

Simplified Steps to Learn QuickTest Professional

INDEX

1.0	<u>Introduction to QuickTest</u>	5
1.1	<u>Over view of Quick Test Pro</u>	5
1.1.1	<u>QuickTest Pro Environment Support</u>	5
1.1.2	<u>QuickTest Pro Configurations</u>	6
2.0	<u>Record and Playback</u>	7
2.1	<u>Create and Execute Basic Scripts</u>	7
2.1.1	<u>Recording Tests</u>	7
2.1.2	<u>Running a Test</u>	9
2.2	<u>Understand Recording Levels</u>	9
2.2.1	<u>Standard Recording</u>	10
2.2.2	<u>Analog Recording</u>	10
2.2.3	<u>Low Level Recording</u>	10
2.3	<u>Understand QuickTest Results</u>	10
3.0	<u>How QuickTest identifies objects</u>	14
3.1	<u>Object Identification</u>	14
3.1.1	<u>Object Identification While Recording</u>	14
3.1.2	<u>Object Identification During Test Run</u>	14
3.2	<u>Object Repository Introduction</u>	14
3.2.1	<u>Identifying the Object</u>	15
3.2.2	<u>Viewing the Object's Properties</u>	15
3.3	<u>Use the Object Spy</u>	16
3.3.1	<u>To view object properties:</u>	16
3.3.2	<u>To view object methods:</u>	17
4.0	<u>Synchronization</u>	18
4.1	<u>Synchronizing Your Tests</u>	18
4.2	<u>Options to Synchronize Tests</u>	18
4.1.1	<u>4.2.1 Inserting Synchronization Point</u>	18
4.1.2	<u>4.2.2 Adding Exist and Wait Statements</u>	19
4.1.3	<u>Global synchronization Settings</u>	19
4.3	<u>Transactions</u>	19
4.1.4	<u>4.3.1 Inserting Transactions</u>	19
4.1.5	<u>4.3.2 Ending Transactions</u>	20
5.0	<u>Checkpoints</u>	21
5.1	<u>About Checkpoints</u>	21
5.2	<u>Adding Checkpoints to a test</u>	21
5.1.1	<u>5.2.1 To add checkpoints while recording:</u>	21
5.1.2	<u>To add a checkpoint while editing your test:</u>	21
5.1.3	<u>5.2.2.1 From Menu bar</u>	21
5.3	<u>Types of Checkpoints</u>	21
5.1.4	<u>5.3.1 QuickTest Professional Checkpoint Types</u>	21

5.1.5	5.3.2 Creating a Standard Checkpoint	22
5.1.6	5.3.3 Creating a Text Checkpoint	22
5.4	Use regular expressions	22
5.1.7	5.4.1 To define a constant property value as a regular expression:	23
5.1.8	5.4.2 To parameterize a property value using regular expressions:	23
5.1.9	5.4.3 To define a regular expression in an object checkpoint:	23
5.1.10	5.4.4 Common options to create regular expressions.	23
6.0	Creating Tests with Multiple Actions	25
6.1	Benefits of Test Modularity	25
6.2	Creating Tests with Multiple Actions	25
6.1.1	6.2.1 Creating New Actions	25
6.1.2	6.2.2 Inserting Existing Actions	26
6.1.3	6.2.3 Nesting Actions	26
6.1.4	6.2.4 Splitting Actions	27
6.3	Miscellaneous	27
6.1.5	6.3.1 Setting Action Properties	27
6.1.6	6.3.2 Sharing Action Information	27
6.1.7	6.3.3 Exiting an Action	27
6.1.8	6.3.4 Removing Actions from a Test	27
6.1.9	6.3.5 Renaming Actions	27
6.1.10	6.3.6 Renaming Actions	27
7.0	Data Driving a Test	29
7.1	Parameterize tests	29
7.1.1	7.1.1 Parameterize test Manually	29
7.1.2	7.1.2 DataTable Parameters	29
7.1.3	7.1.3 Using Environment Variable Parameters	29
7.2	Create data-driven tests	30
7.3	Local and Global Data Tables	30
7.1.4	7.3.1 Using the Data Driver to Parameterize Your Test	31
8.0	Working with Data Tables	33
8.1	Introduction	33
8.2	Working with Global and Action Sheets	33
8.3	Editing and Saving Data Table	33
8.4	Importing Data from a Database	34
8.5	Using Formulas in the Data Table	34
8.6	Using Data Table Scripting Methods	35
9.0	Output and Correlation	36
9.1	About Outputting Values	36
7.1.5	9.1.1 Creating Page Output Values	36
7.1.6	9.1.2 Creating Text Output Values	36
7.1.7	9.1.3 Creating Standard Output Values	36
7.1.8	9.1.4 Creating Image Output Values	37
7.1.9	9.1.5 Creating XML Output Values	37
7.1.10	9.1.6 Creating Table Output Values	37
7.1.11	9.1.7 Creating Database Output Values	37
9.2	Capture and Reuse Run Time data	37
7.1.12	9.2.1 Adding a Standard Output Value	37
7.1.13	9.2.2 Creating Image Output Values	37
7.1.14	9.2.3 Creating Table Output Values	38
10.0	Alternatives to Standard Recording	39
10.1	Analog Recording	39
7.1.15	10.1.1 Analog Recording	39
7.1.16	10.1.2 Recording in Analog Mode	39
10.2	Low-Level Recording	40

