OPTIMIZE

MERCURY WINRUNNER™
VERSION 9.2
Tutorial

MERCURY™
BUSINESS TECHNOLOGY OPTIMIZATION

Mercury WinRunner
Tutorial
Version 9.2

Document Release Date: February 26, 2007

MERCURY™

Mercury WinRunner Tutorial, Version 9.2

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

WRTUT9.2/01

Table of Contents

Table of Contents

Welcome to the WinRunner Tutorial

Welcome to the WinRunner tutorial, a self-paced guide that teaches you the basics of testing your application with WinRunner. This tutorial will familiarize you with the process of creating and running automated tests and analyzing the test results.

Using This Tutorial

The tutorial is divided into 11 lessons. In each lesson, you will create and run tests on the sample Flight Reservation application located in the WinRunner program group.

The sample Flight Reservation application comes in two versions: Flight 4A and Flight 4B. Flight 4A is a fully working product, while Flight 4B has some "bugs" built into it. These versions are used together in the WinRunner tutorial to simulate the development process, in which the performance of one version of an application is compared with that of another.

After you complete the tutorial, you can apply the skills you learn to your own application.

Lesson 1, Introducing WinRunner compares automated and manual testing methods. It introduces the WinRunner testing process and familiarizes you with the WinRunner user interface.

Lesson 2, Setting Up the GUI Map explains how WinRunner identifies GUI (Graphical User Interface) objects in an application and describes the two modes for organizing GUI map files.

Lesson 3, Recording Tests teaches you how to record a test script and explains the basics of Test Script Language (TSL)—Mercury's C-like programming language designed for creating scripts.

Lesson 4, Synchronizing Tests shows you how to synchronize a test so that it can run successfully even when an application responds slowly to input.

Lesson 5, Checking GUI Objects shows you how to create a test that checks GUI objects. You will use the test to compare the behavior of GUI objects in different versions of the sample application.

Lesson 6, Checking Bitmaps shows you how to create and run a test that checks bitmaps in your application. You will run the test on different versions of the sample application and examine any differences, pixel by pixel.

Lesson 7, Programming Tests with TSL shows you how to use visual programming to add functions and logic to your recorded test scripts.

Lesson 8, Creating Data-Driven Tests shows you how to run a single test on several sets of data from a data table.

Lesson 9, Creating Batch Tests shows you how to create a batch test that automatically runs the tests you created in earlier lessons.

Lesson 10, Maintaining Your Test Scripts teaches you how to update the GUI object descriptions learned by WinRunner, so that you can continue to use your test scripts as the application changes.

Lesson 11, Where Do You Go from Here? tells you how to get started testing your own application and where you can find more information about WinRunner.

Typographical Conventions

This book uses the following typographical conventions:

1, 2, 3	Bold numbers indicate steps in a procedure.
>	The greater-than sign separates menu levels (for example, **File** > **Open**).
Stone Sans	The **Stone Sans** font indicates names of interface elements (for example, the **Run** button) and other items that require emphasis.
Bold	**Bold** text indicates method or function names.
Italics	*Italic* text indicates method or function arguments, file names in syntax descriptions, and book titles. It is also used when introducing a new term.
<>	Angle brackets enclose a part of a file path or URL address that may vary from user to user (for example, **<MyProduct installation folder>\bin**).
Arial	The Arial font is used for examples and text that is to be typed literally.
Arial bold	The **Arial bold** font is used in syntax descriptions for text that should be typed literally.
SMALL CAPS	The SMALL CAPS font indicates keyboard keys.
...	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.
[ ]	Square brackets enclose optional arguments.
|	A vertical bar indicates that one of the options separated by the bar should be selected.

1

Introducing WinRunner

In this lesson, you will learn about:	On page:
The Benefits of Automated Testing	11
Understanding the Testing Process	12
Exploring the WinRunner Window	13

The Benefits of Automated Testing

If you have ever tested software manually, you are aware of its drawbacks. Manual testing is time-consuming and tedious, requiring a heavy investment in human resources. Worst of all, time constraints often make it impossible to manually test every feature thoroughly before the software is released. This leaves you wondering whether serious bugs have gone undetected.

Automated testing with WinRunner addresses these problems by dramatically speeding up the testing process. You can create test scripts that check all aspects of your application, and then run these tests on each new build. As WinRunner runs tests, it simulates a human user by moving the mouse cursor over the application, clicking Graphical User Interface (GUI) objects, and entering keyboard input—but WinRunner does this faster than any human user.

With WinRunner you can also save time by running batch tests overnight.

Benefits of Automated Testing	
Fast	WinRunner runs tests significantly faster than human users.
Reliable	Tests perform precisely the same operations each time they are run, thereby eliminating human error.
Repeatable	You can test how the software reacts under repeated execution of the same operations.
Programmable	You can program sophisticated tests that bring out hidden information from the application.
Comprehensive	You can build a suite of tests that covers every feature in your application.
Reusable	You can reuse tests on different versions of an application, even if the user interface changes.

Understanding the Testing Process

The WinRunner testing process consists of six main phases:

1 Teaching WinRunner the objects in your application

WinRunner must learn to recognize the objects in your application to run tests. The preferred way to teach WinRunner your objects depends on the GUI map mode you select. The two GUI map modes are described in detail in subsequent lessons.

2 Creating additional test scripts that test your application's functionality

WinRunner writes scripts automatically when you record actions on your application, or you can program directly in Mercury's Test Script Language (TSL).

3 Debugging the tests

You debug the tests to check that they operate smoothly and without interruption.

4 Running the tests on a new version of the application

You run the tests on a new version of the application to check the application's behavior.

5 Examining the test results

You examine the test results to pinpoint defects in the application.

6 Reporting defects

If you have access to Quality Center, you can report any defects to a database. Quality Center is the Mercury software test management tool.

Exploring the WinRunner Window

Before you begin creating tests, you should familiarize yourself with the WinRunner main window.

Note: To open WinRunner, your screen display settings must be set to use at least 256 colors.

To start WinRunner:

Choose **Start** > **Programs** > **WinRunner** > **WinRunner**. The WinRunner Add-in Manager dialog box opens.
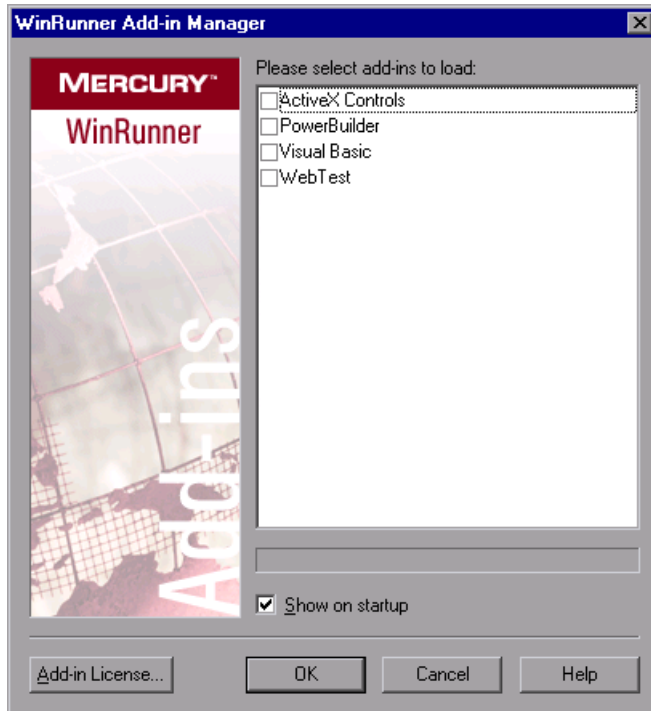
Note: The first time you start WinRunner, the "What's New in WinRunner" Help also opens.

The WinRunner Add-in Manager dialog box contains a list of the add-ins available on your computer. For the purposes of this tutorial, ensure that all of the add-in check boxes are cleared, and click **OK**.

The Welcome to WinRunner window opens. From the Welcome to WinRunner window, you can create a new test, open an existing test, or view an overview of WinRunner in your default browser.

Tip: If you do not want the Welcome to WinRunner window to open the next time you start WinRunner, clear the **Show on startup** check box.

For the purposes of this tutorial, click **Create a New Test** to close the Welcome to WinRunner window and begin working in WinRunner.

WinRunner displays each test you create or run in a test window. You can open many tests at one time.

1   *2*   *3*

1   The WinRunner window displays all open tests.

2   Each test appears in its own test window. You use this window to record, program, and edit test scripts.

3   Buttons on the toolbars help you open, run, save, and debug tests.

4   The User toolbar provides easy access to test creation tools.

5   The status bar displays information about selected commands and the current test run.

6   The execution arrow indicates the next script line to be performed.

7   The Function Viewer enables you to view, edit and use loaded functions in your tests.

8   The Debug Viewer enables you to view the Watch List, Breakpoints, and Call Chain of your test.

8   *7*   *6*   *5*   *4*
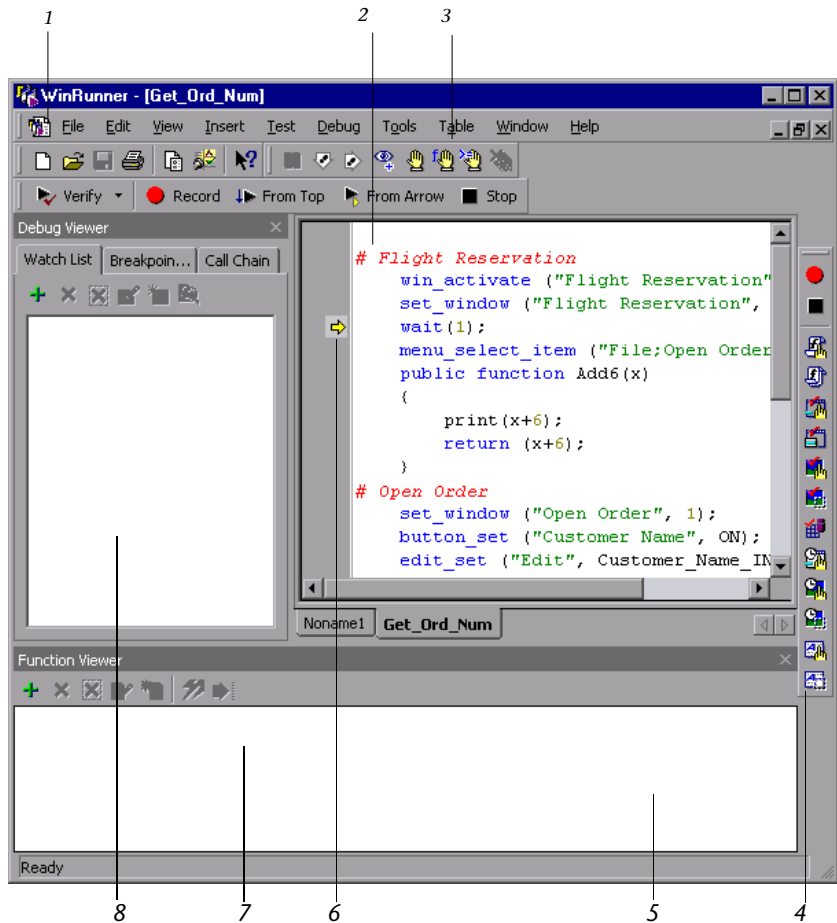
The **File** toolbar provides easy access to frequently performed tasks, such as opening and saving tests, and viewing test results.
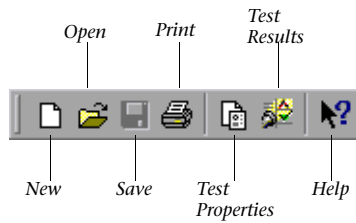
The **Test** toolbar provides easy access to buttons used while running and maintaining tests.
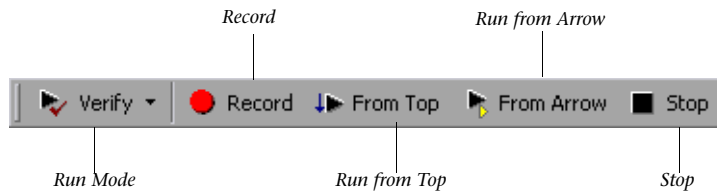
The **Debug** toolbar provides easy access to buttons used while debugging tests.
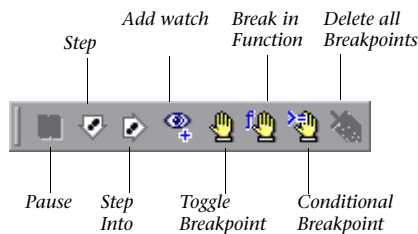
The **User** toolbar displays the tools you frequently use to create test scripts.

Record - Context Sensitive

Stop

Insert Function for Object/Window

Insert Function from Function Generator

GUI Checkpoint for Object/Window

GUI Checkpoint for Multiple Objects

Bitmap Checkpoint for Object/Window

Bitmap Checkpoint for Screen Area

Default Database Checkpoint

Synchronization Point for Object/Window Property

Synchronization Point for Object/Window Bitmap

Synchronization Point for Screen Area Bitmap

Get Text from Object/Window

Get Text from Screen Area

When you create tests, you can minimize the WinRunner window and work exclusively from the toolbar. The commands on the File toolbar, the Test toolbar, and the User toolbar are described in detail in subsequent lessons.

By default, the User toolbar is hidden. To display the User toolbar, choose **View** > **User Toolbar**. The User toolbar is customizable. You can add or remove buttons using the **View** > **Customize User Toolbar** menu option. When you re-open WinRunner, the User toolbar appears as it was when you last closed it.

Note that you can also execute many commands using **softkeys**. Softkeys are keyboard shortcuts for carrying out menu commands. You can configure the softkey combinations for your keyboard using the Softkey Configuration utility in the WinRunner program group. For more information, see the "WinRunner at a Glance" chapter in the *Mercury WinRunner User's Guide*.

Now that you are familiar with the main WinRunner window, take a few minutes to explore these window components before proceeding to the next lesson.

2

Setting Up the GUI Map

In this lesson, you will learn about:	On page:
How WinRunner Identifies GUI Objects	19
Spying on GUI Objects	20
Choosing a GUI Map Mode	23
Using the RapidTest Script Wizard	25

How WinRunner Identifies GUI Objects

GUI applications are made up of GUI objects such as windows, buttons, lists, and menus.

When WinRunner learns the description of a GUI object, it looks at the object's physical properties. Each GUI object has many physical properties such as **class**, **label**, **width**, **height**, **handle**, and **enabled** to name a few. WinRunner only learns the properties it needs to uniquely distinguish an object from all other objects in the application. For more information regarding properties, refer to the "Configuring the GUI Map" chapter in the *Mercury WinRunner User's Guide*.

For example, when WinRunner looks at an **OK** button, it might recognize that the button is located in an Open window, belongs to the **push_button** object class, and has the text label **OK**.

Spying on GUI Objects

To help you understand how WinRunner identifies GUI objects, examine the objects in the sample Flight Reservation application.

flight4a.exe

1 Start the Flight Reservation application.

Choose **Start** > **Programs** > **WinRunner** > **Sample Applications** > **Flight 4A**. The Login window opens.
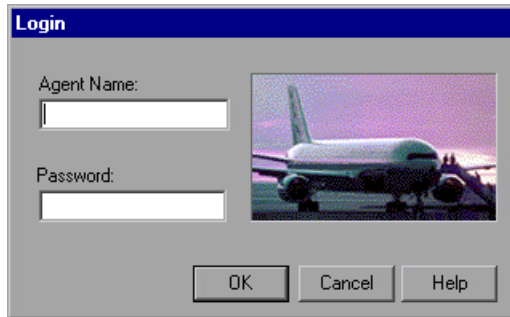
2 Start WinRunner.

Choose **Start** > **Programs** > **WinRunner** > **WinRunner**. In the Welcome window, click the **Create a New Test** button. If the Welcome window does not open, choose **File** > **New**.

3 Open the GUI Spy. This tool enables you to spy on the properties of GUI objects.

Choose **Tools** > **GUI Spy**. The GUI Spy opens. Select the **Hide WinRunner** check box.
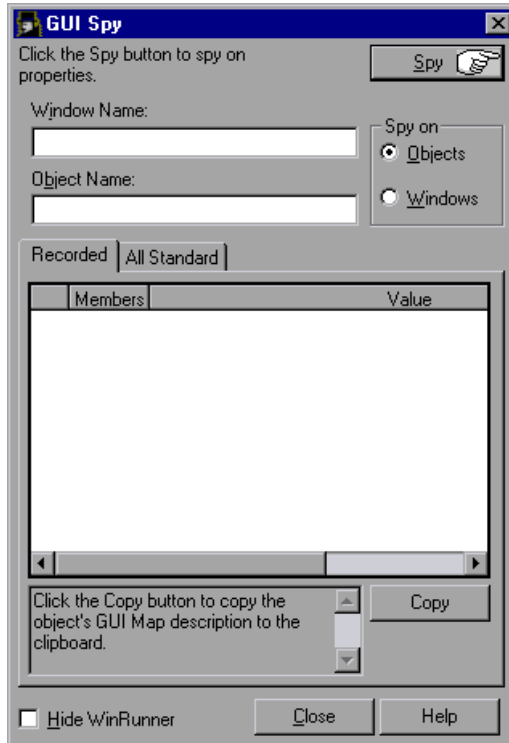
4 View the properties that provide a unique description of the OK button.

