XRunner®

Tutorial

Version 6.0

MERCURY INTERACTIVE

XRunner Tutorial—Version 6.0

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

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

XRTUT6.0/01

Table of Contents

Welcome to the XRunner Tutorial

Welcome to the XRunner Tutorial, a self-paced guide which teaches you the basics of testing your application with XRunner. This tutorial is designed to familiarize you with the process of creating and executing automated tests, and analyzing the test results.

The tutorial is divided into 11 short lessons. In each lesson you will create and run tests on the sample Airspace flight reservation application.

This tutorial focuses on context sensitive recording. In order to use the tutorial, make sure that the Airspace application is linked to Mercury Interactive's Context Sensitive libraries.

After completing the tutorial, you can apply the skills you learned to your own application.

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

Lesson 2, Getting Started with Test Wizard shows you how to use the Test Wizard to quickly generate tests and to teach XRunner descriptions of the GUI (Graphical User Interface) objects in an application. When you execute tests, XRunner uses these descriptions to locate the objects in your application. After using the wizard, you will run a test and examine the results.

Lesson 3, Recording Tests teaches you how to record a test script and explains the basics of Test Script Language (TSL)—Mercury Interactive'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, Reading Text teaches you how to read and check text found in GUI objects and bitmaps.

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

Lesson 10, Maintaining Your Test Scripts teaches you how to update the GUI object descriptions learned by XRunner, 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 XRunner.

As you work through the lessons, keep in mind that you are learning the basics for testing any application. Each lesson should take approximately 20 minutes, but you can take as much time as you need.

1

Introducing XRunner

This lesson:

- describes the benefits of automated testing
- introduces the XRunner testing process
- explains how to set up your UNIX environment for XRunner
- takes you on a short tour of the XRunner user interface

The Benefits of Automated Testing

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

Automated testing with XRunner dramatically speeds 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 XRunner executes tests, it simulates a human user by moving the mouse cursor over the application, clicking on GUI objects, and entering keyboard input—but it does this faster than any human user.

With XRunner you can also save time by running batch tests overnight. You simply start the batch run before you leave work in the evening, and review the results when you return in the morning.

Benefits of Automated Testing	
Fast	XRunner executes tests faster than human users.
Reliable	Tests perform 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 pull 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 XRunner testing process consists of 5 main phases:

1. Start by running the Test Wizard on your application in order to teach XRunner a description of every GUI object the application contains. The wizard automatically generates a series of tests which you can immediately run on your application.
2. Create additional test scripts that simulate use of the application and test its functionality. Use recording and/or programming to build test scripts written in Mercury Interactive's Test Script Language (TSL).
3. Debug the tests to check that they operate smoothly and without interruption.
4. Run the tests on a new version of the application in order to verify the application's behavior.

5 Examine the test results to pinpoint defects in the application.

Setting up the XRunner Environment

XRunner uses two environment variables: M_ROOT and MERCURY_ELMHOST. Make sure that both of these variables are defined, and add $M_ROOT and $M_ROOT/bin to your PATH before attempting to run XRunner with the xrun command.

Test results in XRunner's report forms are color-coded to illustrate when the test, or parts of it, pass or fail. The default "pass" and "fail" colors are green and red (XR_PASS_COLOR and XR_FAIL_COLOR configuration parameters, respectively). If you are using a monochrome display, you may find it necessary to change the color configuration parameters to lighter or darker colors. You can open the configuration form by selecting the Options > Configure command in the main XRunner menu.

In addition, if you are using a monochrome display, add the following line to your .Xdefaults file:

```
xrunner*topShadowColor: black
```

The Airspace Application

The sample application under test (AUT) provided with XRunner is called Airspace, and it is located in $M_ROOT/tutorial. The tutorial directory contains several versions of the application so that both successful and unsuccessful test results can be demonstrated. In this tutorial you will be using Airspace1a and Airspace1b. The tutorial directory must be your current directory when you run the Airspace applications.

Exploring the XRunner Window

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

To open XRunner:

At the command prompt, type

```
xrun &
```

Each test you create or run is displayed by XRunner in a scrollable test area. You can open many tests at one time, and select which one is displayed by using the File > Switch to command.

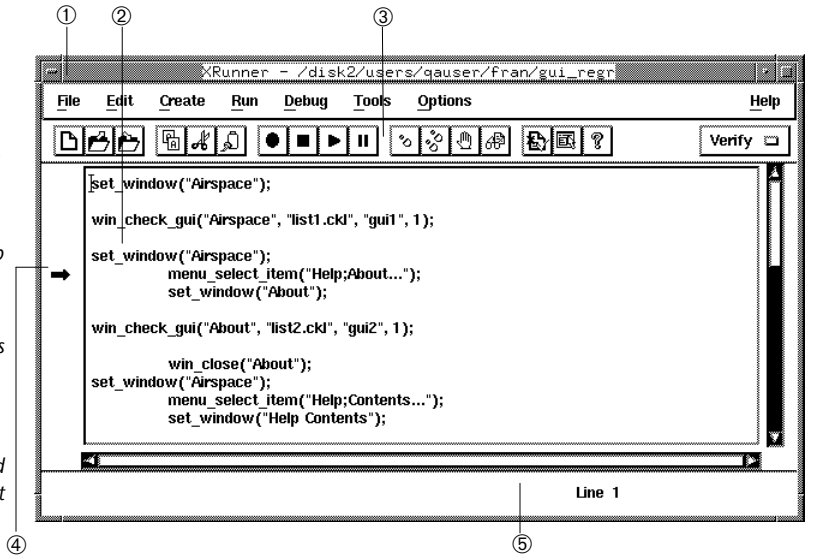
1 *The XRunner window displays one of the open tests*

2 *You use the scrollable test area to record, program, and edit test scripts.*

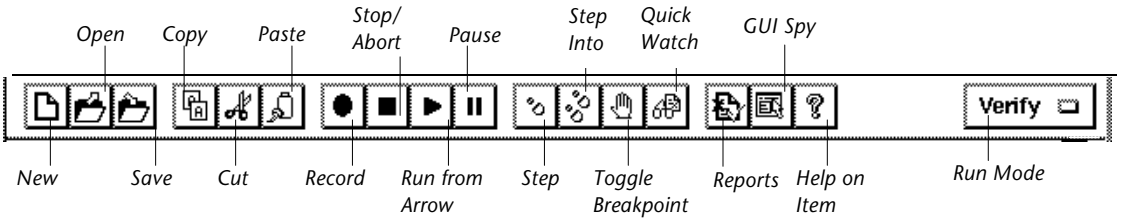
3 *Buttons in the toolbar help you quickly open, execute and save tests.*

4 *The execution arrow marks the line currently being executed by XRunner*

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



The toolbar provides easy access to frequently performed testing tasks, such as opening, executing, and saving tests, and viewing test results.



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

2

Getting Started with Test Wizard

This lesson:

- describes how XRunner identifies GUI objects in an application
- explains how to use the Test Wizard to learn descriptions of GUI objects and to generate tests
- shows you how to execute a test
- helps you analyze the test results

How Does XRunner Identify GUI Objects?

GUI applications are made up of GUI objects, such as windows, buttons, lists, and menus. Before you begin creating and running tests on an application, you should use the Test Wizard to learn a description of all the GUI objects it contains. The wizard opens windows, examines their GUI objects, and saves the object descriptions in a *GUI map file*. Later, when you run your tests, XRunner uses this file to identify and locate objects.

When XRunner learns a description of a GUI object, it looks at the object's physical properties, or *attributes*. Each GUI object has many attributes such as "class", "label", "width", "height", "handle", and "enabled." However, XRunner only learns the attributes which uniquely distinguish an object from all other objects in the application.

For example, when XRunner looks at an OK button, it might see that the button is located in an Open window, belongs to the pushbutton object class, and has the text label "OK".

Spying on GUI Objects

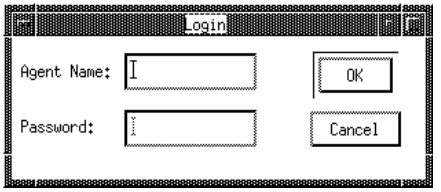
To help you understand how XRunner identifies GUI objects, examine the objects in the sample Airspace application.

1 Open the Airspace application.

At the command prompt, enter the following two commands:

```
cd $M_ROOT/tutorial
./airspace1a &
```

A Login window opens.



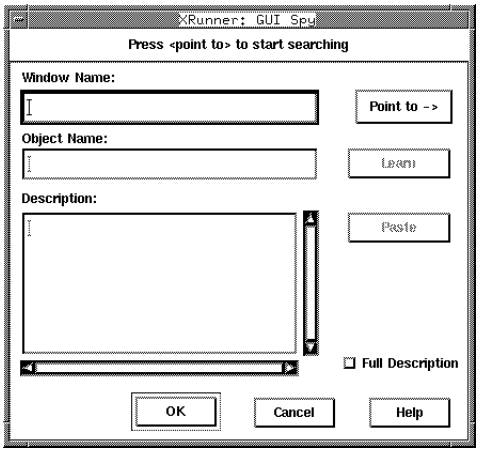
2 Open XRunner.

If XRunner is not already open, type xrun & at the command prompt.

3 Open the GUI Spy. This tool lets you "spy" on the attributes of GUI objects.

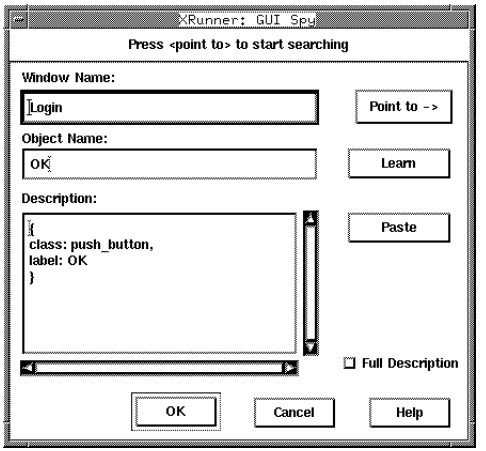
Open the XRunner Tools menu and select the GUI Spy command, or select the GUI Spy button on the toolbar. The GUI Spy opens. Position the GUI

Spy on the desktop so that both the Login window and GUI Spy are clearly visible.



4 View the attributes that provide a unique description of the OK button.

In the GUI Spy, click on the Point to button and select Object. 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 attributes. Place the pointer over the OK button and when the description is visible in the GUI Spy, press F6. This freezes the OK button's description in the GUI Spy.



5 **Examine the attributes of the OK button.**

In the Description field, the attribute names are listed on the left; attribute values are listed after the colon. For example, “label: OK” indicates that the button has the text label “OK”, and “class: push_button” indicates that the button belongs to the pushbutton object class. At the top of the form, the GUI Spy also displays the name of the window in which the object is located.

As you can see, XRunner needs only a few attributes to uniquely identify the object.

6 **View an expanded list of attributes for the OK button.**

Activate the “Full Description” checkbox. Then click on the Point to button and select Object. Move the pointer over the OK button. After the button flashes, the expanded list of attributes of the button, including those not used by XRunner, is displayed in the Description field. Scroll through the list to view all the attributes.

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

Move the pointer over other GUI objects in the Login window to observe the expanded list of their attributes. Press F6 to freeze an object description in the GUI Spy.

8 **Exit the GUI Spy.**

Click Cancel.

Using the Test Wizard

The Test Wizard enables you to quickly start the testing process. You should run this wizard before starting to create test scripts.

The Test Wizard performs two important tasks:

- It systematically opens the windows in your application and learns a description of every GUI object. The wizard stores this information in a GUI map file.

➤ It automatically generates tests based on the information it learned as it navigated through the application.

To observe XRunner's learning and test creation processes, use the Test Wizard on the Airspace application.

1 Log in to the Airspace application.

If the Login window is open, type your name and the password *mercury* and press OK.

If the Login window is not already open on your desktop, type the following commands:

```
cd $M_ROOT/tutorial
./airspace1a &
```

and then log in.

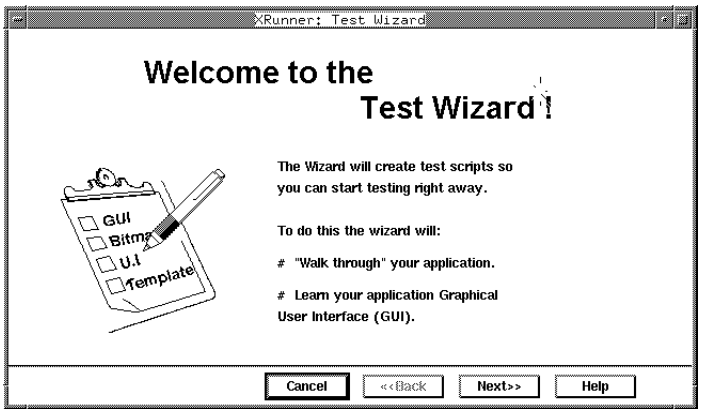
2 Open XRunner and a new test.