7.1.17	10.2.1 Recording in Low-Level mode	40
10.3	Configuring Web Event Recording	41
7.1.18	10.3.1 To set Web Event Recording Configuration:	41
10.4	Define a Virtual Object	41
7.1.19	10.4.1 To define a virtual object:	41
11.0	Introduction to the Expert View	44
11.1	Object Model in the Expert View	44
11.2	Using QuickTest Professional's online books	44
12.0	Working in the Expert View	46
12.1	VBScript Language Overview	46
7.1.20	12.1.1 VBScript Data Types	46
7.1.21	12.1.2 VBScript Variables	46
7.1.22	12.1.3 VBScript Constants	47
7.1.23	12.1.4 VBScript Operators	47
7.1.24	12.1.5 Using Conditional Statements	48
7.1.25	12.1.6 Looping Through Code	49
7.1.26	12.1.7 VBScript Procedures	50
12.2	Working with the Data Table Object	51
7.1.27	12.2.1 AddSheet Method	51
7.1.28	12.2.2 DeleteSheet Method	51
7.1.29	12.2.3 Export Method	51
7.1.30	12.2.4 ExportSheet Method	51
7.1.31	12.2.5 GetCurrentRow Method	51
7.1.32	12.2.6 GetRowCount Method	51
7.1.33	12.2.7 GetSheet Method	51
7.1.34	12.2.8 GetSheetCount Method	52
7.1.35	12.2.9 Import Method	52
7.1.36	12.2.10 ImportSheet Method	52
7.1.37	12.2.11 SetCurrentRow Method	52
7.1.38	12.2.12 SetNextRow Method	52
7.1.39	12.2.13 SetPrevRow Method	52
7.1.40	12.2.14 GlobalSheet Property	52
7.1.41	12.2.15 LocalSheet Property	52
7.1.42	12.2.17 RawValue Property	52
7.1.43	12.2.18 Value Property	53
12.3	Working with TextUtil Object	53
7.1.44	12.3.1 GetText Method	53
7.1.45	12.3.2 GetTextLocation Method	53
12.4	Working with Reporter Objects	53
7.1.46	12.4.1 ReportEvent Method	53
7.1.47	12.4.2 Filter Property	53
7.1.48	12.4.3 ReportPath Property	53
13.0	Object Recognition and Smart Identification	54
13.1	Object Repository Custom Configuration	54
13.2	Introduction to Smart Identification	54
13.2.1	13.2.1 Base filter properties	55
13.2.2	13.2.2 Optional filter properties	55
13.3	Understanding the Smart Identification Process	55
13.4	Smart Identification Configuration	55
14.0	Enhance Test Cases with Descriptive Programming	57
14.1	Interact with Test Objects not stored in the Object Repository	57
14.1.1	14.1.1 Entering Programmatic Description Directly into Test Statements	57
14.1.2	14.1.2 Using Description Objects for Programmatic Descriptions	58
14.1.3	14.1.3 Retrieving ChildObjects	58
14.1.4	14.1.4 Using Programmatic Descriptions for the WebElement Object ..	59

14.1.5	14.1.5 Using the Index Property in Programmatic Descriptions	59
14.2	Access Dynamic Objects during run-time	59
14.2.1	14.2.1 Retrieving Run-Time Object Properties	59
14.2.2	14.2.2 Activating Run-Time Object Methods	60
15.0	Enhance Test Cases with User-Defined Functions	61
15.1	Utilize external Windows API functions in Test Cases	61
15.1.1	15.1.1 Extern Object	61
15.2	Create QuickTest user-defined functions	63
16.0	Database Verification	65
16.1	Review of database concepts	65
16.1.1	Understanding relational tables	65
16.1.2	About SQL	66
16.1.3	Using SQL to interact with a database	66
16.1.4	Connection String	66
16.2	How to add a database checkpoint in QuickTest	67
17.0	Recovery Manager and Scenarios	73
18.0	Scripting in Real Time Environments	99
18.1	QuickTest Pro Coding Standards & Best Practices	99
18.1.1	Introduction:	99
18.2	Naming Conventions	99
18.2.1	Local scope variables	99
18.2.2	Global scope variables	99
18.2.3	Constants	100
18.2.4	Functions/Actions	100
18.2.5	Reusable Actions	100
18.2.6	Scripts	100
18.2.7	Function Libraries	100
18.2.8	Object Repository Files	101
18.3	Coding Rules	101
18.3.1	Commenting Code	101
18.3.2	Formatting Code	103
18.3.3	Using Shared Object Repository	103
18.3.4	Using Relative paths	104
18.3.5	Using Global Variables	104

1.0 Introduction to QuickTest Professional

1.1 Over view of Quick Test Professional

QuickTest Professional is functional enterprise testing tool from HP.

QuickTest Professional is a fresh approach to automated software and application testing .It is designed to provide a robust application verification solution without the need for advanced technical or programming

QuickTest Professional is similar to Astra QuickTest. The key difference lies in the number of environments that QuickTest Professional supports (e.g. ERP/CRM, Java applets and applications, multiple multimedia environments, etc.).

QuickTest Professional enables you to test standard Windows applications, Web objects, ActiveX controls, Visual Basic applications, and multimedia objects on Web pages.

We Can Use QuickTest add-ins for a number of special environments (such as Java, Oracle, SAP solutions, .NET Windows and Web Forms, Siebel, PeopleSoft, Web services, and terminal emulator applications).

1.1.1 QuickTest Professional Environment Support

Windows Applications (MFC) <ul style="list-style-type: none"> • Visual Basic • Java • ActiveX 	Enterprise Applications <ul style="list-style-type: none"> • SAP • Oracle • PeopleSoft • Siebel 	Web Technologies <ul style="list-style-type: none"> HTML • DHTML • JavaScript
Browsers <ul style="list-style-type: none"> IE • Netscape • AOL 	Emerging Technologies <ul style="list-style-type: none"> • .Net Winforms, Webforms, Web services • J2EE Web services • XML, WSDL, UDDI 	Terminal Emulators <ul style="list-style-type: none"> • 3270 • 5250 • VT100
Server Technologies <ul style="list-style-type: none"> • Oracle • Microsoft <ul style="list-style-type: none"> • IBM • BEA • ODBC • COM/COM+ 	Multimedia <ul style="list-style-type: none"> • RealAudio/RealVideo • Windows Media Player <ul style="list-style-type: none"> • Flash 	Languages <ul style="list-style-type: none"> • European • Japanese • Chinese (traditional and simplified) • Korean

1.1.2 QuickTest Pro Configurations

QuickTest Pro	QuickTest Pro Add-ins
<p>Web Environments IE NS AOL ActiveX XML DHTML HTML</p> <p>Client/Server Windows Win32/MFC Visual Basic</p> <p>Operating Systems Windows 98, 2000, NT, ME, XP</p>	<p>.Net Add-in Winforms, Webforms, Net Controls</p> <p>Java Add-in JDK 1.1-1.4.2</p> <p>Terminal Emulator Add-in 3270,5250,vt100</p> <p>MySAP Add-in SAP GUI, Web, Portals 6.2</p> <p>Oracle Add-in 11i</p> <p>PeopleSoft Add-in 8.0-8.8</p> <p>Siebel Add-in 7.0 & 7.5</p> <p>Webservices Add-in WSDL,.Net,J2EE</p>

2.0 Record and Playback

2.1 Create and Execute Basic Scripts

2.1.1 Recording Tests

1. Start QuickTest and open a new test.
 - If QuickTest is not currently open, choose Start > Programs > QuickTest Professional > QuickTest Professional.