In the GUI Spy, click the **Spy** button. WinRunner is minimized so that you can see the Login window. Move the pointer over objects in the Login window. Notice that each object flashes as you move the pointer over it, and the GUI Spy displays its properties.

Place the pointer over the **OK** button and press LEFT CTRL + F3. This freezes the **OK** button's description in the GUI Spy.

5 Examine the properties of the OK button.

At the top of the dialog box, the GUI Spy displays the name of the window in which the object is located and the object's logical name.

In the Recorded tab, the property names and values that would be recorded are listed. For example, **label** OK indicates that the button has the text label **OK**, and **class** push_button indicates that the button belongs to the **push_button** object class.

As you can see, WinRunner needs only a few properties to uniquely identify the object.

6 **Take a few minutes to view the properties of other GUI objects in the Login window.**

Click the **Spy** button and move the pointer over other GUI objects in the Login window.

If you would like to view an expanded list of properties for an object, press **Left Ctrl** + **F3** to stop the current Spy, and then click the **All Standard** tab.

7 **Exit the GUI Spy.**

Press **Left Ctrl** + **F3** to stop the current Spy and click **Close**.

Choosing a GUI Map Mode

Before you start teaching WinRunner the GUI of an application, you should consider whether you want to organize your GUI map files in the GUI Map file per test mode or the Global GUI Map file mode.

The GUI Map file per test mode

In the GUI Map file per test mode, a GUI map file is created automatically every time you create a new test. The GUI map file that corresponds to your test is automatically saved whenever you save your test and automatically loaded whenever you open your test.

If you are new to WinRunner or to testing, you may want to consider working in the GUI Map file per test mode. This is the simplest mode for inexperienced testers and for ensuring that updated GUI map files are saved and loaded.

The Global GUI Map File Mode

In the Global GUI Map file mode, you can use a single GUI map for a group of tests. When you work in the Global GUI Map file mode, you need to save the information that WinRunner learns about the properties into a GUI map file. When you run a test, you must load the appropriate GUI map file.

If you are familiar with WinRunner or with testing, it is probably most efficient to work in the Global GUI Map file mode.

Setting Your Preferred GUI Map File Mode

By default, WinRunner is set to the Global GUI Map file mode. To change the mode to the GUI Map file per test mode, choose **Tools > General Options,** select the **General** category, and select **GUI Map File per Test.** Click **OK** to close the dialog box.

Note: If you change the GUI Map file mode, you must restart WinRunner for the changes to take effect.

Getting Started

The remaining sections in this lesson can be performed only in the Global GUI Map file mode. If you choose to work in the GUI Map file per test mode, change the mode setting as described above and proceed to Lesson 3, "Recording Tests" on page 29.

If you choose to work in the Global GUI Map file mode, proceed to the section, "Using the RapidTest Script Wizard" on page 25.

Important Notes:

It is recommended that you select one GUI map file mode and use it for the duration of the tutorial, rather than switching between modes in different lessons. If you want to practice using both modes, perform the entire tutorial once for each mode.

If you choose to work in Global GUI Map file mode and you close WinRunner before completing all lessons in this tutorial, be sure to save the GUI map file before closing WinRunner, as described on page 37. When you reopen WinRunner, confirm that the GUI map file is loaded as described on page 31, before continuing the lessons.

Using the RapidTest Script Wizard

If you choose the Global GUI Map file mode, the RapidTest Script wizard is usually the easiest and quickest way to start the testing process.

Note: The RapidTest Script wizard is not available when you work in the GUI Map file per test mode.

The RapidTest Script wizard systematically opens the windows in your application and learns the description of every GUI object. The wizard stores this information in a GUI map file. To observe WinRunner's learning process, use the RapidTest Script wizard on the Flight Reservation application.

Note: The RapidTest Script wizard is not available when the Terminal Emulator, WebTest, or Java add-ins are loaded. Therefore, if you are using one or more of these add-ins, skip the remaining sections of this lesson, or close and reopen WinRunner without loading these add-ins.

---

flight4a.exe

1 Log in to the Flight Reservation application.

If the Login window is not already open on your desktop, choose **Start > Programs** > **WinRunner** > **Sample Applications** > **Flight 4A**.

Type your name (at least four characters) in the **Agent Name** box, type mercury in the **Password** box, and click **OK**. The Flight Reservation application opens.

2 Start WinRunner.

If WinRunner is not already open, choose **Start > Programs** > **WinRunner** > **WinRunner**. If the Welcome window is open, click the **New Test** button. Otherwise, choose **File** > **New**. A new test window opens.

3 Start the RapidTest Script wizard.

Choose **Insert** > **RapidTest Script Wizard**. Click **Next** in the wizard's Welcome screen to advance to the next screen.
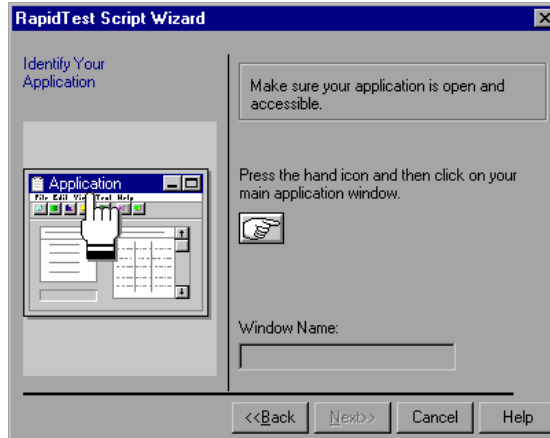
4 Point to the application you want to test.

Click the 👉 button and then click anywhere in the **Flight Reservation** application. The application's window name appears in the wizard's **Window Name** box. Click **Next**.

5 Make sure that all the check boxes are cleared.

For the purposes of this exercise, confirm that all the check boxes are cleared. You will use the wizard only to learn the GUI of the Flight Reservation application. Click **Next**.

6 Accept the default navigation controls.

Navigation controls tell WinRunner which GUI objects are used to open windows. The Flight Reservation application uses the default navigation controls (... and > >), so you do not need to define additional controls. Click **Next**.

7 Set the learning flow to Express.

The learning flow determines how WinRunner walks through your application. Two modes are available: **Express** and **Comprehensive**. Comprehensive mode enables you to customize how the wizard learns GUI object descriptions. First-time WinRunner users should use Express mode.

Click the **Learn** button. The wizard begins walking through the application, pulling down menus, opening windows, and learning object descriptions. This process takes a few minutes.

If a pop-up message notifies you that an interface element is disabled, click the **Continue** button in the message box.

If the wizard cannot close a window, it will ask you to show it how to close the window. Follow the directions on the screen.

8 Accept No in the Start Application screen.

You can choose to have WinRunner automatically open the Flight Reservation application each time you start WinRunner. Accept the default **No**. Click **Next**.

9 Save the GUI information and a startup script.

The wizard saves the GUI information it learned in a **GUI map file**.

The wizard also creates a startup script. This script runs automatically each time you start WinRunner. It contains a command which loads the GUI map file so that WinRunner will be ready to test your application.

Accept the default paths for the files or define different ones. Accept the default GUI map file name and path: **<WinRunner installation folder>\ dat\flight4a.gui.** Make sure that you have write permission for the selected folders. Click **Next**.

10 Click OK in the Congratulations screen.

The information WinRunner learned about the application is stored in a GUI map file.

3

Recording Tests

Choosing a Record Mode

By recording, you can quickly create automated test scripts. You work with your application as usual, clicking objects with the mouse and entering keyboard input. WinRunner records your operations and generates statements in TSL, Mercury's Test Script Language. These statements appear as a script in a WinRunner test window.

Before you begin recording a test, you should plan the main stages of the test and select the appropriate record mode. Two record modes are available: **Context Sensitive** and **Analog**.

Context Sensitive

Context Sensitive mode records your operations in terms of the GUI objects in your application. WinRunner identifies each object you click (such as a window, menu, list, or button), and the type of operation you perform (such as press, enable, move, or select).

For example, if you record a mouse click on the **OK** button in the Flight Reservation Login window, WinRunner records the following TSL statement in your test script:

button_press ("OK");

When you run the script, WinRunner reads the command, looks for the **OK** button, and presses it.

Analog

In Analog mode, WinRunner records the exact coordinates traveled by the mouse, as well as mouse clicks and keyboard input. For example, if you click the **OK** button in the Login window, WinRunner records statements that look like this:

When this statement is recorded...	it really means:
move_locator_track (1);	*mouse track*
mtype ("<T110><kLeft>-");	*left mouse button press*
mtype ("<kLeft>+");	*left mouse button release*

When you run the test, WinRunner retraces the recorded movements using absolute screen coordinates. If your application is located in a different position on the desktop, or the user interface has changed, WinRunner is not able to execute the test correctly.

Note: You should record in Analog mode only when exact mouse movements are an important part of your test, for example, when recreating a drawing.

When choosing a record mode, consider the following points:

Choose Context Sensitive if...	Choose Analog if...
The application contains GUI objects.	The application contains bitmap areas (such as a drawing area).
Exact mouse movements are not required.	Exact mouse movements are required.
You plan to reuse the test in different versions of the application.	

If you are testing an application that contains both GUI objects and bitmap areas, you can switch between modes as you record. This is discussed later in the lesson.

Recording a Context Sensitive Test

In this exercise, you will create a script that tests the process of opening an order in the Flight Reservation application. You will create the script by recording in Context Sensitive mode.

1 Start WinRunner and load the GUI map.

If WinRunner is not already open, choose **Start** > **Programs** > **WinRunner** > **WinRunner**. If the Welcome window is open, click the **New Test** button. Otherwise, choose **File** > **New**. A new test window opens.

If you are working in the Global GUI Map file mode, confirm that the GUI map is loaded. To do this, choose **Tools** > **GUI Map Editor**. In the GUI Map Editor, choose **View** > **GUI Files** and confirm that **flight4a.GUI** is contained in the **GUI File** list. If the GUI map is not loaded, in the GUI Map Editor choose **File** > **Open**, locate and select the GUI map, and click **Open**.

flight4a.exe

2 Start the Flight Reservation application and log in.

Choose **Start** > **Programs** > **WinRunner** > **Sample Applications** > **Flight 4A**. In the Login window, type your name and the password mercury, and click **OK**. The name you type must be at least four characters long. The Flight Reservation window opens.
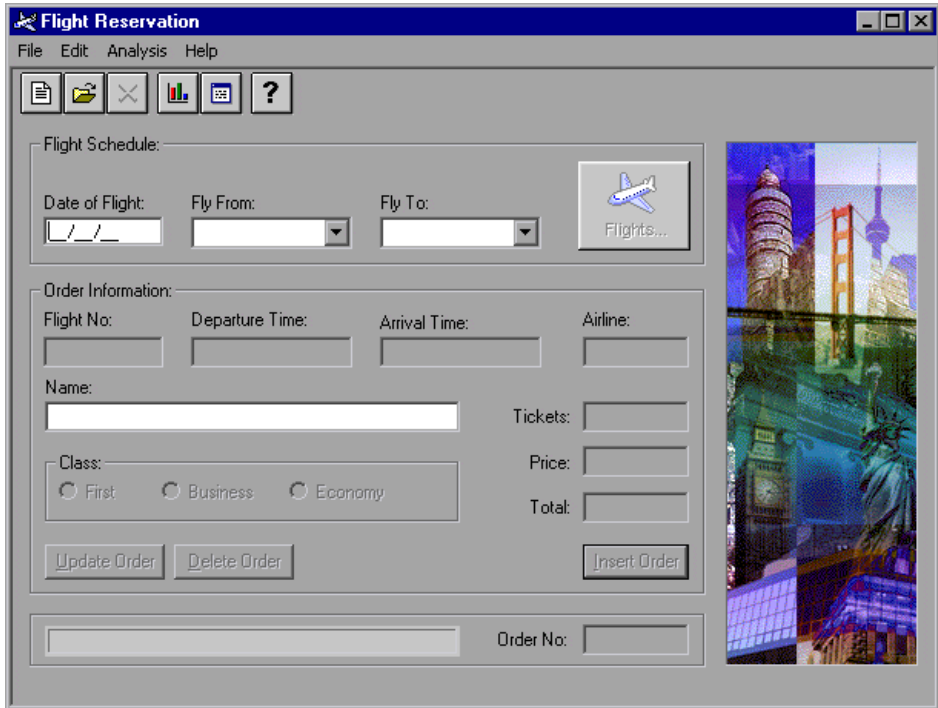
Position the Flight Reservation application and WinRunner so that they are both clearly visible on your desktop.

3 Start recording in Context Sensitive mode.
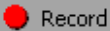
Record

In WinRunner, choose **Test** > **Record - Context Sensitive** or click the **Record** button in the toolbar. From this point on, WinRunner records all mouse clicks and keyboard input. Note that the text, REC, appears in blue below the recording icon. This indicates that you are recording in Context Sensitive mode. The status bar also informs you of your current recording mode.

4 Open order #3.

In the Flight Reservation application, choose **File** > **Open Order**. In the Open Order dialog box, select the **Order No.** check box. Type 3 in the adjacent box, and click **OK**.

Watch how WinRunner generates a test script in the test window as you work.

5 Stop recording.

In WinRunner, choose **Test** > **Stop Recording** or click the **Stop** button in the toolbar.

6 Save the test.

Choose **File** > **Save** or click the **Save** button in the toolbar. Save the test as **lesson3** in a convenient location on your hard drive.

Note that WinRunner saves the **lesson3** test in the file system as a folder, and not as an individual file. This folder contains the test script and the results that are generated when you run the test.

Understanding the Test Script

In the previous exercise, you recorded the process of opening a flight order in the Flight Reservation application. As you worked, WinRunner generated a test script similiar to the following:

```
# Flight Reservation
    set_window ("Flight Reservation", 3);
    menu_select_item ("File;Open Order...");


# Open Order
    set_window ("Open Order", 1);
    button_set ("Order No.", ON);
    edit_set ("Edit_1", "3");
    button_press ("OK");
```

As you can see, the recorded TSL statements describe the objects you selected and the actions you performed. For example, when you selected a menu item, WinRunner generated a **menu_select_item** statement.

The following points will help you understand your test script:

➤ When you click an object, WinRunner assigns the object a **logical name**, which is usually the object's text label. The logical name makes it easy for you to read the test script. For example, when you selected the **Order No.** check box, WinRunner recorded the following statement:

button_set ("Order No.", ON);

Order No. is the object's logical name.

➤ By default, WinRunner automatically adds a comment line each time you begin working in a new window so that your script is easier to read. For example, when you clicked on the Flight Reservation window, WinRunner generated the following comment line:

 # Flight Reservation

➤ WinRunner generates a **set_window** statement each time you begin working in a new window. The statements following a **set_window** statement perform operations on objects within that window. For example, when you opened the Open Order dialog box, WinRunner generated the following statement:

set_window ("Open Order", 1);

➤ When you enter keyboard input, WinRunner generates a **type**, an **obj_type**, or an **edit_set** statement in the test script. For example, when you typed 3 in the **Order Number** box, WinRunner generated the following statement:

edit_set ("Edit", "3");

For more information about the different ways in which WinRunner records keyboard input, choose **Help** > **TSL Online Reference**.

Recording in Analog Mode

In this exercise, you will test the process of sending a fax. You will start recording in Context Sensitive mode, switch to Analog mode to add a signature to the fax, and then switch back to Context Sensitive mode.

1 In the lesson3 test, place the cursor below the last line of the script.

You will add the new test segment to the **lesson3** test. If the test is not already open, choose **File** > **Open** and select the test. In the **lesson3** test window, place the cursor below the last line of the test.

2 Start Recording in Context Sensitive mode.
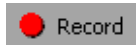
🔴 Record

Choose **Test** > **Record - Context Sensitive** or click the **Record** button in the toolbar.

3 Open the Fax Order form and fill in a fax number.

In the Flight Reservation application, choose **File** > **Fax Order**. In the Fax Number box, type 4155551234.

Fax Order No. 3				_ □ ✕

Fax

Name:		Order:	Flight:	Date:
Mary Parker		3	4194	07/13/99

From:	Departure:	To:	Arrival:
Los Angeles	08:00 AM	Portland	10:00 AM

Class:	# Tickets:	Ticket Price:	Total:
Economy	5	160.60	803.00

Fax Number: Agent Signature:

(415)555-1234

☑ Send Signature with order

Preview Fax	Send	Cancel	Clear Signature

4 Select the Send Signature with order check box.

5 Sign the fax in Context Sensitive mode.

Use the mouse to sign your name in the **Agent Signature** box.

Watch how WinRunner records your signature.

6 Clear the signature.

Click the **Clear Signature** button.

7 Move the Fax Order window to a different position on your desktop.

Before switching to Analog mode, reposition the window in which you are working.

8 Sign the fax again in Analog mode.

Press F2 on your keyboard or click the **Record** button again to switch to Analog mode. Note that the text, **Rec** appears in red above the recording button. This indicates that you are recording in Analog mode. Sign your name in the **Agent Signature** box.

Watch how WinRunner records your signature.

9 Switch back to Context Sensitive mode and send the fax.

Press F2 or click the **Record** button to switch back to Context Sensitive mode. Click **Send**. The application simulates the process of sending the fax.

10 Stop Recording.

Choose **Test** > **Stop Recording** or click the **Stop** button.