If XRunner is not already open, type xrun & at the command prompt. The main XRunner window is displayed.

If XRunner is already open, select File > New to create a new test.

3 Start the Test Wizard.

In XRunner, select Create > Test Wizard. Click Next in the wizard's Welcome screen to advance to the next screen.



4 Indicate the application you want to test.

Click on the Point to button and then click on the Airspace application. The application name appears in the wizard. Click Next.

5 Select the GUI Regression test.

The wizard can generate tests automatically. You can select from the following tests:

GUI Regression Test - uses checkpoints to examine the user-interface elements of different versions of an application.

Bitmap Regression Test - compares bitmaps between different versions of an application.

User Interface Test - runs WidgetLint checks on Motif widgets in an application.

Test Template - creates a test script template for future use.

For the purposes of the exercise, check that the GUI Regression test is the only selected test. For more information about GUI checkpoints, see Chapter 5, Checking GUI Objects. Click Next.

6 Accept the default navigation controls.

Navigation ("Go To") controls tell XRunner which GUI objects are used to open windows. The Airspace application uses the default navigation controls (... and > >) so you do not need to define additional controls. Make sure that the checkbox for "Pause to confirm for each window" is off. Click Next.

7 Set the learning flow to "Express."

The learning flow determines how XRunner will walk through your application. Two modes are available: *Express* and *Comprehensive*. Comprehensive mode lets you customize how the wizard learns descriptions of GUI objects. First-time XRunner 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 or not currently displayed, press the Continue button in the message box.

8. **Accept "Do not invoke" in the Start Application screen.**

   You can choose to have XRunner automatically open the Airspace application each time you start XRunner. Accept the default "Do not invoke application." Click Next.

9. **Save the GUI information and a startup script.**

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

   The wizard also creates a startup script. This script runs automatically each time you start XRunner. It contains a statement in the following format that loads the GUI map file so that XRunner will be ready to test your application:

   ```
   GUI_load ("/u/qauser/fran/air_gui" );
   ```

10. **Specify a full path and the file name air_gui for the GUI map file (for example, /u/qauser/fran/air_gui).**

    Each time you start XRunner, XRunner looks for a script in a directory named **tslinit** under your home directory, to be used as the startup script. To insure that the startup script you have just created is used in future sessions, save it as **tslinit** your home directory (for example, /u/qauser/fran/tslinit). XRunner will create a directory **tslinit** if it does not yet exist, and store the script in this directory. Click Next.

11. **Save the GUI Regression test.**

12. **Specify a path and file name (GUI_regr) for the GUI Regression test and click Next.**

13. **Press OK in the Congratulations screen.**

    The GUI Regression test is displayed in the XRunner window.

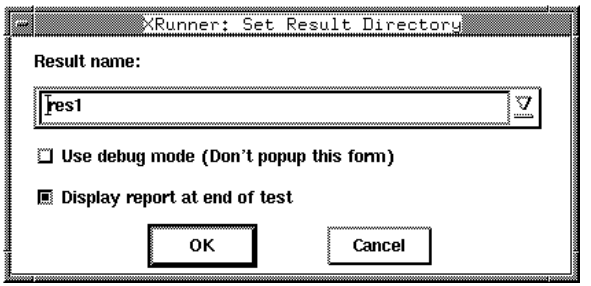
Running the GUI Regression Test

You are now ready to run the GUI Regression test script on the Airspace application. The GUI Regression test creates a detailed description of the GUI elements of your application. You can later use it to run regression tests on subsequent versions of the application to verify that the GUI elements have not been affected.

To run the GUI Regression test:

1. **Check that XRunner and the Airspace application are still open on your desktop.**
2. **Make sure that the GUI Regression test is the current XRunner test.**
3. **If necessary, switch to the GUI Regression test by choosing the File > Switch to command and selecting GUI_regr from the list.**
4. **Select Run from Top.**

   Select Run > Run from Top. The Set Results Directory form opens.



5. **Choose a Results Directory name.**

   At this point you are asked to define the name of the directory in which XRunner will store the results of the test. Use the default name “res1.”

   Note that at the bottom of the form is a “Display report at end of test” checkbox. When selected, XRunner automatically displays the test results when the test run is completed. Make sure that this checkbox is selected.

6. **Run the GUI Regression test.**

   Click OK in the Set Results Directory form. XRunner immediately begins running the GUI Regression test. Watch how XRunner opens each window in the Airspace application.

7. **Review the test results.**

   When the test run is completed, the test results automatically appear in the XRunner report form. 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 XRunner report form. XRunner color-codes results (green = passed, red = failed) so that you can quickly draw conclusions about the success or failure of a test.

1 **Make sure that the XRunner report form is open and displays the results of the GUI Regression test.**

If the XRunner report form is not currently open, select Tools > Reports, or click the Reports button.

2 **Review the results that summarize the analysis of the GUI objects of the Airspace application.**

For a more detailed report, select the *Test Log* option in the Display list.

3 **Close the report.**

Select File > Exit in the XRunner report form.

4 **Close the GUI Regression test.**

Select File > Close.

5 **Close the Airspace application.**

Select File > Exit.

3

Recording Tests

This lesson:

- describes Analog and Context Sensitive record modes
- shows you how to record a test script
- helps you read the test script
- lets you run the recorded test and analyze the results

Choosing a Record Mode

By recording, you can quickly create automated test scripts. You simply work with your application as you normally would, clicking on objects with the mouse and entering keyboard input. XRunner records the operations you perform and generates statements in TSL, Mercury Interactive's Test Script Language. These statements appear as a script in the XRunner 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 the operations you perform in terms of the GUI objects in your application. XRunner identifies each object you operate on (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 Airspace Login window, XRunner records this TSL statement in your test script:

```
button_press ("OK");
```

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

Note that you can only record in Context Sensitive mode if an application under test (AUT) is linked to Mercury Interactive's Context Sensitive libraries. If you do not know if the application is linked, consult with your system administrator.

Analog

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

Recorded statements	meaning...
move_locator_track (1);	*mouse track*
mtype ("<T110><kLeft>-");	*left mouse button press*
mtype ("<kLeft>+");	*left mouse button release*

When you run the test, XRunner retraces the recorded movements using absolute screen coordinates. In order for XRunner to execute the test correctly, your application must be located in the same position on the desktop as when the test was created, and have the same user interface.

Analog mode is most useful when the exact mouse movements are an important part of your test, for example, when creating a drawing.

When choosing a record mode, consider the following points:

Choose Context Sensitive if...	Choose Analog if...
You can link to Context Sensitive libraries.	You cannot link to Context Sensitive libraries.
Exact mouse movements are not required.	Exact mouse movements are required.
You plan to maintain your test scripts and reuse them on different versions of the application.	You do not plan to reuse your test scripts.
Your application is built using a toolkit supported by XRunner.	Your application is built using a toolkit not supported by XRunner
	The application contains bitmap areas.

If you are testing an application which contains both GUI objects and bitmap areas, you can switch between modes as you record.

Recording an Analog Test

In this exercise you will create a script which tests the process of opening an order in the Airspace application. You will record the script in Analog mode.

1 **Open XRunner.**

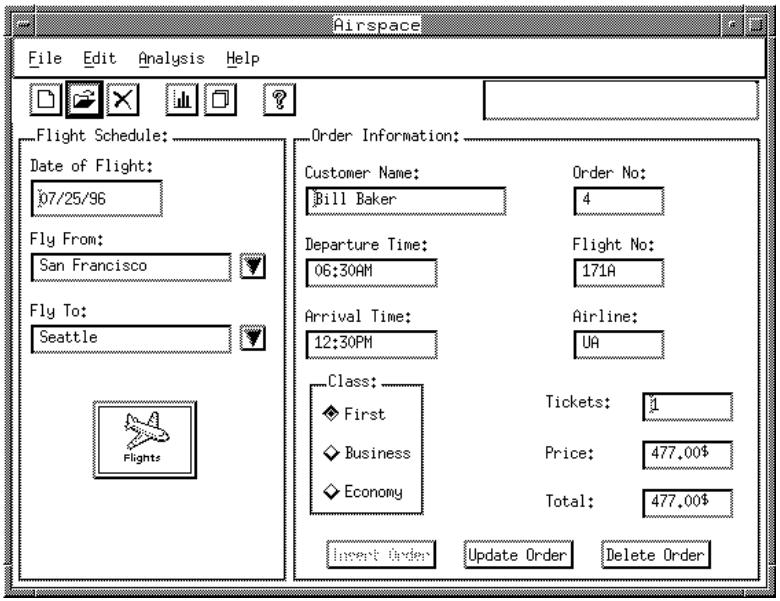
At the command prompt, type xrun &. A blank test opens in XRunner.

2 **Open the Airspace application and log in.**

3 **At the command prompt, enter the following two commands:**

```
cd $M_ROOT/tutorial
./airspace1a &
```

In the Login window, type your name and the password *mercury*, and click OK. The Airspace main window opens.



4 Start Recording in Analog mode.

Select Create > Record—Analog. From this point on, XRunner will record all mouse movements, as well as mouse clicks and keyboard input. You may notice that the application responds more slowly to your input as XRunner processes the actions being recorded.

5 Open order #5.

In the Airspace application, select File > Open order. In the Open Order form, click the Order number checkbox. In the Value field type 5, and press OK.

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

6 Stop Recording.

Use the Stop Record softkey (F6) to stop recording.

7 Save the test.

Select File > Save As or click the Save button. Save the test as *lesson3a* in a convenient location on your hard drive. Click OK to close the Save As form.

After you have recorded the test, keep the Airspace application and XRunner open, and leave the application in its current position on your desktop.

XRunner saves the *lesson3a* test in the file system as a *directory*, not a file. This directory stores the test script and the results generated when you run the test.

Understanding the Analog Test Script

In the previous exercise you recorded the process of opening a flight order in the Airspace application. As you worked, XRunner generated a test script similar to the following:

```
move_locator_track (1);
mtype ("<T496><kLeft>-<T186><kLeft>+", 1);
move_locator_track (2);
mtype ("<T868><kLeft>-<T124><kLeft>+", 2);
move_locator_track (3);
mtype ("<T62><kLeft>-<T186><kLeft>+");
move_locator_track (4);
mtype ("<T186><kLeft>-<T186><kLeft>+");
type ("<t1>3");
move_locator_track (5);
```

These recorded TSL statements correspond to mouse movements (move_locator_track), mouse button clicks (mtype) and keyboard input (type). The script thus simulates the actions of a user navigating through the various buttons and windows of the application.

Running the Test and Analyzing the Results

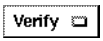
You are now ready to run your recorded test script and to analyze the test results. XRunner provides three modes for running tests. You select a mode from the toolbar.

- Use *Verify mode* when running a test to check the behavior of your application and when you want to save the test results in a directory.
- Use *Debug mode* when you want to check that the test script runs smoothly without errors in syntax. See Lesson 7 for more information.
- Use *Update 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 XRunner and the Airspace application are open on your desktop.**
2. **Make sure that lesson3a is the current test displayed in XRunner.**

   Choose the File > Switch to command to change the current test.



3. **Make sure that Verify mode is selected in the toolbar.**
4. **Select Run from Top.**

   Select Run > Run from Top. The Set Results Directory form opens.

5. **Specify a Results Directory name.**

   Accept the default name “res1.” Also check that the “Display report at end of test” checkbox is selected.

6. **Run the test.**

   Click OK in the Set Results Directory form. XRunner starts executing the test. *Watch how XRunner opens windows and selects objects.*

   Note that if you are running XRunner on a Sun machine under an MIXTrap X server, the script may not be able to execute the signature you created in Analog mode as expected.

7 **Review the results.**

When the test run is completed, the test results appear in the XRunner report form. Note that the test result is “pass”, indicating that the test was run successfully.

8 **Close the results.**

In the report form, select File > Exit.

9 **Close the *lesson3a* test.**

Select File > Close.

Recording a Context Sensitive Test

You will now create a new script that tests the process of opening an order using Context Sensitive recording.

1 **Open XRunner and a new test.**

If XRunner is not already open, type xrun & at the command prompt. The main XRunner window is displayed.

If XRunner is already open, select File > New to create a new test.

2 **Start the Airspace application and log in.**

At the command prompt, enter the following two commands:

```
cd $M_ROOT/tutorial
./airspace1a &
```

In the Login window, type your name and the password *mercury*, and click OK. Reposition the Airspace application and XRunner so that they are both clearly visible on your desktop.

3 **Begin a new test.**

In XRunner, check that no tests are displayed in the test area. If you have a previously created file open, select File > New.

4 **Start Recording in Context Sensitive mode.**

Select Create > Record—Context Sensitive or click the Record button on the toolbar.

