

After you record a basic test and enhance it with checkpoints, parameters, and output values, you may also want to add statements containing test object methods using the Step Generator (**Insert > Step Generator**). You can also enter VBScript programming statements in the Expert View. This enables you to include operations in your test that you did not, or could not, add while recording. These can include retrieving property values, checking that objects exist, enumerating objects, and checking other elements of your application.

The Java Add-in supports IntelliSense and statement completion in the Expert View. The Java Add-in also supports generating statements that access run-time methods and properties in the Step Generator.

You can use Java test object identification property values to confirm that objects in your application look and behave as expected. In addition to checking property values using standard checkpoints, or you can retrieve the values of identification properties during the run session using the **GetROProperty** or **GetROProperties** methods.

Tip: You can use the Object Spy to view all identification properties for any Java test object (except for SWT-based JavaMenu objects).

For more information about the Keyword View, Expert View, and Step Generator, refer to the *QuickTest Professional User's Guide*.

For detailed information, syntax, and examples of all test object methods and properties, refer to the **Java** section of the *QuickTest Professional Object Model Reference*.

Activating Methods Associated with a Java Object

In addition to the Java-specific test objects and methods, you can also use the **Object** property to activate a protected, private, or public Java method for any Java object. The **Object** property is available for all Java objects.

Note: When working with the Microsoft Internet Explorer Virtual Machine, you can activate only public methods using the **Object** property.

If you are not sure which methods your object uses or which arguments you need to send to the method, you can use the Object Spy to view the run-time methods of any object in your applet or application (except for SWT-based JavaMenu objects). For more information, refer to the *QuickTest Professional User's Guide*.

Activating a method for a Java object has the following syntax:

JavaTestObject.**Object**.*Method_to_activate*()

For example, suppose the **getBounds** method is supported for your JButton. To activate the **getBounds** method, insert the following statement into your test script:

```
Set rect=Browser("Browser").Page("PushButtonDemo").JavaApplet  
("pushbutton.html").JavaButton("Enter").Object.getBounds()
```

By storing the Java object returned from a prior **Object** property statement in an object reference, you can later use that object to activate its methods. You activate the methods of a returned object directly (without using the **Object** property).

For example, you can use the following statement to return the BtnObj:

```
Set BtnObj = Browser("Flight Reservation").Page("Flight Reservation").
JavaApplet("FlightLogin").JavaButton("OK").Object
```

Then you can invoke the BtnObj's **getBounds** method, store the returned rectangle, and invoke the returned rectangle's **toString()** method.

```
Set Rect = BtnObj.getBounds()
MsgBox Rect.toString()
```

Note: A recommended alternative to using the **Object** property is to extend QuickTest support for the required Java object using QuickTest Java Add-in Extensibility. For more information, refer to the *QuickTest Professional Java Add-in Extensibility Developer's Guide*.

Setting and Retrieving Java Object Properties

You can set or retrieve the value of any Java object using the **Object** property. If you are not sure which properties your object has, you can use the Object Spy to view the run-time properties of any object in your applet or application (except for SWT-based JavaMenu objects).

Note: When working with the Microsoft Internet Explorer Virtual Machine, you can set and retrieve only public properties using the **Object** property.

Setting the value of a property for a Java object has the following syntax:

```
JavaTestObject.Object.Property=Value
```

For example, suppose the label property is supported for your JavaButton. To set a new value for this property, insert the following statement in your script:

```
Browser("Browser").Page("PushButtonDemo").JavaApplet("pushbutton.html").
JavaButton("Enter").Object.label = "Click me"
```

Retrieving the value of a property for a Java object has the following syntax:

```
val=JavaTestObject.Object.Property
```

or

```
Set val=JavaTestObject.Object.Property
```

Alternatively, suppose the height property is supported for your JButton. To retrieve a value for this property, insert the following statement in your script:

```
height=  
Browser("Browser").Page("PushButtonDemo").JavaApplet("pushbutton.html").  
JButton("Enter").Object.height
```

The **Object** property is also useful for checking the value of properties that are not available using a standard Java checkpoint.

