



WinRunner®
*Java Add-in Installation
and User's Guide*
Version 6.0

Online Guide




Books
Online


Find

Find
Again


Help



Table of Contents

Welcome to the Java Add-in.....	5
Using this Guide	5
Typographical Conventions	7

PART I: INSTALLING THE JAVA ADD-IN

Chapter 1: Before You Install	9
Checking Your Java Add-In Package	9
System Requirements	10
Chapter 2: Setting Up the Java Add-in	11
Running the Setup Program	12
Configuring the Web Server	23
Modifying Your Selected JDK Version.....	24
Chapter 3: Verifying Your Java Add-in Installation.....	26
About the Java Add-in Verifier.....	26
Using the Java Add-in Verifier	27
Chapter 4: Disabling or Uninstalling the Java Add-in.....	38
Disabling the Java Add-in Temporarily	38
Uninstalling the Java Add-in	40



PART II: WORKING WITH THE JAVA ADD-IN

Chapter 5: Testing Standard Java Objects	43
About Testing Standard Java Objects	43
Activating the Java Add-in	44
Recording Context Sensitive Tests	45
Enhancing Your Script with TSL.....	46
Invoking a Java Method.....	47
Setting the Value of a Java Bean Property.....	49
Configuring How WinRunner Learns Object Descriptions and Runs Tests	51
Activating a Java Edit Object.....	55
Chapter 6: Configuring Custom Java Objects.....	56
About Configuring Custom Java Objects.....	56
Adding Custom Java Objects to the GUI Map.....	57
Configuring Custom Java Objects with the Custom Object Wizard..	59
Chapter 7: Using Java Direct Call (JDC) Mechanism.....	66
About Java Direct Call Mechanism.....	66
Using the JDC Mechanism	67
Preparing a TSL Script for use with JDC.....	69
Using JDC: An Example	69

Books
Online

Find

Find
Again

Help

Top of
Chapter

Back

Chapter 8: Troubleshooting Java Add-in Recording Problems	72
Handling General Problems Testing Applets.....	72
Handling Specific Java Add-in Problems.....	73
 Index	 75



Welcome to the Java Add-in

Welcome to WinRunner with add-in support for Java. The Java Add-in enables you to test cross-platform Java applets and applications.

Using this Guide

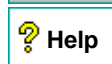
This guide explains everything you need to know to install the Java Add-in and to use WinRunner to successfully test Java applications and applets. It should be used in conjunction with the *WinRunner User's Guide* and the *TSL Online Reference*.

This guide contains 2 parts:

Part I: Installing the Java Add-in

Details the process of installing and verifying the Java Add-in, including:

- [Before You Install](#)
- [Setting Up the Java Add-in](#)
- [Verifying Your Java Add-in Installation](#)
- [Disabling or Uninstalling the Java Add-in](#)



Part II: Working with the Java Add-in

Explains how to use the Java Add-in to test Java applications and applets including:

- **Testing Standard Java Objects**
- **Configuring Custom Java Objects**
- **Using Java Direct Call (JDC) Mechanism**
- **Troubleshooting Java Add-in Recording Problems**



Typographical Conventions

This book uses the following typographical conventions:

1, 2, 3	Bold numbers indicate steps in a procedure.
•	Bullets indicate options and features.
>	The greater than sign separates menu levels (for example, File > Open).
Bold	Bold text indicates function names.
<i>Italics</i>	<i>Italic</i> text indicates variable names.
Helvetica	The Helvetica font is used for examples and statements that are to be typed in literally.
[]	Square brackets enclose optional parameters.
{ }	Curly brackets indicate that one of the enclosed values must be assigned to the current parameter.
...	In a line of syntax, three dots indicate that more items of the same format may be included. In a program example, three dots are used to indicate lines of a program that were intentionally omitted.
	A vertical bar indicates that either of the two options separated by the bar should be selected.



Installing the Java Add-in



Before You Install

Before you install the Java Add-in, please review the following installation procedures.

Checking Your Java Add-In Package

In addition to this guide, please make sure your Java Add-in package contains the items below. If any item is missing or damaged, contact Mercury Interactive or your local distributor.

- Java Add-in CD-ROM
- Registration Card



System Requirements

In order to successfully run WinRunner with the Java Add-In, you need the following minimum system requirements:

Platform	An IBM-PC or compatible with a Pentium® /100MHz or higher microprocessor.
Memory	32 MB of RAM memory.
Disk Space	24 MB of free disk space for a minimum installation, or 66 MB for a typical or complete installation. (An additional 5 MB of free disk space is required in your browser installation folder.)
Operating System	Windows 95, Windows 98, or Windows NT 4.0.
Prerequisites	WinRunner 6.0 standalone installation. Netscape 4.05 Professional Edition or higher, or Internet Explorer 4.01 or higher, or JDK or JRE 1.15 or higher

Note: If the CLASSPATH variable in the *autoexec.bat* file contains more than 400 characters, modify this line to less than 400 characters prior to installation.

If you want to install from the network, map the installation drive before installing and run the installation from the mapped drive.



Setting Up the Java Add-in

The Java Add-in installation process involves two main stages:

- Running the Java Add-in setup program
- Configuring the Web server (for installations with Mercury classes on the Web Server)

Note: Before you install the Java Add-in, you must have a WinRunner 6.0 standalone installation on your computer. If you intend to test applets in a browser, you must have the browser installed. If you intend to test a Java Plug-in or Jinitiator, these must be installed. If you intend to test a Java Application or use the AppletViewer, you must have the Java Development Kit (JDK) or the Java runtime environment (JRE) installed.

If you re-install or install a new version of a Web browser or other Java Environment, you must re-install the Java Add-in after the browser, JDK or plug-in has been installed.

Before you re-install, however, be sure to close any browsers, Java applications and WinRunner.



Books
Online



Find

Find
Again



Help



Top of
Chapter



Back

Running the Setup Program

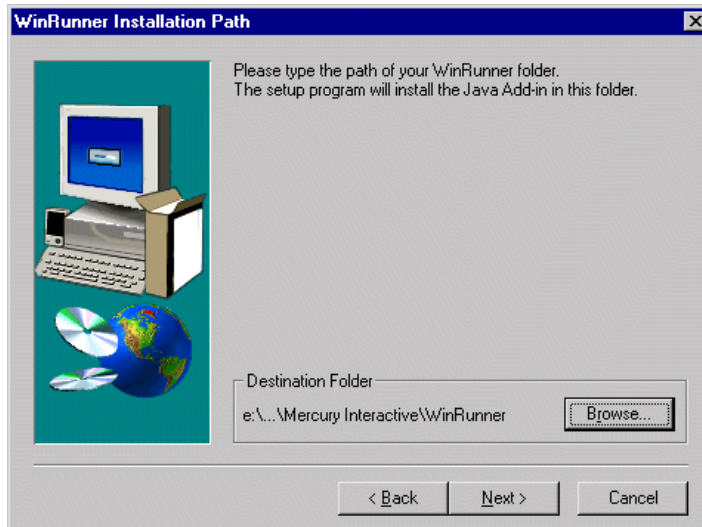
The setup program installs the Java Add-in support in your WinRunner installation folder.

To run the Java Add-in setup program:

- 1 Insert the CD-ROM into the drive from which you want to install. If you are installing from a network drive, connect to it.
- 2 Select **Run** on the **Start** menu.
- 3 Type the location from which you are installing, and setup.exe. For example, type d:\setup.exe.
- 4 Click **OK**. The WinRunner Java Add-in setup program starts.
- 5 Read the Welcome dialog box. Click **Next**.



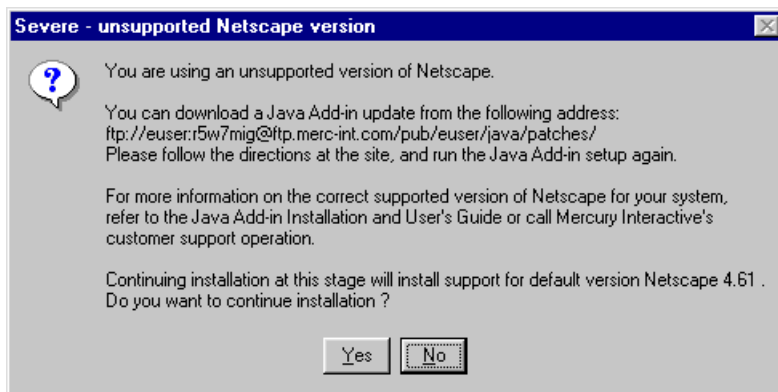
- Specify the folder in which to install the Java Add-in. The destination folder must be the WinRunner installation folder. If the installation folder that appears is not the WinRunner installation folder, click **Browse** to find the correct destination folder. Click **Next**.