11 Save the test.

Choose **File** > **Save** or click the **Save** button.

12 If you are working in the Global GUI Map file mode, save the new objects to the GUI map.

When you ran the RapidTest Script wizard in the previous lesson, it learned all the windows and objects it was able to access. The fax order dialog box, however, can be open only when an order has already been opened, as you did in step 4 of "Recording a Context Sensitive Test" on page 33. Therefore, when you opened the fax order dialog box in step 3 on page 35, WinRunner added the new window, and the objects you recorded within that window, to the temporary GUI map. The temporary GUI map is discarded whenever you close WinRunner, so it is important to save new windows and objects to the GUI map file that your test uses.

Choose **Tools** > **GUI Map Editor**. Choose **View** > **GUI Files**. Note that the Fax Order No. 3 window is displayed in the **L0 <Temporary>** GUI map file. Choose **File** > **Save**. The New Windows dialog box opens. Confirm that the **flight4a.GUI** file is displayed in the **Loaded GUI Files** box. Click **OK**. The Fax Order No. 3 window and all objects under that window are moved from the temporary GUI map to the **flight4a.GUI** map file. Choose **File** > **Exit** to close the GUI Map Editor.

Note: If you are working in the GUI Map File per Test mode, the new objects are added to and saved with the file that is automatically saved when you save your test. You should not manually save objects to your GUI map.

Running the Test

You are now ready to run your recorded test script and to analyze the test results. WinRunner provides three modes for running tests. You select a mode from the toolbar.

➤ Use **Verify run mode** when running a test to check the behavior of your application, and when you want to save the test results. This is the mode that you will use in this lesson.

➤ Use **Debug run mode** when you want to check that the test script runs smoothly without errors in syntax. See Lesson 7 for more information.

➤ Use **Update run mode** when you want to create new expected results for a GUI checkpoint or bitmap checkpoint. See Lessons 5 and 6 for more information.

To run the test:

1 Check that WinRunner and the main window of the Flight Reservation application are open on your desktop.

2 Make sure that the lesson3 test window is active in WinRunner.

Click the title bar of the **lesson3** test window. If the test is not already open, choose **File** > **Open** and select the test.

3 Make sure the main window of the Flight Reservation application is active.

If any dialog boxes are open, close them.

4 Make sure that Verify run mode is selected in the toolbar.

5 Choose Run from Top.
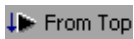
Choose **Test** > **Run from Top** or click the **From Top** button. The Run Test dialog box opens.
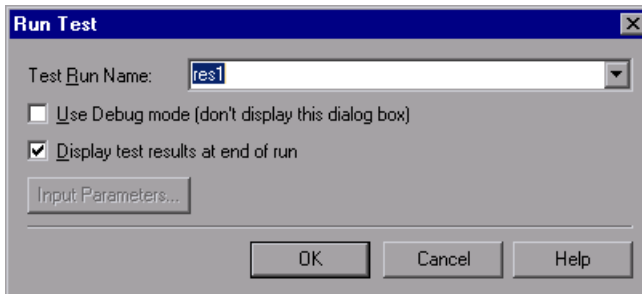
6 Choose a Test Run name.

This step defines the name of the folder in which WinRunner will store the results of the test. Accept the default folder name **res1**. The results folder will be stored within the test's folder.

Note the **Display test results at end of run** check box at the bottom of the dialog box. When this check box is selected, WinRunner automatically displays the test results when the test run is completed. Make sure that this check box is selected.

7 Run the test.

Click **OK** in the Run Test dialog box. WinRunner immediately begins running the test.

Watch how WinRunner opens each window in the Flight Reservation application.

8 Review the test results.

When the test run is completed, the test results automatically appear in the WinRunner Test Results window. See the next section to learn how to analyze the test results.

Analyzing Test Results

Once a test run is completed, you can immediately review the test results in the WinRunner Test Results window.

Understanding WinRunner Test Result Views

WinRunner provides two styles of test results viewers:

➤ **WinRunner report view.** A windows-style viewer. This is the viewer that was available in previous versions of WinRunner.

➤ **Unified report view.** An HTML-style viewer. This viewer is the same as that used in QuickTest Professional.

By default, WinRunner opens the Test Results window using the WinRunner report view, but also creates the data necessary for displaying the results in the unified report view.

This tutorial assumes that you are using the WinRunner report view. If you are using the unified report view, the Test Results window will look significantly different than the images displayed in this tutorial.

To confirm that you are using the WinRunner report view:

1 Choose **Tools** > **General Options** and select the **Run** category.

2 Select **WinRunner report view** and click **OK**.

Viewing Test Results

The following steps show you how to open and view the test results window. In later lessons, you will also analyze the information shown in the test results to identify problems with your test, or with the application you are testing.

1 Make sure that the WinRunner Test Results window is open and displays the test results.

If the WinRunner Test Results window is not currently open, first click the test window to activate it, and then choose **Tools** > **Test Results** or click the **Test Results** button.

1 Displays the name of the current test.

2 Shows the current test results name.

3 Shows whether a test run passed or failed.

4 Includes general information about the test run such as date, operator name, and total run time. To view these details, double click the Information icon.

5 The test log section lists the major events that occurred during the test run. It also lists the test script line at which each event occurred.
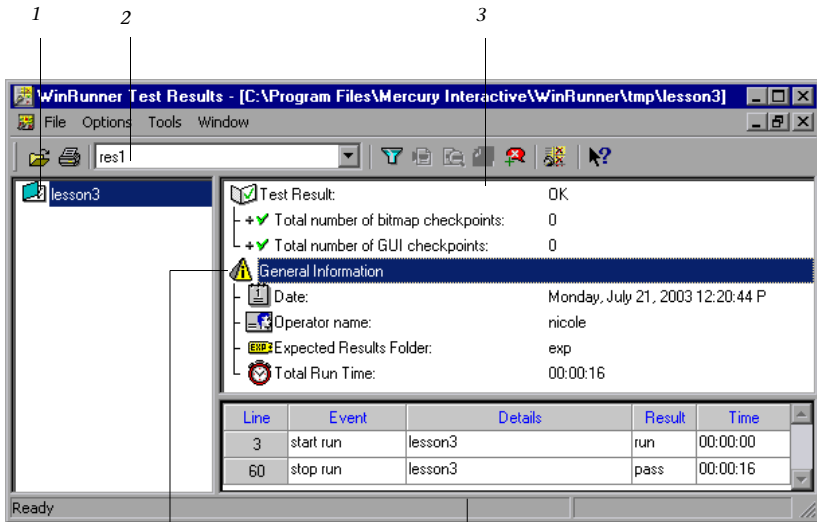
2 Review the results.

3 Close the Test Results window.

Choose **File** > **Exit** in the WinRunner Test Results window.

4 **Close the test.**

Choose **File** > **Close** in the Winrunner main window.

5 **Close the Flight Reservation application.**

Choose **File** > **Exit**.

Recording Tips

➤ Before starting to record, close applications that are not required for the test.

➤ Create the test so that it ends where it started. For example, if the test opens an application, make sure that it also closes the application at the end of the test run. This ensures that WinRunner is prepared to run repeated executions of the same test.

➤ When recording in Analog mode, avoid holding down the mouse button if this results in a repeated action. For example, do not hold down the mouse button to scroll a window. Instead, scroll by clicking the scrollbar arrow repeatedly. This enables WinRunner to accurately run the test.

➤ Before switching from Context Sensitive mode to Analog mode during a recording session, always move the current window to a new position on the desktop. This ensures that when you run the test, the mouse pointer will reach the correct areas of the window during the Analog portion of the test.

➤ When recording, if you click a non-standard GUI object, WinRunner generates a generic **obj_mouse_click** statement in the test script. For example, if you click a graph object, it records something similar to:

obj_mouse_click (GS_Drawing, 8, 53, LEFT);

If your application contains a non-standard GUI object that behaves like a standard GUI object, you can map this object to a standard object class so that WinRunner will record more intuitive statements in the test script. For more information refer to the "Configuring the GUI Map" chapter in the *Mercury WinRunner User's Guide*.

➤ When working in the Global GUI Map file mode, if you click an object whose description was not previously learned, WinRunner learns a description of the object and adds it to a temporary GUI map file. For more information, refer to the "Working in the Global GUI Map File Mode" chapter in the *Mercury WinRunner User's Guide*.

➤ To easily switch between Context Sensitive and Analog modes, press F2.

➤ If you are working in the Global GUI Map file mode, always check whether new windows or objects have been added to the temporary GUI map before you close WinRunner. If new objects have been added, save them to the appropriate GUI map file for your test.

4

Synchronizing Tests

In this lesson, you will learn about:	On page:
Synchronizing Your Test and Application	43
Creating a Test	44
Changing the Synchronization Setting	46
Identifying a Synchronization Problem	47
Synchronizing the Test	48
Running the Synchronized Test	51

Synchronizing Your Test and Application

When you run tests, your application may not always respond to input with the same speed. For example, it might take a few seconds:

➤ to retrieve information from a database

➤ for a window to pop up

➤ for a progress bar to reach 100%

➤ for a status message to appear

WinRunner waits a set time interval for an application to respond to input. The default wait interval is up to 10 seconds. If the application responds slowly during a test run, WinRunner's default wait time may not be sufficient, and the test run may unexpectedly fail.

If you discover a synchronization problem between the test and your application, you can either:

➤ Increase the default time that WinRunner waits. To do so, choose **Tools > General Options** to open the General Options dialog box. Select the **Run** category and the **Settings** sub-category. Change the value of **Timeout for checkpoints and CS statements** option. This method affects all your tests and slows down many other Context Sensitive operations.

➤ Insert a *synchronization point* into the test script at the exact point where the problem occurs. A synchronization point tells WinRunner to pause the test run to wait for a specified response in the application. This is the recommended method for synchronizing a test with your application.

In the following exercises, you will:

➤ create a test that opens a new order in the Flight Reservation application and inserts the order into the database

➤ change the synchronization settings

➤ identify a synchronization problem

➤ synchronize the test

➤ run the synchronized test

Creating a Test

In this first exercise, you will create a test that opens a new order in the Flight Reservation application and inserts the order into a database.

1 Start WinRunner and load the GUI map.

If WinRunner is not already open, choose **Start > Programs** > **WinRunner** > **WinRunner**. If the Welcome window is open, click the **New Test** button. Otherwise, choose **File** > **New**. A new test window opens.

If you are working in the Global GUI Map file mode, confirm that the GUI map is loaded. To do this, choose **Tools** > **GUI Map Editor**. In the GUI Map Editor, choose **View** > **GUI Files** and confirm that **flight4a.GUI** is contained in the **GUI File** list.

2 Start the Flight Reservation application and log in.

Choose **Start** > **Programs** > **WinRunner** > **Sample Applications** > **Flight 4A**. In the Login window, type your name and the password mercury, and click **OK**. Reposition the Flight Reservation application and WinRunner so that they are both clearly visible on your desktop.

3 Start recording in Context Sensitive mode.

Choose **Test** > **Record - Context Sensitive** or click the **Record** button in the toolbar. WinRunner will start recording the test.

4 Create a new order.

Choose **File** > **New Order** in the Flight Reservation application.

5 Fill in flight and passenger information.

1 Enter a date in MM/DD/YY format.

2 Select Los Angeles.

3 Select San Francisco.

4 Click the Flights button, then double-click a flight.

5 Enter your name.

6 Select First Class.

45

Note: Make sure that the date you enter is far enough in the future to enable you to complete the tutorial before that date. Since the Flight Reservation application enables you to reserve flights only for future dates, the test scripts in this tutorial will not work if you try to run them after the date has passed.

---

6 Insert the order into the database.

Click the **Insert Order** button. When the insertion is complete, the status bar displays **Insert Done**.

7 Delete the order.

Click the **Delete Order** button and click **Yes** in the message window to confirm the deletion.
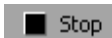
8 Stop recording.

Choose **Test** > **Stop Recording** or click the **Stop** button.

9 Save the test.

Choose **File** > **Save**. Save the test as **lesson4** in a convenient location on your hard drive.

Changing the Synchronization Setting

The default interval that WinRunner waits for an application to respond to input is 10 seconds. In the next exercise, you will identify a synchronization problem and add a synchronization point to solve it. To run the test you have just recorded with a synchronization problem, you need to change the default synchronization setting.

1 Open the General Options dialog box.

Choose **Tools** > **General Options**.

2 Select the Run category and the Settings sub-category.
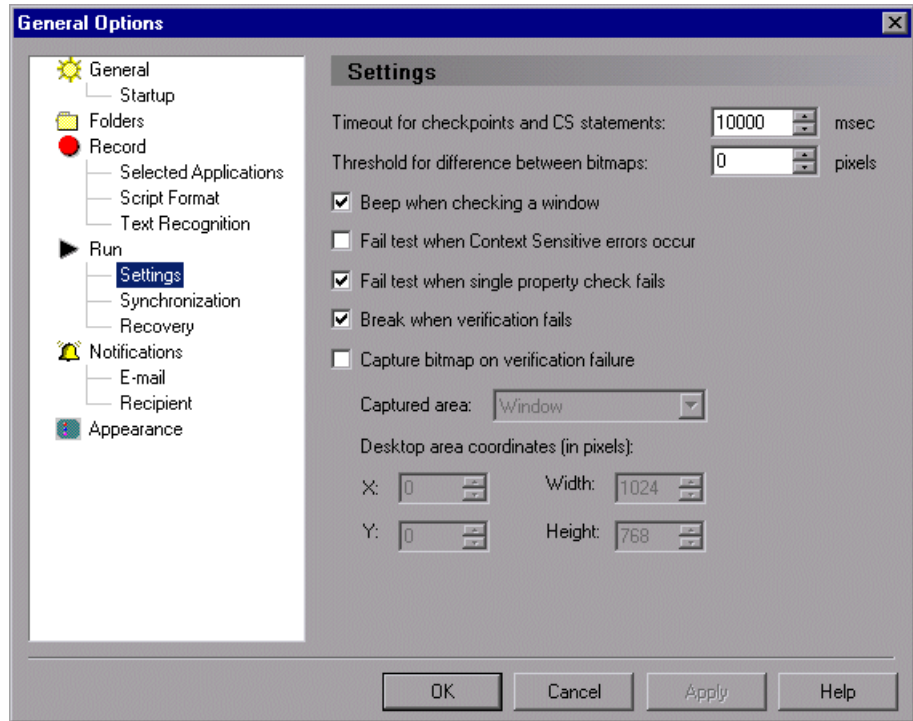
3 Change the value to 1000 milliseconds (1 second).

In the **Timeout for checkpoints and CS statements** box, change the value to **1000**.

4 Click OK to close the dialog box.

Identifying a Synchronization Problem

You are now ready to run the **lesson4** test. As the test runs, look for a synchronization problem.

1 Make sure that the lesson4 test window is active in WinRunner.

Click the title bar of the **lesson4** test window.

 2 Choose Run from Top.

Choose **Test** > **Run from Top** or click the **From Top** button. The Run Test dialog box opens. Accept the default test run name **res1**. Make sure that the **Display test results at end of run** check box is selected.

3 Run the test.

Click **OK** in the Run Test dialog box. WinRunner starts running the test. *Watch what happens when WinRunner attempts to click the Delete button.*

4 Pause the test run.

Click **Pause** in the WinRunner message window. WinRunner fails to click the **Delete Order** button because the button is still disabled. This error occurred because WinRunner did not wait until the Insert Order operation was completed. Note that the execution arrow has paused opposite the command to click the **Delete Order** button.

```
        set_window ("Flight Reservation", 7);
  ⇨     button_press ("Delete Order");

# Flight Reservations
        set_window ("Flight Reservations", 3);
        button_press ("Yes");
```

Synchronizing the Test

In this exercise, you will insert a synchronization point into the **lesson4** test script. The synchronization point will capture a bitmap image of the Insert Done message in the status bar. Later on when you run the test, WinRunner will wait for the Insert Done message to appear before it attempts to click the **Delete Order** button.

1 Make sure that the Flight Reservation window is visible on your screen.

Reposition the Flight Reservation application and WinRunner so that they are both clearly visible on your desktop.

2 Make sure that the lesson4 test window is active in WinRunner.

Click the title bar of the lesson4 test window.

3 Place the cursor at the point where you want to synchronize the test.

Add a blank line below the button_press ("Insert Order"); statement. Place the cursor at the beginning of the blank line.

4 Synchronize the test so that it waits for the Insert Done message to appear in the status bar.

Choose **Insert** > **Synchronization Point** > **For Object/Window Bitmap** or click the **Synchronization Point for Object/Window Bitmap** button on the User toolbar.

Use the ⟨⟨ᵐ⟩ pointer to click the message Insert Done in the Flight Reservation window. WinRunner automatically inserts an **obj_wait_bitmap** synchronization point into the test script. This statement instructs WinRunner to wait 1 second for the Insert Done message to appear in the status bar.

5 Manually change the 1 second wait in the script to a 10 second wait.

The one-second wait that was inserted in the previous step is not long enough. Find the statement:

obj_wait_bitmap("Insert Done...", "Img1", 1);

Change the **1** at the end of the statement to **10**, to indicate a 10 second wait.

6 Save the test.

Choose **File** > **Save** or click the **Save** button.

7 If you are working in the Global GUI Map file mode, add the new objects from the local GUI map to the flight4a.GUI map.