The following example uses the **Object** property to access a JButton object, retrieve its name and size, and display this information in message boxes.

```
set btnObj = Browser("Java Examples").Page("Java Examples").  
Frame("MAIN").JavaApplet("PushButtonAWTApplet").  
JButton("Text button").Object  
msgbox btnObj.label  
msgbox btnObj.width  
msgbox btnObj.height
```

Creating Objects in Your Applet or Application (Advanced)

You can use the **CreateObject** method to create an instance of any Java object within your applet or application. The **CreateObject** method returns an object reference to the newly created Java object.

The **CreateObject** method has the following syntax:

```
JavaTestObject.CreateObject( ClassName, [consArg1 , ... , consArgX] )
```

The *ClassName* argument is the Java class name. *consArg1...consArgX* are the required arguments for that object constructor.

You can activate the methods of an object you create in the same way as you would activate the methods of any returned object from a prior call. Because the **CreateObject** method returns an object reference, there is no need to use the **Object** property when activating methods of the created object.

For example, you can use the **CreateObject** method to create a rectangle object. The return value is an object reference.

```
Set Rect =
Browser("Periodic").Page("Periodic").JavaApplet("Periodic").JavaObject
("Panel").CreateObject ("java.awt.Rectangle", 10, 20)
```

Note: The **CreateObject** method can be performed on any Java test object. The Java test object serves as an anchor object. The class loader of the anchor object is used to load the class of the newly created Java object.

It is recommended to use the **CreateObject** method on a Java test object from the same toolkit as the object you want to create. For example, to create a Swing/JFC object, use the **CreateObject** method on an existing Swing/JFC Java test object.

Working with Static Members

You can invoke any static method, or you can set or retrieve the value of any static property of a Java class using the **GetStatics** method.

The **GetStatics** method has the following syntax:

```
JavaTestObject.GetStatics(ClassName)
```

GetStatics returns a reference to an object that can access static members of the specified class. The **GetStatics** method must be invoked on a Java test object that serves as an anchor object. Note that it is the class loader of the anchor object that is used to load the class specified as a parameter of the **GetStatics** method.

For example, to invoke the `gc` method of `class.java.lang.System`, which runs the garbage collector on the application, you can insert a statement similar to the following:

```
Browser("Browser").Page("Page").JavaApplet("mybuttonapplet.htm").  
JavaObject("MyButton").GetStatics("java.lang.System").gc
```

To retrieve the value of the `out` property of the `java.lang.System` class, you can insert a statement similar to the following:

```
Set OutStream=  
Browser("Browser").Page("Page").JavaApplet("mybuttonapplet.htm").  
JavaObject("MyButton").GetStatics("java.lang.System").out
```

To print a message to the Java console, you can insert a statement similar to the following:

```
Set OutStream=  
Browser("Browser").Page("Page").JavaApplet("mybuttonapplet.htm").  
JavaObject("MyButton").GetStatics("java.lang.System").out  
OutStream.println "Hello, World!"
```

Firing Java Events

You can simulate an event on a Java object during a run session with the `FireEvent` and `FireEventEx` methods. The `FireEvent` method simulates an event on a Java object using one of several pre-defined event constants (refer to the **Java** section of the *QuickTest Professional Object Model Reference* for a complete list). If the list of pre-defined constants does not cover the event you want to fire, you can use the `FireEventEx` method to fire any Java event.