- 7 Select the Java environment(s) that you intend to use. Click **Next**.



- 8 If you have a Java environment installed that is not supported by the Java Add-in installation, you will receive a warning message similar to the one below.



This message will tell you which environment and version you are using that is not supported and will provide you with the URL from which you can download a Java Add-in update.

- If you want to download the update now, and then restart the Java Add-in installation, click **No**.

Create an *updates* folder under your WinRunner installation folder. Download the Java Add-in update into the *updates* folder, and then run the Java Add-in installation from the beginning. When you run the Java Add-in installation, it will automatically install any updates located in the *updates* folder.

- If you want to continue the Java Add-in installation now and download the update at a later time, click **Yes**.

Note: If you choose to download the update at a later time, you must re-install the Java Add-in after you download the update.

9 Choose **Patch Browser** or **Install classes on the Web server**.

If your Java applets require that all Java classes come from the Web server, the Mercury classes must be installed in the same location as the Java classes.



Books
Online



Find

Find
Again



Help



Navigation arrows

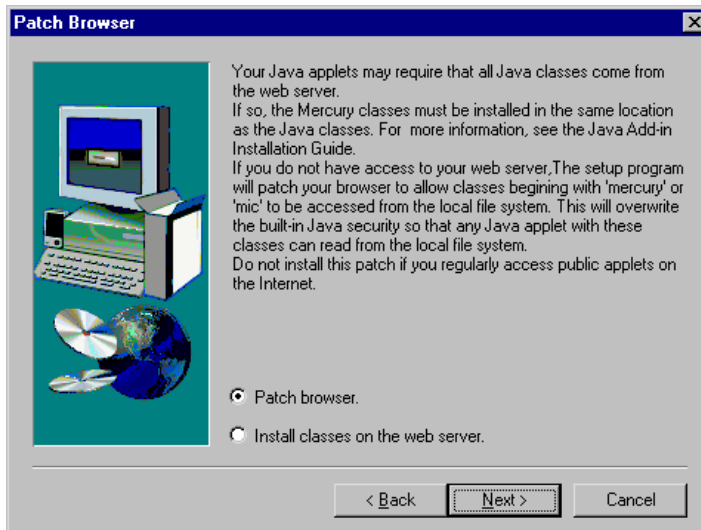


Top of
Chapter



Back

If you select **Install classes on the Web server** option, you must configure your Web server after the setup program is completed. For more information, see [Configuring the Web Server](#) on page 23.

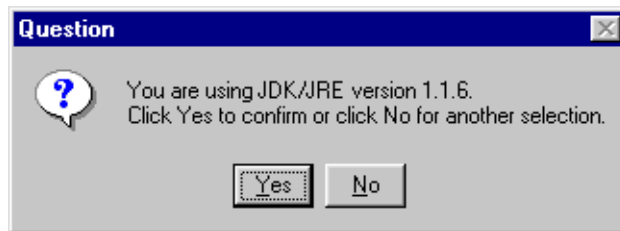


If you do not have access to your Web server, you can patch your browser so that classes beginning with “mercury” or “mic” can be accessed from the local file system. This overwrites the built-in Java security so that any Java applet with these classes can read from the local file system. To patch your browser, select **Patch Browser**.



Click **Next**. If you chose to work with AppletViewer or a Java application, the setup program checks whether you have the Java Development Kit (JDK) or Java Runtime Environment (JRE) installed on your computer.

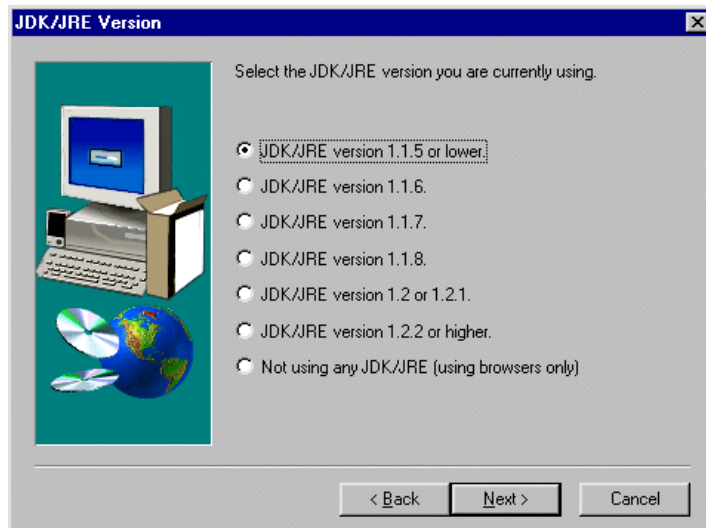
- 10 If you have JDK or JRE installed on your computer, and you selected **Java Application (no browser)** or **AppletViewer** as one of the Java environments you intend to use, then the setup program prompts you to confirm the version you are using.



- If the version displayed in the message is correct, click **Yes**.
- If the version is incorrect, click **No**.



- 11 If you select **No**, the JDK/JRE dialog box opens. In the JDK/JRE dialog box, select the JDK/JRE version you wish to use. Note that selecting a JDK/JRE version which is different from the version you intend to use may cause unpredictable results when you try to record or replay tests with WinRunner.



Click **Next**.

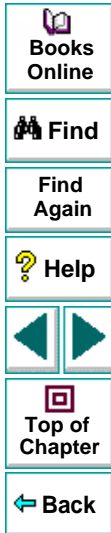
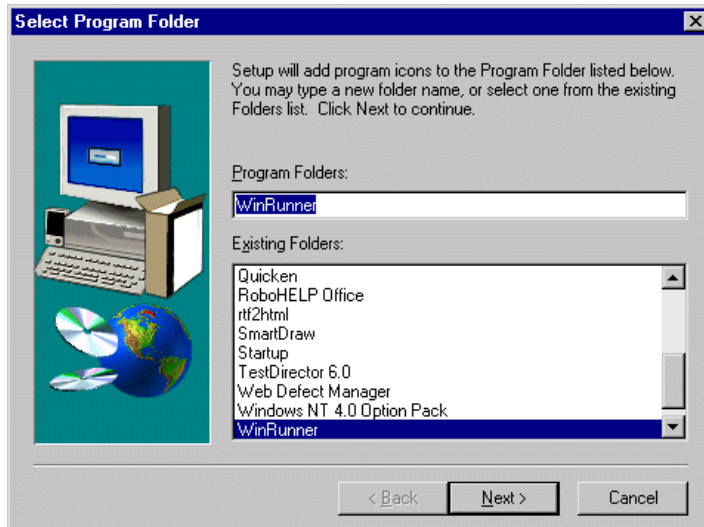


Note: If you need to change the JDK/JRE version you use at another time, you can use the Java Add-in Configuration Tool to quickly modify the Mercury Environment to match the new JDK/JRE version. For more information refer to [Modifying Your Selected JDK Version](#) on page 24.

- 12 The installation process begins. To pause or quit the installation process, click **Cancel**.



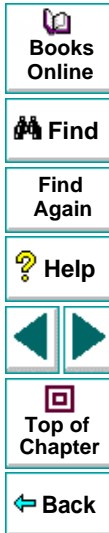
- 13 Select a **Program Folder**. The default is WinRunner. Click **Next**.



- 14 To complete the installation process, click **Finish**.



You must reboot your computer before using the Java Add-in.



Configuring the Web Server

If you chose to install Mercury classes on the Web server during the setup program, you must configure the Web server so that the Mercury classes are installed in the same location as the Java classes.

To configure your Web server to work with your toolkit:

- 1 On the Web server machine, find the location of your Java toolkit classes. The location of these classes depends on the Web server you are using. Please consult the Web server administrator to find the location.
- 2 Copy the content of the folder *WinRunner installation folder\classes\sr* from the client machine to the location of your Java toolkit classes on the Web server.
- 3 Confirm that the folders: *jclass*, *mercury*, *oracle* and *symantec* are now located on your Web server parallel to your Java toolkit classes.