During this test you performed a synchronization point on a new object in the Flight Reservation window (the **Insert Done** bitmap). You should add this object to your GUI map.

To add a new object to your GUI map for a window that already exists, choose **Tools** > **GUI Map Editor**. Choose **View** > **GUI Files**. Note that the new object is displayed in the **L0 <Temporary>** GUI map file. (If it is not displayed, choose **View** > **Expand Objects Tree**.) Choose **File** > **Save**.

A WinRunner message informs you that the new objects from the existing window will be added to the **flight4a.GUI** map that contains this window. Click **Yes**.

8 Save the modified flight4a.GUI map.

In the GUI file box, select **L1 flight4a.GUI**. Note that the GUI map name has an asterisk next to it. This indicates that the file has been changed, (you added the **Insert Done** bitmap) and has not been saved. Choose **File** > **Save** to save the GUI map.

9 Close the GUI Map Editor.

Choose **File** > **Exit** to close the GUI Map Editor.

FYI:

Synchronization points appear as **obj_wait_bitmap** or **win_wait_bitmap** statements in the test script. For example:

obj_wait_bitmap("Insert Done...", "Img1", 10);

Insert Done... is the object's logical name.

lmg1 is the file containing a captured image of the object.

10 is the time (in seconds) that WinRunner waits for the image to appear in the application. This time is added to the default time defined by the *timeout-msec* testing option. (In the above exercise, WinRunner waits a total of 11 seconds).

Running the Synchronized Test

In this exercise, you will run the synchronized test script and examine the test results.

1 Confirm that the lesson4 test window is active in WinRunner.

Click the title bar of the lesson4 test window.

2 Confirm that Verify run mode is selected in the Test toolbar.
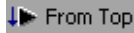
Verify run mode stays in effect until you choose a different mode.

⬇▶ From Top   **3 Choose Run from Top.**

Choose **Test** > **Run from Top** or click the **From Top** button. The Run Test dialog box opens. Accept the default name **res2**. Make sure that the **Display test results at end of run** check box is selected.

4 Run the test.

Click **OK** in the Run Test dialog box. WinRunner starts running the test from the first line in the script.

Watch how WinRunner waits for the Insert Done message to appear in the status bar.

5 Review the results.

When the test run is completed, the test results appear in the WinRunner Test Results window. Note that a **wait for bitmap** event appears in green in the test log section. This indicates that synchronization was performed successfully. You can double-click this event to see a bitmap image of the status bar displaying the Insert Done message.

Img1.bmp (expected)   _ □ ×
Insert Done...

6 Close the Test Results window.

Choose **File** > **Exit**.

7 Close the lesson4 test.

Choose **File** > **Close** in WinRunner.

8 Close the Flight Reservation application.

Choose **File** > **Exit**.

9 Change the timeout value back to 10000 milliseconds (10 seconds).

Choose **Tools** > **General Options** to open the General Options dialog box. Select the **Run** category and the **Settings** sub-category. In the **Timeout for checkpoints and CS statements** box, change the current value to **10000**. Click **OK** to close the dialog box.

To learn about additional synchronization methods, read "Synchronizing the Test Run" in the *Mercury WinRunner User's Guide*.

5
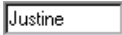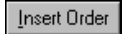
Checking GUI Objects

In this lesson, you will learn about:	On page:
Checking GUI Objects	53
Adding GUI Checkpoints to a Test Script	55
Running the Test	59
Running the Test on a New Version	61
GUI Checkpoint Tips	63

Checking GUI Objects

When working with an application, you can determine whether it is functioning properly according to the behavior of its GUI objects. If a GUI object does not respond to input as expected, a defect probably exists somewhere in the application's code.

You check GUI objects by creating GUI checkpoints. A GUI checkpoint examines the behavior of an object's properties. For example, you can check:

➤ the content of an edit box

➤ whether a radio button is selected or cleared

➤ whether a push button is enabled or disabled

To create a GUI checkpoint for a single object, you first point to it in your application.

➤ If you single-click the object, a checklist with the default checks for the object you selected is inserted into your test script. A *checklist* contains information about the GUI object and the selected properties to check.

➤ If you double-click the object, the Check GUI dialog box opens and displays the object you selected. Select the properties you want to check, and click **OK** to insert a checklist for the object into your test script.

This dialog box opens when you double-click the Insert Order push button.

Select the properties you want to check. The default check for a push button is Enabled.
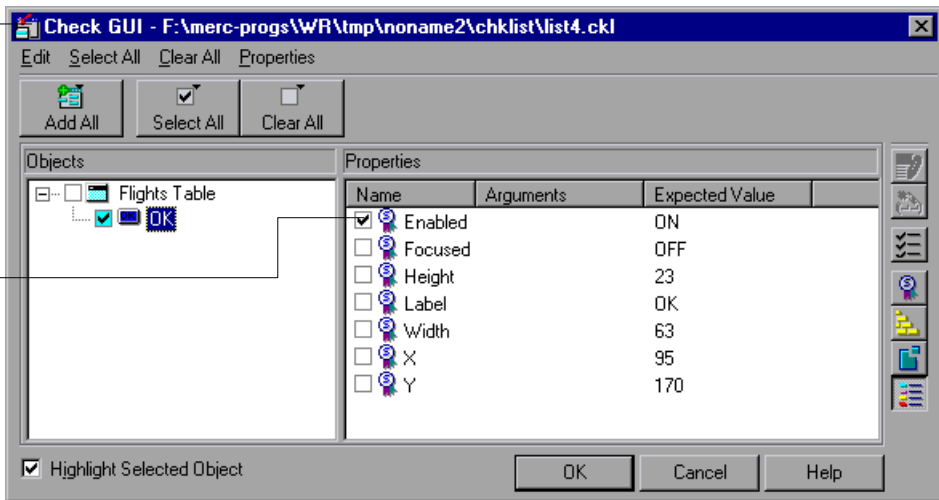
Whether you choose to check an object's default properties or you specify the properties of an object you want to check, WinRunner captures the current values of those properties and saves this information as *expected* results. It then inserts an **obj_check_gui** statement into the test script if you are checking an object, or a **win_check_gui** statement if you are checking a window.

When you run this test on a new version of the application, WinRunner compares the object's expected behavior with its *actual* behavior in the application.

Adding GUI Checkpoints to a Test Script

In this exercise, you will check that objects in the Flight Reservation Open Order dialog box function properly when you open an existing order.

1 Start WinRunner and load the GUI map.

If WinRunner is not already open, choose **Start** > **Programs** > **WinRunner** > **WinRunner**. If the Welcome window is open, click the **New Test** button. Otherwise, choose **File** > **New**. A new test window opens.

If you are working in the Global GUI Map file mode, confirm that the GUI map is loaded. To do this, choose **Tools** > **GUI Map Editor**. In the GUI Map Editor, choose **View** > **GUI Files** and confirm that **flight4a.GUI** is contained in the **GUI File** list.

2 Start the Flight Reservation application and log in.

flight4a.exe

Choose **Start** > **Programs** > **WinRunner** > **Sample Applications** > **Flight 4A**. In the Login window, type your name and the password mercury, and click **OK**. Reposition the Flight Reservation application and WinRunner so that they are both clearly visible on your desktop.

3 Start recording in Context Sensitive mode.

● Record

Choose **Test** > **Record - Context Sensitive** or click the **Record** button in the toolbar.

4 Open the Open Order dialog box.

In the Flight Reservation application, choose **File** > **Open Order**.

5 Create a GUI checkpoint for the Order No. check box.

In the main WinRunner window, choose **Insert** > **GUI Checkpoint** > **For Object/Window**, or click the **GUI Checkpoint for Object/Window** button on the User toolbar.

Note: By default, the User toolbar is hidden in new installations. To open the User toolbar choose **View > User Toolbar**. For more information about the User toolbar, see "Exploring the WinRunner Window" on page 13.

Use the ⏚ pointer to *double-click* the **Order No.** check box. The Check GUI dialog box opens and displays the available checks. Note that this dialog box does not open if you only *single-clicked* the **Order No.** check box. Accept the default check, **State**. This check captures the current state (OFF) of the check box and stores it as expected results.
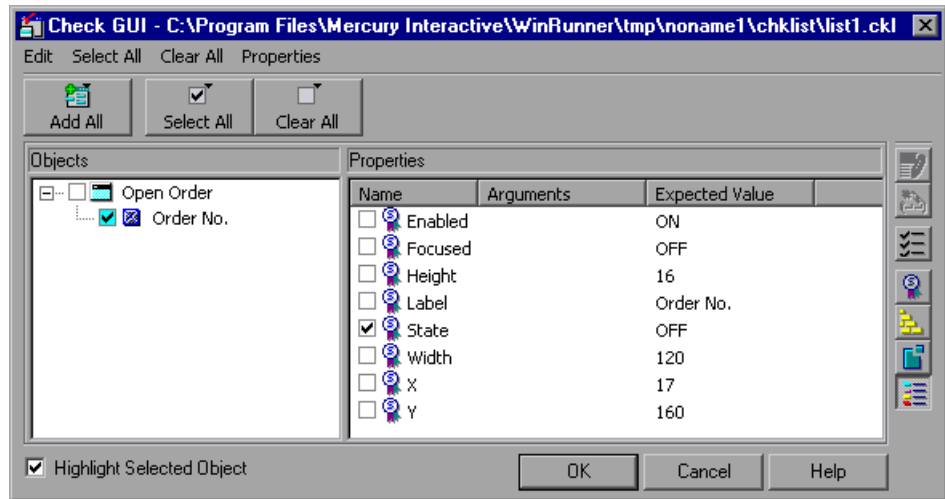
Click **OK** in the Check GUI dialog box to insert the checkpoint into the test script. The checkpoint appears as an **obj_check_gui** statement.

6 Enter 4 as the Order No.

In the Open Order window, select the **Order No.** check box and type in 4 in the **Order No.** text box.

7 Create another GUI checkpoint for the Order No. check box.

Choose **Insert** > **GUI Checkpoint** > **For Object/Window** or click the **GUI Checkpoint for Object/Window** button on the User toolbar.

Use the ⏚ pointer to *single-click* the **Order No.** check box. WinRunner immediately inserts a checkpoint into the test script (an **obj_check_gui** statement) that checks the default check **State**. (Use the single-click option when you want to use only the default check for an object.) This check captures the current state (ON) of the check box and stores it as expected results.

Note: To see the objects and properties in a checkpoint, you must open the Check GUI dialog box. For additional information on this and other GUI checkpoint dialog boxes, refer to the "Checking GUI Objects" chapter in the *Mercury WinRunner User's Guide*.

---

8 Create a GUI checkpoint for the Customer Name check box.

Choose **Insert** > **GUI Checkpoint** > **For Object/Window** or click the **GUI Checkpoint for Object/Window** button on the User toolbar.

Use the 🖑 pointer to double-click the **Customer Name** check box. The Check GUI dialog box opens and displays the available checks. Accept the default check **State** and select **Enabled** as an additional check. The **State** check captures the current state (OFF) of the check box; the **Enabled** check captures the current condition (OFF) of the check box.
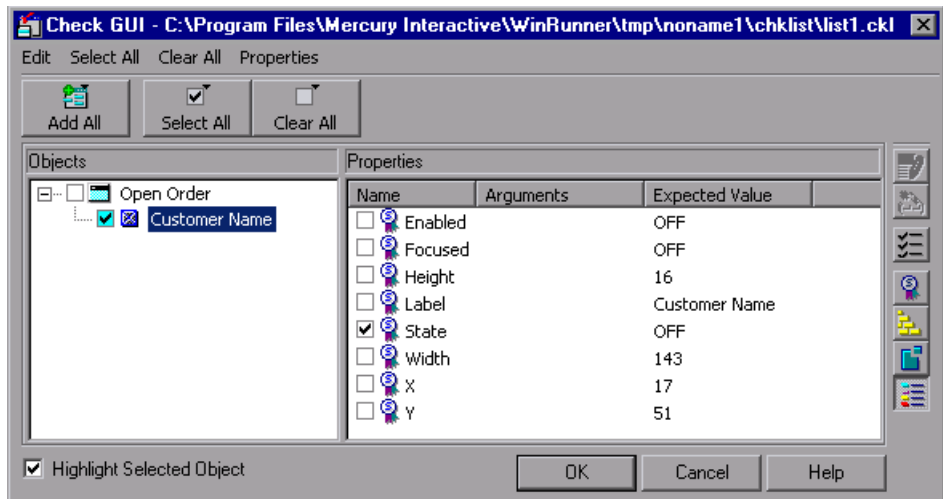
Click **OK** in the Check GUI dialog box to insert the checkpoint into the test script. The checkpoint appears as an **obj_check_gui** statement.

9 Click OK in the Open Order dialog box to open the order.

10 Stop recording.

■ Stop

Choose **Test** > **Stop Recording** or click the **Stop** button.

🖫 **11 Save the test.**

Choose **File** > **Save** or click the **Save** button. Save the test as **lesson5** in a convenient location on your hard drive.

FYI:

GUI checkpoints appear as **obj_check_gui** or **win_check_gui** statements in the test script. For example:

obj_check_gui("Order No.", "list1.ckl", "gui1", 1);

Order No. is the object's logical name.

list1.ckl is the checklist containing the checks you selected.

gui1 is the file containing the captured GUI data.

1 is the time (in seconds) needed to perform the check. This time is added to the value of the *timeout_msec* test option. See Lesson 4 "Synchronizing Tests" on page 43 for more information.

Running the Test

You will now run the **lesson5** test to verify that the test runs smoothly.

1 Make sure that the Flight Reservation application is open on your desktop.

▶ Verify ▾   **2 In WinRunner, check that Verify run mode is selected in the Test toolbar.**

⬇▶ From Top   **3 Choose Run from Top.**

Choose **Test** > **Run from Top**, or click the **From Top** button. The Run Test dialog box opens. Accept the default test run name **res1**. Make sure that the **Display test results at end of run** check box is selected.

4 Run the test.

Click **OK** in the Run Test dialog box.

5 Review the results.

When the test run is completed, the test results appear in the WinRunner Test Results window. In the test log section, all **end GUI checkpoint** events should appear in green (indicating success).

Double-click the last end GUI checkpoint event to view detailed results of that GUI checkpoint. The GUI Checkpoint Results dialog box opens. Select **Customer Name** to display the dialog box as follows:

Names the window containing the objects

Indicates whether an object passed or failed

Lists the objects in the checkpoint

Indicates whether a property check passed or failed

Lists the property checks performed

Lists the specified arguments

Lists expected results

Lists actual results

Note: You can specify the arguments for a check on selected properties. For more information, refer to the "Checking GUI Objects" chapter in the *Mercury WinRunner User's Guide*.

---

6 **Close the test results.**

Click **OK** to close the GUI Checkpoint Results dialog box. Then choose **File** > **Exit** to close the Test Results window.

7 **Close the Flight Reservation application.**

Choose **File** > **Exit**.

Running the Test on a New Version

In this exercise, you will run the **lesson5** test on a new version of the Flight Reservation application to check the behavior of its GUI objects.

1 **Open version 4B of the Flight Reservation application.**

flight4b.exe

Choose **Start** > **Programs** > **WinRunner** > **Sample Applications** > **Flight 4B**. In the Login window, type your name and the password **mercury**, and click **OK**. Position the Flight Reservation application and WinRunner so that they are both clearly visible on your desktop.

2 **Make sure that lesson5 is the active test.**

Click in the **lesson5** test window in WinRunner.

3 **Check that Verify run mode is selected in the toolbar.**

4 **Choose Run from Top.**

Choose **Test** > **Run from Top**, or click the **From Top** button. The Run Test dialog box opens. Accept the default test run name **res2**. Make sure that the **Display test results at end of run** check box is selected.

5 **Run the test.**

Click **OK**. The test run begins. This process might take a while.

If a mismatch is detected at a GUI checkpoint, click **Continue** in the message window.

6 Review the results.

When the test run is completed, the test results appear in the WinRunner Test Results window. In the test log section, one **end GUI checkpoint** statement appears in red and its **Result** box lists **mismatch**. This indicates that one or more of the checks performed on the object failed.

Double-click the red **end GUI checkpoint** event to view detailed results of the failed check. The GUI Checkpoint Results dialog box opens. Select Customer Name to display the dialog box as follows:

The check on the Customer Name check box failed.

The check on the Enabled property of the Customer Name check box failed.

The expected result is **OFF**.

The actual result is **ON**.

7 Close the Test Results window.

Click **OK** in the GUI Checkpoint Results dialog box and then choose **File** > **Exit** to close the Test Results window.

8 Close the lesson5 test.

Choose **File** > **Close**.

9 Close version 4B of the Flight Reservation application.

Choose **File** > **Exit**.

GUI Checkpoint Tips

➤ You can create a single GUI checkpoint in your test that checks several or all objects in a window. Choose **Insert** > **GUI Checkpoint** > **For Multiple Objects.** (This menu command is only available when a test is open.) The Create GUI Checkpoint dialog box opens, which enables you to add objects to the GUI checkpoint and to specify the checks you want to perform on those objects. When you finish creating the checkpoint, WinRunner inserts a **win_check_gui** statement into the test, which includes a checklist for the selected objects.

