

Title: Introduction to winrunner.

Objective:

- ☐ Describes the benefits of automated testing
- ☐ Understand the WinRunner testing process
- ☐ Work with WinRunner user interface

Theory:

Understanding the Testing Process

The WinRunner testing process consists of 6 main phases:

1 Teaching WinRunner the objects in your application

WinRunner must learn to recognize the objects in your application in order to run tests.

The preferred way to teach WinRunner your objects depends on the GUI map mode you select.

The two GUI map modes are described in detail in subsequent lessons.

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

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

3 Debugging the tests

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

4 Running the tests on a new version of the application

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

5 Examining the test results

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

6 Reporting defects

If you have the TestDirector 7.0i, the Web Defect Manager (TestDirector 6.0), or the Remote Defect Reporter (TestDirector 6.0), you can report any defects to a database. The Web Defect Manager and the Remote Defect Reporter are included in TestDirector, Mercury Interactive's software test management tool.

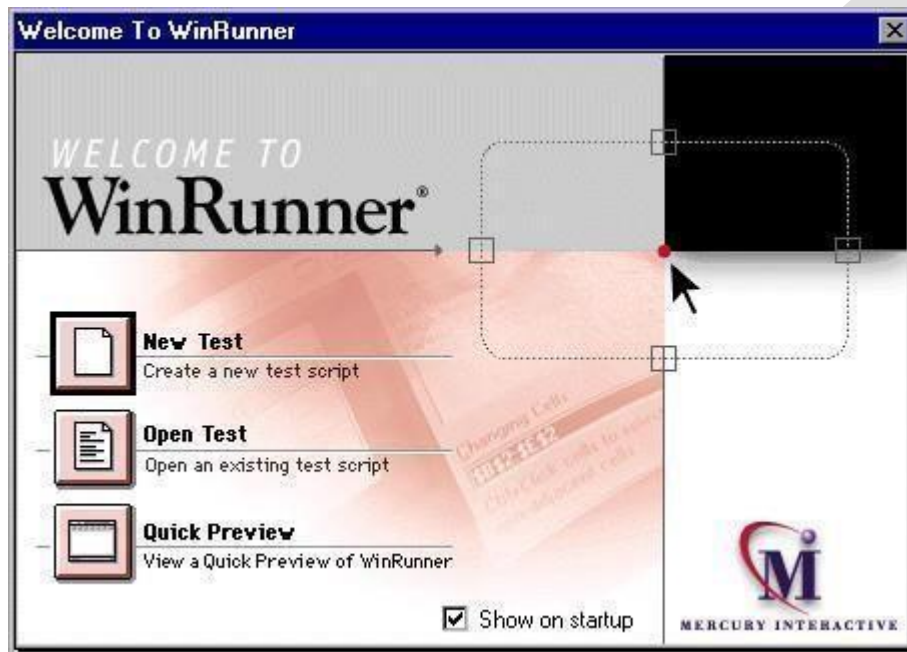
Exploring the WinRunner Window

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

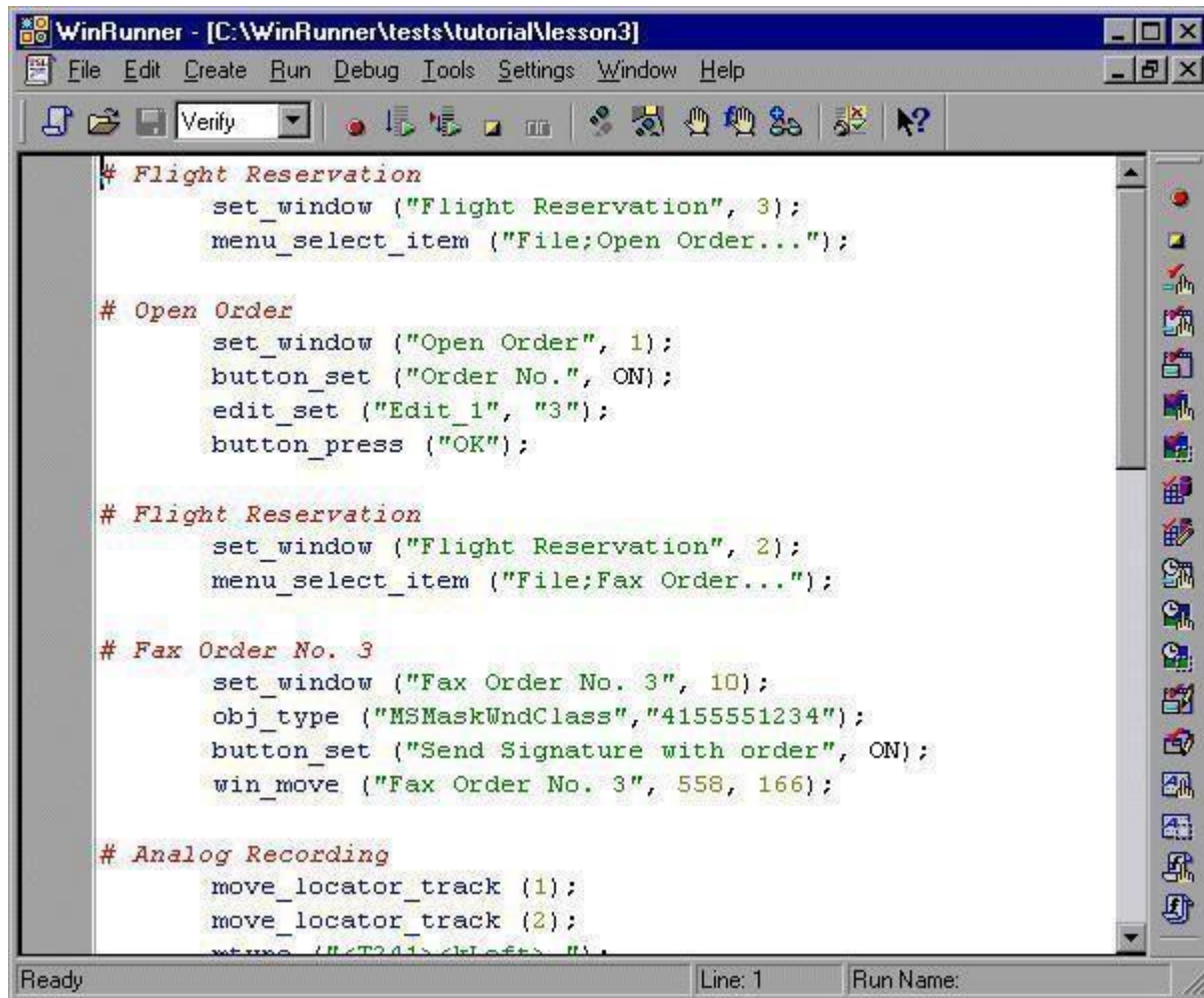
To start WinRunner:

Choose **Programs > WinRunner > WinRunner** on the **Start** menu.

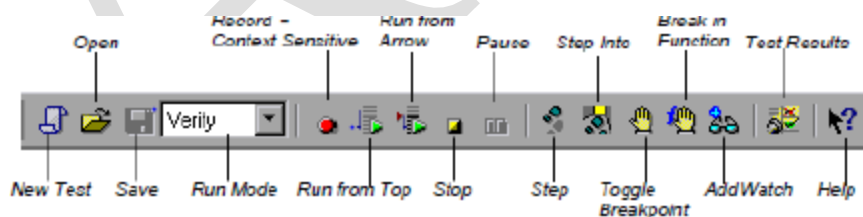
The first time you start WinRunner, the Welcome to WinRunner window opens. From the welcome window you can create a new test, open an existing test, or view an overview of WinRunner in your default browser.



The first time you select one of these options, the WinRunner main screen opens with the “What’s New in WinRunner” section of the help file on top. If you do not want the welcome window to appear the next time you start WinRunner, clear the **Show on startup** check box. Each test you create or run is displayed by WinRunner in a test window. You can open many tests at one time.



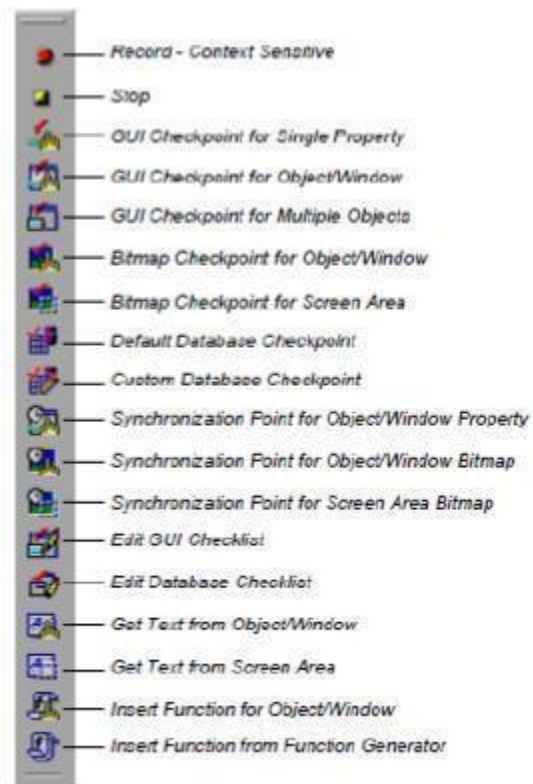
The *Standard toolbar* provides easy access to frequently performed tasks,



The *User toolbar* displays the tools you frequently use to create test scripts.

By default, the User toolbar is hidden.

To display the User toolbar choose **Window > User Toolbar**. When you create tests, you can minimize the WinRunner window and work exclusively from the toolbar.



The User toolbar is customizable. You choose to add or remove buttons using the **Settings > Customize User Toolbar** menu option. When you re-open WinRunner, the User toolbar appears as it was when you last closed it.

Experiment No :1
Date:

Recording in context sensitive mode and analog mode

Title: Recording test in analog and context sensitive mode

Objective: Student should be able to

- ☐ Describes Context Sensitive and Analog record modes
- ☐ Record a test script
- ☐ Read the test script
- ☐ Run the recorded test and analyze the results

Theory:

Choosing a Record Mode

By recording, you can quickly create automated test scripts. You work with your application as usual, clicking objects with the mouse and entering keyboard input.

WinRunner records your operations and generates statements in TSL, Mercury Interactive's Test Script Language. These statements appear as a script in a WinRunner test window.

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

Context Sensitive

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

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

`button_press ("OK");`

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

When choosing a record mode, consider the following points

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

Recording a Context Sensitive Test:-

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

- 1 Start WinRunner.**
- 2 Open a new test.**
- 3 Start the Flight Reservation application and log in.**
- 4 Start recording in Context Sensitive mode.**
- 5 Open order #3.**
- 6 Stop recording.**
- 7 Save the test.**

Recording in Analog Mode:-

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

- 1 Open the Fax Order form and fill in a fax number.**
- 2 Select the Send Signature with Order check box.**
- 3 Sign the fax again in Analog mode.**
- 4 Stop Recording.**
- 5 Save the test.**

Running the Test

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

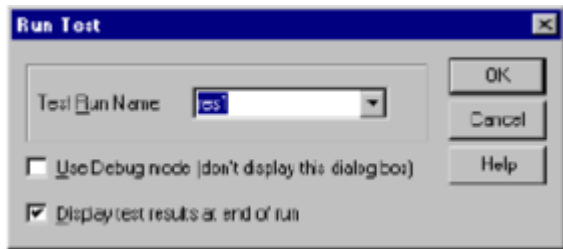
WinRunner provides three modes for running tests. You select a mode from the toolbar.