In the Add-in Manager, confirm that the Web Add-in is selected, and clear all other add-ins. Click OK to close the Add-in Manager and open QuickTest.

Note: While QuickTest loads your selected add-ins, the QuickTest splash screen is displayed. This may take a few seconds. If the Welcome window opens, click Blank Test.

Otherwise, choose File > New, or click the New button .

A blank test opens.

- If QuickTest is already open, check which add-ins are loaded by selecting Help > About QuickTest Professional. If the Web Add-in is not loaded, you must exit and restart QuickTest. When the Add-in Manager opens, select the Web Add-in, and clear all other add-ins.

Choose File > New, or click the New button .

A blank test opens.

Note: If the Add-in Manager does not open when starting QuickTest, choose Tools > Options. In the General tab, select Display Add-in Manager on startup. When you exit and restart QuickTest, the Add-in Manager opens.

2. Start recording.
 - Choose Test > Record or click the Record button. The Record and Run Settings dialog box opens.

In the Web tab, select Open the following browser when a record or run session begins.

- Choose a browser from the Type list and confirm that the URL in the Address box is for example . <http://newtours.mercuryinteractive.com>.
- Confirm that Close the browser when the test is closed is selected.
- In the Windows Applications tab, confirm that Record and run on these applications is selected, and that there are no applications listed.

This setting prevents you from inadvertently recording operations performed on various Windows applications (such as e-mail) during a recording session.

- Click OK.

QuickTest begins recording, and your browser opens to the Mercury Tours Web site.

3. Login to the Mercury Tours Web site.

In the User Name and Password boxes, type the name and password you registered with Mercury Tours.

Click Sign-In.

The Flight Finder page opens.

4. Enter flight details.

Change the following selections:

Departing From: New York

On: Dec 29

Arriving In: San Francisco

Returning: Dec 31

Service Class: Business class

Click CONTINUE to accept the other default selections. The Select Flight page opens.

Note: When entering dates while recording this test, do not click the View Calendar button, which opens a Java-based calendar. Your test will not record the date selected using this calendar because you did not load the Java Add-in for this tutorial.

To check which add-ins have been loaded, click Help > About QuickTest Professional. To change the available add-ins for your tests, you must close and reopen QuickTest Professional.

5. Select a flight.

Click CONTINUE to accept the default flight selections. The Book a Flight page opens.

6. Enter required passenger and purchase information.

Enter the required information (fields with red text labels) in the Passengers and Credit Card sections. (You may enter fictitious information.)

In the Billing Address section, select Ticketless Travel.

At the bottom of the page, click SECURE PURCHASE. The Flight Confirmation page opens.

7. Review and complete your booking.

Click BACK TO HOME. The Mercury Tours home page opens.

8. Stop recording.

In QuickTest, click Stop on the test toolbar to stop the recording process.

You have now reserved an imaginary business class ticket from New York to San Francisco. QuickTest recorded your Web browser operations from the time you clicked the Record button until you clicked the Stop button.

Save your test.

Select File > Save or click the Save button. The Save dialog box opens to the Tests folder.

Create a folder named Tutorial, select it, and click Open.

Type Recording in the File name field.

Confirm that Save Active Screen files is selected.

Click Save. The test name (Recording) is displayed in the title bar of the main QuickTest window.

2.1.2 Running a Test

Here you will run the test you recorded.

- 1) Start QuickTest and open the Recording test.

If QuickTest is not already open, choose Start > Programs > QuickTest Professional > QuickTest Professional.

- If the Welcome window opens, click Open Existing.
- If QuickTest opens without displaying the Welcome window, choose File > Open or click the Open button.

In the Open Test dialog box, locate and select the Recording test, then click Open.

- 2) Confirm that all images are saved to the test results.

QuickTest allows you to determine when to save images to the test results. In this lesson, all images should be saved to the test results.

Choose Tools > Options and select the Run tab. In the Save step screen capture to test results option, select always.

Click OK to close the Options dialog box.

- 3) Start running your test.

Click Run or choose Test > Run. The Run dialog box opens.

Select New run results folder. Accept the default results folder name.

Click OK to close the Run dialog box.

Watch carefully as QuickTest opens your browser and starts running the test. In the browser, you can see QuickTest perform each step you recorded; a yellow arrow in the left margin of the test tree indicates the step that QuickTest is running.

2.2 Understand Recording Levels

2.2.1 Standard Recording

Records the test in terms of GUI objects

2.2.2 Analog Recording

enables you to record the exact mouse and keyboard operations you perform in relation to either the screen or the application window.

2.2.3 Low Level Recording

This mode records at the object level and records all run-time objects as Window or WinObject test objects.