➤ For overnight test runs, you can instruct WinRunner not to display a message when a GUI mismatch is detected. Choose **Tools** > **General Options**. In the General Options dialog box, select the **Run** category and the **Settings** sub-category, and clear the **Break when verification fails** check box. This enables the test to run without interruption.

For more information on setting test run options, refer to the "Setting Global Testing Options" and "Setting Testing Options from a Test Script" chapters in the *Mercury WinRunner User's Guide*.
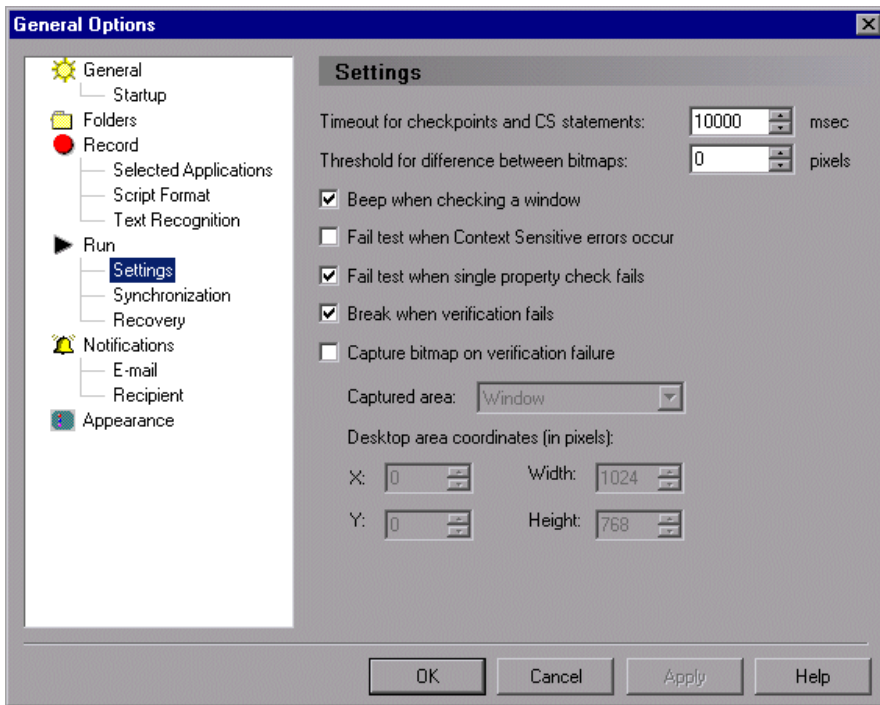
➤ If you want to create new expected results for a GUI checkpoint, run the test in Update run mode. WinRunner overwrites the existing expected GUI data with new data captured during the Update run.

For more information on GUI checkpoints, refer to the "Checking GUI Objects" chapter in the *Mercury WinRunner User's Guide*.

6

Checking Bitmaps

Checking a Bitmap

If your application contains bitmap areas, such as drawings or graphs, you can check these areas using a bitmap checkpoint. A bitmap checkpoint compares captured bitmap images pixel by pixel.

To create a bitmap checkpoint, you indicate an area, window, or object that you want to check. For example:

WinRunner captures a bitmap image and saves it as *expected* results. It then inserts an **obj_check_bitmap** statement into the test script if it captures an object, or a **win_check_bitmap** statement if it captures an area or window.

When you run the test on a new version of the application, WinRunner compares the expected bitmap with the actual bitmap in the application. If any differences are detected, you can view a picture of the differences from the Test Results window.

Adding Bitmap Checkpoints to a Test Script

In this exercise, you will test the Agent Signature box in the Fax Order dialog box. You will use a bitmap checkpoint to check that you can sign your name in the box. Then you will use another bitmap checkpoint to check that the box clears when you click the **Clear Signature** button.

1 Start WinRunner, open a new test and load the GUI map.

If WinRunner is not already open, choose **Start** > **Programs** > **WinRunner** > **WinRunner**. If the Welcome window is open, click the **New Test** button. Otherwise, choose **File** > **New**. A new test window opens.

If you are working in the Global GUI Map file mode, confirm that the GUI map is loaded. To do this, choose **Tools** > **GUI Map Editor**. In the GUI Map Editor, choose **View** > **GUI Files** and confirm that **flight4a.GUI** is contained in the **GUI File** list.

2 Start the Flight Reservation application and log in.

flight4a.exe

Choose **Start** > **Programs** > **WinRunner** > **Sample Applications** > **Flight 4A**. In the Login window, type your name and the password mercury, and click **OK**. Reposition the Flight Reservation application and WinRunner so that they are both clearly visible on your desktop.

3 Start recording in Context Sensitive mode.

Choose **Test** > **Record - Context Sensitive** or click the **Record** button in the toolbar.

4 Open order #6.

In the Flight Reservation application, choose **File** > **Open Order**. In the Open Order dialog box, select the **Order No.** check box and type 6 in the adjacent box. Click **OK** to open the order.

5 Open the Fax Order dialog box.

Choose **File** > **Fax Order**.

6 Enter a 10-digit fax number in the Fax Number box.

You do not need to type in parentheses or dashes.

7 Move the Fax Order dialog box.

Position the dialog box so that it least obscures the Flight Reservation window.

8 Switch to Analog mode.

Press F2 on your keyboard or click the **Record** button to switch to Analog mode.

9 Sign your name in the Agent Signature box.

10 Switch back to Context Sensitive mode.

Press F2 on your keyboard or click the **Record** button to switch back to Context Sensitive mode.

11 Insert a bitmap checkpoint that checks your signature.

Choose **Insert** > **Bitmap Checkpoint** > **For Object/Window** or click the **Bitmap Checkpoint for Object/Window** button on the User toolbar.

Use the 🖑 pointer to click the **Agent Signature** box. WinRunner captures the bitmap and inserts an **obj_check_bitmap** statement into the test script.

12 Click the Clear Signature button.

The signature is cleared from the Agent Signature box.

13 Insert another bitmap checkpoint that checks the Agent Signature box.

Choose **Insert** > **Bitmap Checkpoint** > **For Object/Window** or click the **Bitmap Checkpoint for Object/Window** button on the User toolbar.

Use the 🖑 pointer to click the **Agent Signature** box. WinRunner captures a bitmap and inserts an **obj_check_bitmap** statement into the test script.

14 Click the Cancel button on the Fax Order dialog box.

15 Stop recording.

Choose **Test** > **Stop Recording** or click the **Stop** button.

16 Save the test.

Choose **File** > **Save** or click the **Save** button. Save the test as **lesson6** in a convenient location on your hard drive.

17 If you are working in the Global GUI Map file mode, add the new objects from the local GUI map to the flight4a.GUI map.

To add a new object to your GUI map for a window that already exists, choose **Tools > GUI Map Editor**. Choose **View > GUI Files**. Note that the new object is displayed in the **L0 <Temporary>** GUI map file. (If it is not displayed, choose **View > Expand Objects Tree**.) Choose **File > Save**. A WinRunner message informs you that the new objects from the existing window will be added to the **flight4a.GUI** map that contains this window. Click **Yes**.

18 Save the modified flight4a.GUI map.

In the GUI file box, select **L1 flight4a.GUI**. Note that the GUI map name has an asterisk next to it. This indicates that the file has been changed, and has not been saved. Choose **File > Save** to save the GUI map.

19 Close the GUI Map Editor.

Choose **File > Exit** to close the GUI Map Editor. For more information on saving new windows and new objects, see step 7 on page 49.

FYI:

Bitmap checkpoints appear as **obj_check_bitmap** or **win_check_bitmap** statements in the test script. For example:

obj_check_bitmap("(static)", "Img1", 1);

static is the object or area's logical name.

Img1 is the file containing the captured bitmap.

1 is the time (in seconds) needed to perform the check. This time is added to the value of the *timeout_msec* test option. See Lesson 4 "Synchronizing Tests" on page 43 for more information.

Viewing Expected Results

You can now view the expected results of the **lesson6** test.

1 Open the WinRunner Test Results window.

Choose **Tools** > **Test Results** or click the **Test Results** button. The Test Results window opens.

2 View the captured bitmaps.

In the test log section, double-click the first **capture bitmap** event, or select it and click the **Display** toolbar button.

Next, double-click the second **capture bitmap** event, or select it and click the **Display** toolbar button.
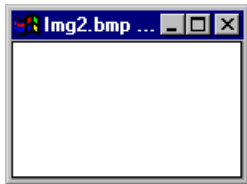
3 Close the Test Results window.

Close the bitmaps and choose **File** > **Exit** to close the Test Results window.

Running the Test on a New Version

You can now run the test on a new version of the Flight Reservation application.

1 Close Flight Reservation 4A.

Choose **File** > **Exit**.

2 Start Flight Reservation 4B.

flight4b.exe

Choose **Start** > **Programs** > **WinRunner** > **Sample Applications** > **Flight 4B**. In the Login window, type your name and the password mercury, and click **OK**. Reposition the Flight Reservation application and WinRunner so that they are both clearly visible on your desktop.

3 Make sure that lesson6 is the active test.

Click in the **lesson6** test window.

4 Check that Verify run mode is selected in the Test toolbar.

5 Choose Run from Top.

Choose **Run** > **Run from Top**, or click the **Run from Top** button. The **Run Test** dialog box opens. Accept the default test run name **res1**. Make sure that the **Display test results at end of run** check box is selected.

6 Run the test.

Click **OK**. The test run begins.

If a mismatch is detected at a bitmap checkpoint, click **Continue** in the message window.

7 Review the results.

When the test run is completed, the test results appear in the WinRunner Test Results window.

The test failed because the Agent Signature box did not clear when WinRunner clicked the Clear Signature button.

Double-click the failed bitmap checkpoint to view the expected, actual, and difference bitmaps.

WinRunner Test Results - [C:\Program Files\Mercury Interactive\Win...
File   Options   Tools   Window

lesson6	Test Result:	fail
	Total number of bitmap checkpoints:	2
	Total number of GUI checkpoints:	0
	General Information	

ine	Event	Details	Result	Time
3	start run	lesson6	run	00:00:00
43	bitmap checkpoint	Img1:1	mismatch	00:00:21
48	bitmap checkpoint	Img2:1	mismatch	00:00:31
51	stop run	lesson6	fail	00:00:31

Ready

8 Close the Test Results window.

Choose **File** > **Exit** to close the Test Results window.

9 Close the lesson6 test.

Choose **File** > **Close**.

10 Close version 4B of the Flight Reservation application.

Choose **File** > **Exit**.

Bitmap Checkpoint Tips

➤ To capture an area, choose **Insert** > **Bitmap Checkpoint** > **For Screen Area** or click the **Bitmap Checkpoint for Screen Area** button on the User toolbar. (This menu command is only available when a test is open.) Use the crosshairs pointer to mark the area that you want WinRunner to capture. WinRunner inserts a **win_check_bitmap** statement into your test script. This statement includes additional parameters that define the position (x- and y-coordinates) and size (width and height) of the area.

➤ For overnight test runs, you can instruct WinRunner not to display a message when a bitmap mismatch is detected. Choose **Tools** > **General Options**. In the General Options dialog box, select the Run category and the Settings sub-category. Clear the **Break when verification fails** check box. This enables the test to run unattended.
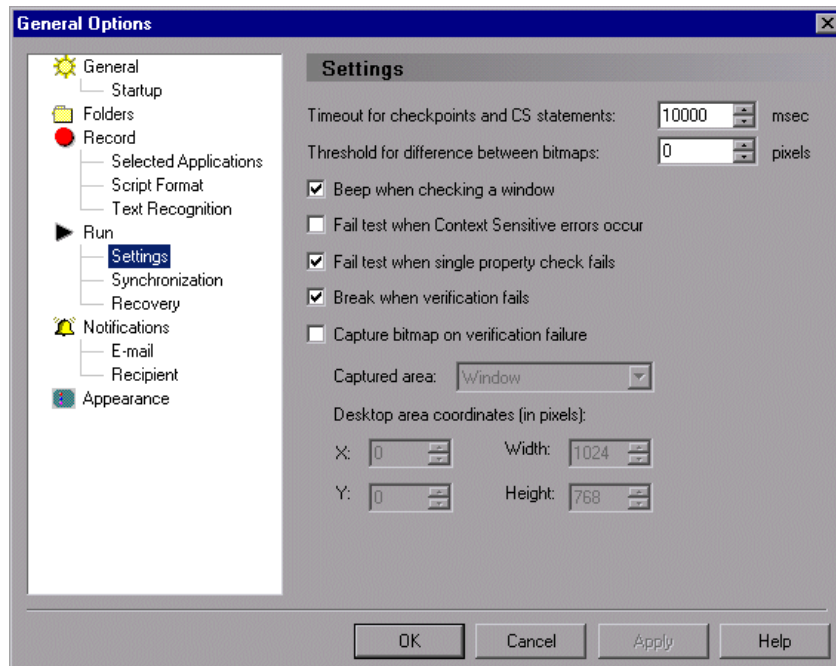
➤ When running a test that includes bitmap checkpoints, make sure that the screen display settings and display driver are the same as when the test script was created. If the display settings are different, WinRunner may report a bitmap mismatch.

➤ If you want to create new expected results for a bitmap checkpoint, run the test in Update run mode. WinRunner overwrites the existing expected bitmaps with new expected bitmaps captured during the Update run.

For more information on bitmap checkpoints, refer to the "Checking Bitmaps" chapter in the *Mercury WinRunner User's Guide*.

7

Programming Tests with TSL

Programming Tests with TSL

When you record a test, WinRunner generates TSL statements in a test script each time you click a GUI object or type on the keyboard. In addition to the recorded TSL functions, TSL includes many other built-in functions which can increase the power and flexibility of your tests. You can quickly add these functions to a test script using WinRunner's visual programming tool, the Function Generator. All functions located in the Function Generator are explained in the *TSL Online Reference* and the *TSL Reference Guide*.

The Function Generator enables you to add TSL functions in two ways:

➤ You can point to a GUI object and let WinRunner "suggest" an appropriate function. You can then insert this function into the test script.

➤ You can select a function from a list. Functions appear by category and alphabetically.

You can further enhance your test scripts by adding logic. Simply type programming elements such as conditional statements, loops, and arithmetic operators directly into the test window.

In the following exercises, you will create a test that:

➤ opens an order

➤ opens the Fax Order dialog box

➤ checks that the total is equal to the number of tickets ordered multiplied by the price per ticket

➤ reports whether the total is correct or incorrect

Recording a Basic Test Script

Start by recording the process of opening an order in the Flight Reservation application and opening the Fax Order dialog box.

1 Start WinRunner, open a new test and load the GUI map.

If WinRunner is not already open, choose
Programs > **WinRunner** > **WinRunner** on the **Start** menu. If the Welcome window is open, click the **New Test** button. Otherwise, choose **File** > **New**. A new test window opens.

If you are working in the Global GUI Map file mode, confirm that the GUI map is loaded. To do this, choose **Tools** > **GUI Map Editor**. In the GUI Map Editor, choose **View** > **GUI Files** and confirm that **flight4a.GUI** is contained in the **GUI File** list.

2 Start the Flight Reservation application and log in.

Choose **Programs** > **WinRunner** > **Sample Applications** > **Flight 4A** on the **Start** menu. In the Login window, type your name and the password **mercury**, and click **OK**. Reposition the Flight Reservation application and WinRunner so that they are both clearly visible on your desktop.

3 Start recording in Context Sensitive mode.

Choose **Test** > **Record - Context Sensitive** or click the **Record** button in the toolbar.

4 Open order #3.

In the Flight Reservation application, choose **File** > **Open Order**. In the Open Order dialog box, select the **Order No.** check box and type 3 in the adjacent box. Click **OK** to open the order.

5 Open the Fax Order dialog box.

Choose **File** > **Fax Order**.

6 Click Cancel to close the dialog box.

7 Stop recording.

Choose **Test** > **Stop Recording** or click the **Stop** button.

8 Save the test.

Choose **File** > **Save** or click the **Save** button. Save the test as **lesson7** in a convenient location on your hard drive.

9 If you are working in the Global GUI Map file mode, add the new objects from the local GUI map to the flight4a.GUI map.

To add a new object o your GUI map for a window that already exists t, choose **Tools** > **GUI Map Editor**. Choose **View** > **GUI Files**. Note that the new object is displayed in the **L0 <Temporary>** GUI map file. (If it is not displayed, choose **View** > **Expand Objects Tree**.) Choose **File** > **Save**. A WinRunner message informs you that the new objects from the existing window will be added to the **flight4a.GUI** map that contains this window. Click **Yes**.

10 Save the modified flight4a.GUI map.

In the GUI file box, select **L1 flight4a.GUI**. Note that the GUI map name has an asterisk next to it. This indicates that the file has been changed, and has not been saved. Choose **File > Save** to save the GUI map.

11 Close the GUI Map Editor.

Choose **File > Exit** to close the GUI Map Editor.

Using the Function Generator to Insert Functions

You are now ready to add functions to the test script which query the **# Tickets**, **Ticket Price**, and **Total** boxes in the Fax Order dialog box.

1 Insert a blank line above the button_press ("Cancel"); statement and place the cursor at the beginning of this line.

2 Open the Fax Order dialog box.

Choose **File > Fax Order** in the Flight Reservation application.

3 Query the # Tickets box.

Choose **Insert > Function > For Object/Window** or click the **Insert Function for Object/Window** button on the User toolbar. Use the 🖑 pointer to click the **# Tickets** box.