Modifying Your Selected JDK Version

The Java Add-in Configuration Tool makes it possible to test in a multi-JDK environment by quickly modifying the Mercury environment to match the JDK you want to use, without requiring re-installation of the Java Add-in.

The Java Add-in Configuration Tool is installed during the Java Add-in installation, and is included in the Java Add-in program group.

To modify your selected JDK version:

- 1 Close any open Java applets or applications.
- 2 Choose **Programs > WinRunner > Java Add-in > Java Add-in Configuration Tool** from the **Start** menu. The Select JDK Version dialog box opens.
- 3 Select the JDK version you want to use. Click **Next**.
- 4 Browse to, or type, the path of your WinRunner folder or accept the path listed in the window. Click **Next**.
- 5 If you choose JDK version 1.2.x in step 3, the JDK 1.2 folder dialog box opens. Browse to, or type, the name of your JDK/JRE 1.2.x folder or accept the path listed in the window. Click **Next**.



- The Mercury environment is modified according to your selection and a confirmation message appears. Click **OK**.

Note: The Java Add-in Configuration Tool enables you to modify the Mercury environment only for SUN JDK 1.1.5 - 1.1.8 and 1.2 - 1.2.2 environments. If you need to modify the Mercury Environment for any other Java environment such as browsers or plug-ins, run the Java Add-in installation and select your required environment(s) from the Select Components dialog box. For more information see [page 14](#).



Verifying Your Java Add-in Installation

This chapter explains how to run and analyze the Java Add-in Verifier in order to isolate installation problems.

About the Java Add-in Verifier

If you experienced problems installing the Java Add-in, or you successfully installed it but experience difficulties when you try to launch WinRunner, record a Java applet or application with WinRunner, or run a test script, you can verify your Java Add-in installation by running the Java Add-in Verifier.

The Java Add-in Verifier checks the environment settings and installations that will affect the performance and operation of the Java Add-in. It collects information about the WinRunner version, the Java Add-in, and Java development kits (JDKs) installed on your computer.

After the verification process is complete, you can view a report. The Java Add-in Verifier saves this report as a log file that you can view in any text editing tool.



Using the Java Add-in Verifier

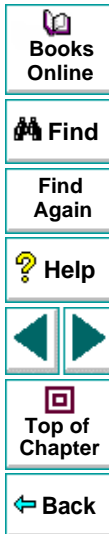
To verify your Java Add-in installation:



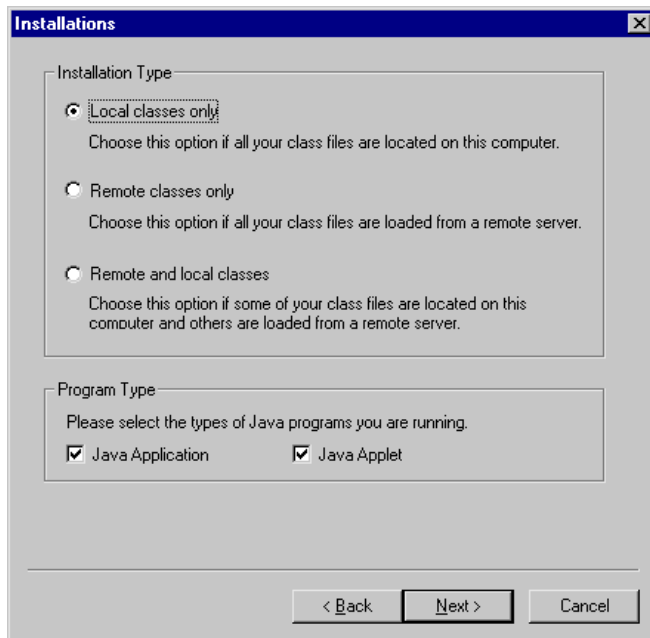
- 1 Launch the Java Add-in Verifier. Choose **Programs > WinRunner > Java Verifier** in the **Start** menu. The Java Add-in Verifier dialog box opens.



- 2 Read the opening screen. Click **Next**.



- 3 In the Installations dialog box, select an installation type and specify the types of Java applications you run on your system.



Select one of the following installation types:

- **Local classes only:** select this option if all your classes in the classpath are located on your computer.
- **Remote classes only:** select this option if all your class files are located on a remote server.
- **Remote and local classes:** select this option if some of your class files are located on your computer, and some of your class files are located on a remote server.

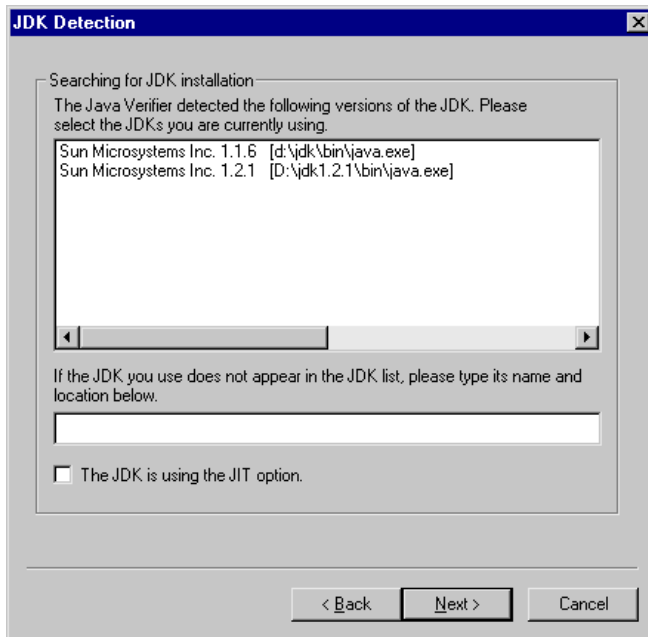
Also, select the types of Java programs you run on your system:

- **Java application**
- **Java applet**

Click **Next**. The Java Add-in Verifier scans your computer's hard disk, and creates a list of the JDKs installed on your computer.



- 4 In the JDK Detection dialog box, select the JDKs you use from the list.



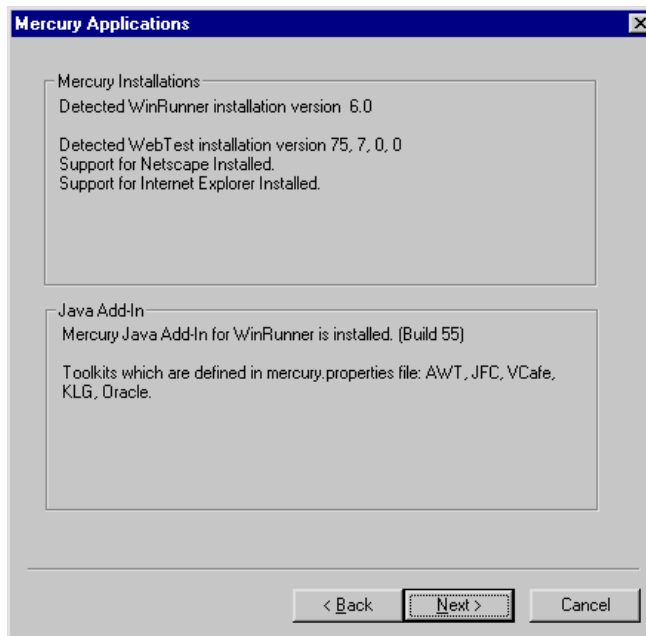
Note: If your JDK is not displayed in the list, first check your JDK installation and make sure that your Java bin folder is in the system path variable. If the Java Add-in Verifier is still unable to recognize the JDK, try to reinstall the JDK. If this does not work, enter the name and location of the JDK in the text box or contact Mercury Interactive Customer Support.

Select **The JDK is using the JIT option** check box if you are using your JDK's "just in time" compiler option. Note that by default JDK 1.1.6 and higher use JIT.

Click **Next**.



- 5 The Java Add-in Verifier lists all of the WinRunner and Java Add-in installation options the Java Add-in Verifier detected.
- The Mercury Installations list displays a list of Mercury Interactive products installed on your computer such as: WinRunner, LoadRunner, and the WebTest Add-in.
 - The Java Add-in list displays a list of toolkits which have been defined in the *mercury.properties* file.