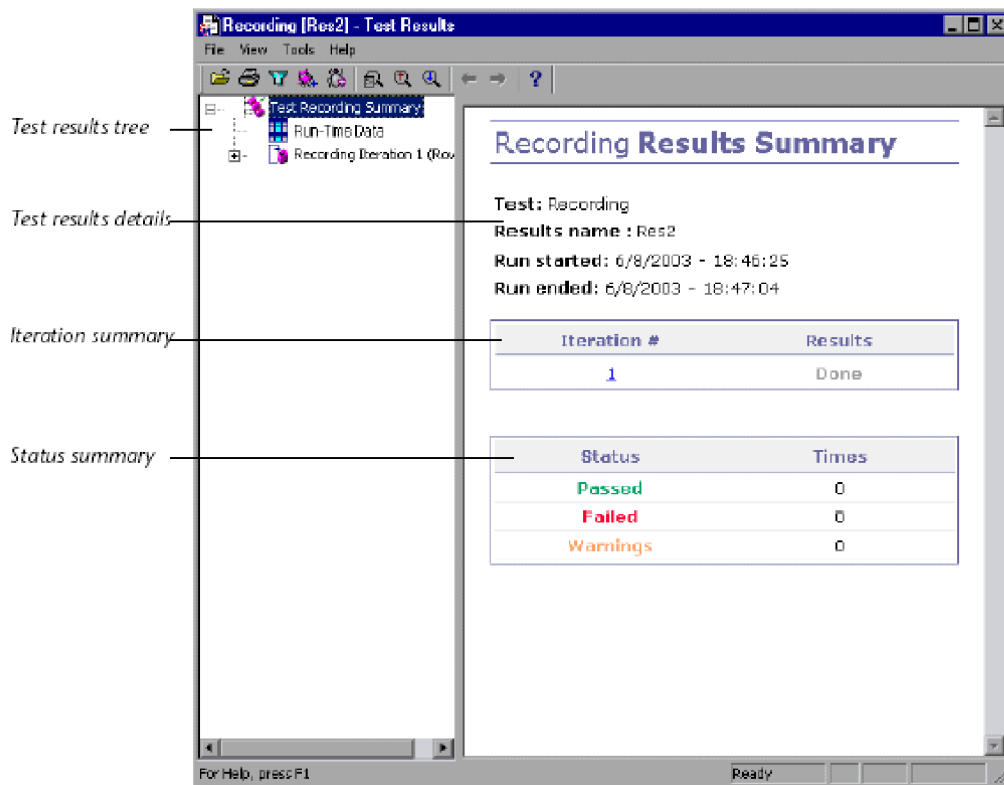
2.3 Understand QuickTest Results

When QuickTest finishes running the test, the Test Results window opens.

Initially, the Test Results window contains two panes for displaying the key elements of your test run.

- The left pane displays the test results tree, an icon-based view of the steps that were performed while the test was running. Similar to the test tree in QuickTest main screen, it is organized according to the Web pages visited during the test run and can be expanded (+) to view each step. The steps performed during the test run are represented by icons in the tree. You can instruct QuickTest to run a test or action more than once using different sets of data in each run. Each test run is called an iteration and each iteration is numbered. (The test you ran had only one iteration.)
- The right pane displays the test results details. The iteration summary table indicates which iterations passed and which failed. The status summary table indicates the number of checkpoints or reports that passed, failed, and raised warnings during the test.

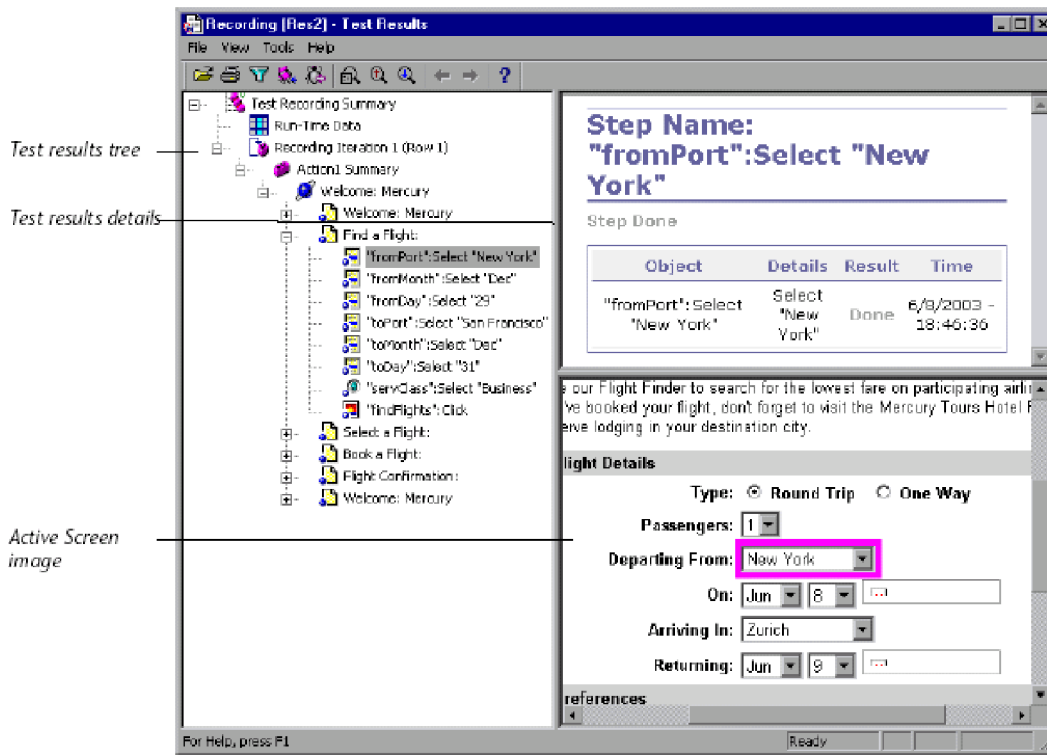
Your test run succeeded because QuickTest was able to navigate the Mercury Tours site just as the test was originally recorded. In this section, you will inspect the steps QuickTest performed when running your test, and how the application window appeared when a step was performed.



1. View the test results for a specific step.

In the test results tree, expand (+) Test Recording Summary > Recording Iteration 1 (Row 1) > Action1 Summary > Welcome Mercury > Find a Flight. Highlight "fromPort":Select "New York" in the test results

tree.



The Test Results window now contains three panes, displaying:

- the test results tree, with one step highlighted
- the test results details of the highlighted step
- the Active Screen, showing a screen capture of the Web page on which the step was performed.

When you click a page in the test results tree, QuickTest displays the corresponding page in the application view. When you click a step (an operation performed on an object) in the test results tree, the corresponding object is highlighted in the application view. In this case, the Departing From text box is highlighted.

2. Close the Test Results window.

Choose File > Exit.

Test Results Tree Symbols


✓ indicates a step that succeeded.


✗ indicates a step that failed.


! indicates a warning, meaning that the step did not succeed, but it did not cause the action or test to fail.

! ✗ indicates a step that failed unexpectedly, such as when an object is not found for a checkpoint.