The `FireEvent` method has the following syntax:

```
JavaObjectName.FireEvent ( EventType [, EventParam(s)] )
```

The `FireEventEx` method has the following syntax:

```
JavaObjectName.FireEventEx ( JavaClassName, EventID [, EventParam(s)] )
```

For example, you can use the **FireEvent** method to fire a `MouseClicked` event on the `JavaObject` called `MyButton_0`.

```
Browser("Browser").Page("Page").Applet("mybuttonapplet.htm").JavaObject
("MyButton_0").FireEvent micMouseClicked, 0, "BUTTON1_MASK", 4, 4, 1, "OFF":
```

Alternatively, you can use the **FireEventEx** method to fire the same event as follows:

```
Browser("Browser").Page("Page").Applet("mybuttonapplet.htm").JavaObject
("MyButton_0").FireEventEx "java.awt.event.MouseEvent",
"MOUSE_CLICKED", 0, "BUTTON1_MASK", 4,4, 1, "False"
```

Note that you can pass any Java constant that is used as one of the event's parameters using its string, rather than its value. In the example above, the `"java.awt.event.MouseEvent"` Java constant `MOUSE_CLICKED` is supplied as a string argument instead of its value (500 in this example).

5

Troubleshooting Testing Java Applets and Applications

This chapter is intended to help pinpoint and resolve some common problems that may occur when testing Java applets and applications.

This chapter describes:	On page:
Identifying and Solving Common Problems and Solutions	72
Checking Java Environment Variables Settings	74
Locating the Java Console	75
Running an Application or Applet with the Same Settings	78
Running the Java Add-in on Java 2, Java 5, and Java 6 Environments	78
Disabling Dynamic Transformation Support (Advanced)	80

Identifying and Solving Common Problems and Solutions

The QuickTest Professional Java Add-in provides a number of indicators that help you identify whether your add-in is properly installed and functioning. The following table describes the indicators you may see when your add-in is not functioning properly and suggests possible solutions:

Indicator	Solution
<p>You cannot record or run tests on Java applets or applications, or the Object Spy identifies Java objects as Standard Windows objects.</p>	<p>Make sure that the Java Add-in is loaded with QuickTest. To check this, choose Help > About QuickTest Professional 9.1 and verify that the Java Add-in check box is selected.</p> <p>You load the Java Add-in using the Add-in Manager. For more information, see “Loading QuickTest with Java Add-in Support” on page 16.</p>
<p>The Java console does not display a line containing the text "Loading Mercury Interactive Java Support".</p>	<p>Check that the settings in your environment correspond to the environment settings defined in this chapter, or check for a batch file that may override the settings.</p> <p>For more information, see:</p> <ul style="list-style-type: none"> • “Checking Java Environment Variables Settings” on page 74 • “Locating the Java Console” on page 75
<p>A different applet or application works with the QuickTest Professional Java Add-in, but the application you want to test does not work.</p>	<p>First check whether you can record and run tests if you invoke the other Java applet or application using exactly the same settings.</p> <p>Check that the settings in your environment correspond to the environment settings defined in this chapter, or check for a batch file that may override the settings.</p> <p>For more information, see:</p> <ul style="list-style-type: none"> • “Running an Application or Applet with the Same Settings” on page 78 • “Checking Java Environment Variables Settings” on page 74

Indicator	Solution
The add-in does not function properly with applications that run with the -Xincgc option.	<p>Either remove the -Xincgc option, or run without dynamic transformation support.</p> <p>For more information, see:</p> <ul style="list-style-type: none"> • “Running the Java Add-in on Java 2, Java 5, and Java 6 Environments” on page 78 • “Disabling Dynamic Transformation Support (Advanced)” on page 80
Your Java console contains the line: Could not find -Xrun library: jvmhook.dll.	<p>Check that the jvmhook.dll is located within your java.library.path For more information, see “Running the Java Add-in on Java 2, Java 5, and Java 6 Environments” on page 78.</p>
Your SWT- or Eclipse-based application crashes on startup.	<p>Make sure that the ActiveX Add-in is not loaded with QuickTest. To check this, choose Help > About QuickTest Professional 9.1 and verify that the ActiveX Add-in check box is cleared.</p> <p>Note: If you want to test an ActiveX object that is embedded in your SWT- or Eclipse-based application, you must first disable Java support. To do this, rename the _JAVA_OPTIONS or IBM_JAVA_OPTIONS environment variables. This instructs QuickTest to treat the ActiveX object as if it is embedded in a Standard Windows application. In addition, the container application objects will be identified as Standard Windows objects.</p>

If, after reviewing the above indicators and solutions, you are still unable to record and run tests on your Java applet or application, contact Mercury Customer Support.