- ☐ Use *Verify mode* when running a test to check the behavior of your application, and when you want to save the test results.
- ☐ 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.

To run the test:

- 1 Check that WinRunner and the main window of the Flight Reservation application are open on your desktop.**
- 2 Make sure that the *saved* test window is active in WinRunner.**
- 3 Make sure the main window of the Flight Reservation application is active.**
- 4 Make sure that Verify mode is selected in the toolbar.**
- 5 Choose Run from Top.**

Choose **Run > Run from Top** or click the **Run from Top** button. The **Run Test** dialog box opens.



6 Choose a Test Run name.

7 Run the test.

8 Review the test results.

Conclusion:-

WinRunner Results - H:\Program Files\Mercury
Interactive\WinRunner\tmp\noname14

=====

Expected results folder: H:\Program Files\Mercury
Interactive\WinRunner\tmp\noname14\exp
Test Results Name: H:\Program Files\Mercury
Interactive\WinRunner\tmp\noname14\res1
Operator Name:
Date: Tue Mar 17 12:00:21 2015
Summary:

Test Result: OK
Total number of bitmap checks: 0
Total number of GUI checks: 0
Total Run Time: 00:00:04
Detailed Results Description
Line Event Result Details Time

3 start run run noname14 00:00:00

23 stop run pass noname14 00:00:04
WinRunner Results - H:\Program Files\Mercury
Interactive\WinRunner\tmp\noname14

=====

Expected results folder: H:\Program Files\Mercury
Interactive\WinRunner\tmp\noname14\exp
Test Results Name: H:\Program Files\Mercury
Interactive\WinRunner\tmp\noname14\res1
Operator Name:
Date: Tue Mar 17 12:00:21 2015
Summary:

Test Result: OK
Total number of bitmap checks: 0
Total number of GUI checks: 0
Total Run Time: 00:00:04
Detailed Results Description
Line Event Result Details Time

3 start run run noname14 00:00:00
23 stop run pass noname14 00:00:04

Experiment No: 2,3,4
Date:

GUI checkpoint for single property
GUI checkpoint for single object/window
GUI checkpoint for multiple objects

Title: Checking GUI Objects.

Objective: Student should be able to

- ☐ Explain how to check the behavior of GUI objects
- ☐ Create a test that checks GUI objects
- ☐ 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 creating *GUI checkpoints*. A GUI checkpoint examines the behavior of an object's properties.

For example, you can check: the content of a field whether a radio button is on or off whether a pushbutton is enabled or disabled

Adding GUI Checkpoints to a Test Script

- 1 Start WinRunner and open a new test.
- 2 Start the Flight Reservation application and log in.
- 3 Start recording in Context Sensitive mode.
- 4 Open the Open Order dialog box.
- 5 Create a GUI checkpoint for the Order No. check box.
- 6 Enter "4" as the Order No.
- 7 Create another GUI checkpoint for the Order No. check box.
- 8 Create a GUI checkpoint for the Customer Name check box.
- 9 Click OK in the Open Order dialog box to open the order.
- 10 Stop recording.
- 11 Save the test.

Conclusion

WinRunner Results - H:\Program Files\Mercury Interactive\WinRunner\tmp\noname15

=====

Expected results folder: H:\Program Files\Mercury Interactive\WinRunner\tmp\noname15\exp

Test Results Name: H:\Program Files\Mercury Interactive\WinRunner\tmp\noname15\res2

Operator Name:

Date: Tue Mar 17 12:06:04 2015

Summary:

Test Result: OK

Total number of bitmap checks: 0

Total number of GUI checks: 3

Total Run Time: 00:00:02

Detailed Results Description

Line Event Result Details Time

3 start run run noname15 00:00:00

12 start GUI checkpoint--- gui1 00:00:01

12 end GUI checkpointOK gui1 00:00:01

23 start GUI checkpoint--- gui2 00:00:02

23 end GUI checkpointOK gui2 00:00:02

24 start GUI checkpoint--- gui3 00:00:02

24 end GUI checkpointOK gui3 00:00:02

25 stop run pass noname15 00:00:02

Experiment No : 5
Date:

- a) Bitmap checkpoint for object/window
- b) Bitmap checkpoint for screen area

Title: Checking Bitmap Objects

Objective: Student should be able to

- ☐ Explains how to check bitmap images in an application
- ☐ Create a test that checks bitmaps
- ☐ Run the test in order to compare bitmaps in different versions of an application
- ☐ Analyze the results

Theory:

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.

Adding Bitmap Checkpoints to a Test Script

- 1 Start WinRunner and open a new test.
- 2 Start the Flight Reservation application and log in.
- 3 Start recording in Context Sensitive mode.
- 4 Open order #6.
- 5 Open the Fax Order dialog box.
- 6 Enter a 10-digit fax number in the Fax Number box. 7 Move the Fax Order dialog box.
- 8 Switch to Analog mode.
- 9 Sign your name in the Agent Signature box. 10 Switch back to Context Sensitive mode.
- 11 Insert a bitmap checkpoint that checks your signature. 12 Click the Clear Signature button.
- 13 Insert another bitmap checkpoint that checks the Agent Signature box. 14 Click the Cancel button on the Fax Order dialog box.
- 15 Stop recording. 16 Save the test.

Conclusion:

WinRunner Results - H:\Documents and Settings\JNEC-12\Desktop\bit

=====

===

Expected results folder: H:\Documents and Settings\JNEC-12\Desktop\bit\exp Test Results Name: H:\Documents and Settings\JNEC-12\Desktop\bit\res3 Operator Name:
Date: Tue Mar 17 11:44:42 2015

Summary:

Test Result: OK

Total number of bitmap
checks: 2 Total number of
GUI checks: 0 Total Run
Time: 00:00:38 Detailed
Results Description Line
Event Result Details Time

3 start run run bit 00:00:00
58 bitmap checkpoint OK Img1 00:00:37
67 bitmap checkpoint OK Img2 00:00:38
76 stop run pass bit 00:00:38

Experiment No : 6
Date:

Database checkpoint for Default check

When you create a default check on a database, you create a standard database checkpoint that checks the entire result set using the following criteria:

- The default check for a multiple-column query on a database is a case sensitive check on the entire result set by column name and row index.
- The default check for a single-column query on a database is a case sensitive check on the entire result set by row position.