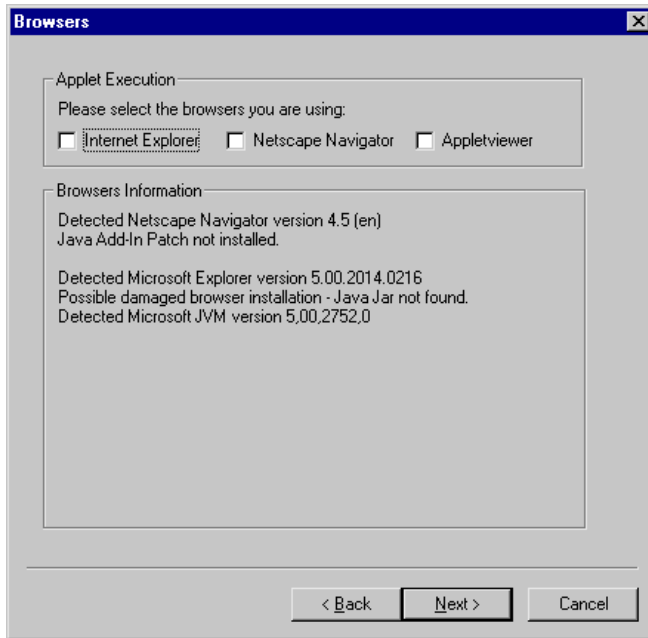


Note: If the Java Add-in Verifier failed to detect a Mercury Interactive application, first check the application separately. If it does not work properly, reinstall the application. If the Java Add-in Verifier still fails to detect the application, contact Mercury Interactive Customer support.

Read the list, and click **Next**.

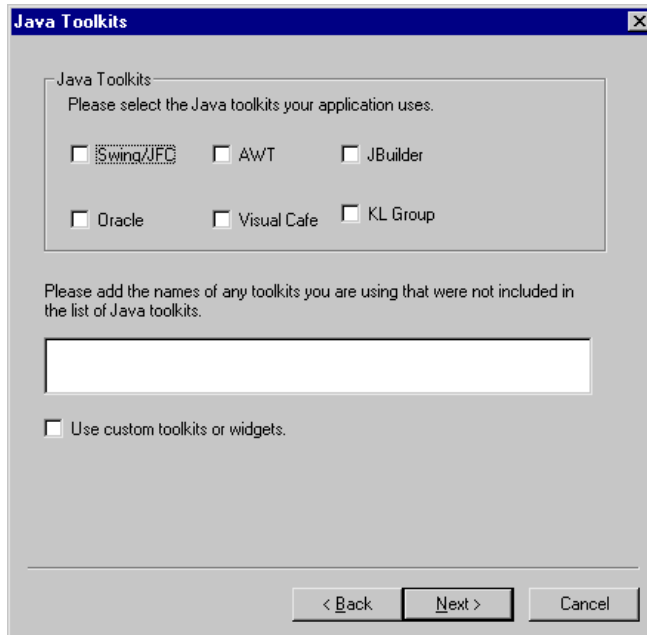


- In the Browsers dialog box, select the Web browser(s) you are using to execute Java applets. Click **Next**.



Note: If the Java Add-in Verifier failed to detect a browser installed on your computer, first check the browser separately. If it does not work properly, reinstall the browser. If the Java Add-in Verifier still fails to detect the browser, contact Mercury Interactive Customer Support.

- 7 In the Java Toolkits dialog box, select all Java toolkits used by your Java applet or application.



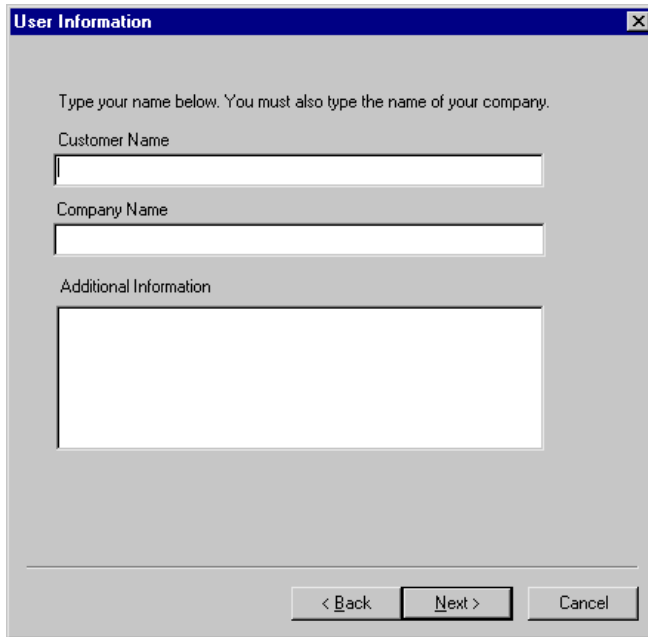
If your toolkit is not listed, you can add the name of the toolkit in the text box.

If you are using your own custom toolkits and widgets, select the **Use custom toolkits and widget** check box.

Click **Next**.



- 8 In the User Information dialog box, type your name, the name of your company, and any additional information you want to add to the log file. This information will be included in the Java Add-in Verifier log file. Click **Next**.



The image shows a dialog box titled "User Information" with a close button in the top right corner. The dialog box contains the following text and fields:

Type your name below. You must also type the name of your company.

Customer Name

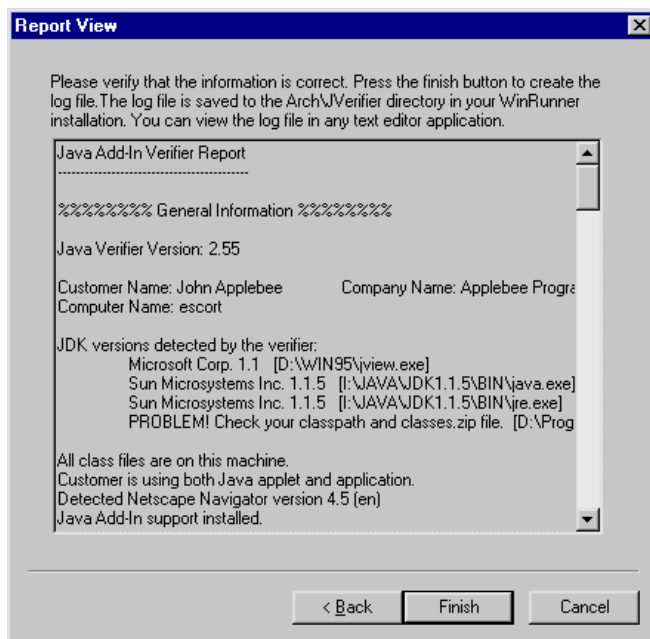
Company Name

Additional Information

At the bottom of the dialog box, there are three buttons: "< Back", "Next >", and "Cancel".



- 9 View the generated report in the Report View dialog box.



Click the **Finish** button to close the Java Add-in Verifier and generate the log file. The log file is saved in the *WinRunner installation folder\arch\JVVerifier* folder. You can view the file in any text editor application.

Disabling or Uninstalling the Java Add-in

If you decide to use WinRunner without the Java Add-in, you have the option of temporarily disabling the add-in, or uninstalling it.

Disabling the Java Add-in Temporarily

There are two ways to temporarily disable the Java Add-in:

- Disable the toolkit support in the `mercury.properties` file
- Remove the WinRunner classpaths

To temporarily disable the Java Add-in via the `mercury.properties` file:

- 1 Select **Programs > WinRunner > Java Add-in > `mercury.properties`** from the **Start** menu.
- 2 In the `mercury.properties` file, place a pound sign (#) before the line starting with `mic_toolkit=`. For example:

`# mic_toolkit=AWT, JFC, VCafe, KLG, Oracle`
- 3 Save and close the file.
- 4 If you use Jinitiator, open the `mercury.properties` file from the *Jinitiator Installation folder\classes* folder and repeat steps 2 and 3 for this file.



To temporarily disable the Java Add-in by removing WinRunner classpaths:

- 1 Remove *WinRunner installation folder\classes* and *WinRunner installation folder\classes\srcv* from the classpath.



Uninstalling the Java Add-in

To uninstall the Java Add-in:

- 1 Select **Programs > WinRunner > Java Add-in > Uninstall Java Add-in** from the **Start** menu.