Checking Java Environment Variables Settings

This section describes the environment variables that need to be set when you load your Java application with QuickTest Java Add-in support. For all of the environments, you need to set one or more environment variables to the short path name of the Java Add-in support classes folder.

Sun Java 2 or Later (Version 1.2 or Later) or IBM Java 2 or Later (Version 1.2 or Later)

Set the `_JAVA_OPTIONS` environment variable (Sun) or the `IBM_JAVA_OPTIONS` environment variable (IBM) as follows:

```
-Xrunjvmhook -Xbootclasspath/a:<common_files>\MERCUR~1\FUNCTI~1\Java\classes;<common_files>\MERCUR~1\FUNCTI~1\Java\classes\jasmine.jar
```

The above settings should appear on one line (no newline separators).

Note: `<common_files>` denotes the short path of the **Common Files** folder located under **Program Files**. For example, if the **Common Files** folder is located in `C:\Program Files\Common Files`, then the value for `-Xbootclasspath` is as follows:

```
-Xbootclasspath/a:C:\PROGRA~1\COMMON~1\MERCUR~1\FUNCTI~1\Java\classes;C:\PROGRA~1\COMMON~1\MERCUR~1\FUNCTI~1\Java\classes\jasmine.jar
```

Tip: If needed, you can temporarily remove Java support by renaming the `_JAVA_OPTIONS` or `IBM_JAVA_OPTIONS` environment variable. For example, you must remove Java support if you want to test ActiveX controls that are embedded in SWT- or Eclipse-based applications.

If you are working with Sun Java 6 (version 1.6), you must set an additional environment variable, `JAVA_TOOL_OPTIONS`, with the following value:
-agentlib:jvmsHook

Microsoft Java Virtual Machine (JVM) - Internet Explorer/JView

Make sure that the `MSJAVA_ENABLE_MONITORS` variable is set to **1**. This is relevant when the user who installed QuickTest with Java Add-in support is not the user running the application.

Netscape Browser 8.x

Netscape 8.x uses the Java 2, Java 5, or Java 6 Virtual Machine as an external plug-in. For more information, see “Sun Java 2 or Later (Version 1.2 or Later) or IBM Java 2 or Later (Version 1.2 or Later)” on page 74.

Mozilla Firefox

Mozilla Firefox uses the Java 2, Java 5, or Java 6 Virtual Machine as an external plug-in. For more information, see “Sun Java 2 or Later (Version 1.2 or Later) or IBM Java 2 or Later (Version 1.2 or Later)” on page 74.

Locating the Java Console

The Java console is the window in which your Java application or applet displays messages. The location of the Java console changes according to your application setup. Your Java application can be:

- a standalone application
- run in an applet viewer
- an applet run in Microsoft Internet Explorer, Netscape Browser, or Mozilla Firefox

If your Java application is a standalone application:

- Open the batch file or shortcut that invokes the application and look for the command that launches Java (**java.exe**, **javaw.exe**, **jre.exe**, or **jrew.exe**).
- If the application was run with **java.exe** or **jre.exe**, it will load with a console (Command prompt window).
- If the application was run with **javaw.exe** or **jrew.exe**, it will not load with a console (the console is unavailable). You can check for Java Add-in support by invoking the application with **java.exe** or **jre.exe**. Do this by altering your batch file or the shortcut invoking your application.