! indicates an optional step that failed and therefore was ignored. Note that this does not cause the test to fail.

 indicates that the Smart Identification mechanism successfully found the object.

 indicates that a recovery scenario was activated.

 indicates that the test run was stopped before it ended.

3.0 How QuickTest identifies objects

3.1 Object Identification

3.1.1 Object Identification While Recording

Stores Object as Test Object, Determining the class it fits.

For each test object class, QuickTest always learns a list of mandatory properties. Checks whether this description is enough to uniquely identify the object. If it is not, QuickTest adds assistive properties, one by one, to the description, until it has a unique description.

3.1.2 Object Identification During Test Run

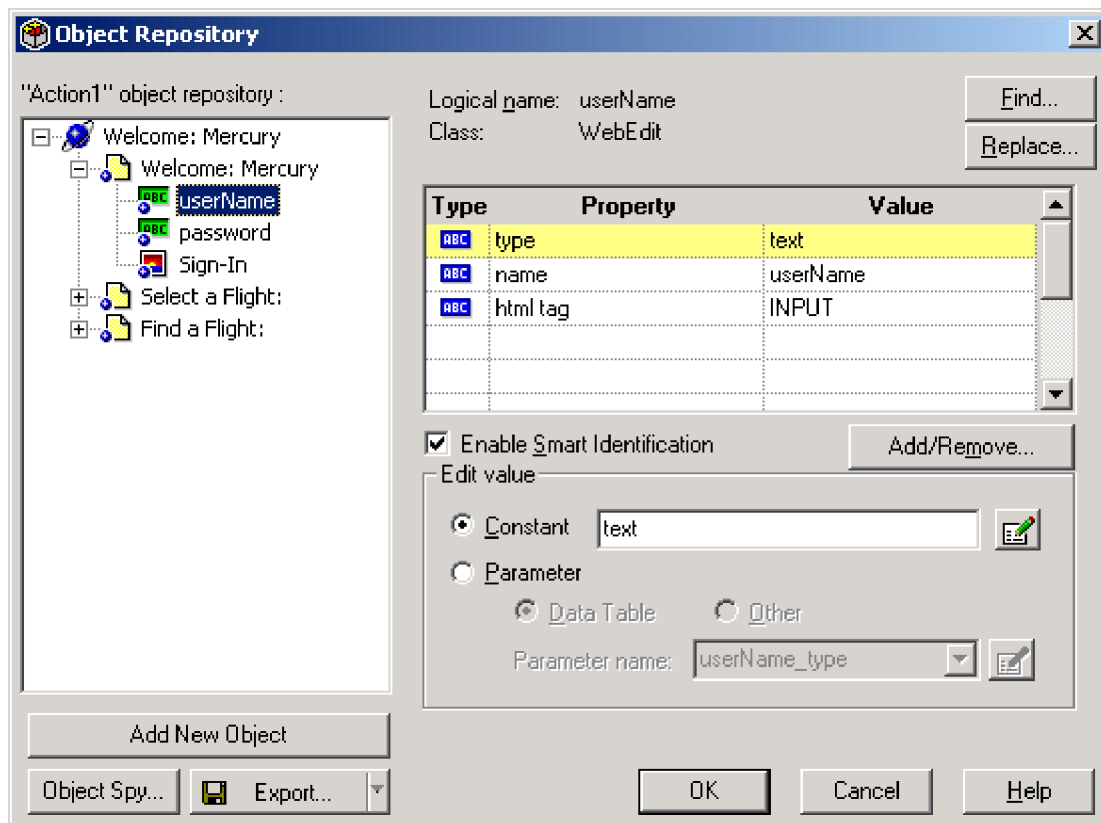
Searches for a run-time object that exactly matches the description of the test object

It expects to find a perfect match for both the mandatory and any assistive properties of test object

Uses Smart Identification mechanism to identify an object, even when the recorded description is no longer accurate.

3.2 Object Repository Introduction

The Object Repository dialog box displays a test tree of all objects in the current action or the entire test.



You can use the Object Repository dialog box to view or modify the properties of any test object in the repository or to add new objects to your repository.





3.2.1 Identifying the Object

The top part of the dialog box displays information about the object:

Information	Description
Logical name	The name that QuickTest assigns to the object.
Class	The class of the object.
Find	Opens the Find dialog box, where you can find a property or value that occurs several times in the same action.
Replace	Opens the Replace dialog box, where you can modify a property or value that occurs several times in the same action.

3.2.2 Viewing the Object's Properties

The default properties for the object are listed in the Properties pane of the dialog box. The pane includes the properties, their values, and their types:

Pane Element	Description
Type	<p>The  icon indicates that the value of the property is currently a constant.</p> <p>The  icon indicates that the value of the property is currently a Data Table parameter.</p> <p>The  icon indicates that the value of the property is currently an environment variable parameter.</p> <p>The  icon indicates that the value of the property is currently a random number parameter.</p>
Property	The name of the property.
Value	The value of the property.
Enable Smart Identification	<p>Indicates whether or not QuickTest uses Smart Identification to identify this object during the test run if it is not able to identify the object using the test object description. Note that this option is available only if Smart Identification properties are defined for the object's class in the Object Identification dialog box.</p> <p>Note: When you select Disable Smart Identification during the test run in the Run tab of the Test Settings dialog box, this option is disabled, although the setting is saved. When you clear the Disable Smart Identification during the test run check box, this option returns to its previous on or off setting after the test run.</p>

Add/Remove	Opens the Add/Remove Properties dialog box which lists the properties that can be used to identify the object.
------------	--

3.3 Use the Object Spy

Using the Object Spy, you can view the properties of any object in an open application. In addition to viewing object properties, the Object Spy also enables you to view both the run-time object methods and the test object methods associated with an object and to view the syntax for a selected method.

3.3.1 To view object properties:

Open your browser or application to the page containing the object on which you want to spy.

Choose Tools > Object Spy to open the Object Spy dialog box and display the Properties tab. Alternatively, click the Object Spy button from the Object Repository dialog box.

In the Object Spy dialog box, click the pointing hand. Both QuickTest and the Object Spy are minimized so that you can point to and click on any object in the open application.