5 Open order #3.

In the Airspace application, select File > Open order. In the Open Order form, click the Order number checkbox. In the adjacent field type 3, and press OK.

6 Stop Recording.

In XRunner, select Create > Record—Stop or click the Stop button on the toolbar.

7 Save the test.

Select File > Save As or click the Save button. Save the test as *lesson3b* in a convenient location on your hard drive. Click OK to close the Save As form.

Understanding the Context Sensitive Test Script

In the previous exercise, you recorded the process of opening a flight order in the Airspace application in Context Sensitive mode. As you worked, XRunner generated a test script similar to the following:

```
set_window ("Airspace", 10);
menu_select_item ("File;Open order...");
set_window ("Open Order", 10);
button_set ("Order number", ON);
edit_set("Order Value","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, XRunner generated a **menu_select_item** statement.

The following points will help you understand your test script:

- When you click on an object, XRunner assigns the object a *logical name*, usually the object's text label. The logical name makes it easy for you to read the test script.

  For example, when you clicked on the Order number button, XRunner recorded the statement:

```
button_set ("Order number", ON);
```

"Order number" is the object's logical name.

- XRunner generates a **set_window** statement each time you begin working in a new window. The statements following **set_window** perform operations on objects within this window. For example, when you began using the Open Order window, XRunner recorded the statement:

```
set_window ("Open Order", 10);
```

- When you enter keyboard input, XRunner generates an **edit_set** statement in the test script. For example, when you typed 3 in the Order Number field, XRunner generated the statement:

```
edit_set("Order Value","3");
```

Using Both Analog and Context Sensitive Modes

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

1 **In the *lesson3b* test, place the cursor below the last line of the script.**

You will add the new test segment to the *lesson3b* test. If the test is not already open, select File > Open and specify the location of the test. If the test is open but not displayed, select File > Switch to and select the test. Place the cursor below the last line of the test.

2 **Start Recording in Context Sensitive mode.**

Select Create > Record—Context Sensitive or click the Record button on the toolbar.

3 **Open the Fax Order form.**

In the Airspace application, select File > Fax order.

4 **Click the "Send Signature With Order" checkbox.**

5 **Sign the fax in Context Sensitive mode.**

6 **Sign your name in the Agent Signature field.**

7 **Clear the signature.**

Click the Clear Signature button.

8 **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.

9 **Sign the fax again in Analog mode.**

Press F5 on your keyboard or click the Record button again to switch to Analog mode. Sign your name in the Agent Signature area.

10 **Switch back to Context Sensitive mode and cancel the fax.**

Press F5 to switch back to Context Sensitive mode. Click Cancel.

11 **Stop Recording.**

Select Create > Record—Stop or click the Stop button.

12 **Save the test.**

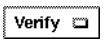
Select File > Save or click the Save button.

Running the Test and Analyzing the Results

You are now ready to run the second test script and to analyze the test results.

To run the test:

1. **Check that XRunner and the Airspace application are open on your desktop.**
2. **Make sure that *lesson3b* is the current test displayed in XRunner.**
3. **If the test is not already open, select File > Open. If it is open but not displayed, select File > Switch to, to change the current test.**



4. **Make sure that Verify mode is selected in the toolbar.**
5. **Select Run from Top.**

   Select Run > Run from Top. The Set Results Directory form opens. Accept the default results directory name “res1.” Also check that the “Display report at end of test” checkbox is active.

6. **Run the test.**

   Click OK in the Set Results Directory form. XRunner starts executing the test.

   Watch how XRunner opens windows and selects objects. Also watch what happens when XRunner draws the fax signature in Context Sensitive mode, and in Analog mode.

   Note that if you are running XRunner on a Sun machine under an MIXTrap X server, the script may not be able to execute the signature you created in Analog mode as expected.

7. **Review the results.**

   When the test run is completed, the test results appear in the XRunner report form. Note that the test result is “pass”, indicating that the test was run successfully.

8. **Close the results.**

   In the report form, Select File > Exit.

9 **Close the *lesson3b* test.**

Select File > Close.

10 **Close the Airspace application.**

Select File > Exit.

Recording Tips

- Before starting to record, you should close applications that are not required for the test.
- Make sure the test leaves the application under test in the state it was in when the test 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 XRunner 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 XRunner to accurately execute 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 on an object whose description was not learned by the Test Wizard, XRunner learns a description of the object and adds it to a temporary GUI map file. For more information, refer to Chapter 4, “Creating the GUI Map” in your *XRunner User’s Guide*.
- In XRunner there are several ways to stop recording: by the Create > Record—Stop menu command, by the Stop button on the toolbar, and by a softkey (F6). Similarly, there are several ways to switch between Analog and Context Sensitive recording modes: by the Create > Record menu commands, by the Record button on the toolbar, and by a softkey (F5). It is recommended that you use the softkeys (F6 or F5) when leaving Analog mode, to insure that no actions caused by accessing the menu commands or the toolbar are inserted into the test script.

4

Synchronizing Tests

This lesson:

- describes when you should synchronize a test
- shows you how to synchronize a test
- lets you run the test and analyze the results

When Should You Synchronize?

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

XRunner waits a set amount of time—by default, 10 seconds—for an application to respond to input. If the application responds slowly during a test run, the XRunner's default wait time may not be enough, and XRunner may try to continue the test before the application is ready. The test run will then unexpectedly fail.

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

- increase the default time that the XRunner waits. To do so, you change the value of the XR_TIMEOUT parameter in the Configuration form (Options >

Configure). This method affects all your tests and will slow many other Context Sensitive operations.

or:

- insert a *synchronization point* into the test script at the exact point that the problem occurs. A synchronization point tells XRunner to pause the test run in order 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 Airspace application and inserts the order into the database
- run the test and identify a synchronization problem
- add a synchronization point to the test
- run the test again

Creating a Test

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

1 **Open XRunner and a new test.**

If XRunner is not already open, type xrun & at the command prompt. The main XRunner window is displayed.

If XRunner is already open, select File > New to create a new test.

2 **Set the Progress Bar Delay variable for Airspace.**

For this lesson, it is necessary to set an environment variable named AIR_BAR_DELAY, which will enable Airspace to simulate a lengthy transaction. Enter the following at the command prompt:

```
setenv AIR_BAR_DELAY 20
```

3 Open the Airspace application and log in.

At the command prompt, enter the following two commands:

```
cd $M_ROOT/tutorial
./airspace1a &
```

In the Login window, type your name and the password *mercury*, and click OK. Reposition the Airspace application and XRunner so that they are both clearly visible on your desktop.

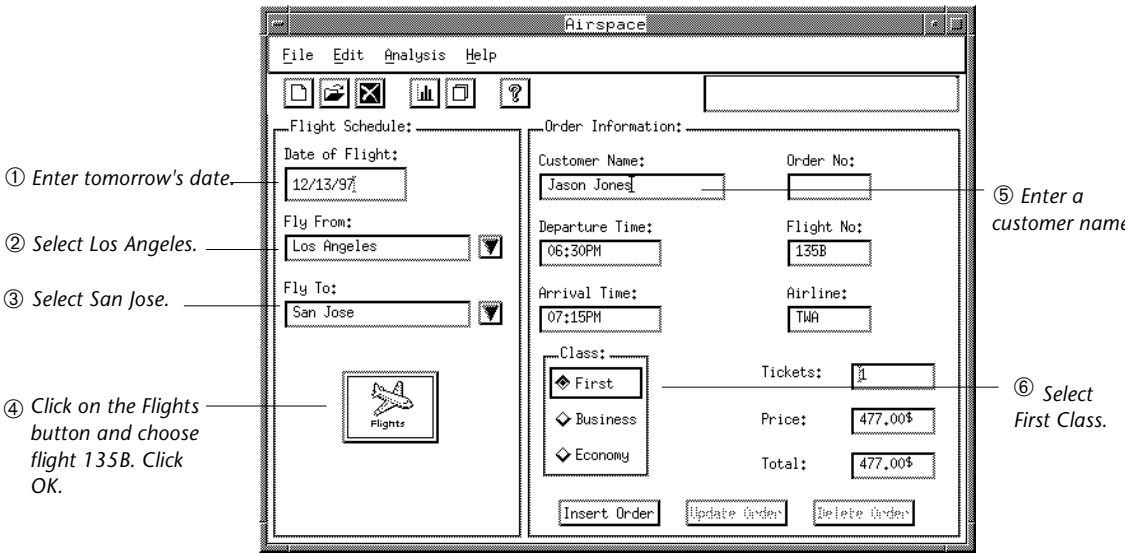
4 Start Recording in Context Sensitive mode.

Select Create > Record—Context Sensitive or click the Record button on the toolbar.

5 Create a new order.

Select File > New order in the Airspace application.

6 Fill in flight and passenger information.



7 Insert the order into the database.

Click the Insert Order button. When the insertion is complete, the message "Insert Done" appears in the status area at the top of the window.

8 Stop Recording.

Select Create > Record—Stop or click the Stop Recording button.

9 Save the test.

Select File > Save As. Save the test as *lesson4* in a convenient location on your hard drive. Click OK to close the Save As form.

Identifying a Synchronization Problem

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

1 Make sure that *lesson4* is the current XRunner test.

Use the File > Switch to command to display *lesson4* if it is not displayed.

2 Select Run from Top.

Select Run > Run from Top. The Set Results Directory window opens. Accept the default name "res1."

3 Run the test.

Click OK in the Set Results Directory form. XRunner starts executing the test. *Watch what happens when XRunner attempts to press the Delete button.*

4 Click Pause in the XRunner message window.

XRunner failed to press the Delete Order button because the button is still disabled. *This error occurred because XRunner did not wait until the Insert Order operation was completed.*

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, XRunner will wait for the Insert Done message to appear before it attempts to press the Delete Order button.

1 **Make sure that the *lesson4* test window is the current XRunner test.**

Use the File > Switch to command to display lesson4 if it is not yet displayed.

2 **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.

3 **Synchronize the test so that it waits for the "Insert Done" message to appear in the status area.**

Select the Create > Wait Bitmap > Object command.

Using the ☞ pointer, click on the status area at the top right corner of the Airspace window. XRunner automatically inserts an **obj_wait_bitmap** synchronization point into the test script.

This statement instructs XRunner to wait 10 seconds for the "Insert Done" message to appear in the status area.

> Synchronization points appears as **obj_wait_bitmap** and **win_wait_bitmap** statements in the test script. For example:
>
> obj_wait_bitmap("Done", "Img1", 10);
>
> *Done* is the object's logical name.
>
> *Img1* is the file containing a captured image of the object.
>
> *10* is the time (in seconds) that XRunner waits for the image to appear in the application. This time is added to the default time defined by the XR_TIMEOUT configuration parameter. (In this example, XRunner waits a total of 20 seconds.)

Change the timeout value in the **obj_wait_bitmap** statement from 10 seconds to 30 seconds to insure that the Airspace application has enough time to complete the insertion of the new order.