The Function Generator opens and suggests the **edit_get_text** function.
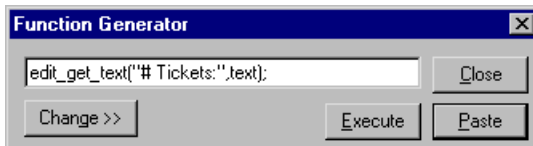
This function reads the text in the **# Tickets** box and assigns it to a variable. The default variable name is text.

Change the variable name **text** to **tickets** by typing in the box:

edit_get_text("# Tickets:",tickets);

Click **Paste** to add the function to the test script.

4 Query the Ticket Price box.

Choose **Insert** > **Function** > **For Object/Window** or click the
Insert Function for Object/Window button on the User toolbar. Use the 🖑
pointer to click the **Ticket Price** box.

The Function Generator opens and suggests the **edit_get_text** function.
Change the name of the **text** variable to **price**:

edit_get_text("Ticket Price:",price);

Click **Paste** to add the function to the test script.

5 Query the Total box.

Choose **Insert** > **Function** > **For Object/Window** or click the
Insert Function For Object/Window button on the User toolbar. Use the 🖑
pointer to click the **Total** box.

The Function Generator opens and suggests the **edit_get_text** function.
Change the name of the **text** variable to **total**:

edit_get_text("Total:",total);

Click **Paste** to add the function to the test script.

6 Close the Fax Order dialog box.

Click **Cancel** to close the dialog box in the Flight Reservation application.

7 Save the test.

Choose **File** > **Save** or click the **Save** button.

8 **If you are working in the Global GUI Map file mode, add the new objects from the local GUI map to the flight4a.GUI map.**

To add a new object from a window that already exists to your GUI map, choose **Tools** > **GUI Map Editor**. Choose **View** > **GUI Files**. Note that the new object is displayed in the **L0 <Temporary>** GUI map file. (If it is not displayed, choose **View** > **Expand Objects Tree**.) Choose **File** > **Save**. A WinRunner message informs you that the new objects from the existing window will be added to the **flight4a.GUI** map that contains this window. Click **Yes**.

9 **Save the modified flight4a.GUI map.**

In the GUI file box, select **L1 flight4a.GUI**. Note that the GUI map name has an asterisk next to it. This indicates that the file has been changed, and has not been saved. Choose **File** > **Save** to save the GUI map.

10 **Close the GUI Map Editor.**

Choose **File** > **Exit** to close the GUI Map Editor.

For more information on saving new windows and new objects, see step 12 on page 37 and step 7 on page 49.

Adding Logic to the Test Script

In this exercise, you will program decision-making logic into the test script using an if/else statement. This enables the test to:

➤ check that the total is equal to the number of tickets ordered multiplied by the price per ticket

➤ report whether the total is correct or incorrect

1 **Place the cursor below the last edit_get_text statement in the lesson7 script.**

2 **Add the following statements to the test script exactly as they appear below.**

Note that the tabs or spaces at the beginning of the second and fourth lines are optional.

```
if (tickets*price == total)
            tl_step ("total", 0, "Total is correct.");
else
            tl_step ("total", 1, "Total is incorrect.");
```

In plain English, these statements mean: If **tickets** multiplied by **price** equals **total**, report that the total is correct, otherwise (else) report that the total is incorrect. See "Understanding tl_step" on page 82 for more information on the **tl_step** function.

3 Add a comment to describe what this section of the script will do.

Insert a blank line above the if statement you added in the previous step and place the cursor at the beginning of this line. Choose **Edit** > **Comment**. After the # sign, type check that the total ticket price is calculated correctly.

4 Save the test.

Choose **File** > **Save** or click the **Save** button.

For more information on saving new windows and new objects, see step 12 on page 37 and step 7 on page 49.

FYI:

You can use the Function Generator to quickly insert **tl_step** statements into the test script. Choose **Insert** > **Function** > **From Function Generator** or choose **Insert Function from Function Generator** on the User toolbar.

Understanding tl_step

In most cases when you run a test, WinRunner reports an overall test result of pass or fail. By adding **tl_step** statements to your test script, you can determine whether a particular operation within the test passed or failed, and send a message to the report.

For example:

tl_step ("total", 1, "Total is incorrect.");

total is the name you assign to this operation.

1 causes WinRunner to report that the operation failed. If you use *0*, WinRunner reports that the operation passed.

Total is incorrect is the message sent to the report. You can write any message that will make the test results meaningful.

For more information regarding the **tl_step** function, refer to the *TSL Online Reference* in WinRunner.

Debugging the Test Script

After enhancing a test with programming elements, you should check that the test runs smoothly, without errors in syntax and logic. WinRunner provides debugging tools which make this process quick and easy.

You can:

➤ run the test line by line using the Step commands

➤ define breakpoints that enable you to stop running the test at a specified line or function in the test script

➤ monitor the values of variables and expressions using the Watch List

When you debug a test script, you should run your test in Debug mode. (To run a test in Debug mode, select **Debug** from the Run Mode list on the Standard toolbar.) The test results are saved in a *debug* directory. Each time you run the test in Debug mode, WinRunner overwrites the previous debug results.

In this exercise, you will control the test run using the Step command. If any error messages appear, examine the test script and try to fix the problem.

1 Select Debug mode from the Run Mode list on the Standard toolbar.

Debug mode will remain in effect until you select a different mode.

2 Place the execution marker –> next to the first line in the test script.

Click in the left margin, next to the first line in the test script.

3 Choose Debug > Step or click the Step button to run the first line in the test script.

WinRunner runs the first line of the test.

4 Use the Step button to run the entire test, line by line.

Click the **Step** button to run each line of the test script. Note that your mouse pointer may sometimes move to the flight application as it clicks on objects during the test run.

Stop | **5 Click Stop.**

Click the **Stop** button to tell WinRunner that you have completed the Debug test run.

6 Review the test results in the WinRunner Test Results window.

When you run the test in Debug mode, the test results do not open automatically. Choose **Tools** > **Test Results** or click the **Test Results** button. The WinRunner Test Results window displays the results of the Debug test run.

7 Close the Test Results window.

Choose **File** > **Exit**.

8 Exit the Flight Reservation application.

Choose **File** > **Exit**.

For more information on debugging test scripts, refer to Part VI, "Debugging Tests", in the *Mercury WinRunner User's Guide*.

Running the Test on a New Version

Once the test script is debugged, you can run it on a new version of the Flight Reservation application.

flight4b.exe | **1 Open version 4B of the Flight Reservation application.**

Choose **Programs** > **WinRunner** > **Sample Applications** > **Flight 4B** on the **Start** menu. In the Login window, type your name and the password mercury, and click **OK**. Reposition the Flight Reservation application and WinRunner so that they are both clearly visible on your desktop.

Verify ▾ | **2 Select Verify mode from the Run Mode list on the Standard toolbar.**

Verify mode will remain in effect until you select a different mode.

From Top | **3 Choose Run from Top.**

Choose **Test** > **Run from Top**, or click the **From Top** button. The Run Test dialog box opens. Accept the default test run name **res1**. Make sure that the **Display test results at end of run** check box is selected.

4 Run the test.

Click **OK** in the Run Test dialog box. The test run begins.

5 Review the test results.

When the test run is completed, the test results appear in the WinRunner Test Results window.

The number of tickets multiplied by the price equals the total. Therefore the tl_step statement reports pass.
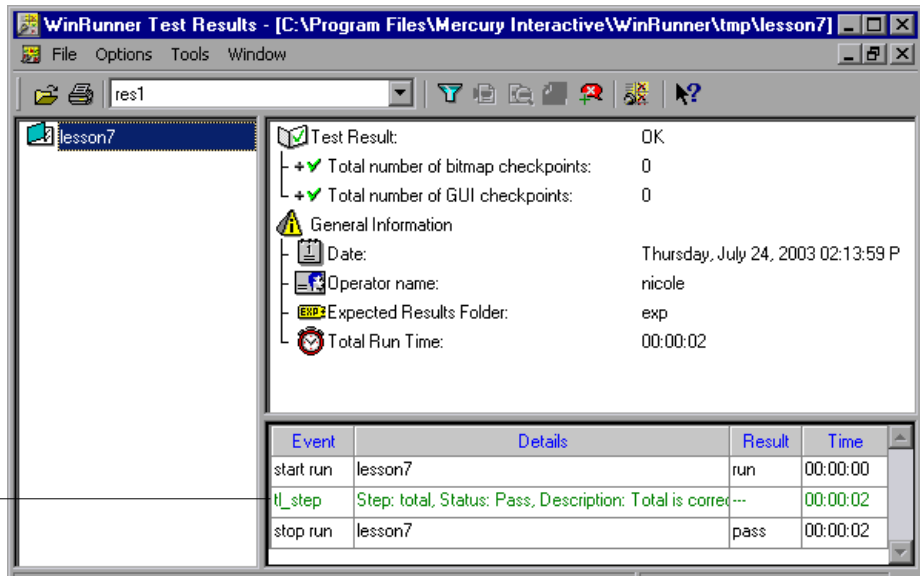
You can double-click the **tl_step** statement in the test log to view the full details:
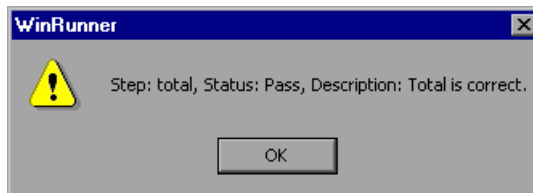
Notice that the message, **Total is correct**, is the same message you wrote in the test script. Click **OK** to close the message.

6 Close the test results.

Choose **File** > **Exit** to close the Test Results window.

7 Close the lesson7 test.

Choose **File** > **Close**.

8 Close version 4B of the Flight Reservation application.

Choose **File** > **Exit**.

8

Creating Data-Driven Tests

Creating Data-Driven Tests

Once you have successfully debugged and run your test, you may want to see how the same test performs with multiple sets of data. To do this, you convert your test to a data-driven test and create a corresponding data table with the sets of data you want to test.

Converting your test to a data-driven test involves the following steps:

➤ Adding statements to your script that open and close the data table.

➤ Adding statements and functions to your test so that it will read from the data table and run in a loop while it applies each set of data.

➤ Replacing fixed values in recorded statements and checkpoint statements with parameters, known as *parameterizing* the test.

You can convert your test to a data-driven test using the DataDriver Wizard or you can modify your script manually.

When you run your data-driven test, WinRunner runs the parameterized part(s) of the test one time (called an *iteration*) for each set of data in the data table, and then displays the results for all of the iterations in a single Test Results window.

In Lesson 7, you created a test that opened a specific flight order and read the number of tickets, price per ticket, and total price from a fax order dialog box to check that the total price was correct. In this lesson, you will create a test that performs the same check on several flight orders to check that your application computes the correct price for various quantities and prices of tickets.

Converting Your Test to a Data-Driven Test

Start by opening the test you created in Lesson 7 and use the DataDriver Wizard to parameterize the test.

1 Create a new test from the lesson7 test and load the GUI map.

If WinRunner is not already open, choose **Start > Programs > WinRunner > WinRunner**. If the Welcome window is open, click the **Open Test** button. Otherwise, choose **File > Open** and select the test you created in Lesson 7. The **lesson7** test opens.

Choose **File > Save As** and save the test as **lesson8** in a convenient location on your hard drive.

If you are working in the Global GUI Map file mode, confirm that the GUI map is loaded. To do this, choose **Tools > GUI Map Editor**. In the GUI Map Editor, choose **View > GUI Files** and confirm that **flight4a.GUI** is contained in the **GUI File** list.

2 Run the Data Driver Wizard.

Choose **Table > Data Driver Wizard**. The DataDriver Wizard welcome window opens. Click **Next** to begin the parameterization process.

3 Create a data table for the test.

In the **Use a new or existing Excel table** box, type lesson8. The DataDriver Wizard creates a Microsoft Excel table with this name and saves it in the test folder.

4 Assign a table variable name.

Accept the default table variable name, **table**.

At the beginning of a data-driven test, the Microsoft Excel data table you wish to use is assigned as the value of the table variable. Throughout the script, only the table variable name is used. This makes it easy for you to assign a different data table to the script at a later time without making changes throughout the script.

**5 Select global parameterization options.**

Select **Add statements to create a data-driven test**. This adds TSL statements to the test that define the table variable name, open and close the data table, and run the appropriate script selection in a loop for each row in the data table.

Select **Parameterize the test** and choose the **Line by line** option. When you select Parameterize the test, you instruct WinRunner to find fixed values in recorded statements and selected checkpoints and to replace them with parameters. The Line by line option instructs the wizard to open a screen for each line of the selected test that can be parameterized so that you can choose whether or not to parameterize that line.

Click **Next**.

6 Select the data to parameterize.

The first line-by-line screen opens. It refers to the **Order Number** radio button.

Test script line to parameterize:

button_set ("Order No.", ON);
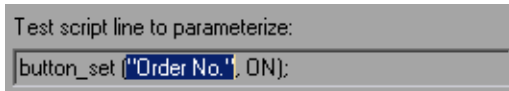
In this test, you are going to open a different fax order in each iteration and the **Order Number** radio button must be selected each time. Thus, for this script line, keep the selection, **Do not replace this data**, and click **Next**.

The next line by line screen refers to the **Order Number** box. This is the box you want to change for each iteration. Note that the value **"3"** is highlighted and listed in the **Argument to be replaced** box to indicate that this is the value selected for parameterization.

Test script line to parameterize:

edit_set ("Edit_1", "3");

Argument to be replaced: "3"   ◄ ►
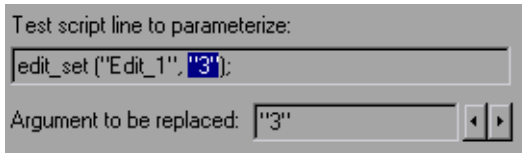
Select **A new column** under **Replace the selected value with data from:** and type Order_Num in the adjacent box. The new column option creates a column titled **Order_Num** in the **lesson8.xls** table, and enters the value **3** in the first row of the column.
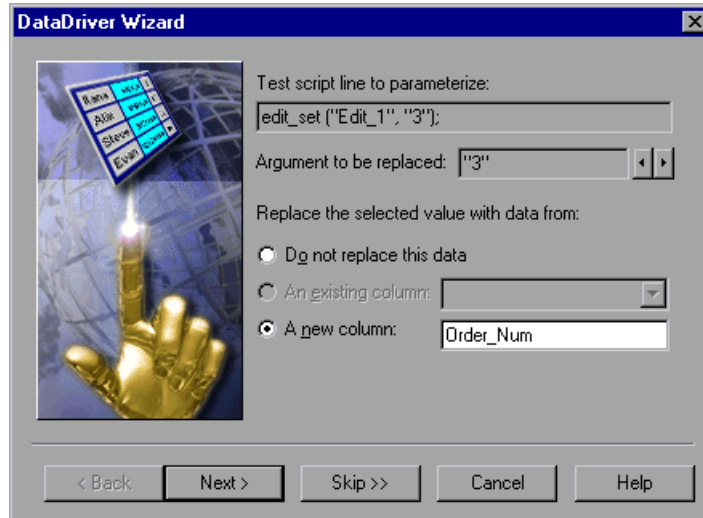
Click **Next** and then click **Finish**. Your test is parameterized.

FYI:

The following elements are added or modified in your parameterized test:

The **table =** line defines the table variable.

The **ddt_open** statement opens the table, and the subsequent lines confirm that the data-driven test opens successfully.

The **ddt_get_row_count** statement checks how many rows are in the table, and therefore, how many iterations of the parameterized section of the test to perform.

The **for** statement sets up the iteration loop.

The **ddt_set_row** statement tells the test which row of the table to use on each iteration.

In the **edit_set** statement, the value, **"3"** is replaced with a **ddt_val** statement.

The **ddt_close** statement closes the table.
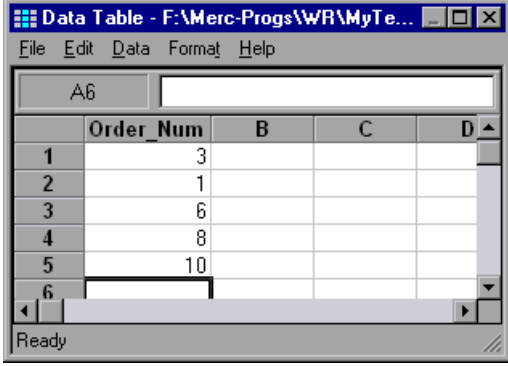
Adding Data to the Data Table

Now that you have parameterized your test, you are ready to add the data that the parameterized test will use.

1 Open the data table.

Choose **Table > Data Table**. The **lesson8.xls** table opens. Note that there is one column named **Order_Num**, and that the first row in the column contains the value **3**.

2 Add data to the table.

In rows **2**, **3**, **4**, and **5** of the **Order_Num** column, enter the values, 1, 6, 8, and 10 respectively.

3 Save and close the table.

Click an empty cell and choose **File** > **Save** from the data table menu. Then choose **File** > **Close** to close the table.

4 Save the test.

Choose **File** > **Save** or click the **Save** button.

Adjusting the Script with Regular Expressions

Your test is almost finished. Before running the test you should look through it to see if there are any elements that may cause a conflict in a data-driven test. The DataDriver wizard finds all fixed values in selected checkpoints and recorded statements, but it does not check for things such as object labels that also may vary based on external input.