Note: If the window on which you want to spy is partially hidden by another window, hold the pointing hand over the partially hidden window for a few seconds. The window comes into the foreground. You can now point and click on the object you want. You can configure the length of time required to bring a window into the foreground in the General tab of the Options dialog box.

If the object on which you want to spy can only be displayed by performing an event (such as a right-click or a mouse-over to display a context menu), hold the Ctrl key. The pointing hand temporarily turns into a standard arrow and you can perform the event. When the object on which you want to spy is displayed, release the Ctrl key. The arrow becomes a pointing hand again.

Click the object for which you want to view properties. The Object Spy returns to focus and displays the object hierarchy tree and the properties of the object that is selected within the tree.

To view the properties of the test object, click the Test Object Properties radio button. To view the properties of the run-time object, click the Run-Time Object Properties radio button.

Tip: You can use the Object property to retrieve the values of the run-time properties displayed in the Object Spy.

You can use the GetTOProperty and SetTOProperty methods to retrieve and set the value of test object properties for test objects in your test. You can use the GetROProperty to retrieve the current property value of the objects in your application during the test run.

If you want to view properties for another object within the displayed tree, click the object in the tree.

If you want to copy an object property or value to the clipboard, click the property or value. The value is displayed in the selected property/value box. Highlight the text in the selected property/value box

and use Ctrl + C to copy the text to the clipboard or right-click the highlighted text and choose Copy from the menu.

Note: If the value of a property contains more than one line, the Values cell of the object properties list indicates multi-line value. To view the value, click the Values cell. The selected property/value box displays the value with delimiters indicating the line breaks.

3.3.2 To view object methods:

Open your browser or application to the page containing the object on which you want to spy.

Choose Tools > Object Spy to open the Object Spy dialog box. Alternatively, click the Object Spy button from the Object Repository dialog box.

Click the Methods tab.

Click the pointing hand. Both QuickTest and the Object Spy are minimized so that you can point to any object on the open application.

Note: If the object you want is partially hidden by another window, hold the pointing hand over the partially hidden window for a few seconds. The window comes into the foreground. You can now point and click on the object you want. You can configure this option in the Options dialog box.

If the object on which you want to spy can only be displayed by performing an event (such as a right-click or a mouse-over to display a context menu), hold the Ctrl key. The pointing hand temporarily turns into a standard arrow and you can perform the event. When the object on which you want to spy is displayed, release the Ctrl key. The arrow becomes a pointing hand again.

Click the object for which you want to view the associated methods. The Object Spy returns to focus and displays the object hierarchy tree and the run-time object or test object methods associated with the object that is selected within the tree.

To view the methods of the test object, click the Test Object Methods radio button. To view the methods of the run-time object, click the Run-Time Object Methods radio button.

Tip: You can use the Object property to activate the run-time object methods displayed in the Object Spy.

If you want to view methods for another object within the displayed tree, click the object on the tree.

If you want to copy the syntax of a method to the clipboard, click the method in the list. The syntax is displayed in the selected method syntax box. Highlight the text in the selected method syntax box and use Ctrl + C to copy the text to the clipboard, or right-click the highlighted text and choose Copy from the menu.

4.0 Synchronization

4.1 Synchronizing Your Tests

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

- For a progress bar to reach 100%
- For a button to become enabled
- For a button to become enabled
- For a window or pop-up message to open

You can handle these anticipated timing problems by synchronizing your test to ensure that QuickTest waits until your application is ready before performing a certain step

4.2 Options to Synchronize Tests

There are several options that you can use to synchronize your test:

- You can insert a synchronization point, which instructs QuickTest to pause the test until an object property achieves the value you specify.
- You can insert Exist or Wait statements that instruct QuickTest to wait until an object exists or to wait a specified amount of time before continuing the test.
- You can also increase the default timeout settings in the Test Settings and Options dialog boxes in order to instruct QuickTest to allow more time for certain events to occur.

4.1.1 4.2.1 Inserting Synchronization Point

- Begin recording your test.
- Display the screen or page in your application that contains the object for which you want to insert a synchronization point.
- In QuickTest choose Insert > Step > Synchronization Point. The mouse pointer turns into a pointing hand.
- Click the object in your application for which you want to insert a synchronization point. If the location you click is associated with more than one object in your application, the Object Selection – Synchronization Point dialog box opens.

Select the object for which you want to insert a synchronization point, and click OK.

The Add Synchronization Point dialog box opens.

- The Property name list contains the test object properties associated with the object. Select the Property name you want to use for the synchronization point.
- Enter the property value for which QuickTest should wait before continuing to the next step in the test.

- Enter the synchronization point timeout (in milliseconds) after which QuickTest should continue to the next step, even if the specified property value was not achieved.
- Click OK. A WaitProperty step is added to your test.

For example, if you insert a synchronization point for the Update Order button, it may look something like this:

```
Window("Flights").WinButton("Update Order").WaitProperty "enabled", 1, 3000
```

4.1.2 4.2.2 Adding Exist and Wait Statements

You can use **Exist** and/or **Wait** statements in the Expert View to instruct QuickTest to wait for a window to open or an object to appear.

Exist statements return a Boolean value indicating whether or not an object currently exists.

Example following statement returns whether Flights Table is displayed:

```
y=Window("Flight Reservation").Dialog("Flights Table").Exist
```

Wait statements instruct QuickTest to wait a specified amount of time before proceeding to the next.

Example following statement waits for 10 seconds:

```
Wait (10)
```

4.1.3 Global synchronization Settings

Modifying Timeout Values

- To modify the maximum amount of time that QuickTest waits for an object to appear, change the Object Synchronization Timeout (Test > Settings > Run tab).
- To modify the amount of time that QuickTest waits for a Web page to load, change the Browser Navigation Timeout (Test > Settings > Web tab).

4.3 Transactions

The time taken by a section of the test to run can be measured by defining **Transaction**.

A transaction represents the business process that you are interested in measuring. You define transactions within your test by enclosing the appropriate sections of the test with **start** and **end** transaction statements.

4.1.4 4.3.1 Inserting Transactions

You define the beginning of a transaction in the Start Transaction dialog box.

To insert a transaction:

- In the test tree, click the step where you want the transaction timing to begin. The page is displayed in the Active Screen tab.
- Click the Start Transaction button or choose Insert > Start Transaction. The Start Transaction dialog box opens.