```
obj_wait_bitmap("(Static)", "Img1", 30);
```

4 Save the test.

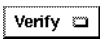
Select File > Save or click the Save button.

Running the Synchronized Test

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

1 Make sure that the *lesson4* test is the current XRunner test.

Use the File > Switch to command to display *lesson4* if it is not yet displayed.



2 Check that Verify mode is selected in the toolbar.

Verify mode will stay in effect until you choose a different mode.

3 Select Run from Top.

Select Run > Run from Top. The Set Results Directory form opens. Accept the default name “res2.” Make sure that the “Display report at end of test” checkbox is selected in the Set Results Directory form.

4 Run the test.

Click OK in the Set Results Directory form. XRunner starts executing the test from the first line in the script. Watch how XRunner waits for the “Insert Done” message to appear in the status bar.

5 Review a summary of the results.

When the test run is completed, the test results appear in the XRunner report form.

6 View the test log.

There are several formats for viewing results: these are options in the Display option menu. You can select *Summary* to view a general survey of the test,

and *All Captures* to view a complete list of GUI data stored in the results directory. Now, select *Test Log*. A list of the test events appears.

Note that a "wait for bitmap" event appears in green. This indicates that synchronization was performed successfully. You can double-click on this event to see a bitmap image of the status bar.

7 Close the report.

Select File > Exit.

8 Close the *lesson4* test.

Select File > Close in XRunner.

9 Close the Airspace application.

Select File > Exit.

10 Reset the AIR_BAR_DELAY **variable.**

If you want the Insert Order, Update Order and Delete Order transactions in Airspace to run faster, completing in less than 20 seconds, reset the AIR_BAR_DELAY environment variable now. For example, to set the delay to 8 seconds, enter the following at the command prompt:

```
setenv AIR_BAR_DELAY 8
```

To learn about additional synchronization methods, read Chapter 13, "Synchronizing Test Execution: Context Sensitive Testing," Chapter 14, "Synchronizing Test Execution: Analog Testing," and Chapter 15, "Enhancing Window Comparison and Synchronization" in your *XRunner User's Guide*.

5

Checking GUI Objects

This lesson:

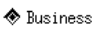
- explains how to check the behavior of GUI objects
- lets you create a test that checks GUI objects
- lets you run the test on different versions of an application and examine the results

How Do You Check GUI Objects?

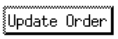
When working with an application, you can determine whether it is functioning properly according to the behavior of its GUI objects. If a GUI object does not respond to input as expected, a defect probably exists somewhere in the application's code.

You check GUI objects by adding GUI checkpoints to your test scripts. A GUI checkpoint examines the behavior of an object's attributes. For example, you can check:

the content of a field

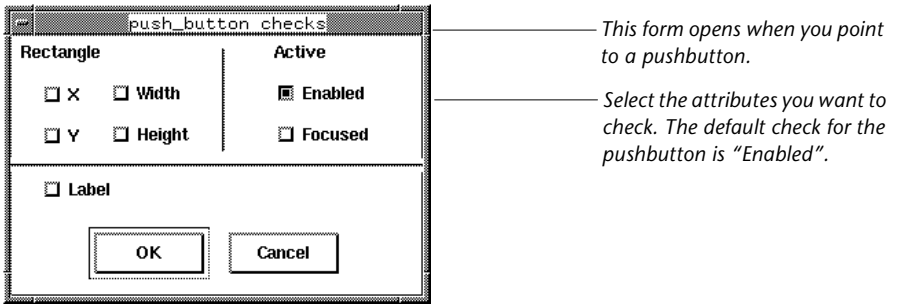


if a radio button is on or off



if a pushbutton is enabled or disabled

To create a GUI checkpoint, you double-click on an object in an application. A check form opens for the object type you selected and you choose the attributes you want to check.



XRunner captures the current values of the selected attributes 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, XRunner compares the *expected* behavior of the object with its *actual* behavior in the application.

Adding GUI Checkpoints to a Test Script

In this exercise you will check that objects in the Airspace Open Order form function properly when you open an existing order.

1 **Open XRunner and a new test.**

If XRunner is not already open, type xrun & at the command prompt. The main XRunner window is displayed.

If XRunner is already open, select File > New to create a new test.

2 **Start the Airspace application and log in.**

At the command prompt, enter the following two commands:

```
cd $M_ROOT/tutorial
./airspace1a &
```

In the Login window, type your name and the password *mercury*, and click OK. Reposition the Airspace application and XRunner so that they are both clearly visible on your desktop.

3 Start Recording in Context Sensitive mode.

Select Create > Record—Context Sensitive or click the Start Recording button on the toolbar.

4 Open the Open Order form.

Select File > Open order in the Airspace application.

5 Create a GUI checkpoint for the Order number button.

Select Create > Check GUI > Object.

Using the pointer, *double-click* on the Order number button. The Checkbutton check form opens and displays the available checks. Accept the default check "State." This check captures the current state (off) of the checkbutton and stores it as expected results.

Click OK in the check form to insert the checkpoint into the test script. The checkpoint appears as an **obj_check_gui** statement.

6 Select the Order number button and enter "4" in the field.

7 Create another GUI checkpoint for the Order number button.

Select Create > Check GUI > Object.

Using the pointer, *single-click* on the Order number button. XRunner immediately inserts a checkpoint into the test script (an **obj_check_gui** statement) which uses the default check "State." (Use this shortcut when you want to use only the default check for an object.) This check captures the current state (on) of the button and stores it as expected results.

8 Create a GUI checkpoint for the Customer Name button.

Select Create > Check GUI > Object.

Using the pointer, *double-click* on the Customer Name button. The Checkbutton check form 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 checkbutton; the "Enabled" check captures the current condition (disabled).

Click OK in the check form to insert the checkpoint into the test script. The checkpoint appears an as **obj_check_gui** statement.

9 Click OK in the Open Order form to open the order.

10 Stop recording.

Select Create > Record—Stop in XRunner.

11 Save the test.

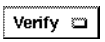
Select File > Save or click the Save button. Name the test *lesson5*. Click OK.

> GUI checkpoints appear as **obj_check_gui** and **win_check_gui** statements in the test script. For example:
>
> obj_check_gui("Order number", "list1.ckl", "gui1", 1)
>
> *Order number* 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 XR_TIMEOUT configuration parameter. See Lesson 4 for more information.

Running the Test

You will now run the *lesson5* test in order to make sure that the test runs smoothly.

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



2 In XRunner, check that Verify mode is selected in the toolbar.

3 Select Run from Top.

Select Run > Run from Top. The Set Results Directory window opens. Accept the default name "res1." Make sure that the "Display report at end of test" checkbox is selected.

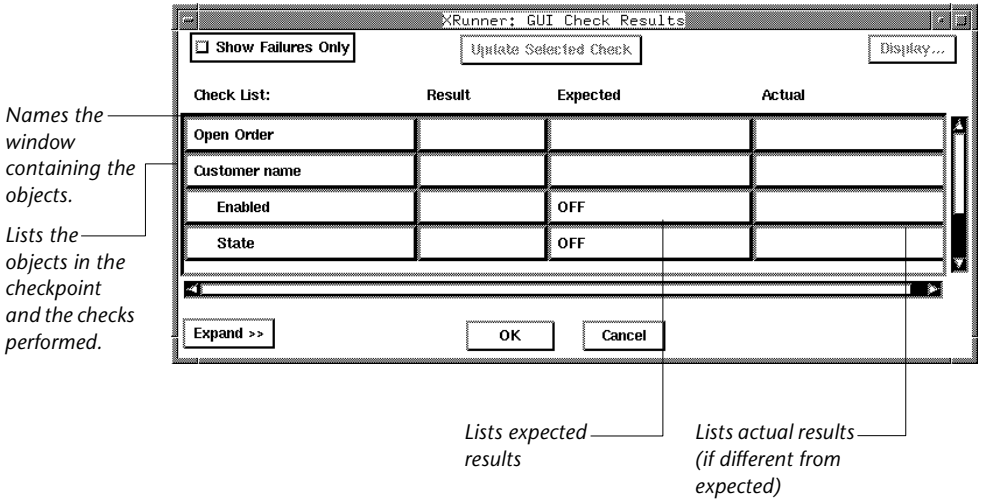
4 Run the test.

Click the OK button.

5 Review the results.

When the test run is completed, the test results appear in the XRunner report form. Select *Test Log* from the Display option menu to view the list of events included in the test, and the result of each event. All "end GUI checkpoint" events should appear in green (indicating success).

Double-click an "end GUI checkpoint" event to view detailed results of a GUI checkpoint. For example, if you double-click on the third end GUI checkpoint, the following table is displayed.



6 Close the results.

Click OK to close the GUI Check Results form and then select File > Exit to close the report.

7 Close the Airspace application.

Select File > Exit.

Running the Test on a New Version

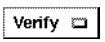
In this exercise you will run the *lesson5* test on a new version of the Airspace application in order to check the behavior of its GUI objects.

1 **Open version 1b of the Airspace application.**

At the command prompt, enter the following two commands:

```
cd $M_ROOT/tutorial
./airspace1b &
```

In the Login window, type your first name and the password *mercury*, and click OK. Reposition the Airspace application and XRunner so that they are both clearly visible on your desktop.



2 **Check that Verify mode is selected in the toolbar.**

3 **Select Run from Top.**

Select Run > Run from Top. The Set Results Directory window opens. Accept the default results directory name "res2." Make sure that the "Display report at end of test" checkbox is selected.

4 **Run the test.**

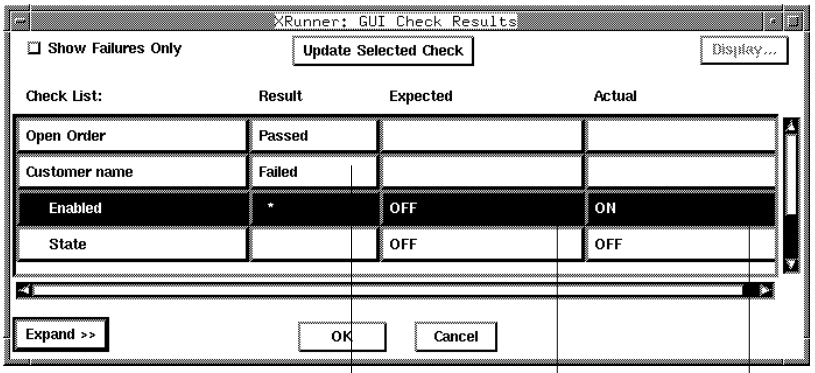
Click OK. The test run begins.

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

5 **Review the results.**

When the test run is completed, the test results appear in the XRunner report form. In the Test Log display, one "end GUI checkpoint" statement appears in red and its Result field lists "mismatch." This indicates that one or more of the checks performed on the object failed.

Double-click on the red “end GUI checkpoint” event to view detailed results.



6 Close the results.

Click OK in the GUI Check Results form and then select File > Exit to close the report.

7 Close the *lesson5* test.

Select File > Close.

8 Close version 1b of the Airspace application.

Select File > Exit.

GUI Checkpoint Tips

- You can create a single GUI checkpoint that checks several or all objects in a window. Select Create > Check GUI > Checklist. A form opens which enables you to point at objects and select checks. When you finish creating the checklist, XRunner inserts a **win_check_gui** statement into the test script.
- For overnight test runs, you can instruct XRunner not to display a message when a GUI mismatch is detected. Select Options > Configure. In the

Configuration form, clear the XR_MISMATCH_BREAK parameter. This enables the test to run without interruption.

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

For more information on GUI Checkpoints, refer to Chapter 8, "Checking GUI Objects" in the *XRunner User's Guide*.

6

Checking Bitmaps

This lesson:

- explains how to check bitmap images in your application
- lets you create a test that checks bitmaps
- lets you run the test in order to compare bitmaps in different versions of an application
- helps you analyze the results

How Do You Check 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.

XRunner 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, XRunner compares the expected bitmap with the actual bitmap in the application. If

any differences are detected, you can view a picture of the differences in the test report.

Expected

Actual

Difference

Adding Bitmap Checkpoints to a Test Script

In this exercise you will test the Agent Signature field in the Fax Order form. You will use a bitmap checkpoint to check that you can sign your name in the field, and then another bitmap checkpoint to check that the field clears when you press the Clear Signature button.

1 Open XRunner and a new test.

If XRunner is not already open, type xrun & at the command prompt. The main XRunner window is displayed.

If XRunner is already open, select File > New to create a new test.

2 Open the Airspace application and log in.

At the command prompt, enter the following two commands:

```
cd $M_ROOT/tutorial
./airspace1a &
```

In the Login window, type your first name and the password *mercury*, and click OK. Reposition the Airspace application and XRunner so that they are both clearly visible on your desktop.

3 Start Recording.

Select Create > Record—Context Sensitive or click the Record button on the toolbar.

4 **Open order #6.**

In the Airspace application select File > Open order. In the Open Order form, click the Order number checkbox and enter “6” in the adjacent field. Click OK to open the order.

5 **Open the Fax Order form.**

Select File > Fax order.

6 **Move the Fax Order form.**

Position the form above the Airspace window.

7 **Switch to Analog mode.**

Press F5 on your keyboard or click the Record button to switch to Analog mode.

8 **Draw an X in the Agent Signature box.**

9 **Switch back to Context Sensitive mode.**

Press F5 on your keyboard to switch back to Context Sensitive mode.

10 **Insert a bitmap checkpoint that checks your signature.**

Select Create > Check Bitmap > Object.

Using the pointer, click the Agent Signature area. XRunner captures the bitmap and inserts an **obj_check_bitmap** statement into the test script.

11 **Press the Clear Signature button.**

The signature is cleared from the Agent Signature area.

12 **Insert another bitmap checkpoint that checks the Agent Signature area.**

Select Create > Check Bitmap > Object.

Using the pointer, click the Agent Signature area. XRunner captures a bitmap and inserts an **obj_check_bitmap** statement into the test script.

13 **Press the Cancel button.**

14 **Stop recording.**

Select Create > Record—Stop or click the Stop button.

15 Save the test.

Select File > Save As or click the Save button. Name the test *lesson6*. Click OK in the Save As form.

> Bitmap checkpoints appear as **obj_check_bitmap** and **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 XR_TIMEOUT configuration parameter. See Lesson 4 for more information.

Viewing Expected Results

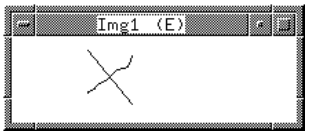
You can now view the expected results of the *lesson6* test.

1 Open the XRunner report.

Select Tools > Reports or click the Reports button. The report form displays a summary of the expected results.

2 View the captured bitmaps.

Select the Test Log display. Click on the first "capture bitmap" event and click the Display button.



Next click on the second “capture bitmap” event and click the Display button.

3 **Close the report.**

Close the bitmaps and select File > Exit to close the report.

Running the Test on a New Version

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

Note that if you are running XRunner on a Sun machine under an MIXTrap X server, the script may not be able to execute the signature you created in Analog mode as expected.

1 **Close Airspace 1a.**

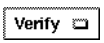
Select File > Exit.

2 **Open Airspace 1b.**

At the command prompt, enter the following two commands:

```
cd $M_ROOT/tutorial
./airspace1b &
```

In the Login window, type your first name and the password *mercury*, and click OK. Reposition the Airspace application and XRunner so that they are both clearly visible on your desktop.



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

4 **Select Run from Top.**

Select Run > Run from Top. The Set Results Directory window opens. Accept the default results directory name “res1.” Make sure that the “Display report at end of test” checkbox is selected.

5 **Run the test.**

Click OK in the Set Results Directory form. XRunner starts executing the test.

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

6 **Review the results.**

When the test run is completed, the test results appear in the XRunner report form.

The test failed because the Agent Signature field did not clear when XRunner pressed the Clear Signature button. Double-click on the failed bitmap checkpoint to view the expected, actual, and difference bitmaps.

7 **Close the results.**

Select File > Exit to close the report.

8 **Close the *lesson6* test.**

Select File > Close.

9 **Close version 1b of the Airspace application.**

Select File > Exit.

Bitmap Checkpoint Tips

- To capture an area, select Create > Check Bitmap > Area. Use the crosshairs pointer to mark the area that you want XRunner to capture. XRunner 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 XRunner not to display a message when a bitmap mismatch is detected. Select Options > Configure. In the Configuration form, clear the XR_MISMATCH_BREAK parameter. This enables the test to run without interruption.
- When running a test that includes bitmap checkpoints, make sure that the screen display settings are the same as when the test script was created. If the screen settings are different, XRunner will report a bitmap mismatch.

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

For more information on bitmap checkpoints, refer to Chapter 9, "Checking Bitmaps" in the *XRunner User's Guide*.

7

Programming Tests with TSL

This lesson:

- shows you how to use visual programming to add functions to your recorded test scripts
- shows you how to add decision-making logic to a test script
- helps you debug a test script
- lets you run a test on a new version of an application and analyze the results

How Do You Program Tests with TSL?

When you record a test, XRunner generates TSL statements in a test script each time you click on 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 XRunner's visual programming tool, the Function Generator.

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

- You can point to a GUI object and let XRunner "suggest" an appropriate function. You can then insert this function into the test script.
- You can select a function from a list. Functions are presented both 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 your goal is to create a test that:

- opens an order
- opens the Fax Order form
- checks that the number of tickets in the fax form matches the number of tickets in the main form
- reports whether the numbers are equal or not

Recording a Basic Test Script

Start by recording the process of opening an order in the Airspace application and opening the Fax Order form.

1 Open XRunner and a new test.

If XRunner is not already open, type xrun & at the command prompt. The main XRunner window is displayed.

If XRunner is already open, select File > New to create a new test.

2 Open the Airspace application and log in.

At the command prompt, enter the following two commands:

```
cd $M_ROOT/tutorial
./airspace1a &
```

In the Login window, type your first name and the password *mercury*, and click OK. Reposition the Airspace application and XRunner so that they are both clearly visible on your desktop.

3 Start Recording.

Select Create > Record—Context Sensitive or click the Record button on the toolbar.

4 Open order #4.

In the Airspace application select File > Open order. In the Open Order form, click the Order number checkbox and enter “4” in the adjacent field. Click OK to open the order.

5 **Open the Fax Order form.**

Select File > Fax order.

6 **Click Cancel to close the form.**

7 **Stop Recording.**

Select Create > Record—Stop or click the Stop button.

8 **Save the test.**

Select File > Save As or click the Save button. Name the test *lesson7* and click OK.

Using the Function Generator to Insert Functions

You are now ready to add functions to the test script which check the Tickets fields in the main Airspace form and in the Fax Order form.

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

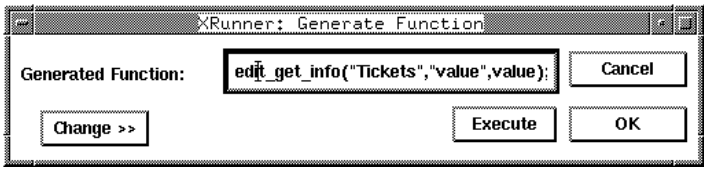
2 **Open the Fax Order form.**

Select File > Fax order.

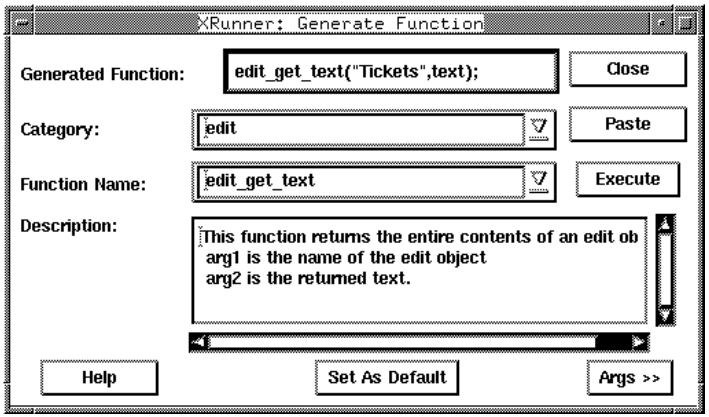
3 **Query the Tickets field in the main form.**

Select Create > Insert Function > Object. Using the ☞ pointer, click on the Tickets edit field in the main Airspace window.

The Function Generator opens and suggests the **edit_get_info** function.



Change this by pressing Change>> and selecting a different function, **edit_get_text**, from the Function Name list.



This function reads the text in the Tickets field and assigns it to a variable. The default variable name is *text*. Change the variable name to *tickets_main* by typing in the Generated Function field.

```
edit_get_text("Tickets:",tickets_main);
```

Click Paste to add the function to the test script and Close to close the Function Generator.

4 Query the # Tickets field in the Fax Order form.

Select Create > Insert Function > Object. Using the ☞ pointer, click on the # Tickets edit field in the Fax Order window.

The Function Generator opens and suggests the **edit_get_info** function. Press Change >>. Change the Function Name to **edit_get_text** and the name of the *text* variable to *tickets_fax*.

```
edit_get_text("# Tickets:",tickets_fax);
```

Click Paste to add the function to the test script and Close to close the Function Generator.

5 Close the Fax Order form.

Click Cancel to close the form.

6 Save the test.

Select File > Save or click the Save button.

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 will enable the test to:

- check that the number of tickets in the fax order form is the same as the number of tickets in the main form
- report whether the two fields match.

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.

```
if (tickets_main == tickets_fax)
                tl_step ("tickets", 0, "# Tickets fields match.");
else
                tl_step ("tickets", 1, "# Tickets fields do not match.");
```

In plain English these statements mean "If *ticket_main* equals *tickets_fax,* report that there is a match, otherwise (else) report that the numbers do not match." See the section "Understanding tl_step" below for more information on the **tl_step** function.

> You can use the Function Generator to quickly insert **tl_step** statements into the test script. Select Create > Insert Function > From List.

3 Save the test.

Select File > Save or click the Save button.

Understanding tl_step

When you run a test, XRunner usually 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 ("tickets", 1, "# Tickets fields do not match.");
```

tickets is the name you assign to this operation.

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

 # Tickets fields do not match is the message sent to the report. You can write any message that will make the test results meaningful.

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. XRunner provides debugging tools which make this process quick and easy.

You can:

- execute the test line by line using the Step commands
- define breakpoints that enable you to stop execution at a specified line or function in the test script
- monitor the values of variables and expressions using the Watch List.

When debugging a test script, you should run the test in Debug mode (select this mode from the run mode list on the toolbar.) The test results are saved in a *debug* directory. Each time you run the test in Debug mode, XRunner 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 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 **Select Run > Step or click the Step button to run the first line in the test script.**

XRunner executes the first line.

4 **Run the entire test, line-by-line, using the Step button.**

Click on the Step button to execute each line of the test script.

5 **Press Stop.**

Press the Stop button to tell XRunner that you have completed the Debug test run.

6 **Review the test results in the XRunner report form.**

Select Tools > Reports or click the Reports icon. The XRunner report displays the results of the Debug test run.

7 **Close the report.**

Select File > Exit.

8 **Exit the Airspace application.**

Select File > Exit.

For more information on debugging test scripts, refer to Part VI, “Debugging Tests” in your *XRunner 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 Airspace application.

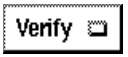
1 **Open version 1b of the Airspace application.**

At the command prompt, enter the following two commands:

```
cd $M_ROOT/tutorial
```

```
./airspace1b &
```

In the Login window, type your name and the password *mercury*, and click OK. Reposition the Airspace application and XRunner so that they are both clearly visible on your desktop.



2 Select Verify mode from the run mode list on the toolbar.

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

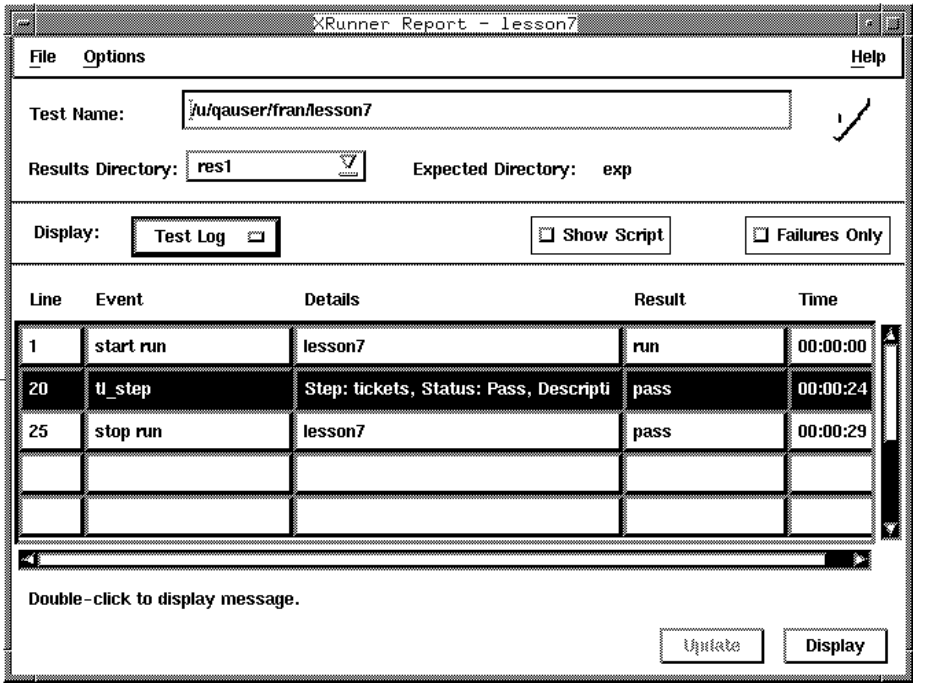
3 Select Run from Top.

The Set Results Directory form opens. Accept the default name "res1." Make sure that the "Display report at end of test" checkbox is selected in the Set Results Directory form.

4 Run the test.

Click OK in the Set Results Directory form. XRunner starts executing the test.

5 **Review the test results.**



The number of tickets in the fax form equals the number in the main form. Therefore the tl_step statement reports "pass".

You can double-click on the **tl_step** statement in the test log to view the full details.

Click OK to close the message.

6 **Close the results.**

Select File > Exit to close the report.

7 **Close the *lesson7* test.**

Select File > Close.

8 **Close version 1b of the Airspace application.**

Select File > Exit.

8

Reading Text

This lesson:

- describes how you can read text from areas of your screen
- shows you how to teach XRunner the fonts used by an application
- lets you create a test which reads and verifies text
- lets you run the test and analyze the results

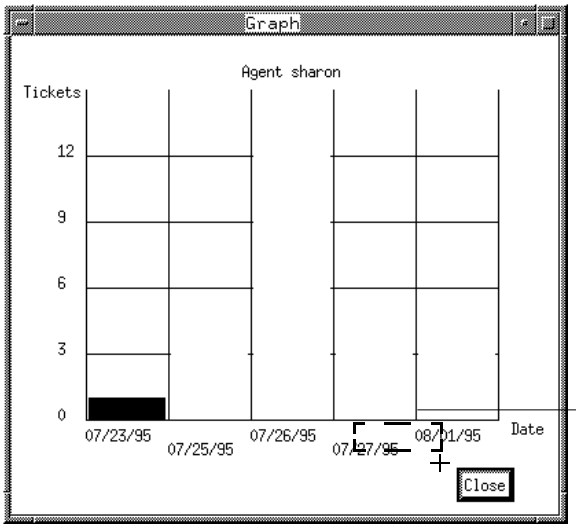
How Do You Read Text from an Application?