If you want to check only part of the contents of a result set, edit the expected value of the contents, or count the number of rows or columns, you should create a custom check instead of a default check. For information on creating a custom check on a database, see “Creating a Custom Check on a Database,”

Creating a Default Check on a Database Using ODBC or Microsoft Query

You can create a default check on a database using ODBC or Microsoft Query.

To create a default check on a database using ODBC or Microsoft Query:

1. Choose **Insert > Database Checkpoint > Default Check** or click the **Default Database Checkpoint** button on the User toolbar. If you are recording in Analog mode, press the CHECK DATABASE (DEFAULT) softkey in order to avoid extraneous mouse movements. Note that you can press the CHECK DATABASE (DEFAULT) softkey in Context Sensitive mode as well.

2. If Microsoft Query is installed and you are creating a new query, an instruction screen opens for creating a query.

If you do not want to see this message next time you create a default database checkpoint, clear the **Show this message next time** check box.

Click **OK** to close the instruction screen.

If Microsoft Query is not installed, the Database Checkpoint wizard opens to a screen where you can define the ODBC query manually. For additional information, see “Setting ODBC (Microsoft Query) Options”

3. Define a query, copy a query, or specify an SQL statement. For additional information, see “Creating a Query in ODBC/Microsoft Query” or “Specifying an SQL Statement”
4. WinRunner takes several seconds to capture the database query and restore the WinRunner window.

WinRunner captures the data specified by the query and stores it in the test's exp folder.

WinRunner creates the msqr*.sql query file and stores it and the database checklist in the test's chklist folder. A database checkpoint is inserted in the test script as a **db_check** statement.

Creating a Default Check on a Database Using Data Junction

You can create a default check on a database using Data Junction.

To create a default check on a database:

1. Choose **Insert > Database Checkpoint > Default Check** or click the **Default Database Checkpoint** button on the User toolbar.

If you are recording in Analog mode, press the CHECK DATABASE (DEFAULT) softkey in order to avoid extraneous mouse movements. Note that you can press the CHECK DATABASE (DEFAULT) softkey in Context Sensitive mode as well.

For information on working with the Database Checkpoint wizard, see "Working with the Database Checkpoint Wizard"

2. An instruction screen opens for creating a query.

If you do not want to see this message next time you create a default database checkpoint, clear the **Show this message next time** check box.

Click **OK** to close the instruction screen.

3. Create a new conversion file or use an existing one. For additional information, see "Creating a Conversion File in Data Junction"
4. WinRunner takes several seconds to capture the database query and restore the WinRunner window.

WinRunner captures the data specified by the query and stores it in the test's exp folder.

WinRunner creates the *.djs conversion file and stores it in the checklist in the test's chklist folder.

A database checkpoint is inserted in the test script as a **db_check** statement

Experiment No : 7
Date:

Database checkpoint for custom check

When you create a custom check on a database, you create a standard database checkpoint in which you can specify which properties to check on a result set.

You can create a custom check on a database in order to:

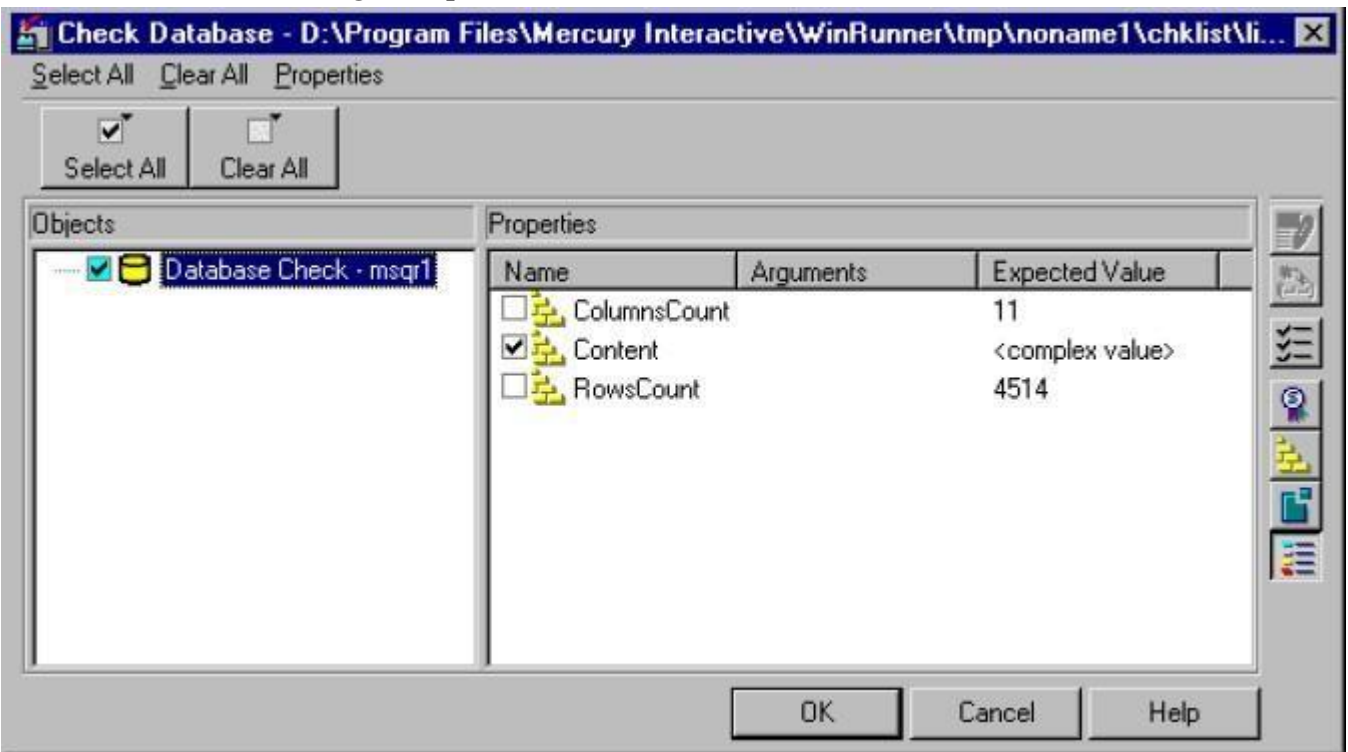
- check the contents of part or the entire result set
- edit the expected results of the contents of the result set
- count the rows in the result set
- count the columns in the result set