Note: Alternatively, you may select Java Add-in from the Add/Remove Programs dialog box (**My computer > Control Panel > Add/Remove Programs**).

- 2 A message asks you to confirm that you want to remove the Java Add-in and all of its components. Click **Yes**.
- 3 When the uninstall process is complete, click **OK**.
- 4 If you selected the Java Plug-in environment during installation of the Java Add-in, go to *Plug-in installation folder\plug\lib\bak*. Move the *rt.jar* file to the *lib* folder and delete the *bak* folder.



- 5 Remove the path *WinRunner installation folder\classes* and the path *WinRunner installation folder\classes\sr* from the classpath.
 - If you are working in Windows NT 4.0, from the Start menu select **Settings > Control Panel > System** from the **Start** menu. In the System properties dialog box, click the **Environment** tab. In both the **System Variables** list and the **User Variables** list, delete *WinRunner installation folder\classes* from the classpath. For example, `c:\program files\WinRunner\classes`. Click **Close**.
 - If you are working in Windows 95 or Windows 98, open the file *[Windows Drive]:\autoexec.bat* in any editing program. Delete *WinRunner installation folder\classes* from the classpath. For example, `c:\program files\WinRunner\classes`. Save and close the file, and reboot your computer.

The Java Add-in is now completely uninstalled.



Working with the Java Add-in



Testing Standard Java Objects

This chapter explains how to activate the Java Add-in and describes how to record standard Java objects and enhance scripts that test Java applets and applications.

This chapter describes:

- **Activating the Java Add-in**
- **Recording Context Sensitive Tests**
- **Enhancing Your Script with TSL**
- **Invoking a Java Method**
- **Configuring How WinRunner Learns Object Descriptions and Runs Tests**
- **Activating a Java Edit Object**

About Testing Standard Java Objects

With the Java Add-in, you can record or write context sensitive scripts on all standard Java objects from the supported toolkits in Netscape, Internet Explorer, AppletViewer, or a standalone Java application.



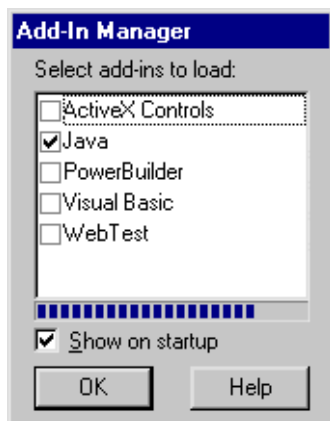
Activating the Java Add-in

Before you begin testing your Java application or applet, make sure that you have installed all the necessary files and made any necessary configuration changes. For more information, refer to Chapter 2, [Setting Up the Java Add-in](#).

To activate the Java Add-in:



- 1 Select **Programs > WinRunner > WinRunner** in the **Start** menu. The **Add-in Manager** dialog box opens.



- 2 Select **Java**.
- 3 Click **OK**. WinRunner opens with the Java Add-in loaded.

For more information on the Add-in Manager, refer to the *WinRunner User's Guide*.



Recording Context Sensitive Tests

Once you start WinRunner with the Java Add-in active, then the Java environments you selected during installation will always open with Mercury Java support active. For more information about selecting Java environments see [page 14](#).

You can confirm that your Java environment has opened properly by viewing the Mercury confirmation message in the Java console.

Note: If you are running your tests from a browser with a Java plug-in, then opening the Java console opens a second virtual machine and WinRunner cannot run tests when 2 virtual machines are open. Close the browser and console and then re-open the browser before running the tests.

If your Java application or applet uses standard Java objects from any of the supported toolkits, then you can use WinRunner to record a Context Sensitive test in Netscape, Internet Explorer, AppletViewer or a standalone Java application, just as you would with any Windows application.

As you record, WinRunner adds standard Context Sensitive TSL statements into the script. If you try to record an action on an unsupported or custom Java object, WinRunner records a generic **obj_mouse_click** or **win_mouse_click** statement. You can configure WinRunner to recognize your custom objects as push buttons, check buttons, static text or edit fields by using the Java Custom Objects wizard. For more information, refer to [Chapter 6, Configuring Custom Java Objects](#).



Enhancing Your Script with TSL

WinRunner includes several TSL functions that enable you to add Java-specific statements to your script. Specifically, you can use TSL functions to:

- Invoke a Java method.
- Set the value of a Java bean property.
- Configure the way WinRunner learns object descriptions and runs tests on Java applets and applications.
- Send the ENTER key as the parameter to the specified Java object.

For more information about TSL functions and how to use TSL, refer to the *TSL Reference Guide* or the *TSL Online Reference*.



Invoking a Java Method

You can invoke the a Java method for a specific object using the **java_activate_method** function. This function has the following syntax:

```
java_activate_method ( object, method, retval [ , param1, ... param8 ] );
```

The *object* parameter is the logical name of the object. The *method* parameter indicates the name of the Java method to invoke. The *retval* parameter is an output variable that holds a return value from the invoked method. Note that this parameter is required even for void Java methods. *param1..8* are the parameters to be passed to the Java method. All of the java method parameters, including *retval*, must belong to one of the following Java types: Boolean, boolean, Integer, int, or String.

Note: If the function returns Boolean or boolean output, the *retval* parameter will return the string representation of the output: “true” or “false”.



Books
Online



Find

Find
Again



Help



Top of
Chapter



Back

For example, you can use the `java_activate_method` function to perform actions on a list:

Add item to the list at position 2:

```
java_activate_method("list", "add", retval, "new item", 2);
```

Get number of visible rows in a list:

```
java_activate_method("list", "getRows", rows);
```

Check if an item is selected:

```
java_activate_method("list", "isIndexSelected", isSelected, 2);
```



Books
Online



Find

Find
Again



Help



Top of
Chapter

← Back

Setting the Value of a Java Bean Property

You can set the value of a Java bean property with the `obj_set_info` function. This function works on all properties that have a set method. The function has the following syntax:

```
obj_set_info ( object, property, value );
```

The *object* parameter is the logical name of the object. The object may belong to any class. The *property* parameter is the object property you want to set. Refer to the *WinRunner Users Guide* for a list of properties. The *value* parameter is the value that is assigned to the property.

Note: When writing the *property* parameter name in the function, convert the capital letters of the *property* to lowercase, and add an underscore before letters that are capitalized within the Java bean property name. Therefore the Java bean property, *MyProp* becomes *my_prop* in the TSL statement.

For example, for a property called *MyProp*, which has method `setMyProp(String)`, you can use the function as follows:

```
obj_set_info(object, my_prop, "Mercury");
```



Books
Online



Find

Find
Again



Help



Top of
Chapter

← Back

The **obj_set_info** function will return `ATTRIBUTE_NOT_SUPPORTED` for the property, *my_prop* if one of the following statements is true:

- The object does not have a method called `setMyProp`.
- The method `setMyProp()` exists, but it has more than one parameter, or the parameter does not belong to one of following Java classes: `String`, `int`, `boolean`, `Integer` or `Boolean`.
- The value parameter is not convertible to one of the above Java classes. For example, the method gets an integer number as a parameter, but the function's value parameter was a "non numeric value".
- The `setMyprop()` method throws a Java exception.



Configuring How WinRunner Learns Object Descriptions and Runs Tests

You can configure how WinRunner learns descriptions of objects, records tests, and runs tests on a Java applet or application with the **set_aut_var** function. The function has the following syntax:

```
set_aut_var ( variable, value );
```

The following variables and corresponding values are available:

EDIT_REPLAY_MODE

Controls how WinRunner performs actions on edit fields. Use one or more of the following values:

“S”-uses the setValue () method to set a value of the edit object.

“P”-sends KeyPressed event to the object for every character from the input string.

“T”-sends KeyTyped events to the object for every character from the input string.

“R”-sends KeyReleased event to the object for every character from the input string.

Default value: “PTR”.

Note that the default action sends a triple event to the edit field (KeyPressed-KeyTyped-KeyReleased).



EVENT_MODEL

Sets the event model that will be used to send events to the AUT objects. Use one of the following values:

“NEW”-for applications written in the new event model

“OLD”-for applications written in the old even model

“DEFAULT”- Uses the OLD event model for AWT objects and NEW for all other toolkit objects.