- Enter a meaningful name in the Name box.
- Decide where you want the transaction timing to begin:
To insert a transaction before the current step, select Before current step.
To insert a transaction after the current step, select After current step.
- Click OK. A tree item with the Start Transaction icon is added to the test tree.

4.1.5 4.3.2 Ending Transactions

You define the end of a transaction in the End Transaction dialog box.

To end a transaction:

- In the test tree, click the step where you want the transaction timing to end. The page opens in the Active Screen.
- Click the End Transaction button or choose Insert > End Transaction. The End Transaction dialog box opens.



- The Name box contains a list of the transaction names you defined in the current test. Select the name of the transaction you want to end.
- Decide where to insert the end of the transaction:
To insert a transaction before the current step, select Before current step.
To insert a transaction after the current step, select After current step.
- Click OK. A tree item with the End Transaction icon is added to the test tree.

5.0 Checkpoints

5.1 About Checkpoints

A checkpoint is a verification point that compares a current value for a specified property with the expected value for that property. This enables you to identify whether your Web site or application is functioning correctly.

5.2 Adding Checkpoints to a test

There are several ways to add checkpoints to your tests.

5.1.1 5.2.1 To add checkpoints while recording:

We can add checkpoints while recording the test. Use the commands on the **Insert** menu, or click the arrow beside the **Insert Checkpoint** button on the Test toolbar. This displays a menu of checkpoint options that are relevant to the selected step in the test tree.

5.1.2 To add a checkpoint while editing your test:

5.1.3 5.2.2.1 From Menu bar

Use the commands on the Insert menu, or click the arrow beside the Insert Checkpoint button on the Test toolbar. This displays a menu of checkpoint options that are relevant to the selected step in the test tree.

5.2.2.2 From Test Tree

Right-click the step in the test tree where you want to add the checkpoint and choose Insert Standard Checkpoint.

5.2.2.3 From the Active Screen

Right-click any object in the Active Screen and choose Insert Standard Checkpoint. This option can be used to create checkpoints for any object in the Active Screen (even if the object is not part of any step in your test tree).

5.3 Types of Checkpoints


A checkpoint is a verification point that compares a current value for a specified property with the expected value for that property. This enables you to identify whether your Web site or application is functioning correctly.

5.1.4 5.3.1 QuickTest Professional Checkpoint Types



Checkpoint Type	Description
Standard Checkpoint	Checks values of an object's properties
Image Checkpoint	Checks the property values of an image
Table Checkpoint	Checks information in a table
Page checkpoint	Checks the characteristics of a Web page

Text / Text Area Checkpoint	Checks that a text string is displayed in the appropriate place in a Web page or application window
Bitmap Checkpoint	Checks an area of a Web page or application after capturing it as a bitmap
Database Checkpoint	Checks the contents of databases accessed by an application or Web site
Accessibility Checkpoint	Identifies areas of a Web site to check for Section xxx compliance
XML Checkpoint	Checks the data content of XML documents

5.1.5 5.3.2 Creating a Standard Checkpoint

- Click the Insert Checkpoint toolbar button  or choose Insert > Checkpoint > Standard Checkpoint.
- Click the object you want to check. The Select an Object dialog box opens.
- Select the item you want to check from the displayed object tree.
- Click OK. The Checkpoint Properties dialog box opens.
- Specify the settings for the checkpoint.
- Click OK to close the dialog box.

5.1.6 5.3.3 Creating a Text Checkpoint

- Display the page, window, or screen containing the text you want to check.
- Choose Insert > Checkpoint > Text Checkpoint,  or click the arrow next to the Insert Checkpoint button  and Choose Text Checkpoint.
- The QuickTest window is minimized and the mouse pointer turns into a pointing hand.
- Click the text string for which you want to create the checkpoint. If the area you defined is associated with more than one object, the Object Selection–Text Checkpoint Properties dialog box opens. Select the required object
- The Text Checkpoint Properties dialog box opens.
- Specify the checkpoint settings.
- Click OK to close the dialog box.

5.4 Use regular expressions

Regular expressions enable QuickTest to identify objects and text strings with varying values.

You can use regular expressions when:

- Defining the property values of an object
- Parameterize a step
- Creating checkpoints with varying values

For example, if a window titlebar's name changes according to a file name, you can use a regular expression to identify a window whose titlebar has the specified product name, followed by a hyphen, and then any other text.

A regular expression is a string that specifies a complex search phrase. By using special characters such as a period (.), asterisk (*), caret (^), and brackets ([]), you can define the conditions of a search. When one of these special characters is preceded by a backslash (\), QuickTest searches for the literal character.

5.1.7 5.4.1 To define a constant property value as a regular expression:

- Open the Object Properties dialog box for the object either from test tree or from Active Screen or from Object Repository
- In the Property column, select the property you want to set as a regular expression.
- In the Edit value section, click Constant.
- Click the Edit Constant Value Options button. The Constant Value Options dialog box opens.
- Select the Regular Expression check box.
- In the Value box, enter the regular expression syntax for the string.
- Click OK to close the Constant Value Options dialog box.
- Click OK to save and close the Object Properties

5.1.8 5.4.2 To parameterize a property value using regular expressions:

Covered in Data Driving a Test

5.1.9 5.4.3 To define a regular expression in an object checkpoint:

- Display the page, window, or screen containing the text you want to check.
- Choose Insert > Checkpoint > Text Checkpoint, or click the arrow next to the Insert Checkpoint button and choose Text Checkpoint.
- Click the text string for which you want to create the checkpoint.
- Select the object for which you are creating the checkpoint. The Text Checkpoint Properties dialog box opens.
- In the Edit value section, click Constant.
- Click the Edit Constant Value Options button. The Constant Value Options dialog box opens.
- Select the Regular Expression check box.
- In the Value box, enter the regular expression syntax for the string.
- Click OK to close the Constant Value Options dialog box.
- Click OK to save and close Text Checkpoint Properties dialog box.

5.1.10 5.4.4 Common options to create regular expressions.