You can create a custom check on a database using ODBC, Microsoft Query or Data Junction.

To create a custom check on a database:

1. Choose **Insert > Database Checkpoint > Custom Check**. If you are recording in Analog mode, press the CHECK DATABASE (CUSTOM) softkey in order to avoid extraneous mouse movements. Note that you can press the CHECK DATABASE (CUSTOM) softkey in Context Sensitive mode as well.
2. Follow the instructions on working with the Database Checkpoint wizard, as described in “Working with the Database Checkpoint Wizard”
3. If you are creating a new query, an instruction screen opens for creating a query.
If you do not want to see this message next time you create a default database checkpoint, clear the **Show this message next time** check box.
4. If you are using ODBC or Microsoft Query, define a query, copy a query, or specify an SQL statement.
If you are using Data Junction, create a new conversion file or use an existing one.
5. If you are using Microsoft Query and you want to be able to parameterize the SQL statement in the **db_check** statement which will be generated, then in the last wizard screen in Microsoft Query, click **View data or edit query in Microsoft Query**. Follow the instructions in “Parameterizing Standard Database Checkpoints”
6. WinRunner takes several seconds to capture the database query and restore the WinRunner window.

The Check Database dialog box opens



The **Objects** pane contains “Database check” and the name of the *.sql query file or *.djs conversion file included in the database checkpoint. The **Properties** pane lists the different types of checks that can be performed on the result set. A check mark indicates that the item is selected and is included in the checkpoint.

7. Select the types of checks to perform on the database. You can perform the following checks:

ColumnsCount: Counts the number of columns in the result set.

Content: Checks the content of the result set, as described in “Creating a Default Check on a Database,”

RowCount: Counts the number of rows in the result set.

If you want to edit the expected value of a property, first select it. Next, either click the **Edit Expected Value** button, or double-click the value in the Expected Value column.

- For **ColumnsCount** or **RowCount** checks on a result set, the expected value is displayed in the **Expected Value** field corresponding to the property check. When you edit the expected value for these property checks, a spin box opens. Modify the number that appears in the spin box.
- For a **Content** check on a result set, <complex value> appears in the **Expected Value** field corresponding to the check, since the content of the result set is too complex to be displayed in this column. When you edit the expected value, the **Edit Check** dialog box opens. In the **Select Checks** tab, you can select which checks to perform on the result set, based on the data captured in the query. In the **Edit Expected Data** tab, you can modify the expected results of the data in the result set.

8. Click **OK** to close the Check Database dialog box.

WinRunner captures the current property values and stores them in the test's exp folder. WinRunner stores the database query in the test's chklist folder. A database checkpoint is inserted in the test script as a **db_check** statement.

ASAP

Experiment No : 8
Date:

Database checkpoint for runtime record check

You can make changes to a checklist you created for a runtime database record checkpoint. Note that a checklist includes the connection string to the database, the SQL statement or a query, the database fields in the data source, the controls in your application, and the mapping between them. It does not include the success conditions of a runtime database record checkpoint.

When you edit a runtime database record checklist, you can:

- modify the data source connection string manually or using ODBC
- modify the SQL statement or choose a different query in Microsoft Query
- select different database fields to use in the data source (add or remove)
- match a database field already in the checklist to a different application control
- match a new database field in the checklist to an application control

To edit an existing runtime database record checklist:

1. Choose **Insert > Edit Runtime Record Checklist**.

The Runtime Record Checkpoint wizard opens.



- Choose the runtime database record checklist to edit.
2. Click **Next** to proceed.
 - The Specify SQL Statement screen opens:



In this screen you can

- modify the connection string manually or by clicking **Edit** to open the ODBC Select Data Source dialog box, where you can select a new *.dsn file in the Select Data Source dialog box to create a new connection string
 - modify the SQL statement manually or redefine the query by clicking the **Microsoft Query** button to open Microsoft Query
3. Click **Next** to continue.
- The Match Database Field screen opens:



"New" icon indicates that this database field was not previously included in the checklist.

- For a database field previously included in the checklist, the database field is displayed along with the application control to which it is mapped. You can use the pointing hand to map the displayed field name to a different application control or text string in a Web page.
 - If you modified the SQL statement or query in Microsoft Query so that it now references an additional database field in the data source, the checklist will now include a new database field. You must match this database field to an application control. Use the pointing hand to identify the control or text that matches the displayed field name.
4. Click **Next** to continue.
 - The Finished screen is displayed.
 5. Click **Finish** to modify the checklist used in the runtime record checkpoint(s).

Experiment No : 9
Date:

- a) Data driven test for dynamic test data submission
- b) Data driven test through flat files
- c) Data driven test through front grids
- d) Data driven test through excel test

Title: Creating data driven test

Objective: Student should be able to

- ☐ Use the DataDriver Wizard to create a data-driven test
- ☐ Use regular expressions for GUI object names that vary with each iteration of a test
- ☐ Run a test with several iterations and analyze the results.