You can read text from any area of an application window by adding text checkpoints to a test script. You then add programming elements to the test script to verify that the text is correct.

For example, you can use a text checkpoint to:

- verify a range of values
- calculate values
- perform certain operations only if specified text is read from the screen

To create a text checkpoint, you indicate the area that contains the text you want to read.



XRunner inserts a **get_text** statement into the test script and assigns the text to a variable. To verify the text you add programming elements to the script.

Keep in mind that if you want to read text from GUI objects which are supported in XRunner, it is recommended to use a GUI checkpoint or TSL functions such as **edit_get_text** or **obj_get_info**. Use a text checkpoint only when you want to read text from a bitmap image or an object not supported in XRunner.

In the following exercises your goal is to create a test that:

- opens a graph and reads the name of the ticket agent
- checks the name of the ticket agent
- reports whether the expected name is displayed or not displayed

Before you can run tests that read text in an application, XRunner must learn the fonts used in the text areas which you plan to test. In this lesson, you will teach XRunner several of the fonts used by the Airspace application, one of which is needed for the exercises described above.

Learning Fonts

There are several steps involved in the process of teaching fonts to XRunner. In this section you will enable XRunner to recognize fonts used in the Airspace application by:

- displaying a list of the fonts used by your application (by using the xmon utility for **Sun/Solaris and HP machines**, and the xscope utility for **IBM machines**)
- selecting the fonts you need
- teaching fonts to XRunner

Displaying the List of Fonts—Sun/Solaris and HP Machines

If you are using a Sun/Solaris or HP machine, use the xmon utility to identify the fonts that Airspace uses.

1. **At the command prompt, enter the command:**

   xmonui | xmond &

   These utilities are found in $M_ROOT/bin.

   The main window of xmonui is displayed.

2. **In the Selected Requests region of this window, select "main" from the Detail options.**
3. **In the Selected Requests list box, select request 45 OpenFont.**
4. **Start the application from the command line with the -display option as follows:**

   cd $M_ROOT/tutorial
   ./airspace1a -display <*machine_name*>:1 &

   For example, if you are running xmonui | xmond on a host machine called venus, type:

   cd $M_ROOT/tutorial
   ./airspace1a -display venus:1 &

The xmon utilities examine the requests passed by Airspace to the <*machine_name*>:1 virtual display, while displaying the application on your local display (as defined by your DISPLAY environment variable).

As the Airspace Login window opens, observe standard output in the shell window from which you entered the command in step 1. The names of the fonts used by Airspace for the Login window are displayed. For example:

```
REQUEST: OpenFont
        name: "fixed"
```

5 **Log in to the Airspace application.**

In the Login window, type your name and the password *mercury*, and click OK.

Note that additional font listings appear in the shell window as the main Airspace window opens. Reposition the windows so that they are clearly visible on your desktop.

6 **Open the Graph window.**

Open the Graph window by choosing the Analysis > Graphs command or using the Graph button on the toolbar.

In the exercises in this lesson, you will be reading text in the Graph window. The names of the fonts used in this window are now displayed in standard output. These are the fonts you will teach XRunner before you create a test that reads text.

Displaying the List of Fonts—IBM Machines

For IBM applications, you can use xscope or a similar utility to determine the fonts used.

1 **At the command prompt, enter the commands:**

```
cd <home_directory>
xscope -v1 -i1 > scope_file
```

<*home_directory*> can be any directory in which you have permission to create a file. The xscope utility is found in $M_ROOT/bin.

2 **Start the application from the command line with the -display option as follows:**

```
cd $M_ROOT/tutorial
./airspace1a -display <machine_name>:1 &
```

For example, if you are running xscope on a host machine called venus, type:

```
cd $M_ROOT/tutorial
./airspace1a -display venus:1 &
```

As you open the various windows of the application, the xscope utility examines the requests passed by Airspace to the <*machine_name*>:1 virtual display, while displaying the application on your local display (as defined by your DISPLAY environment variable).

The -v option of the xscope command, which should be 1, defines the verbosity level of the output. The information will be logged in the file scope_file in the current directory.

3 **Log in to the Airspace application.**

In the Login window, type your name and the password *mercury*, and click OK. Reposition the Airspace application so that it is clearly visible on your desktop.

4 **Open the Graph window.**

In this lesson you will be reading text in the Graph window. To determine which fonts must be learned, open the Graph window by choosing the Analysis > Graphs command or using the Graph button on the toolbar. The xscope utility will determine the fonts used in this window and record them in the output file.

5 **Open the file scope_file using a text editor.**

6 **Search the scope_file file for the string**

"REQUEST: OpenFont."

A few lines below this entry you will find a line with the format:

name: "6x13"

In this example, the string "6x13" is the name of the font used by Airspace.

The output file will contain six such OpenFont statements. You will be teaching XRunner several of the fonts listed here before creating a test which reads text in the Graph window.

Selecting the Fonts That You Need for Your Test

You now have a comprehensive list of fonts used in Airspace, but you need not learn all the fonts. The next step shows you how to attempt to match the text displayed by the X server, in the area you plan to test, with the appropriate font name, in order to eliminate fonts which need not be learned.

Reopen the Airspace graph by selecting Analysis > Graphs. Refer to the list of fonts which are candidates for the font used in the Agent name field. At the command prompt, enter commands in the following format:

xfd –fn <*font_requested_by_application*>

For example:

xfd –fn 6x13

Run the xfd command for several of the fonts in the list: For example, “fixed,” “-adobe-*-bold-r-normal--12-*-100-100-*-*-iso8859-1,” “6x13,” and “cursor”. Each time you run the command, a grid of the character set provided by the server for the application font is displayed.

By comparing the grids with the characters in the Agent name field in the Graph window, try to determine which font Airspace uses to display the Agent name. You may still have several candidates after this step.

Exit from the Airspace application by selecting File > Exit.

Teaching Fonts to XRunner

You will now teach XRunner the selected fonts and assign them to a font group. Before creating a test, you will make this font group the active font group so that it will be accessible to XRunner for text recognition.

Note: The data for each font learned by XRunner is stored in a .mfn file. The default location for these files is $(M_ROOT)/fonts. This directory may be read-only for you. You can change the target location (directory) before a font is learned by changing the variable XR_GLOB_FONT_LIB in the Configuration form (Options > Configure menu command). After opening XRunner and changing the parameter, *exit from XRunner* before continuing to learn the fonts, so that the updated value will be used.

---

1 Teach XRunner a font using xrmkfont.

At the command prompt, enter a command in the following format:

```
xrmkfont fixed
xrmkfont "-adobe-*-bold-r-normal--12-*-100-100-*-*-iso8859-1"
```

2 Repeat for additional fonts.

Repeat the xrmkfont command for any other fonts you want XRunner to learn.

```
xrmkfont 6x13
```

3 Assign the learned fonts to a font group.

After the application fonts have been learned, you must assign them to a font group, using the xrfontgrp utility. Each of the following examples creates a font group called air_fonts that includes two learned fonts:

```
xrfontgrp -add air_fonts fixed 6x13
xrfontgrp -add air_fonts "-adobe-*-bold-r-normal--12-*-100-100
-*-*-iso8859-1" 6x13
```

Note: The file associated with this group will be called air_fonts.grp and will be located in the same directory as the .mfn font files, in the directory defined by the XR_GLOB_FONT_LIB parameter.

---

4 **Specify the Active Font Group in the Configuration form.**

The final step before you can perform any of the functions for checking text is to activate the font group that includes the fonts used by the application.

Select Options > Configure. In the Configuration form (AUT section, Text page), locate the XR_FONT_GROUP parameter. Change it from the default value to air_fonts, and click Save and Close.

Reading Text from an Application

In this first exercise you will record the process of opening the Airspace analysis graph to read the name of the ticket agent who is logged on. In the second exercise you will add programming elements to the test script which determine whether the agent name is displayed in the Graph window.

1 **Open XRunner.**

If XRunner is not already open, type xrun & at the command prompt. A blank test opens in XRunner.

2 **Open the Airspace application and log in.**

At the command prompt, enter the following two commands:

```
cd $M_ROOT/tutorial
./airspace1a &
```

In the Login window, type your name and the password *mercury*, and click OK. Reposition the Airspace application and XRunner so that they are both clearly visible on your desktop.

3 **Start Recording in Context Sensitive mode.**

Select Create > Record—Context Sensitive, or press the Record button on the toolbar.

4 **Open the graph.**

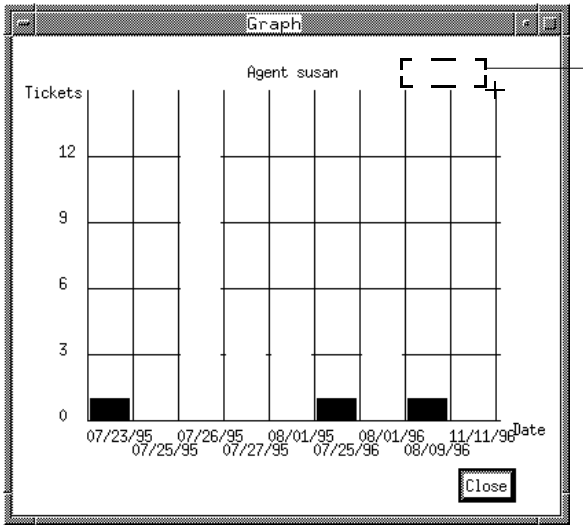
In the Airspace application select Analysis > Graph.

5 Move the graph window to a different location on your desktop.

6 Read the name of the agent.

Select Create > Get Text.

Using the crosshairs pointer and the left mouse button, drag a rectangle around the name of the agent.



Indicate the area that contains the agent name

Click the middle mouse button if you would like to preview the text in the selected area. Click the right mouse button to complete the operation. XRunner inserts a **get_text** statement into the test script. The text appears in

the script as a comment: for example,
susan.

When XRunner reads text from the screen, it inserts a **get_text** statement into the test script. For example:

```
t = get_text(346, 252, 373, 272);
```

t is the variable which stores the text you selected.

346, 252, 373, 272 are the coordinates of the rectangle you marked around the text.

7 Close the graph.

Press Close in the Graph window.

8 Stop recording.

In XRunner, select Create > Record—Stop or click the Stop button.

9 Save the test.

Select File > Save As or click the Save button. Name the test *lesson8*.

Verifying Text

In this exercise you add an if/else statement to the test script in order to determine whether the name of the agent logged in is displayed in the Graph window.

1 In the get_text statement in the *lesson8* test script, change the text variable, *t*, to *agent_name*.

2 Place the cursor below the last line in the script.

3 **Add the following four lines to the test script exactly as they appear below.**

```
if (agent_name == "susan")
        tl_step ("agent", 0, "Agent name is displayed.");
else
        tl_step ("agent", 1, "Agent name is not displayed.");
```

In plain English, these statements mean "If *agent_name* equals "susan", report that the agent name is displayed, otherwise (else) report that the agent name is not displayed."

For a description of the **tl_step** function, review Lesson 7, "Programming with TSL."

4 **Save the test.**

Select File > Save or click the Save button.

Debugging the Test Script

You should now run the test in Debug mode in order to check for errors in syntax and logic. If any error messages appear, look over the test script and try to fix the problem.

1 **Select Debug mode from the run mode list on the toolbar.**

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

2 **Run the test.**

Select Create > Run from Top. If you prefer to run the test line-by-line, use the Step button.

3 **Review the test results in the XRunner report form.**

Select Tools > Reports or click the Reports button. The XRunner report displays the results of the Debug test run.

If the **tl_step** event failed, a problem exists in the test script. Examine the script and try to fix the problem.

4 **Exit the Airspace application.**

Select File > Exit.

Running the Test on a New Version

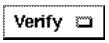
Once the test script is debugged, you can run it on a new version of the Airspace application.

1 **Open version 1b of the Airspace application.**

At the command prompt, enter the following two commands:

```
cd $M_ROOT/tutorial
./airspace1b &
```

In the Login window, type your name and the password *mercury*, and click OK. Reposition the Airspace application and XRunner so that they are both clearly visible on your desktop.