Default value: "DEFAULT"

MAX_TEXT_DISTANCE

Sets the maximum distance in pixels, to look for attached text.

Default value: 100

REPLAY_INTERVAL

Sets the processing time in milliseconds between the execution of two functions.

Default value: 200

RETRY_DELAY

Sets the maximum time in milliseconds to wait before retrying to execute a command.

Default value: 1000



SKIP_ON_LEARN Controls how WinRunner learns a window. Mercury Interactive classes listed in the variable are ignored. May contain a list of Mercury Interactive classes, separated by spaces. By default, only non-“object” objects are learned.

Default value: "object"

TABLE_RECORD_MODE Sets the record mode for a table object (CS or ANALOG).

“CS”: indicates that the record mode is Context Sensitive.

“ANALOG”: records only low-level (Analog) table functions: **tbl_click_cell**, **tbl_dbl_click_cell**, and **tbl_drag**. (JFC JTable object only).

Default value: “CS”

COLUMN_NUMBER Minimum number of columns for an Oracle table to be considered a table object. Otherwise the edit fields are treated as separate objects.

Default value: 2

MAX_COLUMN_GAP The maximum number of pixels between objects in a table to be considered a column.

(Oracle only)

Default value: 12



MAX_LINE_DEVIATION

The maximum number of pixels between objects to be considered to be on a single line. (Oracle only)

Default value: 8

MAX_LIST_COLUMNS

The maximum number of columns in an Oracle LOV object to be considered a list. A larger number constitutes a table. (Oracle only)

Default value: 99

MAX_ROW_GAP

The maximum number of pixels between objects to be considered one table row. (Oracle only)

Default value: 12

RECORD_BY_NUM

Controls how items in list, combo box, and tree view objects are recorded. The variable can be one of the following values: list, combo, tree, or a combination separated by a space. If one of these objects has been detected, numbers are recorded instead of the item names.



Activating a Java Edit Object

You can activate an edit field with the **edit_activate** function. This is the equivalent of a user pressing the ENTER key on an edit field. This function has the following syntax:

```
edit_activate ( object );
```

The object parameter is the logical name of the edit object on which you want to perform the action.

For example, if you want to enter John Smith into the edit field, "Text Fields_0", then you can set the text in the edit field and then use **edit_activate** to send the activate event as in the following script:

```
set_window("swingsetapplet.html", 8);  
edit_set("Text Fields:_0", "John Smith 2");  
edit_activate("Text Fields:_0");
```



Configuring Custom Java Objects

This chapter explains how to add Java objects to the GUI map and to configure custom Java objects as standard GUI objects.

This chapter describes:

- **Adding Custom Java Objects to the GUI Map**
- **Configuring Custom Java Objects with the Custom Object Wizard**

About Configuring Custom Java Objects

With the Java Add-in you can use WinRunner to record test scripts on most Java applications and applets, just like you would in any other Windows application. If you record an action on a custom or unsupported Java object, however, WinRunner maps the object to the general object class in the WinRunner GUI map. When this occurs, you can use the Custom Object wizard to configure the GUI map to recognize these Java objects as a push button, check button, static text or text field. This makes the test script easier to read and makes it easier for you to perform checks on relevant object properties.

After using the wizard to configure a custom object, you can add it to the GUI map, record actions and run it as you would any other WinRunner test.



Adding Custom Java Objects to the GUI Map

Once the Java Add-in is loaded, you can add custom Java objects to the GUI map by recording an action or using the GUI Spy to learn the objects. By default, however, these objects will each be mapped to the general object class, and activities performed on those objects will generally result in generic **obj_mouse_click** or **win_mouse_click** statements. The objects will usually be identified in the GUI map by their label property, or if WinRunner does not recognize the label, by a numbered class_index property.

For example, suppose you wish to record a test on a sophisticated subway routing Java application. This application lets you select your starting location and destination, and then suggests the best subway route to take. The application allows you to select which bus line(s) you prefer to use for your travels.

Since WinRunner cannot recognize the custom Java check boxes as GUI objects, when you check one of the options, the GUI map defines the objects as:

```
{  
class: object,  
label: "M (Nassau St Express)"  
}
```



If you were to record a test in which you selected the “M”, “A” and “Six” lines as your preferred lines, WinRunner would create a test script similar to the following:

```
set_window("Line Selection", 1);  
obj_mouse_click("M (Nassau St Express)", 6, 32, LEFT);  
obj_mouse_click("A (Far Rockaway) (Eighth Av...", 10, 30, LEFT);  
obj_mouse_click("Six (Lexington Ave Local)", 5, 27, LEFT);
```

This test script is difficult to understand. If, instead, you use the Custom Object wizard in order to associate the custom objects with the check button class, WinRunner records a script similar to the following:

```
set_window("Line Selection", 8);  
button_set("M (Nassau St Express)", ON);  
button_set("A (Far Rockaway) (Eighth Av...", ON);  
button_set("Six (Lexington Ave Local)", ON);
```

Now it is easy to see that the objects in the script are check buttons and that the user selected (turned ON) the three check buttons.



Configuring Custom Java Objects with the Custom Object Wizard

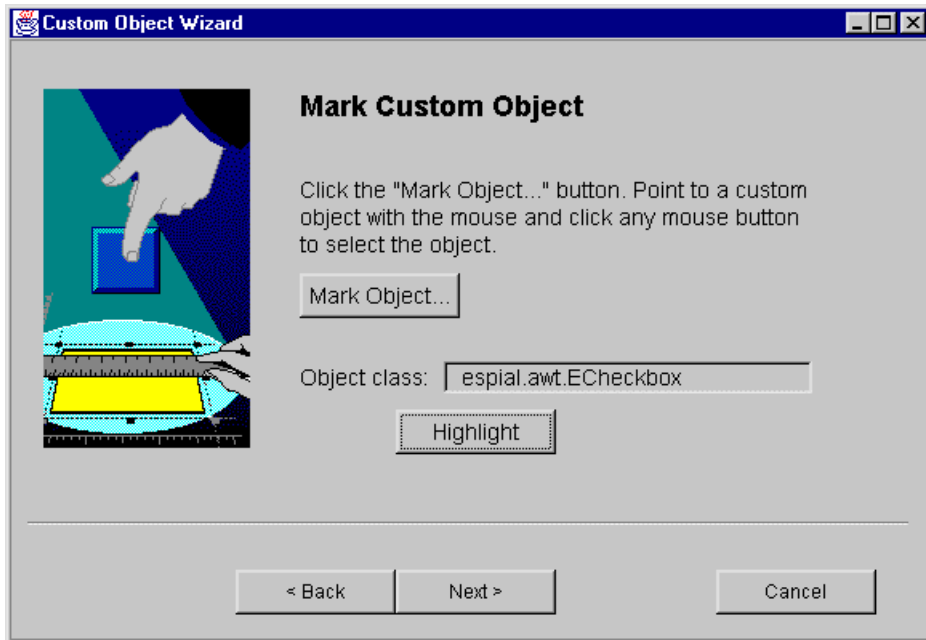
You configure a custom Java object in WinRunner using the Custom Object wizard in order to assign the object to a standard GUI class and to object properties which will uniquely identify the object.

To configure a Java object using the Custom Object wizard:

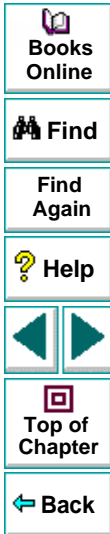
- 1 Open your Java application containing custom Java objects.
- 2 Choose **Tools > Java GUI Map Configuration**. The Custom Object Welcome screen opens. Click **Next**.