Theory:

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

- ☐ Adding statements to your script that open and close the data table.
- ☐ Adding statements and functions to your test so that it will read from the data table and run in a loop while it applies each set of data.
- ☐ Replacing fixed values in recorded statements and checkpoint statements with parameters, known as *parameterizing* the test. You can convert your test to a data-driven test using the DataDriver Wizard or you can modify your script manually. When you run your data-driven test, WinRunner runs the parameterized part(s) of the test one time (called an *iteration*) for each set of data in the data table, and then displays the results for all of the iterations in a single Test Results window.

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

1 Create a new test from the test experiment 6.

2 Run the DataDriver Wizard.

3 Create a data table for the test.

4 Assign a table variable name.

5 Select global parameterization options.

6 Select the data to parameterize.

7 Open the data table.

8 Add data to the table.

9 Save and close the table.

10 Save the test.

11 Locate the Fax Order window in the *flight1a.gui* GUI map file.

12 Modify the window label with a regular expression.

13 Close the Modify dialog box.

14 Modify the *tl_step* statements.

Locate the first **tl_step** statement in your script. Delete the words “total is

correct.” and replace them with, “Correct. "tickets" tickets at \$"price" cost \$"total".”

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

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

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

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

15 Save the test.

Conclusion

Line	Event	Detail	Result	Time
1	start run Lesson8		Run	00:00:00
22	tl_step	Step total: Status: Pass, Description: Correct. 4 tickets at \$321.40 cost \$1293.60	—	00:00:02
22	tl_step	Step total: Status: Pass, Description: Correct. 1 tickets at \$371.00 cost \$371.00	—	00:00:05
22	tl_step	Step total: Status: Pass, Description: Correct. 1 tickets at \$337.40 cost \$337.40	—	00:00:07
22	tl_step	Step total: Status: Pass, Description: Correct. 2 tickets at \$364.54 cost \$729.08	—	00:00:09
22	tl_step	Step total: Status: Pass, Description: Correct. 2 tickets at \$161.80 cost \$321.60	—	00:00:12
28	stop run Lesson8		pass	00:00:12

Experiment No :10
Date:

- Batch testing without parameter passing
- Batch testing with parameter passing

Create and run a test batch

Relevant for: GUI tests and components and API testing

Test Batch Runner enables you to run tests in a collective, successive test run.

Tests are run individually but sequentially in a single session.

In this topic:

- [Open the Test Batch Runner](#)
- [Add batches or tests](#)
- [Run the test batch](#)
- [Run the test batch via the command line](#)
- [View the test batch run results](#)

Open the Test Batch Runner

You do not need to have UFT One open to use the Test Batch Runner.

Start the Test Batch Runner from the Start menu, or from the following path: <UFT One installation folder/bin/UFT OneBatchRunner.exe.

Tip: Once open, if you are using a concurrent license, select **Test > Close UFT One after Test Run** to close UFT One and release the license once the test run is complete.


[↑ Back to top](#)

Add batches or tests


Use Test Batch Runner to create a list of tests that can be used to run the same batch of tests again another time. Save the list as a batch .mtb file.

Tip: You can include or exclude a test in your batch list from running during a particular batch run without affecting the other tests in the batch.

**Add a test batch
file (.mtb)**

- Select **File > Add** or click the **Add** button  .
- Navigate to the folder in which the batch file is saved.

Add individual tests

1. Select **Tests > Add** or click the **Add** button .
2. In the **Browse For Folder** dialog box, select the folder in which your tests are located.

All the tests from the selected folder are added to the **Tests** pane in the main Test Batch Runner window.

Note: When adding tests through the **Tests > Add** menu command, you must select all the tests from the target folder.

If you do not want to run all the tests in the target folder, select the check boxes next to the tests you want to run before you run the test batch.

[↑ Back to top](#)

Run the test batch

Click the **Run** button  to run the test batch.

The Output pane allows you view the results of the test run in run time, including:

- The test's path in the file system
- The progress of the test
- Any errors that occur during the run

Run the test batch via the command line

Run the test batch via the command line to include UFT One tests in a build run, in a continuous integration system.

In the Command Line window, enter **UFTBatchRunnerCMD.exe** and the **source** switch followed by the test batch file (**.mtb**) or folder containing the test.

For example, your command line might contain text like this:

```
UFTBatchRunnerCMD.exe -source "C:\users\MySample.mtb"  
UFTBatchRunnerCMD.exe -source "C:\users\APITest1"  
Pass test parameters in your command
```

To add test parameters to your command, use the following syntax:

```
UFTBatchRunnerCMD.exe -source "C:\users\GUITest1" -parameter  
"Parameter1=UFT;Parameter2=Test"
```

Note: Passing test parameters is supported for running single tests only, and not for all tests in a folder.

Run the test batch using a Runtime Engine license

To run a test batch using a Runtime Engine license only, with UFT One on hidden mode, add the -visible N parameter to your command line.

For example:

```
UFTBatchRunnerCMD.exe -visible N -source "C:\users\MySample.mtb"
```

View the test batch run results

Following the test batch run, the results are saved to a run results file.

This file includes details about whether the test passed or failed and errors in running the test.

In the Tests pane, click the results link for a specific test in the **Report** column.

Experiment No : 11
Date:

Data driven batch

Data driven batch

How Do You Create Data-Driven Tests?

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

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

- Adding statements to your script that open and close the data table.
- Adding statements and functions to your test so that it will read from the data table and run in a loop while it applies each set of data.
- Replacing fixed values in recorded statements and checkpoint statements with parameters, known as parameterizing the test.