Note: **java.exe** and **javaw.exe** are nearly identical, as are **jre.exe** and **jrew.exe**. The only difference between them is whether they launch a console window.

If your Java application runs in an applet viewer:

- Look in the DOS command prompt window that invoked the applet viewer.
- If there is no DOS command prompt window, your applet viewer may be run by a batch file similar to a standalone application. For more information, see the information about **javaw** and **jrew** in the standalone application section above.

If your Java applet runs in Microsoft Internet Explorer, Netscape Browser, or Mozilla Firefox:

- ▶ If your applet runs in Microsoft Internet Explorer or Netscape Browser using the Sun Java plug-in:

Right-click the Java (plug-in) icon in your taskbar tray and click the option that opens the console (for example, Open Console or Show Console, depending on the installed version).

If you do not see the Java (plug-in) icon in your taskbar tray, choose **Start > Settings > Control Panel** and double-click the Java icon or option (choose the Java version used by your application). Then, in the displayed dialog box, select the option to show the Java console (for example, Show console). Note that the actual name of the option, and its location in the dialog box, depend on the Java version used by your application.) Confirm the change (for example, by clicking Apply). Restart the browser.

Note: To find out whether your Microsoft Internet Explorer works with the Sun Java plug-in, select **Tools > Internet Options > Advanced**. Under **Java (Sun)** verify that **Use Java** is selected. Java plug-in version 1.3 or later automatically configures Internet Explorer to work with the Sun Java plug-in.

- ▶ If your applet runs using the Microsoft Internet Explorer internal Virtual Machine:

In Microsoft Internet Explorer, select **Tools > Internet Options**. In the Advanced tab, look for **Microsoft VM**. Select **Java console enabled (requires restart)** and click **OK**. Restart the browser and invoke your application. Select **View > Java Console**.

- ▶ If your applet runs in Mozilla Firefox:

In Mozilla Firefox, select **Tools > Java Console**. If you do not see the **Java Console** option in the **Tools** menu, install the **Open Java Console** extension from <https://addons.mozilla.org/firefox/141/>. This extension provides the menu option on the **Tools** menu for opening the Java Console from Mozilla Firefox. It also provides a toolbar button in the JavaScript Console for opening Java Console.

Running an Application or Applet with the Same Settings

In some cases, running another Java application or applet with the exact same settings helps determine whether you are encountering a general problem with the Java Add-in or an application-specific problem.

To run an application or applet with the same settings:

- ▶ Determine whether the application is a standalone application or an applet.
- ▶ If the application is an applet, check the browser type.
- ▶ If the applet is executed from a shortcut, execute the applet with the same command.
- ▶ If the applet is executed from a batch file, copy the batch file and change only the class file that invokes the applet.

Note: If the classpath must also be changed, add only the new items needed. Do not remove any of the items from the original application or applet classpath.

Running the Java Add-in on Java 2, Java 5, and Java 6 Environments

When you run the Java Add-in on Java 2 or Java 5 environments, the add-in uses a mechanism that supports multiple Java environments (such as, Microsoft Internet Explorer internal virtual machine, SUN JRE, and IBM JRE) and multiple Java versions (such as, JDK 1.3.1, 1.4.2, 1.5.0, and 1.6.0) without requiring any configuration changes. This mechanism, known as the **dynamic transformation support** mechanism, uses the profiler interface of the Java Virtual Machine (JVM) to adjust the Java Add-in support classes according to the Java environment and version used.

The dynamic transformation support mechanism is invoked by the **-Xrunjvmsupport** option, which is supplied to the JVM. If the **-Xrunjvmsupport** option is specified, the JVM hook profiler (part of the Java Add-in support) is loaded with every Java 2 application or applet that loads. The JVM hook profiler dynamically transforms the necessary classes to enable context-sensitive Java support.