2 **Select Verify mode from the run mode list on the toolbar.**

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

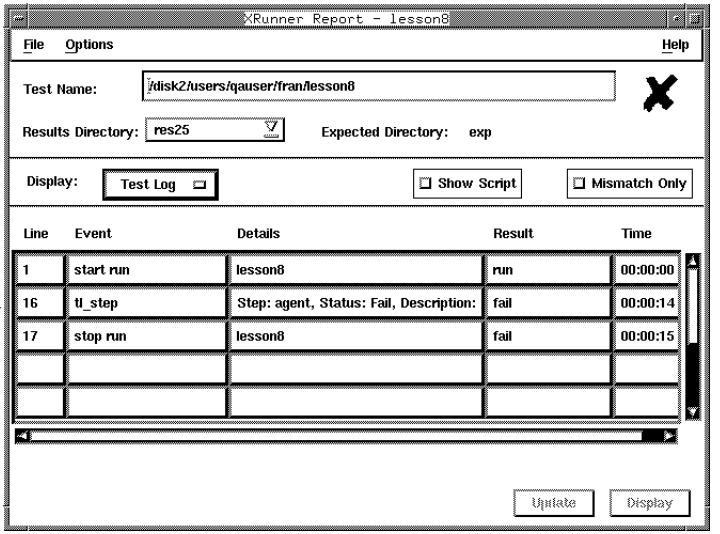
3 **Select Run from Top.**

The Set Results Directory form opens. Accept the default name “res1”. Make sure that the “Display report at end of test” checkbox is selected in the Set Results Directory form.

4 **Run the test.**

Click OK in the Set Results Directory form. XRunner starts executing the test.

5 Review the test results.



The test fails because the agent name is not displayed

Select the *Test Log* display option to view the results of the **tl_step** statement.

XRunner compares the text stored in the variable *agent_name* to the actual text it reads in the application window. The test fails because the agent name is not displayed.

6 Close the report.

Select File > Exit.

7 Close the *lesson8* test.

Select File > Close.

8 Close version 1b of the Airspace application.

Select File > Exit.

Text Checkpoint Tips

- Before you create a script that reads text, determine where the text is located. If the text is part of a GUI object supported in XRunner, use a GUI checkpoint or TSL functions such as **edit_get_text** or **obj_get_info**. If the text is part of a bitmap, use the Create > Get Text command.
- When XRunner reads text from the application, the text appears in the script as a comment (a comment is preceded by #). If the comment

  ```
  #no text was found
  ```

  appears in the script, XRunner does not recognize your application font. Make sure you have completed the procedures at the beginning of this lesson to teach XRunner this font.
- If your application uses very small, italic, or oblique fonts, XRunner may have difficulty recognizing them.
- When specifying a font name at the command line, enclose the font name in double quotes if it contains any special characters such as hyphens or asterisks.
- TSL includes additional functions that enable you to work with text such as **find_text** and **compare_text**. Refer to Chapter 12, "Checking Text" in your *XRunner User's Guide* for more information.

9

Creating Batch Tests

This lesson:

- describes how you can use a batch test to run a suite of tests unattended
- helps you create a batch test
- helps you run the batch test and analyze the results

What Is a Batch Test?

By creating a single batch test, you can run an entire suite of tests unattended. You can start a batch test run, go to lunch, and come back to review the results when the run is finished.

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 "/u/qauser/fran/lesson8"();
  ```

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

- You select the Batch Mode option, XR_BATCH_MODE, in the Configuration form (Options > Configure) before running the test. This option tells XRunner to suppress messages that will interrupt the test. For example, if XRunner detects a bitmap mismatch, it will not ask if you want 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 3 times in order to check how the Airspace application handles the *stress* of repeated execution.

1 **Open XRunner and a new test.**

If XRunner is not already open, type xrun & at the command prompt. The main XRunner window is displayed.

If XRunner is already open, select File > New to create a new test.

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

Type the **call** statements into the new test window. The statements should look like this:

```
call "/u/qauser/fran/lesson5"();
call "/u/qauser/fran/lesson6"();
call "/u/qauser/fran/lesson7"();
```

In your test script, replace /u/qauser/fran with the directory path that contains your tests.

3 **Define a loop so that each test is called 3 times.**

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

```
for (i=0; i<3; i++)
{
call "/u/qauser/fran/lesson5"();
call "/u/qauser/fran/lesson6"();
call "/u/qauser/fran/lesson7"();
}
```

In plain English, this means "Execute *lesson5*, *lesson6*, and *lesson7*, and then loop back and execute each test again. Repeat this process until each test is executed 3 times." Note that the brackets { } define which statements are included in the loop.

4 **Select the Batch Mode option in the Configuration form.**

Select Options > Configure. In the Configuration form (Execution section), select the XR_BATCH_MODE parameter and click Save and Close. All tests will run in Batch mode from this point on, until the option is changed again.

5 **Save the batch test.**

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

Running the Batch Test on Version 1b

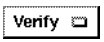
You are now ready to run the batch test in order to check the Airspace application. When you run the test, XRunner will compare the expected results of each test to the actual results in the application. It uses the expected results stored when you created the tests in earlier lessons.

1 **Open version 1b of the Airspace application and log in.**

At the command prompt, enter the following two commands:

```
cd $M_ROOT/tutorial
./airspace1b &
```

In the Login window, type your name and the password *mercury*, and click OK. Reposition the Airspace application and XRunner so that they are both clearly visible on your desktop.



2 **In XRunner, select Verify mode from the run mode list on the toolbar.**

3 **Select Run from Top.**

Select Run > Run from Top. The Set Results Directory window opens. Accept the default name "res1." Make sure that the "Display report at end of test" checkbox is selected.

4 **Run the test.**

Click OK in the Set Results Directory form. XRunner starts executing the test. *Watch how XRunner opens and executes each called test, and loops back to execute the tests again (for a total of 3 times).*

Analyzing the Batch Test Results

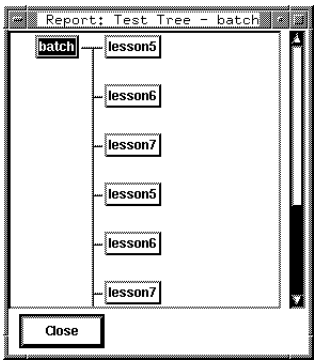
Once the batch test run is completed, you can analyze the results in the XRunner report form. The report form displays the overall result (pass or fail) of the batch test, as well as a result for each called test. The batch test fails if any of the called tests failed.

1 **Open the XRunner report form and display the res1 results of the batch test.**

If the XRunner report form is not currently open, switch to or open the batch test and select Tools > Reports, or click the Reports button.

Select the Test Log display option. An additional window containing the Test Tree will open.

2 View the results of the batch test.

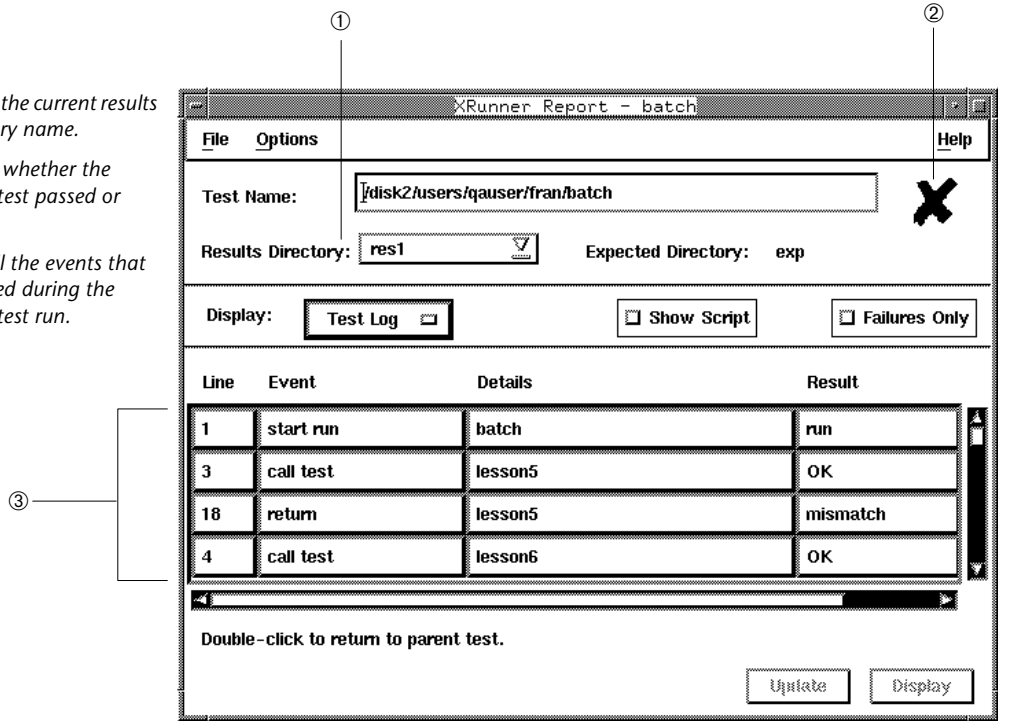


The test tree shows all the tests called during the batch test run. Since each test was called 3 times, the test names appear 3 times in the list.

1 *Shows the current results directory name.*

2 *Shows whether the batch test passed or failed.*

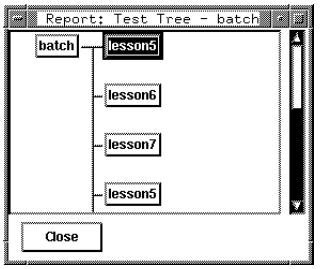
3 *Lists all the events that occurred during the batch test run.*



A “call test” event indicates that a called test was opened and executed, and 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 1b contains some bugs.

3 View the results of the called tests.



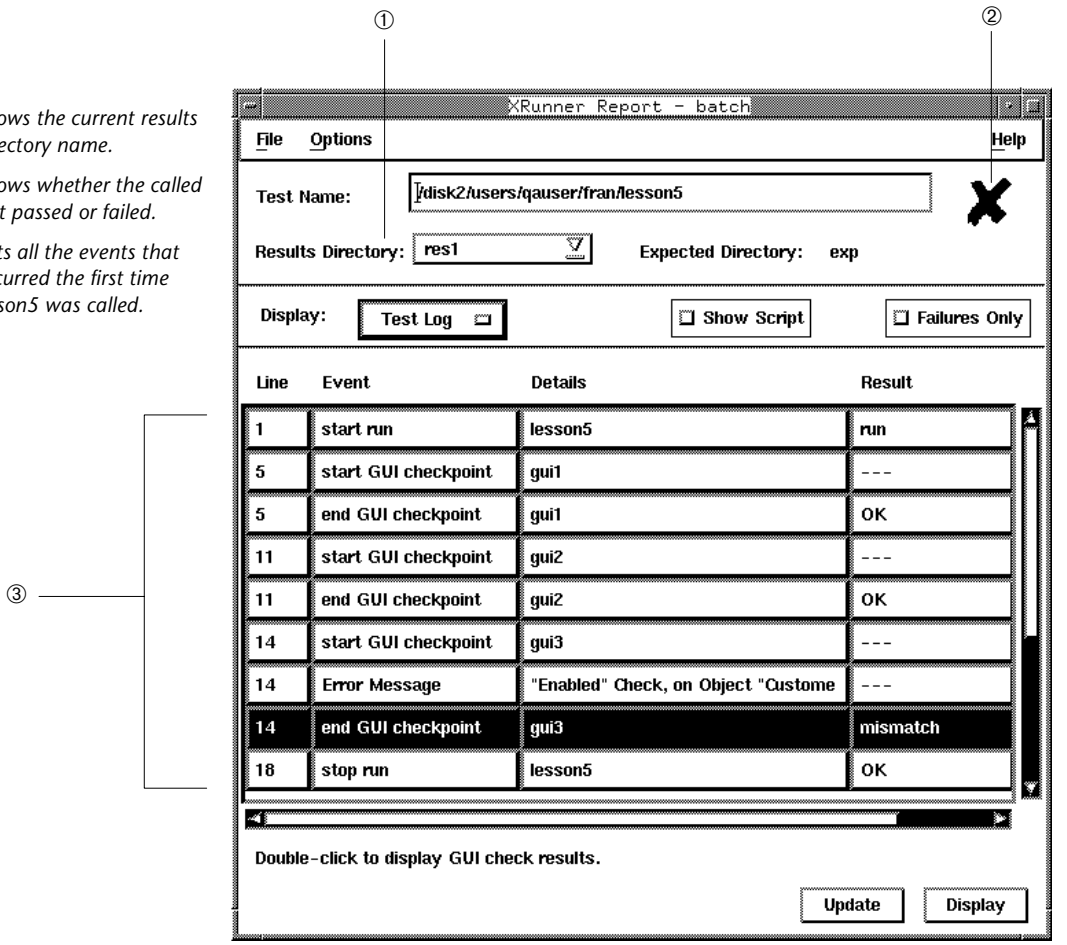
The highlighted test indicates which test results are currently displayed. In this case, lesson5 results appear in the detailed report form.

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

1 *Shows the current results directory name.*

2 *Shows whether the called test passed or failed.*

3 *Lists all the events that occurred the first time lesson5 was called.*



Remember that *lesson5* uses a GUI checkpoint to check the states of GUI objects. Since the Customer Name button was not disabled, the GUI checkpoint detected a mismatch. You can double-click on the failed event to display the expected, actual, and difference results.

4 Close the report form.

Select File > Exit.

5 Close the *batch* test.

Select File > Close.

6 **Clear the Batch Mode parameter in the Configuration form.**

Once you are finished running the batch test, clear the XR_BATCH_MODE parameter. Select Options > Configure. In the Configuration form, clear the XR_BATCH_MODE parameter and click Save and Close.

7 **Close version 1b of the Airspace application.**

Select File > Exit.

Batch Test Tips

- By defining search paths, you can instruct XRunner to search for called tests in certain directories. Select Options > Search Path. In the Search Path form, simply define the paths in which the tests are located. This enables you to include only the test name in a **call** statement. For example:

  ```
  call "lesson6"();
  ```

- 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 Batch Mode option in the Configuration form in order for the batch test to run unattended.

  For more information on creating batch tests, refer to Chapter 19, "Calling Tests" and Chapter 26, "Running Batch Tests" in your *XRunner User's Guide*.

10

Maintaining Your Test Scripts

This lesson:

- explains how the GUI map enables you to continue using your existing test scripts after the user interface changes in your application
- shows you to edit existing object descriptions or add new descriptions to the GUI map
- shows you how to use the Run Wizard to automatically update the GUI map

What Happens When the 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?

XRunner 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. It is created when you use the Test Wizard to learn the objects in your application. This information is saved in a GUI map file.

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 attributes 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, XRunner reads an object’s logical name in the test script and refers to its physical description in the GUI map. XRunner 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 XRunner can continue to 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 Airspace application, the *Delete Order* button is changed to a *Delete* button. In order for you to continue running tests that use the Delete Order button, you must edit the button’s physical description in the GUI map.

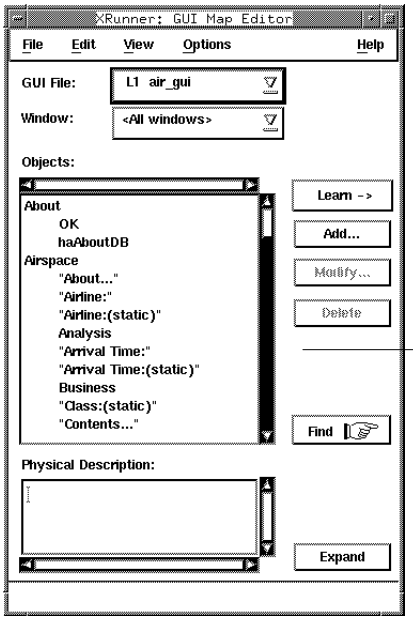
1 **Open XRunner and a new test.**

If XRunner is not already open, type xrun & at the command prompt. The main XRunner window is displayed.

If XRunner is already open, select File > New to create a new test.

2 Open the GUI map.

Select Tools > GUI Map Edit. The GUI Map Editor opens. Make sure that in the View menu, the GUI Files option is selected. The GUI map stored in the GUI file specified at the top of the form is displayed. If the filename in the GUI File field is not air_gui (the GUI file you saved in Lesson 3), select air_gui from the list now. If air_gui is not one of the files in the GUI file list, add air_gui now: Choose File > Open, and specify the location of the air_gui file. Make sure the Load File option at the bottom of the form is selected, and press OK.



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

When All Windows is specified in the Window field, the GUI Map Editor displays the object names in outline form. The objects are grouped according to the window in which they are located. You can specify a window in the Window field to display only the objects located in that window.

3 Find the Delete Order button in the tree.

Select Airspace in the Window option menu.

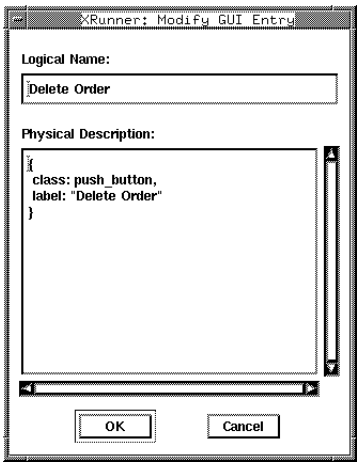
Scroll down until you locate the Delete Order button.

4 View the Delete Order button's physical description.

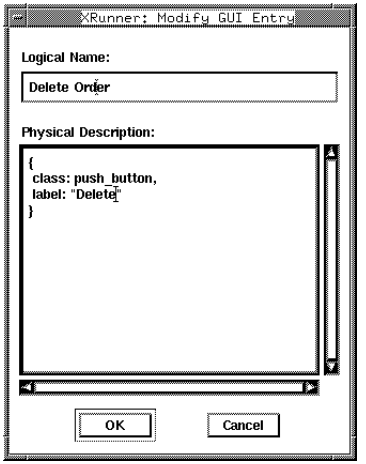
Click on the Delete Order button in the list. The Delete Order button's physical description appears in the Physical Description field below the object list.

5 Modify the Delete Order button's physical description.

Click the Modify button. The Modify GUI Entry form displays the button's logical name and physical description.



In the Physical Description field, change the label attribute from Delete Order to Delete.



Click OK to close the form.

6 Close the GUI Map.

In the GUI Map Editor, select File > Exit.

The next time you run a test that contains the logical name “Delete Order”, XRunner will locate the Delete button in the Airspace window.

Adding GUI Objects to the GUI Map

If your application contains new objects, you can add them to the GUI map without running the Test 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 Airspace Login window to the GUI map.

1 Open the Airspace Login window.

At the command prompt, enter the following two commands:

```
cd $M_ROOT/tutorial
./airspace1a &
```

2 **Open the GUI map.**

Select Tools > GUI Map Edit. The GUI Map Editor opens.

3 **Learn all the objects in the Login window.**

Click the Learn button and select Window. Using the hand, click on the background of the Login window.

A message asks you if you want to learn all the objects in the window. Click Yes.

XRunner learns a description of each object in the Login window and adds it to the GUI Map.

It then notifies you that it has completed the task successfully.

4 **Find the Login window objects in the GUI Map Editor tree.**

5 **Close the GUI Map Editor.**

In the GUI Map Editor, select File > Exit.

6 **Close the Login window.**

Click Cancel.

Updating the GUI Map with the Run Wizard

During a test run, if XRunner 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 asks 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 presses the Insert Order button in the Airspace 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 press the hand button in the wizard and point to the Insert button. The Run Wizard then offers a solution.

When you press OK to accept the solution suggested by XRunner, XRunner automatically modifies the object's physical description in the GUI map, and then resumes the test run.

If you would like see for yourself how the Run Wizard works:

1. **Open the GUI map (Tools > GUI Map Edit).**
2. **Delete the "Fly From" object from the GUI Map Editor tree.**

   The "Fly From" object (Fly From:_1) is listed under the Airspace window. Select this object and press the Delete button in the GUI Map Editor.
3. **Open Airspace 1a.**

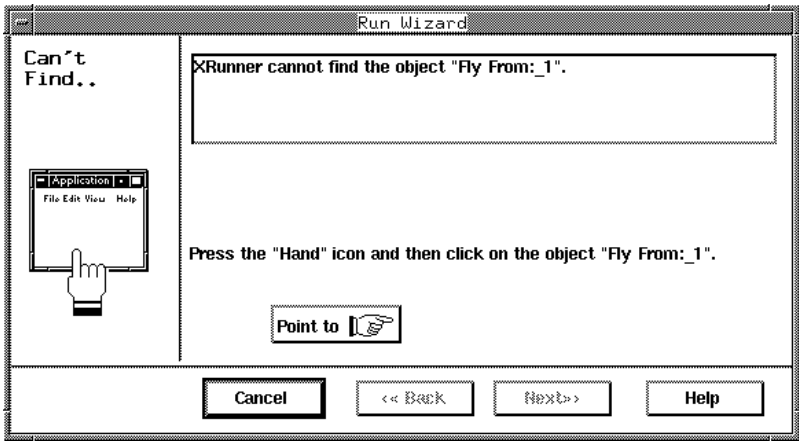
   At the command prompt, enter the following two commands:

   ```
   cd $M_ROOT/tutorial
   ./airspace1a &
   ```

   In the Login window, type your name and the password *mercury*, and click OK. Reposition the Airspace application and XRunner so that they are both clearly visible on your desktop.

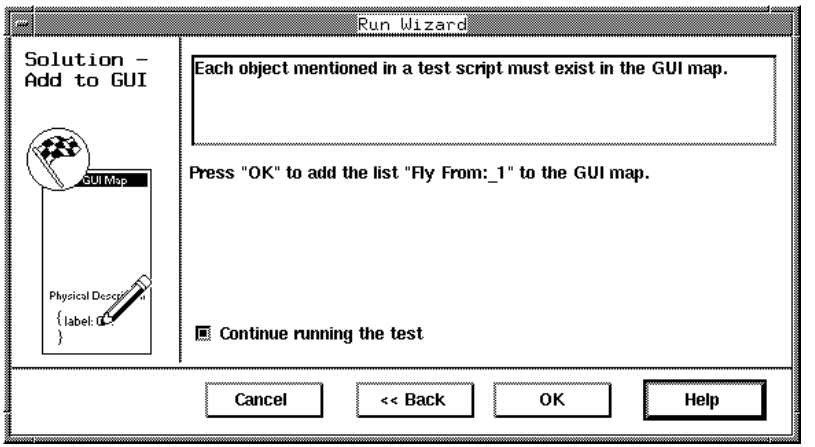
4 Open and run the *lesson4* test.

Watch what happens when XRunner reaches the statement list_select_item ("Fly From:_1", "San Francisco");



5 Follow the Run Wizard instructions.

The Run Wizard asks you to point to the Fly From object. Point to the list itself and not the edit area above the list. XRunner then adds the object description to the GUI map and continues the test run.



6 Find the object description in the GUI map.

When XRunner completes the test run, return to the GUI Map Editor and look for the Fly From object description. You can see that the Run Wizard added the object to the tree.

7 Close the GUI Map.

In the GUI Map Editor, select File > Exit.

8 Close the Airspace application.

Select 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 XRunner concepts and skills you learned to your own application.

This lesson:

- shows you how to start testing your application
- describes where you find additional information about XRunner

Getting Started

In order to start testing your application you should use the Test Wizard to learn a description of every object it contains. However, before doing this, remove the sample application's object descriptions from the GUI map.

To get started:

1. **Close all applications on your desktop except for XRunner and the application you want to test.**
2. **Clear the GUI map.**

   The GUI map currently contains descriptions of objects in the sample application. Since you no longer need these descriptions, you should clear the GUI map.

   To clear the GUI map, open the GUI Map Editor (Tools > GUI Map Edit) and select Edit > Clear All. XRunner will close all open GUI map files and will also delete any descriptions found in the temporary GUI map file.

   Select File > Exit to close the GUI Map Editor.

3 **Run the Test Wizard on your application. Learn object descriptions in Comprehensive mode.**

You should now use the Test Wizard to learn a description of each object in your application. Select Create > Test Wizard and follow the instructions on the screen.

When the wizard asks you to select a learning flow, choose *Comprehensive*. This mode lets you control how XRunner learns object descriptions.

After the learning process is completed, 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, press the Help button on the appropriate screen.

4 **Create tests!**

Once you finish using the wizard, you can start creating tests in XRunner. Use recording, programming, or a combination of both to build your automated test scripts.

Getting Additional Information

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

Documentation Set

In addition to this tutorial, XRunner comes with a complete set of documentation:

XRunner User's Guide provides step-by-step instructions on how to use XRunner to test your application. It describes many useful testing tasks and options not covered in this tutorial.

XRunner Customization Guide shows you how to meet testing requirements that are unique to your application.

XRunner Installation Guide explains how to install XRunner on a single computer or on a network.

Product Release Notes provides last-minute information that is not included in the XRunner User's Guide.

Online Resources

XRunner includes the following online resources:

XRunner Online Help provides quick answers to questions that arise as you work with XRunner. It describes menu commands, forms, and TSL functions and guides you in tailoring XRunner's Context Sensitive testing features to meet the special needs of you application.

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

XRunner Sample Tests includes utilities and sample tests with accompanying explanations.

Books Online are available on Mercury Interactive's Customer Support web site. It includes the complete documentation set in PDF format. Online books can be read using Adobe Acrobat Reader.

Technical Support Online uses your default web browser to open Mercury Interactive's Customer Support web site.

Support Information presents the locations of Mercury Interactive's Customer Support web site and home page, the e-mail address for sending information requests, the name of the relevant news group, the location of Mercury Interactive's public FTP site, and a list of Mercury Interactive's offices around the world.

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

Mercury Interactive Corporation
1325 Borregas Avenue
Sunnyvale, CA 94089 USA

Main Telephone: (408) 822-5200
Sales & Information: (800) TEST-911
Customer Support: (408) 822-5400
Fax: (408) 822-5300

Home Page: www.merc-int.com
Customer Support: web.merc-int.com