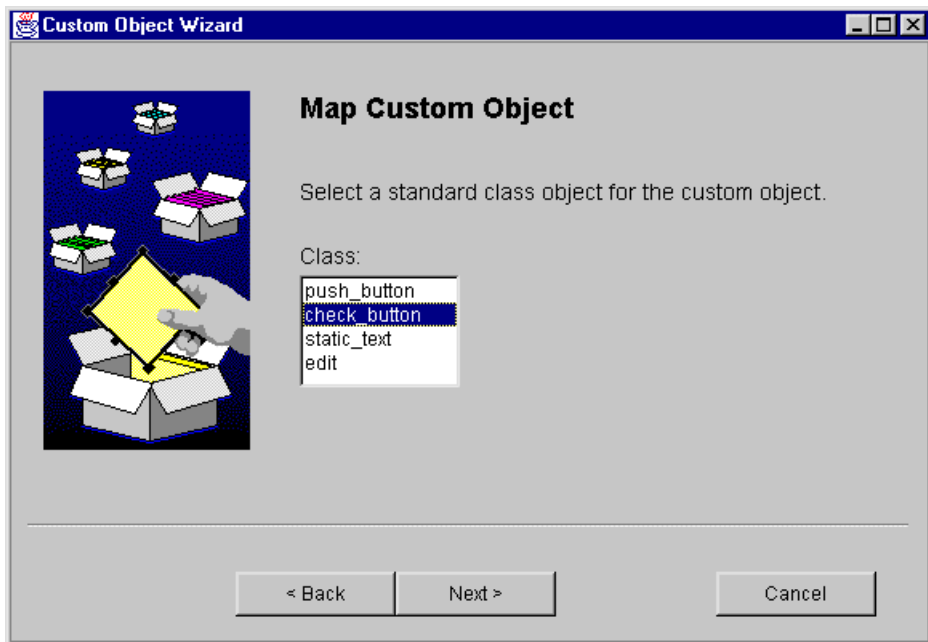
- 3 Click the **Mark Object** button. Point to an object in the Java application. The object is highlighted. Click any mouse button to select the object. A default name appears in the **Object class** field.



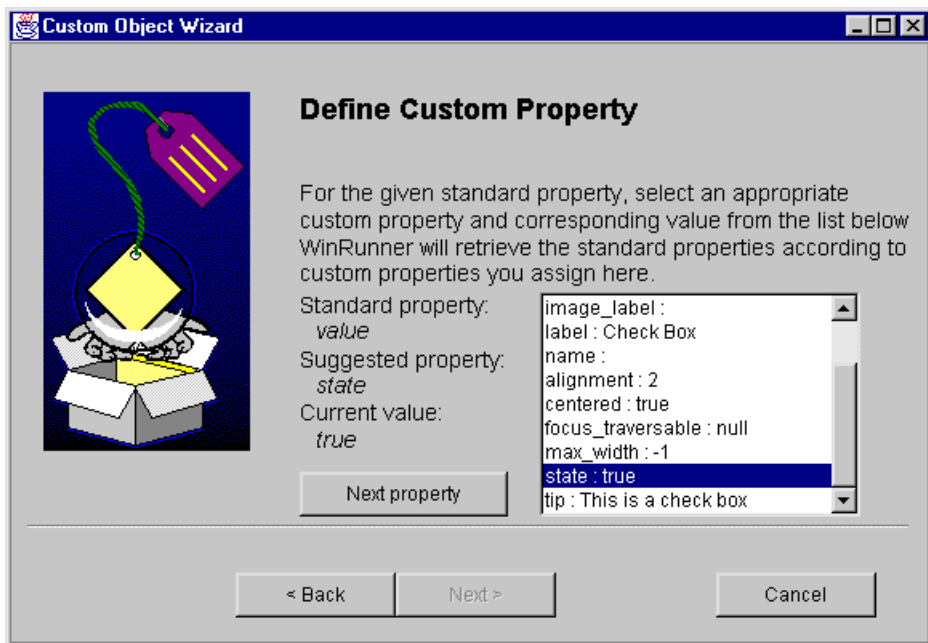
- 4 Click the **Highlight** button if you want to confirm that the correct option was selected. The object you selected is highlighted.
- 5 If you want to select a different object, repeat steps 3 and 3. When you are satisfied with your selection, click **Next**.



- 6 Select a standard class object for the object you selected. Click **Next**.



- 7 Select an appropriate custom property and corresponding property value from the property list on the right to uniquely identify the object, or accept the suggested property and value.



If you selected `check_button` as the standard object, two custom properties are necessary. After selecting the first property, click **Next Property** to select the second property for the object. Click **Next**.

- 8 If you wish to learn another custom Java object, click **Yes**. The wizard returns to the Mark Custom Object screen. Repeat steps 3-6 for each custom object you want to configure. If you are finished configuring custom Java options, click **No**.



- 9 The Finish screen opens. Click the **Finish** button to close the Custom Object wizard.



- 10 Close and reopen your Java application or applet in order to activate the new configuration for the object(s).

Note: The new object configuration settings will not take effect until the Java application or applet is restarted.

Once you have configured a custom Java option using the Custom Object wizard, you can add the objects to the GUI map or record a test as you would in any Windows application. For more information on the GUI map and recording scripts, refer to the *WinRunner User's Guide*.



Using Java Direct Call (JDC) Mechanism

This chapter explains how to use the Java Direct Call (JDC) Mechanism to call Java functions from TSL scripts.

This chapter describes

- [Using the JDC Mechanism](#)
- [Preparing a TSL Script for use with JDC](#)
- [Using JDC: An Example](#)

About Java Direct Call Mechanism

JDC enables you to specify a Java function to execute from the TSL script. This user-defined Java function may contain any standard Java code.

Unlike the `java_activate_method` function described in Chapter 5, [Testing Standard Java Objects](#), JDC functions work on Java applications that do not have any Java User Interface object bound to them. These functions can retrieve string parameters provided in TSL.



Books
Online



Find

Find
Again



Help



Top of
Chapter



Back

Using the JDC Mechanism

You can call use standard Java code to call a Java function from a TSL script.

To enable the JDC mechanism:

- 1 Create a Java class listing and implementing all methods to be called from the TSL script.

All methods must follow the prototype convention:

```
public int jdc_<func_name >( String [ ] params );
```

func_name a name of the function

params an array of parameters passed from WinRunner.

- 2 Register JDC class(es) with WinRunner by using the following TSL statement:

```
set_aut_var("JDC_CLASSES", "foo.bar.class1;foo.bar.class2");
```

Note: You can create as many JDC classes as required. JDC classes must be found in the CLASSPATH.



Books
Online



Find

Find
Again



Help



Top of
Chapter



Back

- 3 Provide an "extern" definition for the JDC function in TSL.

For example, if you have defined a JDC function in your Java class as:

```
public int jdc_print_strings(String[] param);
```

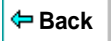
make the following declaration in TSL:

```
extern int jdc_print_strings(in string p1, in string p2);
```

When calling Java, param[0] will contain p1 and param[1] will contain p2.

- 4 Call the JDC function from TSL:

```
jdc_print_strings("str1", "str2");
```



Preparing a TSL Script for use with JDC

Your TSL script must begin with the function **jdc_aut_connect**. This function establishes a connection between WinRunner and Java applications and must be executed at least once. You use this function as follows:

```
jdc_aut_connect ( in_timeout );
```

Using JDC: An Example

The example below shows how a user prepares the Java source file with definitions for two Java functions. Then the user registers the Java functions with WinRunner so that he can call the Java functions from the TSL script.

Preparing the Java Source File

The following sample Java source file defines 2 Java functions for later use in the TSL script.

```
// Sample File of JDC calling mechanism.
```

```
public class JdcExample {
```

```
/**
```

```
    This function will print the first parameter that it  
    receives to the console
```



```
*/
public static int jdc_simple_call(String[] params) {
    String first_param = params[0];
    System.out.println("jdc_simple_call called: Got parameter: " +
        first_param);
    return 0;
}

/**
    This function will return the upper case version of the first
    parameter string in the second parameter.
*/
public static int jdc_out_par_call(String[] params) {
    // Convert input parameters
    String in_par = params[0];
    String out_par = params[1];

    System.out.println("jdc_out_par_call called: Got parameter: " +
        in_par);
    out_par = in_par.toUpperCase();
    // Prepare output parameters
    params[1] = out_par;
    return 0;
}
}
```



Books
Online



Find

Find
Again



Help



Top of
Chapter



Back

Registering JDC with WinRunner and calling Java functions in the TSL Script

The TSL script segment below shows how to define the “extern” declaration, to load and register the JDC classes defined in the Java source code, and then to call the Java functions.

```
# define "extern" declaration
extern int jdc_simple_call(in string str);
extern int jdc_out_par_call(in string str1, out string str2<256>);

# load and register JDC classes
set_aut_var("JDC_CLASSES", "JdcExample");
jdc_aut_connect(10);

# call JDC functions - this will print the parameter to the Java console
r1=jdc_simple_call("my string");
r2=jdc_simple_call(256);

# this will put the Upper Case form of the parameter in the UpperCaseParam var.
r3=jdc_out_par_call("my string", UpperCaseParam);

pause(UpperCaseParam);
```



Once you complete the Java Add-in installation process, you should be able to successfully record from Netscape, Internet Explorer, or a standalone Java application.