When you run the Java Add-in on Java 6 environments, the dynamic transformation support mechanism is invoked by the **-agentlib:jvmsupport**, which is defined in the `JAVA_TOOL_OPTIONS` environment variable.

Note: When working with Java 6, there is no conflict between **-agentlib:jvmsupport** (defined in the `JAVA_TOOL_OPTIONS` environment variable) and **-Xrunjvmsupport** (defined in the `_JAVA_OPTIONS` environment variable) because Java 6 ignores **-Xrunjvmsupport**.

The Java agent searches for the **jvmsupport.dll** according to the **java.library.path** system property. You can identify any override of this system property using the Java command line: **-djava.library.path = <path>**. However, although you can override the **java.library.path** system property, it is recommended to extend the **java.library.path** and not to overwrite it.

By default, the value of the **java.library.path** system property is the system path. If your application is loaded with a different library path, you must either add the **jvmsupport.dll** to a location within the **java.library.path**, or change the **java.library.path** to contain **<Windows installation folder>/system32**.

The **<JRE root folder>/bin** folder is always located in the **java.library.path**. If needed, you can manually copy the **jvmsupport.dll** to this folder. However, if you need to modify more than one computer, it is recommended to modify the batch file that alters the **java.library.path**.

Disabling Dynamic Transformation Support (Advanced)

If the dynamic transformation support mechanism does not work properly, you can disable it and manually configure the Java environment to use the Java Add-in without dynamic transformation support.

In addition, the dynamic transformation support mechanism is not supported when using the incremental garbage collector (**-Xincgc** option). Therefore, if you absolutely must use the **-Xincgc** option, you need to disable dynamic transformation support.

You disable dynamic transformation support by performing the following steps:

- ▶ Saving the dynamically transformed classes, as described on page 80
- ▶ Disabling dynamic transformation support by disabling the JVM hook profiler, as described on page 81

After you perform these steps, the saved transformed classes will be used instead of dynamic transformation.

To save the dynamically transformed classes:

- 1 Specify the folder in which to save the dynamically transformed classes that will be generated during the preliminary launching of your Java applet or application.

To do this, open the registry editor (choose **Start > Run**, type **regedit** in the **Open** box and click **OK**) and navigate to the **JavaAgent** main key, located in: **HKEY_LOCAL_MACHINE\SOFTWARE\Mercury Interactive\JavaAgent**. Define a new string value named **ClassesDumpFolder**, and set its value data to an existing folder (preferably empty) on your computer, for example, **C:\JavaSupportClasses**.

Note: If the **ClassesDumpFolder** string value already exists, you can modify its value data to an existing folder on your computer.

- 2 If you are using the **-Xincgc** option, temporarily remove it from the command line to enable the JVM hook profiler to transform and save the necessary classes.
- 3 Launch your applet or application and perform some basic operations on it. This ensures that all of the necessary classes are transformed and saved. Close your applet or application. All of the dynamically transformed classes are now saved in the folder you specified in the previous step (for example, C:\JavaSupportClasses).
- 4 If you temporarily removed the **-Xincgc** option from the command line in step 2, you can restore it now.

Now that you have saved the transformed classes, you are ready to disable dynamic transformation support.

To disable dynamic transformation support:

- 1 Remove the **-Xrunjvhook** option from the **_JAVA_OPTIONS** (or **IBM_JAVA_OPTIONS** for IBM VM-based applications) environment variable.
- 2 Add the following option instead:

-Xbootclasspath/p:<ClassesDumpfolder>\Final where **<ClassesDumpfolder>** is the value of the folder in which the dynamically transformed classes were saved (step 1 on page 80). For example, after your modification the **_JAVA_OPTIONS** environment variable might look like this:

```
-Xbootclasspath/p:C:\JavaSupportClasses\Final -
Xbootclasspath/a:C:\PROGRA~1\COMMON~1\MERCUR~1\FUNCTI~1\Java\cl
asses;C:\PROGRA~1\COMMON~1\MERCUR~1\FUNCTI~1\Java\classes\
jasmine.jar
```