In the flight application, the name of the Fax Order window changes to reflect the fax order number. If you run the test as it is, the test will fail on the second iteration, because the Flight Application will open a window titled **Fax Order No. 1**, but the script tells it to make the window titled **Fax Order No. 3** active. WinRunner will be unable to find this window.

To solve this problem, you can use a regular expression. A regular expression is a string that specifies a complex search phrase to enable WinRunner to identify objects with varying names or titles.

In this exercise, you will use a regular expression in the physical description of the Fax Order window so that WinRunner can ignore variations in the window's label.

1 Locate the Fax Order window in the flight4a.GUI GUI map file.

Choose **Tools** > **GUI Map Editor.** Choose **View** > **GUI Files.** Select **flight4a.GUI** in the GUI file box. Select the Fax Order No. 3 window icon.

2 Modify the window label with a regular expression.

In the **Physical Description** box, on the **label** property line, add an **!** immediately following the opening quotes to indicate that this is a regular expression. Delete the space and the number **3** at the end of the line and replace this text with ***** to indicate that the text following this phrase can vary.
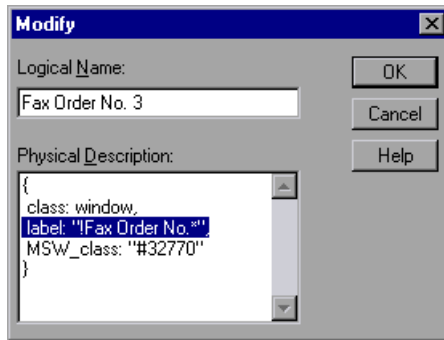
3 Close the Modify dialog box.

Click **OK** to close the Modify window.

4 Save the GUI map (only if you are working in the Global GUI Map file mode) and close the GUI Map Editor.

If you are working in the Global GUI Map file mode, choose **File** > **Save** to save your changes and choose **File** > **Exit** to close the GUI Map Editor.

If you are working in the GUI Map File per Test mode, choose **File** > **Exit** to exit the GUI Map Editor.

Customizing the Results Information

You could run the test now, but it may be difficult for you to interpret the results for each iteration. You can add iteration-specific information to the reporting statements in your script so that you can see which data is the basis for each result.

1 Modify the tl_step statements.

Locate the first **tl_step** statement in your script. Delete the words "total is correct"." and replace them with "Correct. "tickets" tickets at $"price" cost $"total"."

tl_step("total",0, "Correct. "tickets" tickets at $"price" cost $"total".");

Use the same logic to modify the next **tl_step** statement to report an incorrect result. For example:

tl_step("total", 1, "Error! "tickets" tickets at $"price" does not equal $"total". ");

Now you will be able to see which data is used in each iteration when you view the results.

2 Save the test.

Choose **File** > **Save** or click the **Save** button.

Running the Test and Analyzing Results

You run the data-driven test just like any other test in WinRunner. When the test run is completed, the results for all iterations are included in a single Test Results window.

1 Make sure that the Flight 4A Flight Reservation application is open on your desktop.
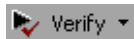
2 In WinRunner, check that Verify run mode is selected in the Test toolbar.

3 Choose Run from Top.

Choose **Run** > **Run from Top**, or click the **Run from Top** button. The **Run Test** dialog box opens. Accept the default test run name. Make sure that the **Display test results at end of run** check box is selected.

4 Run the test.

Click **OK** in the Run Test dialog box. The test will run through the parameterized section of the script five times, once for each row in the data table.

5 Review the results.

When the test run is completed, the test results appear in the WinRunner Test Results window.
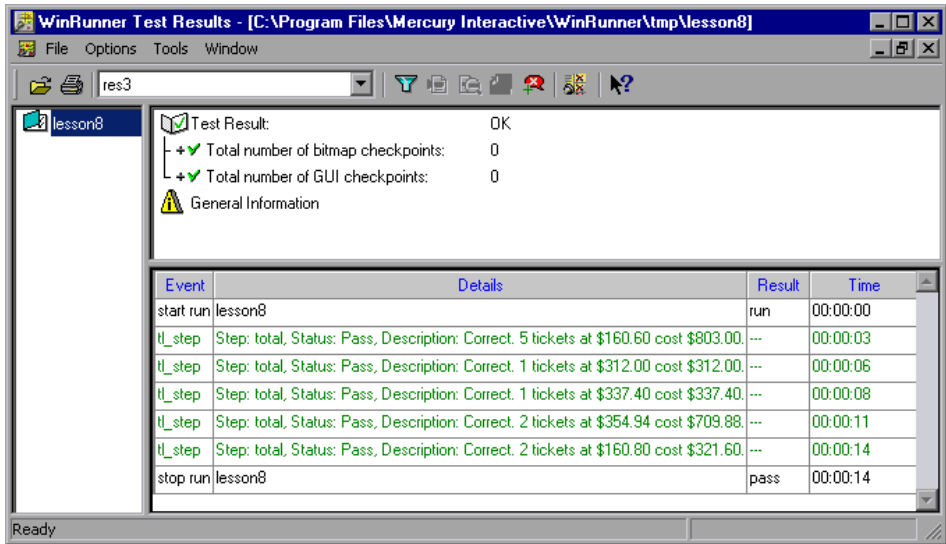
Note that the **tl_step** event is listed five times and that the details for each iteration include the actual number of tickets, price and total cost that was checked.

6 Close the test results.

Choose **File** > **Exit** to close the Test Results window.

7 Close the Flight Reservation application.

Choose **File** > **Exit**.

8 Close the lesson8 test.

Choose **File** > **Close**.

Data-Driven Testing Tips

➤ You can parameterize only part of your test script or a loop within it, and a single data-driven test can contain more than one parameterized loop.

➤ You can open and save data tables other than the **default.xls** data table. This enables you to use several different data tables in one test script.

➤ You can parameterize statements containing GUI checkpoints, bitmap checkpoints, and bitmap synchronization points, and constants.

➤ You can use the data table in the same way as an Excel spreadsheet, including inserting formulas into cells.

➤ Before you run a data-driven test, you should look through it to see if there are any elements that may cause a conflict in a data-driven test. There are two ways to solve most of these conflicts:

> ➤ Use a regular expression to enable WinRunner to recognize objects based on a portion of its physical description. For more information on regular expressions, refer to the "Using Regular Expressions" chapter in the *Mercury WinRunner User's Guide*.

> ➤ Use the GUI Map Configuration dialog box to change the physical properties that WinRunner uses to recognize the problematic object. For more information on GUI Map configuration, refer to the "Configuring the GUI Map" chapter in the *Mercury WinRunner User's Guide*.

➤ You can change the active row, or read from a non-active row during the test run by using TSL statements. For more information, refer to the "Using TSL Functions with Data-Driven Tests" chapter in the *Mercury WinRunner User's Guide*.

➤ It is not necessary for the data table viewer to be open when you run a test.

To learn more about data-driven tests, refer to the "Creating Data-Driven Tests" chapter in the *Mercury WinRunner User's Guide*.

9

Creating Batch Tests

In this lesson, you will learn about:	On page:
Using Batch Tests	99
Programming a Batch Test	100
Running the Batch Test on Version 4B	101
Analyzing the Batch Test Results	102
Batch Test Tips	105

Using Batch Tests

Imagine that you have revised your application and you want to run old test scripts on the revised product. Instead of running each test individually, you can use a batch test to run several tests, leave for lunch, and see the results of all your tests on your screen when you get back.

A batch test looks and behaves like a regular test script, except for two main differences:

➤ It contains **call** statements, which open other tests. For example:

call "c:\\qa\\flights\\lesson9"();

During a test run, WinRunner interprets a **call** statement, and then opens and runs the called test. When the called test is done, WinRunner returns to the batch test and continues the run.

➤ You choose the **Run in batch mode** option in the Run category of the General Options dialog box (**Tools** > **General Options**) before running the test. This option instructs WinRunner to suppress messages that would otherwise interrupt the test. For example, if WinRunner detects a bitmap mismatch, it does not prompt you to pause the test run.

When you review the results of a batch test run, you can see the overall results of the batch test (pass or fail), as well as the results of each test called by the batch test.

Programming a Batch Test

In this exercise, you will create a batch test that:

➤ calls tests that you created in earlier lessons (**lesson5**, **lesson6**, and **lesson7**)

➤ runs each called test three times to check how the Flight Reservation application handles the stress of repeated execution

1 Start WinRunner, open a new test and load the GUI map.

If WinRunner is not already open, choose **Start** > **Programs** > **WinRunner** > **WinRunner**. If the Welcome window is open, click the **New Test** button. Otherwise, choose **File** > **New**. A new test window opens.

If you are working in the Global GUI Map file mode, confirm that the GUI map is loaded. To do this, choose **Tools** > **GUI Map Editor**. In the GUI Map Editor, choose **View** > **GUI Files** and confirm that **flight4a.GUI** is contained in the **GUI File** list.

2 Program call statements in the test script that call lesson5, lesson6, and lesson7.

Type the **call** statements into the new test window in the following format:

```
call "c:\\qa\\flights\\lesson5"();
call "c:\\qa\\flights\\lesson6"();
call "c:\\qa\\flights\\lesson7"();
```

In your test script, replace c:\\qa\\flights with the directory path that contains your tests.

Note: When you type the path, use double backslashes between the directory names.

---

3 Define a loop that calls each test 3 times.

Add a loop around the **call** statements so that the test script looks like this:

```
for (i=0; i<3; i++)
{
call "c:\\qa\\flights\\lesson5"();
call "c:\\qa\\flights\\lesson6"();
call "c:\\qa\\flights\\lesson7"();
}
```

In plain English, this means: Run lesson5, lesson6, and lesson7, and then loop back and run each test again. Repeat this process until each test runs 3 times. Note that the brackets { } define which statements are included in the loop.

4 Choose the Batch Run option in the General Options dialog box.

Choose **Tools** > **General Options**. In the General Options dialog box, choose the Run category. Select the **Run in batch mode** check box and click **OK** to close the General Options dialog box.

5 Save the batch test.

Choose **File** > **Save** or click the **Save** button. Name the test **batch**.

Running the Batch Test on Version 4B

You are now ready to run the batch test to check the Flight Reservation application. When you run the test, WinRunner will compare the expected results of each test to the actual results in the application. It uses the expected results stored when you created the tests in earlier lessons.
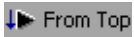
flight4b.exe

1 Open version 4B of the Flight Reservation application and log in.

Choose **Start** > **Programs** > **WinRunner** > **Sample Applications** > **Flight 4B**. In the **Login** window, type your name and the password mercury, and click **OK**. Reposition the Flight Reservation application and WinRunner so that they are both clearly visible on your desktop.

▶ Verify ▾

2 In WinRunner, check that Verify run mode is selected in the Test toolbar.

↓▶ From Top

3 Choose Run from Top.

Choose **Test** > **Run from Top**, or click the **From Top** button. The Run Test dialog box opens. Accept the default test run name **res1**. Make sure that the **Display test results at end of run** check box is selected.

4 Run the test.

Click **OK** in the Run Test dialog box. The test run begins. The test run consists of nine different test executions and may take some time.

Watch how WinRunner opens and runs each called test, and loops back to run the tests again (a total of 3 times).

Analyzing the Batch Test Results

Once the batch test run is completed, you can analyze the results in the WinRunner Test Results window. The Test Results window displays the overall result (pass or fail) of the batch test, as well as a result for each called test. The batch test fails if any of the called tests failed.

1 Open the WinRunner Test Results window and display the res1 results of the batch test.

If the WinRunner Test Results window is not currently open, click in the batch test window and choose **Tools** > **Test Results**, or click the **Test Results** button.

2 View the results of the batch test.

The test tree shows all the tests called during the batch test run. Since each test was called 3 times, the test names appear 3 times in the list.

Lists all the events that occurred during the batch test run.

Displays the current test results name.

Shows whether the batch test passed or failed.

A call test event indicates that a called test was opened and run.

A return event indicates that control was returned to the batch test.

The batch test failed because one or more of the called tests failed. As you have seen in earlier lessons, version 4B contains some bugs.

3 View the results of the called tests.

Select a test name in the test tree to view the results of a called test.

The highlighted test indicates which test results are currently displayed. In this case, lesson6 results appear in the Test Results window.

Lists all the events that occurred when the test was called.

Displays the current test results name.

Shows whether the selected called test passed or failed.

Recall that **lesson6** uses a bitmap checkpoint to check that the Agent Signature box in the Fax Order dialog box clears after WinRunner clicks the **Clear Signature** button. Since the **Signature** box did not clear, the bitmap checkpoint detected a mismatch. You can double-click the failed event to display the expected, actual, and difference results.

4 Close the Test Results window.

Choose **File** > **Exit**.

5 Close the batch test.

Choose **File** > **Close** for each open test. That is, the batch test and the three tests that were called by the batch. Alternatively, you can choose **File** > **Close All**.

6 Clear the Batch Run option in the General Options dialog box.

Once you are finished running the batch test, clear the Batch Run option. Choose **Tools** > **General Options**. In the General Options dialog box, select the Run category. Clear the **Run in batch mode** check box and click **OK**.

7 Close version 4B of the Flight Reservation application.

Choose **File** > **Exit**.

Batch Test Tips

➤ By defining search paths, you can instruct WinRunner to search for called tests in certain directories. Choose **Tools** > **General Options**. In the General Options dialog box, select the Folders category. In the **Search path for called tests** box, define the paths in which the tests are located. This enables you to specify a relative path, such as just the test name in a **call** statement. For example:

call "lesson6"();

For more information on defining search paths for called tests, refer to the "Setting Global Testing Options" chapter in the *Mercury WinRunner User's Guide*.

➤ You can pass parameter values from the batch test to a called test. Parameter values are defined within the parentheses of a call statement.

call test_name ([parameter1, parameter2, ...]);

➤ Remember that you must select the **Run in batch mode** option in the Run category of the General Options dialog box in order for the batch test to run unattended. Otherwise, the test will stop running to display an error message if any errors occur.

For more information on creating batch tests, refer to the "Calling Tests" and "Running Batch Tests" chapters in the *Mercury WinRunner User's Guide*.

10

Maintaining Your Test Scripts

Handling User Interface Changes

Consider this scenario: you have just spent several weeks creating a suite of automated tests that covers the entire functionality of your application. The application developers then build a new version with an improved user interface. They change some objects, add new objects, and remove others. How can you test this new version using your existing tests?

WinRunner provides an easy solution. Instead of manually editing every test script, you can update the *GUI map*. The GUI map contains descriptions of the objects in your application.

An object description in the GUI map is composed of:

➤ a *logical name*, a short intuitive name describing the object. This is the name you see in the test script. For example:

button_press ("Insert Order");

Insert Order is the object's logical name.

➤ a *physical description*, a list of properties that uniquely identify the object. For example:

```
{
class: push_button
label: "Insert Order"
}
```

The button belongs to the **push_button** object class and has the label **Insert Order**.

When you run a test, WinRunner reads an object's logical name in the test script and refers to its physical description in the GUI map. WinRunner then uses this description to find the object in the application under test.

If an object changes in an application, you must update its physical description in the GUI map so that WinRunner can find it during the test run.

In the following exercises, you will:

➤ edit an object description in the GUI map

➤ add objects to the GUI map

➤ use the Run Wizard to automatically detect user interface changes and update the GUI map

Editing Object Descriptions in the GUI Map

Suppose that in a new version of the Flight Reservation application, the **Insert Order** button is changed to an **Insert** button. To continue running tests that use the **Insert Order** button, you must edit the label in the button's physical description in the GUI map. You can change the physical description using regular expressions. For additional information, refer to "Adjusting the Script with Regular Expressions" on page 93 of this tutorial and to the "Using Regular Expressions" chapter in the *Mercury WinRunner User's Guide*.

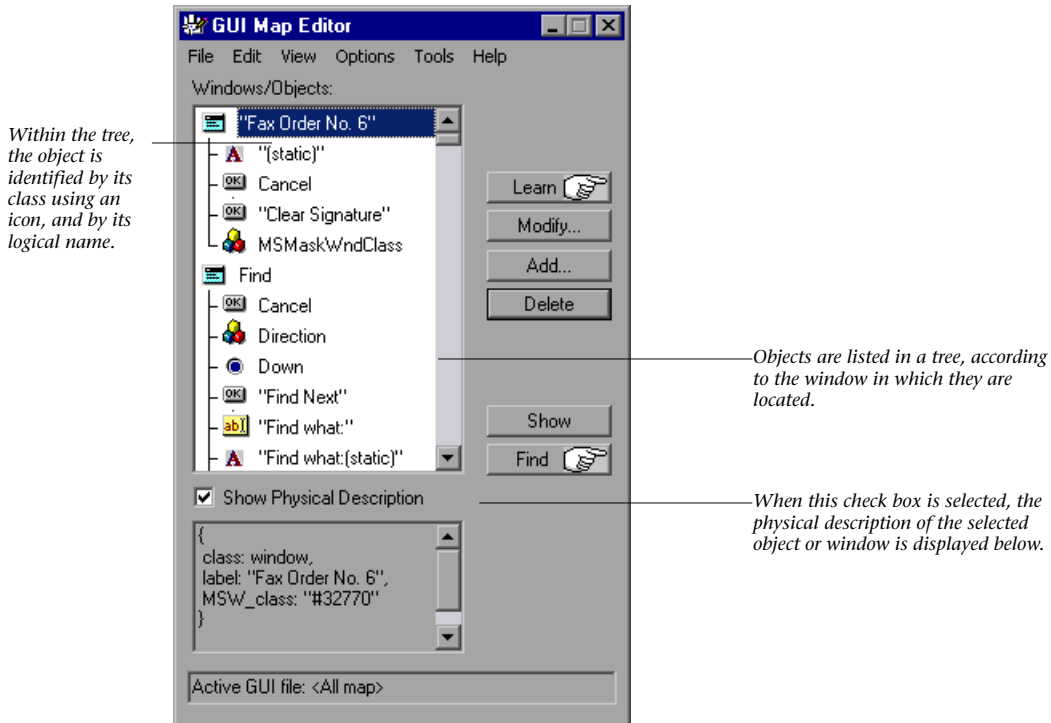**1 Start WinRunner, open a test, and load the GUI map.**

If WinRunner is not already open, choose **Start** > **Programs** > **WinRunner** > **WinRunner**. If the Welcome window is open, click the **New Test** button. Otherwise, choose **File** > **New**. A new test window opens.