Handling General Problems Testing Applets

To analyze problems testing applets from Netscape or Internet Explorer:

- 1 Perform each of the following checks:
 - View the Java console and confirm that one of the following confirmation messages appears: “Mercury Java support is active” or “Init Mercury support”.
 - Confirm that you are able to test your applet with the AppletViewer.
 - Run the Java Add-in Verifier in order to verify your Java Add-in installation.
 - For more information about running the Java Add-in Verifier, see [Verifying Your Java Add-in Installation](#) on page 26.
- 2 If any of the above checks are not successful, close WinRunner and all browsers and re-install the Java Add-in.
- 3 If you still have problems testing applets from Netscape or Internet Explorer, please contact Mercury Support.



Handling Specific Java Add-in Problems

If you are having specific problems installing or recording, try the relevant solutions from the following list:

- If the Java Add-in setup program displays the following message:

The Setup program was unable to locate a version of JDK/JRE on this computer.

Map to the installation drive from Network Neighborhood or the Windows Explorer.

- If the Java console and a Java plug-in are open simultaneously, the Java add-in support will not function properly as this scenario results in two virtual machines and WinRunner cannot distinguish between them.

Close the browser and Java console, then re-open the browser and try again.

- If you have JDK 1.1.x installed and you want to test an applet, enter:

```
AppletViewer <URL address>
```

- If you have JDK 1.2.x installed and you want to test an applet, enter:

```
AppletViewer -J-Xbootclasspath:<WinRunner installation folder>\classes;<WinRunner installation folder>\classes\srv;<Java Add-in installation folder>\jre\lib\rt.jar;%classpath% <URL address>
```



Note: To test a Java application running under JDK 1.2.x, you must start the application with the following line:

```
java -Xbootclasspath:<WinRunner installation folder>\classes;<WinRunner installation folder>\classes\srv;<Java installation folder>\jre\lib\rt.jar;%classpath% <Application class>
```

To test Java applets running in AppletViewer under JDK 1.2.x, you must start the applet with the following line:

```
AppletViewer -J-Xbootclasspath:<WinRunner installation folder>\classes;<WinRunner installation folder>\classes\srv;<Java installation folder>\jre\lib\rt.jar;%classpath% <Applet URL>
```

- If you have JRE installed and you want to test an application, enter:

```
jre -cp %classpath% <Application class>
```

Then check the classpath and verify that it contains the *WinRunner installation folder\classes* folder and *WinRunner installation folder\classes\srv* folder.

- If you have Microsoft JView and you want to set the classpath, use only the classpath environment variable. Do not use the /cp /cp:p or /cp:a options.



Books
Online



Find

Find
Again



Help



Top of
Chapter



Back

Index

A

- activate changes [65](#)
- activating an edit field [55](#)
- Add-in Manager [44](#)
- adding custom Java objects to the GUI map
[57–58](#)
- AppletViewer [14](#)

C

- check button [61](#), [62](#)
- classes
 - Java [16](#)
 - local [29](#)
 - Mercury [16](#)
 - remote [29](#)
 - remote and local [29](#)
- configuring custom Java objects [56–65](#)
- configuring the way WinRunner learns [51](#)
- configuring the web server [23](#)
- conventions. See [typographical conventions](#)
- Custom Object Wizard [59–65](#)
- custom property [62](#)

D

- disabling or uninstalling the Java Add-in
[38–41](#)
- disabling the Java Add-in temporarily [38](#)

E

- edit field [61](#)
- edit_activate function [55](#)
- ENTER key [55](#)
- extern definition [68](#)

G

- GUI spy [57](#)

H

- highlight [60](#)

I

- installation, verifying [26–37](#)
- installing the Java Add-in [11–25](#)
- Internet Explorer [14](#)
- invoking a Java method [47](#)
- invoking a java method [47](#)



A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

J

- Java Add-In package [9](#)
- Java Add-in Verifier, launching [27](#)
- Java Add-in, starting the [44](#)
- Java application [14](#)
- Java bean properties, setting the value of [49](#)
- Java Direct Call Mechanism [66–71](#)
- Java environment(s) [14](#)
- java method [47](#)
- java method, invoking [47](#)
- Java Plug-in [14](#)
- Java Wizard [59–65](#)
- java_activate_method function [47](#)
- JDC [66–71](#)
- jdc_aut_connect function [69](#)
- JDK [18](#)
- JDK detection [30](#)
- JInitiator [14](#)
- JIT option [31](#)
- JRE [18](#)
- JView [74](#)

L

- local access [17](#)
- local classes [29](#)

M

- mark object [60](#)
- mercury.properties file [38](#)
- Microsoft JView [14](#)

N

- Netscape [14](#)
- no browser [14](#)

O

- obj_mouse_click function [45](#)
- obj_mouse_click statement [57](#)
- obj_set_info function [49](#)
- ATTRIBUTE_NOT_SUPPORTED return value [50](#)
- object configuration, activate changes in [65](#)
- Oracle [14](#)

P

- patch browser [17](#)
- program folder [21](#)
- push button [61](#)

Books
Online

Find

Find
Again

Help

Top of
Chapter

Back

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

R

registering JDC classes with WinRunner 67
 remote classes 29

S

sending the ENTER key 55
 set_aut_var 51
 set_aut_var function
 COLUMN_NUMBER variable 53
 EDIT_REPLAY_MODE variable 51
 EVENT_MODEL variable 52
 MAX_COLUMN_GAP variable 53
 MAX_LINE_DEVIATION variable 54
 MAX_LIST_COLUMNS variable 54
 MAX_ROW_GAP variable 54
 MAX_TEXT_DISTANCE variable 52
 RECORD_BY_NUM variable 54
 REPLAY_INTERVAL variable 52
 RETRY_DELAY variable 52
 SKIP_ON_LEARN variable 53
 TABLE_RECORD_MODE variable 53
 setting the value of a Java bean property 49
 setup program, running the 12
 standard class object 61
 static text 61
 system requirements 10

T

troubleshooting
 handling specific Java Add-in problems 73
 Java Add-in recording problems 72–74
 Java Add-in setup 73
 Java console 73
 JDK/JRE 73
 JView 74
 testing applets 72
 TSL functions, using with Java applications and
 applets 43–55
 typographical conventions in this guide 7

U

uninstall Java Add-in 38–41
 uninstall Java Add-in, how to 40

V

variables, for set_aut_var 51–54
 Verifier
 report 37
 Verifier, launching 27
 verifying your Java Add-in installation 26–37



Books
Online



Find

Find
Again



Help



Top of
Chapter



Back

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

W

- win_mouse_click function [45](#)
- win_mouse_click statement [57](#)



A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

WinRunner - Java Add-in Installation and User's Guide, Version 6.0

© Copyright 1994 - 1999 by Mercury Interactive Corporation

All rights reserved. All text and figures included in this publication are the exclusive property of Mercury Interactive Corporation, and may not be copied, reproduced, or used in any way without the express permission in writing of Mercury Interactive. Information in this document is subject to change without notice and does not represent a commitment on the part of Mercury Interactive.

Mercury Interactive may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents except as expressly provided in any written license agreement from Mercury Interactive.

WinRunner, XRunner, LoadRunner, TestDirector, TestSuite, and WebTest are registered trademarks of Mercury Interactive Corporation in the United States and/or other countries. Astra, Astra SiteManager, Astra SiteTest, RapidTest, QuickTest, Visual Testing, Action Tracker, Link Doctor, Change Viewer, Dynamic Scan, Fast Scan, and Visual Web Display are trademarks of Mercury Interactive Corporation in the United States and/or other countries.

This document also contains registered trademarks, trademarks and service marks that are owned by their respective companies or organizations. Mercury Interactive Corporation disclaims any responsibility for specifying which marks are owned by which companies or organizations.

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

Mercury Interactive Corporation
1325 Borregas Avenue
Sunnyvale, CA 94089
Tel. (408) 822-5200 (800) TEST-911
Fax. (408) 822-5300