You can convert your test to a data-driven test using the DataDriver Wizard or you can modify your script manually. When you run your data-driven test, WinRunner runs the parameterized part(s) of the test one time (called an iteration) for each set of data in the data table, and then displays the results for all of the iterations in a single Test Results window.

Converting Your Test to a Data-Driven Test

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



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

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

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

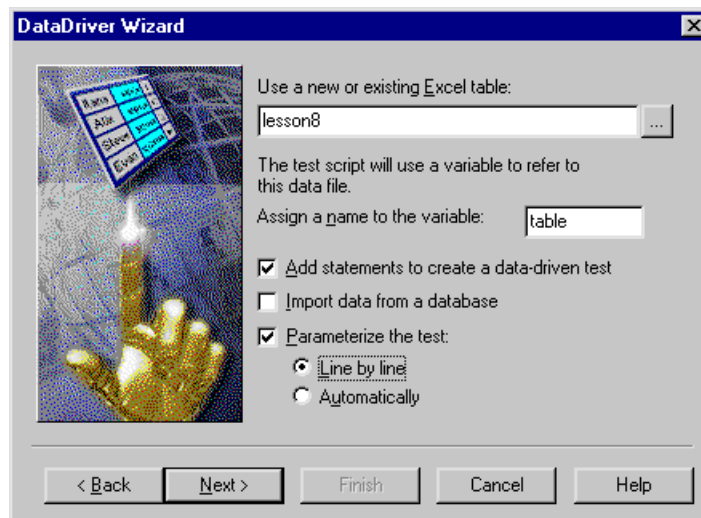
If you are working in the Global GUI Map File mode, confirm that the GUI map is loaded. To do this, choose **Tools > GUI Map Editor**. In the GUI Map Editor choose **View > GUI Files** and confirm that **flight4a.GUI** is contained in the **GUI File** list.

2 Run the Data Driver Wizard.

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

3. Create a data table for the test.

In the **Use a new or existing Excel table** box, type **lesson8**. The DataDriver Wizard creates a Microsoft



Excel table with this name and saves it in the test folder.

3 Assign a table variable name.

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

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

4 Select global parameterization options.

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

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

Click **Next**.

5 Select the data to parameterize.

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

In this test you are going to open a different fax order in each iteration and

Test script line to parameterize:
button_set ("Order No.", ON);

the **Order Number** radio button must be selected each time. Thus, for this script line, keep the selection, **Do not replace this data**, and click **Next**.

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

Test script line to parameterize:
edit_set ("Edit_1", "3");
Argument to be replaced: "3"

Select **A new column** under **Replace the selected value with data from:** and type **Order_Num** in the adjacent box. The new column option creates a column titled **Order_Num** in the **lesson8.xls** table, and enters the value **3** in the first row of the column.

Click **Next** and then click **Finish**. Your test is parameterized.

FYI:

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

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

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

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

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

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

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

The **ddt_close** statement closes the table.

Adding Data to the Data Table

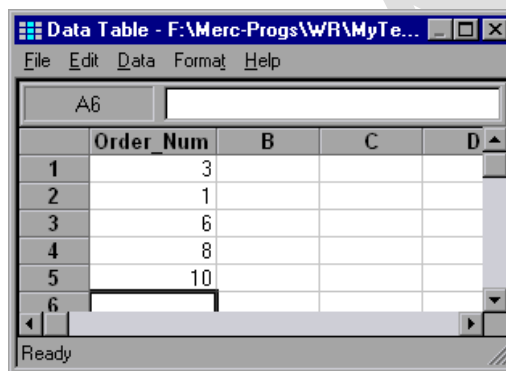
Now that you have parameterized your test, you are ready to add the data that the parameterized test will use.

1 Open the data table.

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

2 Add data to the table.

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



	Order_Num	B	C	D
1	3			
2	1			
3	6			
4	8			
5	10			
6				

2 Save and close the table.

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



4 Save the test.

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

Adjusting the Script with Regular Expressions

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

In the flight application, the name of the Fax Order window changes to reflect the fax order number. If you run the test as it is, the test will fail on the second iteration, because the Flight Application will open a window

titled **Fax Order No. 1**, but the script tells it to make the window titled **Fax Order No. 3** active. WinRunner will be unable to find this window.

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

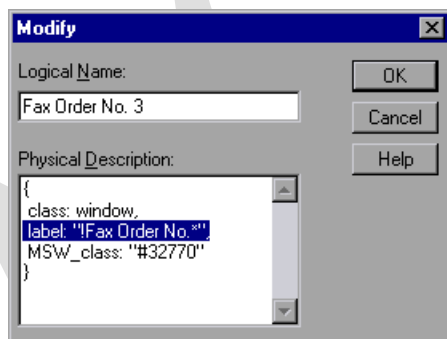
In this exercise you will use a regular expression in the physical description of the Fax Order window so that WinRunner can ignore variations in the window's label.

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

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

- 2 Modify the window label with a regular expression.

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



- 3 Close the Modify dialog box.