Index

Numerics

1_APP_ENV variable 48

1_DIR_ENV variable 48

A

action parameters 59

Active Screen

 defining capture settings 43

Add-in Manager dialog box 16

add-ins

 loading with add-in support 16

 loading without add-in support 18

 repairing 19

 uninstalling external 21

administrator privileges for installing 5

application details environment variables 48

B

BROWSER_ENV variable 48

C

cell editors, JTable

 working with 51

checking

 Java objects 61

checkpoints

 accessibility options 43

CMDLINE_ENV variable 48

component parameters 59

conventions. *See* typographical conventions

Custom Active Screen Capture Settings

 dialog box 43

Customer Registration screen 10

customer support site 10

D

disabling dynamic transformation support
 80

disk space requirements 4

documentation updates xi

dynamic transformation support mechanism
 78

dynamic transformation support, disabling
 80

E

environment variables

 application details 48

 checking settings 74

EXEPATH_ENV variable 48

Expert View 49, 63

external add-ins

 uninstalling 21

F

Firefox, Mozilla

 environment variables 75

 Java console 77

full object hierarchy, viewing 59

G

GetStatics method 67

GetTOProperty method 63

H

hard disk space requirements 4

Index

I

IBM Java
 environment variables 74
installation requirements 4

J

Java
 classes, working with static members
 67
 console 75
 environment variables 74
 objects, methods and events 63
Java Add-in
 Advanced Java Options dialog box 34
 checking environment settings 74
 Java tab, Options dialog box 32
 troubleshooting 71
Java console, locating 75
JTable cell editors
 custom 51
 finding the toolkit class using a
 QuickTest script 54
 finding the toolkit class using the
 Object Spy 52
 recording on 51
 standard 51
JView
 environment variables 75
JVM
 environment variables 75

K

Keyword View 63
Knowledge Base x

L

License Agreement screen 8
loading
 with add-in support 16
 without add-in support 18
locating the Java console 75

M

Mercury Best Practices x
Mercury Customer Support Web site x
Mercury Home Page x
methods 63
 test object 63
Microsoft Internet Explorer
 environment variables 75
 Java console 77
Microsoft Internet Explorer Virtual Machine
 Java console 77
Mozilla Firefox
 environment variables 75
 Java console 77

N

Netscape Browser
 environment variables 75
 Java console 77

O

object hierarchy
 full 59
 recorded 49
Object Spy 67
online documentation viii
online resources x
Options dialog box
 Java tab 32
outputting
 Java values 61

P

parameters 59
prerequisites, installation 4
properties
 unique to Java test objects 63

Q

QuickTest
 loading with add-in support 16
 loading without add-in support 18

R

- Readme viii
- Record and Run Settings dialog box
 - optimizing other tab settings 48
 - Web tab 48
 - Windows Applications tab 48
- recorded object hierarchy 49
- recording and running tests on Java objects,
 - about 28
- repairing add-ins 19
- requirements, installation 4
- running tests on Java objects 56

S

- Select Updates screen 11
- SetCellData method 51
- setup
 - Customer Registration screen 10
 - License Agreement screen 8
 - running 5
 - Setup Complete screen 14
- statements 63
- Step Generator 63
- Sun Java
 - environment variables 74
- SWT-based Java objects 55
- system requirements 4

T

- table_external_editors_list variable 51
- test object model 29
- test objects 63
- test parameters 59
- toolkit class, of a cell editor 51
- Tree View 49
- troubleshooting, testing Java objects 71
- typographical conventions xii

U

- uninstalling external add-ins 21
- updates, documentation xi
- URL_ENV variable 48

V

- VBScript 63

W

- Web tab, Record and Run Settings dialog box
 - 48
- Windows Applications tab, Record and Run Settings dialog box 48
- WORKDIR_ENV variable 48