If you are working in the GUI Map File per Test mode, open the **lesson4** test.

If you are working in the Global GUI Map file mode, confirm that the GUI map is loaded. To do this, choose **Tools** > **GUI Map Editor**. In the GUI Map Editor, choose **View** > **GUI Files** and confirm that **flight4a.GUI** is contained in the **GUI File** list.

2 Open the GUI Map Editor.

Choose **Tools** > **GUI Map Editor**. The GUI Map Editor opens. Choose **View** > **GUI Map**. The Windows/Object list displays the current contents of the GUI Map. (If you are working in the GUI Map File per Test mode, the GUI Map Editor will contain fewer objects than shown below.)

Within the tree, the object is identified by its class using an icon, and by its logical name.

Objects are listed in a tree, according to the window in which they are located.

When this check box is selected, the physical description of the selected object or window is displayed below.

The GUI Map Editor displays the object names in a tree. Preceding each name is an icon representing the object's type. The objects are grouped according to the window in which they are located. You can double-click a window icon to collapse or expand the view of its objects.

3 Find the Insert Order button in the tree.

In the GUI Map Editor, choose **View** > **Collapse Objects Tree** to view only the window titles. (If you are working in the GUI Map File per Test mode, the GUI Map Editor will contain fewer objects than shown below.)
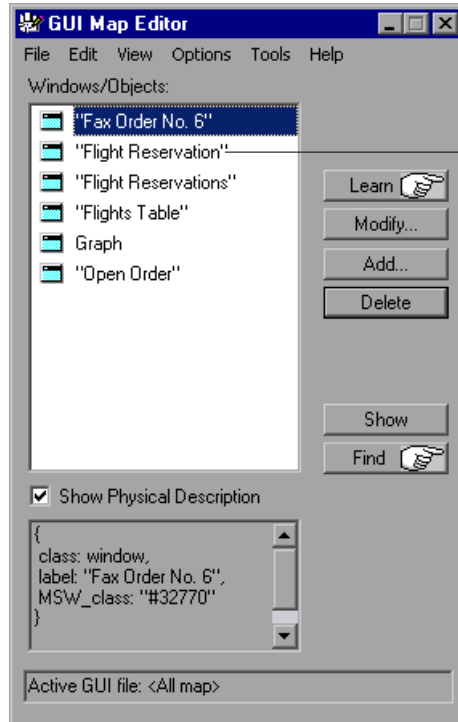

When you collapse the tree, only window titles are listed.

Double-click the Flight Reservation window to view its objects. Scroll down the alphabetical object list until you locate the **Insert Order** button.

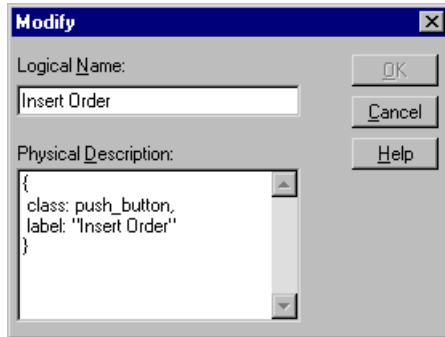**4 View the Insert Order button's physical description.**

Click the **Insert Order** button in the tree. (If you are working in the GUI Map File per Test mode, the GUI Map Editor will contain fewer objects than shown below.)
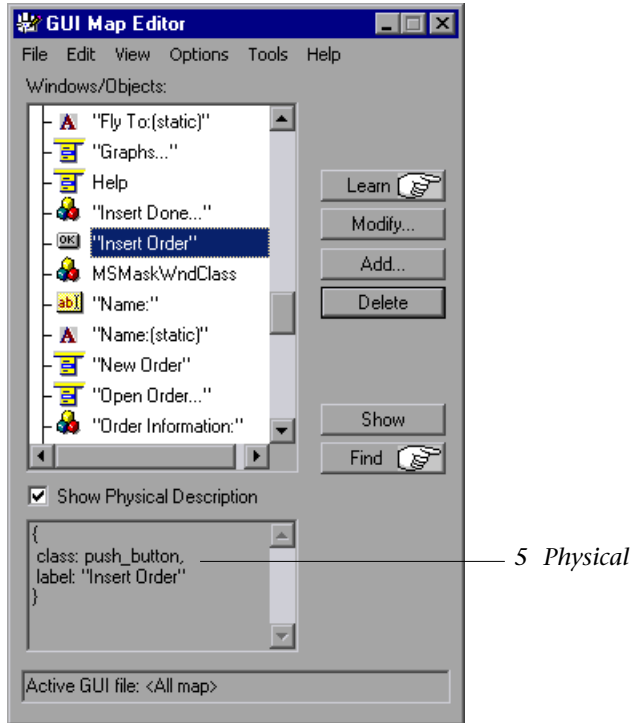

The physical description of the object is displayed in the bottom pane of the GUI Map Editor.

6 Modify the Insert Order button's physical description.

Click the **Modify** button or double-click the **Insert Order** button. The Modify dialog box opens and displays the button's logical name and physical description.

In the **Physical Description** box, change the **label** property from Insert Order to Insert.
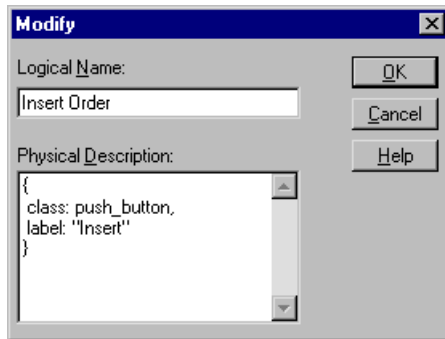
Click **OK** to apply the change and close the dialog box.

7 Close the GUI Map Editor.

In the GUI Map Editor, choose **File > Save** to save your changes and then choose **File > Exit**. If you are working in the GUI Map File per Test mode, choose **File > Exit** in the GUI Map Editor and then **File > Save** in WinRunner.

The next time you run a test that contains the logical name **Insert Order**, WinRunner will locate the **Insert** button in the Flight Reservation window.

If you are working in the GUI Map File per Test Mode, go back and perform steps 1 through 6 for the **lesson9** test.

One of the main advantages of working in the Global GUI Map file mode is the ability to modify an object in one place and all tests using the same GUI Map file benefit from the update. When working in the GUI Map File per Test Mode, you must update the GUI Map file of each test containing the modified object.

Adding GUI Objects to the GUI Map

Note: If you are working in the GUI Map File per Test mode, skip this exercise, since new objects are saved in your test's GUI map automatically when you save your test.

If your application contains new objects, you can add them to the GUI map without running the RapidTest Script wizard again. You simply use the Learn button in the GUI Map Editor to learn descriptions of the objects. You can learn the description of a single object or all the objects in a window.

In this exercise, you will add the objects in the Flight Reservation Login window to the GUI map.

 1 **Open the Flight Reservation Login window.**

flight4a.exe

Choose **Start** > **Programs** > **WinRunner** > **Sample Applications** > **Flight 4A**.
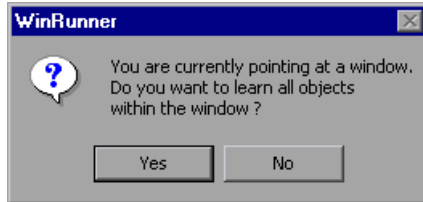
 2 **Open the GUI Map Editor.**

In WinRunner, choose **Tools** > **GUI Map Editor**. The GUI Map Editor opens. Choose **View** > **GUI Files**.

3 Learn all the objects in the Login window.

Click the **Learn** button. Use the 🖑 pointer to click the title bar of the Login window.

A message prompts you to learn all the objects in the window. Click **Yes**.

Watch as WinRunner learns a description of each object in the Login window and adds it to the temporary GUI map.

4 Save the new objects in the GUI map.

Choose **File** > **Save** in the GUI Map Editor. Click **Yes** and/or **OK** to add the new objects and/or new windows to your GUI map. Choose **File** > **Exit** to close the GUI Map Editor.

For more information on saving new windows and new objects, see step 7 on page 49.

5 Close the Login window.

Click **Cancel**.

Updating the GUI Map with the Run Wizard

Note: If you are working in the GUI Map File per Test mode, skip this exercise, since new objects are automatically saved in your test's GUI map when you save your test.

During a test run, if WinRunner cannot locate an object mentioned in the test script, the Run Wizard opens. The Run Wizard helps you update the GUI map so that your tests can run smoothly. It prompts you to point to the object in your application, determines why it could not find the object, and then offers a solution. In most cases, the Run Wizard will automatically modify the object description in the GUI map or add a new object description.

For example, suppose you run a test that clicks the **Insert Order** button in the Flight Reservation window:

button_press ("Insert Order");

If the **Insert Order** button is changed to an **Insert** button, the Run Wizard opens during a test run and describes the problem.
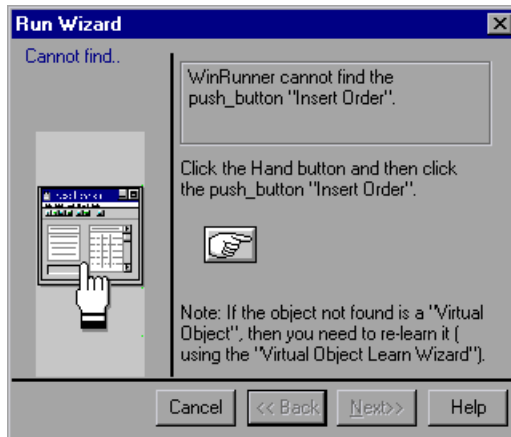
You click the hand button in the wizard and click the **Insert** button in the Flight Reservation program. The Run Wizard then offers a solution:
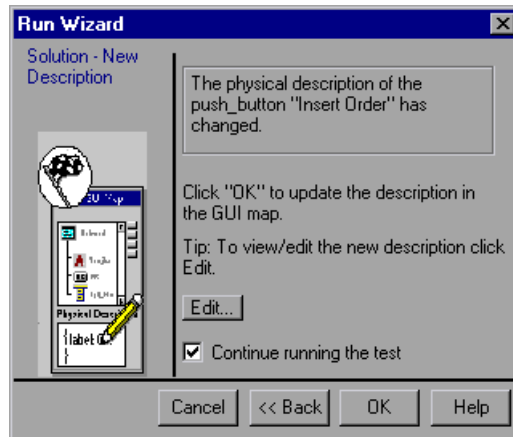
When you click **OK**, WinRunner automatically modifies the object's physical description in the GUI map and then resumes the test run.

If you want to see how the Run Wizard works:

1 Open the GUI Map Editor.

Choose **Tools** > **GUI Map Editor**. Choose **View** > **GUI Files**. Select **flight4a.GUI** in the **GUI File** box.

2 Delete the "Fly From" list object from the GUI Map Editor tree.

The "**Fly From**" object is listed under the Flight Reservation window. Select this object and click the **Delete** button in the GUI Map Editor. Save the GUI map file and exit the GUI Map Editor.

3 Open Flight Reservation 4A.

Choose **Start** > **Programs** > **WinRunner** > **Sample Applications** > **Flight 4A**. In the Login window, type your name and the password mercury, and click **OK**. Reposition the Flight Reservation application and WinRunner so that they are both clearly visible on your desktop.

4 In WinRunner, open the lesson4 test and run it.

Watch what happens when WinRunner reaches the statement

list_select_item ("Fly From:", "Los Angeles");

5 Follow the Run Wizard instructions.

The Run Wizard asks you to point to the **Fly From** object and then adds the object description to the GUI map. WinRunner then continues the test run. WinRunner also fails to find the **Insert Order** button because we changed the object's label description to **Insert**. Follow the instructions in the Run Wizard to fix the Insert Order object description.

6 Save the new Fly From object to the GUI map.

When WinRunner completes the test run, return to the GUI Map Editor and look for the Fly From object description. If you are working in the Global GUI Map file mode, the Run Wizard adds the object to the temporary GUI map. Choose **File > Save** to add the object back to the **flight4a.GUI** map file.

7 Close the GUI map.

In the GUI Map Editor, choose **File** > **Exit**.

8 Close the Flight Reservation application.

Choose **File** > **Exit**.

11

Where Do You Go from Here?

Now that you have completed the exercises in Lessons 1 through 10, you are ready to apply the WinRunner concepts and skills you learned to your own application.

In this lesson, you will learn about:	On page:
Getting Started	119
Getting Additional Information	121

Getting Started

Before you start testing your application, you must decide which GUI map mode you want to use.

If you used the GUI Map File per Test mode throughout this tutorial and you want to continue in this mode, you can start recording tests right away.

If you want to use the Global GUI Map file mode, you should use the RapidTest Script wizard to learn a description of every object your application contains. Before doing this, however, remove the sample application's object descriptions from the GUI map.

To get started in Global GUI Map file mode:

1 **Close all applications on your desktop except for WinRunner and the application you want to test.**

2 **Close the flight4a.GUI map.**

Choose **Tools** > **GUI Map Editor**. Choose **View** > **GUI Files**. Make sure you are viewing the **flight4a.GUI** map file. Choose **File** > **Close** to close the **flight4a.GUI** map file. Choose **File** > **Exit** to close the GUI Map editor.

3 **Remove the flight4a.GUI map from your startup script.**

In WinRunner, choose **File** > **Open.** Browse to <**WinRunner Installation path**>**dat**. Select the **myinit** file. Delete the line:

GUI_load("<WInRunner installation path>\\dat\\flight4a.GUI");

Choose **File** > **Save** to save the startup script.

4 **Run the RapidTest Script Wizard on your application. Learn object descriptions in Comprehensive mode.**

You should now use the RapidTest Script Wizard to learn a description of each object in your application. Choose **Insert** > **RapidTest Script Wizard** and follow the instructions on the screen.

When the wizard asks you to choose a learning flow, choose **Comprehensive**. This mode enables you to control how WinRunner learns object descriptions. It enables you to customize logical names and map custom objects to a standard object class.

After the learning process is complete, the wizard creates a GUI map file and a startup script. If you are working in a testing group, store this information on a shared network drive.

If you need help while using the wizard, click the **Help** button in the appropriate screen.

5 Create tests.

Once you finish using the wizard, you can start creating tests in WinRunner in the Global GUI Map file mode. Use recording, programming, or a combination of both to build your automated test scripts.

To get started in GUI Map File per Test mode:

1 Close all applications on your desktop except for WinRunner and the application you want to test.

2 Create tests.

Plan the main stages of the test you wish to create, and select the appropriate record mode. Use recording, programming, or a combination of both to build your automated test scripts.

Getting Additional Information

For more information on WinRunner and TSL, refer to the user's guides and online resources provided with WinRunner.

Online Resources

WinRunner includes the following online resources:

Read Me provides last-minute news and information about WinRunner (**Start** > **Programs** > **WinRunner** > **Read Me**).

What's New in WinRunner describes the newest features in your version of WinRunner.

Books Online displays the complete documentation set in PDF format. Online books can be read and printed using Adobe Acrobat Reader. It is recommended that you use version 5.0 or later. You can download Adobe Acrobat Reader from www.adobe.com. Check Mercury's Customer Support Web site for updates to WinRunner online books.

WinRunner Context Sensitive Help provides immediate answers to questions that arise as you work with WinRunner. It describes menu commands and dialog boxes, and shows you how to perform WinRunner tasks. Check the Mercury Customer Support Web site for updates to WinRunner help files.

TSL Online Reference describes Test Script Language (TSL), the functions it contains, and examples of how to use the functions. Check the Mercury Customer Support Web site for updates to the *TSL Online Reference*.

Sample Tests includes utilities and sample tests with accompanying explanations. Check the Mercury Customer Support Web site for updates to WinRunner sample tests.

WinRunner Quick Preview opens an overview of WinRunner in your default browser.

Technical Support Online uses your default web browser to open the Mercury Customer Support Web site.

Support Information presents the Mercury home page, its Customer Support Web site, and links to help you find and contact Mercury offices around the world.

Mercury Interactive on the Web uses your default Web browser to open the Mercury home page. This site provides you with the most up-to-date information on Mercury, its products and services. This includes new software releases, seminars and trade shows, customer support, and training.