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

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

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

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

Customizing the Results Information

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

1 Modify the `tl_step` statements.

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

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

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

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

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



2 Save the test.

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

Running the Test and Analyzing Results

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

1 Make sure that the Flight 4A Flight Reservation application is open on your desktop.

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

3 Choose Run from Top.



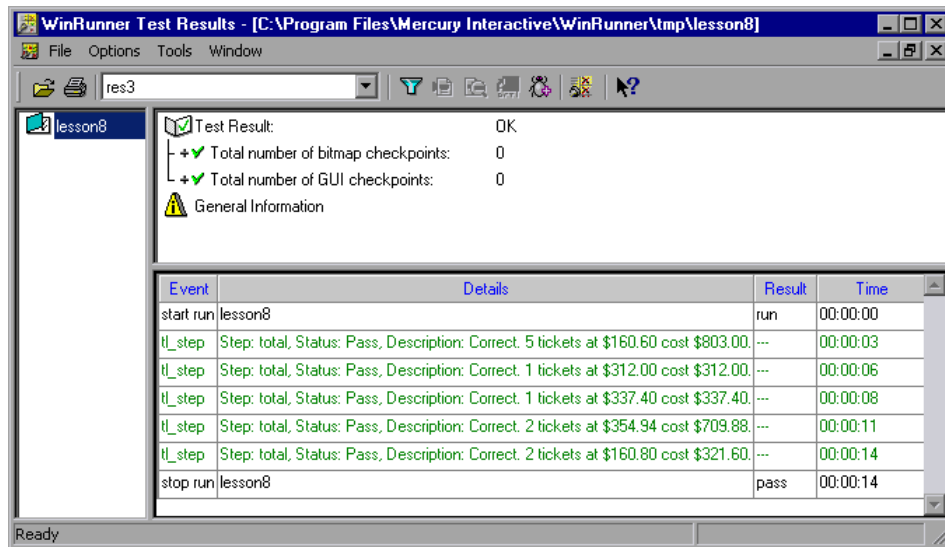
Choose **Run > Run from Top**, or click the **Run from Top** button. The **Run Test** dialog box opens. Accept the default test run name. Make sure that the **Display test results at end of run** check box is selected.

4 Run the test.

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

5 Review the results.

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



Note that the **tl_step** event is listed five times and that the details for each iteration include the actual number of tickets, price and total cost that was checked.

6 Close the test results.

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

7 Close the Flight Reservation application.

Choose **File > Exit**.

8 Close the lesson8 test.

Choose **File > Close**.

Experiment No : 12
Date:

Silent mode test execution without any interruption

Running silent installation

1. Deploy the TestExecute installation package on the target computer.
2. On the target computer, run the installation using the needed command-line arguments. For example:

```
<path_to_installation_package>\TestExecute1490_Release.exe -SilentInstall
```

3. The installation program will install TestExecute without user interaction.

To run the installation using specific command-line arguments, administrator permissions are required.

Supported command-line arguments

The installation package supports the following command-line arguments:

```
<TestExecute1490_Release.exe> [< mode> <options> <modules>] [ /? ]
```

Mode

Must be one of the following:

-Install

Commands the installation program to run in interactive (GUI) mode.

-SilentInstall

Commands the installation program to run in silent mode.

-SilentUninstall

Commands the installation program to uninstall the product in silent mode.

-Uninstall

Commands the installation program to uninstall the product in interactive (GUI) mode.

Notes:

- If you specify the *-Uninstall* or *-SilentUnInstall* argument, the installation program will ignore the other command-line arguments and uninstall TestExecute.

- To uninstall TestExecute, use the installation package of the TestExecute version that is currently installed.

Options

The following arguments are optional. If the argument value includes spaces, enclose it in double quotes. For example:

```
-log="C:\my install files"
```

```
-Log=Log_Path
```

Saves the installation log to the specified folder. For example:

```
-log=C:\source
```

If the folder name is omitted, the log will be saved to the folder where the installation package resides:

```
-log
```

```
-InstallPath=Application_Path
```

TestExecute is installed into the specified folder.

```
-ExternalData=Data_Path
```

TestExecute is installed by using the specified external installation package (.exe).

```
-CleanUp
```

TestExecute and all its features are removed from the computer and the installation information is removed from the **Programs and Features** feature of Windows. Use it if the ordinary uninstallation has failed to remove all product files from the computer.

Modules

```
-IntelligentQuality, -Desktop, -Web, -Mobile
```

The specified modules and add-ons will be installed and activated.

By installing the Intelligent Quality add-on in silent mode, you confirm that you consent to the [third-party terms of use](#).

If a module or add-on is not specified as an argument parameter, it will not be installed. For example, the Desktop and Mobile modules will be installed, but the Web module and the add-ons will be skipped:

```
<path_to_installation_package>\TestExecute1490_Release.exe -SilentInstall -Desktop-Mobile
```

The Desktop module and the Intelligent Quality add-on will be installed, but the Web and Mobile modules will be skipped:

```
<path_to_installation_package>\TestExecute1490_Release.exe -SilentInstall -Desktop -IntelligentQuality
```

If no modules or add-ons are specified during the installation, all the available modules and add-ons will be installed.

Get help on the installation

To view information on available command-line arguments, run the installation package with the `/?` argument:

```
<TestExecute1490_Release.exe> /?
```

Any other command-line arguments will be ignored.

Check the silent installation success

In silent mode, the installation wizard shows no messages. You can use the `-Log` command-line argument to configure the installation to create a log file and write all the progress information to it. After the installation is over, you can examine that log file to learn what was happening during the installation and the installation result.

Experiment No :13**Date:**

Test case for calculator in windows application

1. Check if the calculator is a normal calculator or a scientific calculator.
2. Verify that all the buttons are present and text written on them is readable.
3. Check the arithmetic operations are working fine- +, -, /, * etc.
4. Verify that BODMAS is applied in case of complex queries and the correct result is returned.
5. Verify that the calculator gives the correct result in case of operations containing decimal numbers.
6. Check if the calculator is battery operated or works on solar power.
7. Verify the outer body material of the calculator.
8. Verify the spacing between the two buttons, the buttons should not be too closely placed.
9. Check the pressure required to press a button, the pressure required should not be too high.
10. Verify the number of digits allowed to enter in the calculator for any operation.
11. Verify the limit of the response value.
12. Verify the functioning of memory functions.
13. Check if the calculator allows navigating through previous calculations.
14. Verify that hitting 'C' cancels any digits or operation added.
15. Verify the working of the ON-OFF button in the calculator.
16. Check if keeping the calculator unused for a certain period of time, turns it off automatically.
17. Verify that on pressing two operators one after the other, the latest one will override the previous operator.
18. Verify the state of the calculator when two buttons are pressed simultaneously.
19. Verify if the user can delete digits one by one using the backspace key.