.	Matching Any Single Character
---	-------------------------------

[xy]	Matching Any Single Character in a List
[^xy]	Matching Any Single Character Not in a List
[x-y]	Matching Any Single Character within a Range
*	Matching Zero or More Specific Characters
+	Matching One or More Specific Characters
?	Matching Zero or One Specific Character
()	Grouping Regular Expressions
	Matching One of Several Regular Expressions
^	Matching the Beginning of a Line
\$	Matching the End of a Line
\w	Matching Any AlphaNumeric Character Including the Underscore
\W	Matching Any Non-AlphaNumeric Character

You can combine regular expression operators in a single expression to achieve the exact search criteria you need.

6.0 Creating Tests with Multiple Actions

6.1 Benefits of Test Modularity

- Makes code reusable.
- Scripts are easy to maintain.
- Scripts are efficient.
- Saves development time.

6.2 Creating Tests with Multiple Actions

You can divide your test into multiple actions by creating new actions or by inserting existing actions. There are three kinds of actions:

- **Non-reusable action**—an action that can be used only in the test in which it was created, and only once.
- **Reusable action**—an action that can be called multiple times by the test in which it was created (the local test) as well as by other tests.
- **External action**—a reusable action created in another test. External actions are read-only in the calling test. They can be modified only in the test in which they were created.

6.1.1 6.2.1 **Creating New Actions**

You can add new actions to your test during a recording session or while designing your test.

You can add the action as a top-level action, or you can add the action as a sub-action (or nested action) of an existing action in your test.

To create a new action in your test:

If you want to insert the action within an existing action, click the step after which you want to insert the new action.

- Choose Insert > New Action or click the New Action button. The Insert New Action dialog box opens.
- Type a new action name or accept the default name.
- If you wish, add a description of the action. You can also add an action description at a later time in the Action Properties dialog box.
- Select Reusable Action if you want to make the action reusable. You can also set or modify this setting at a later time in the Action Properties dialog box.
- Decide where to insert the action and select At the end of the test or After the current step.
- Click OK.

A new action is added to your test and is displayed at the bottom of the test tree or after the current step. You can move your action to another location in your test by dragging it to the desired location.

6.1.2 6.2.2 Inserting Existing Actions

You can insert an existing action by inserting a copy of the action into your test, or by inserting a call to the original action.

6.2.2.1 Inserting Copies Actions

When you insert a copy of an action into a test, the action is copied in its entirety, including checkpoints, parameterization, and the corresponding action tab in the Data Table. The action is inserted into the test as an independent, non-reusable action

Once the action is copied into your test, you can add to, delete from, or modify the action just as you would with any other recorded action. Any changes you make to this action after you insert it affect only this action, and changes you make to the original action do not affect the inserted action. You can insert copies of both reusable and non-reusable actions.

Steps to insert a copy of an action:

- Choose Insert > Copy of Action, right-click the action and select Insert Copy of Action, or right-click any step and select Action > Insert Copy. The Insert Copy of Action dialog box opens.
- Type a meaningful name for the action in the New action name box and give action description
- Specify where to insert the action : At the end of the test or After the current step.
- Click OK. The action is inserted into the test as an independent, nonreusable action.

6.2.2.2 Inserting Call to Actions

You can insert a call (link) to a reusable action that resides in your current test (local action), or in any other test (external action).

When you insert a call to an external action, the action is inserted in read-only format. You can view the components of the action in the action tree, but you cannot modify them.

Steps to insert a call to an action:

- Choose Insert > Call to Action, right-click the action and select Insert Call to Action, or right-click any step and select Action > Insert Call. The Insert Call to Action dialog box opens.
- In the Select an action box, select the action you want to insert from the list.
- Specify where to insert the action : At the end of the test or After the current step.
- Click OK. The action is inserted into the test as a call to the original action

6.1.3 6.2.3 Nesting Actions

Sometimes you may want to run an action within an action. This is called *nesting*.

Nesting actions Help you maintain the modularity of your test. Enable you to run one action or another based on the results of a conditional statement.

6.1.4 6.2.4 **Splitting Actions**

You can split an existing action into two sibling actions or into parent-child nested actions.

You cannot split an action and the option is disabled

- When an external action is selected
- When the first line of the action is selected
- While recording a test
- While running a test
- When you are working with a read-only test

6.3 **Miscellaneous**

6.1.5 6.3.1 **Setting Action Properties**

The Action Properties dialog box enables you to modify an action name, add or modify an action description, and set an action as reusable.

6.1.6 6.3.2 **Sharing Action Information**

There are several ways to share or pass values from one action to other actions:

- Store values from one action in the global Data Table and use these values as Data Table parameters in other actions.
- Set a value from one action as a user-defined environment variable and then use the environment variable in other actions.
- Add values to a Dictionary object in one action and retrieve the values in other actions.

6.1.7 6.3.3 **Exiting an Action**

You can add a line in your script in the Expert View to exit an action before it runs in its entirety.

There are four types of exit action statements you can use:

- ExitAction - Exits the current action, regardless of its iteration attributes.
- ExitActionIteration - Exits the current iteration of the action.
- ExitRun - Exits the test, regardless of its iteration attributes.
- ExitGlobalIteration - Exits the current global iteration.

6.1.8 6.3.4 **Removing Actions from a Test**

We can remove Non-reusable actions, External Actions, Reusable Actions, or Calls to External or Reusable actions.

6.1.9 6.3.5 **Renaming Actions**

You can rename actions from the Tree View or from the Expert View.

6.1.10 6.3.6 **Renaming Actions**

If you want to include one or more statements in every new action in your test, you can create an action template.

Steps to create an action template:

- Create a text file containing the comments, function calls, and other statements that you want to include in your action template.
- Save the text file as *ActionTemplate.mst* in your *<QuickTest Installation Folder>\dat* folder.

7.0 Data Driving a Test

7.1 Parameterize tests

You can use the parameter feature in QuickTest to enhance your tests by parameterizing values in the test. A parameter is a variable that is assigned a value from various data sources or generators.

Two different groups of parameters

- Data Table parameters - parameter is taken from data table.
- Other parameter types - parameter values are taken from random number, external user-defined file or environment variables etc.

7.1.1 7.1.1 Parameterize test Manually

- You can parameterize a step recorded in your test or a checkpoint added to your test.
- You can parameterize steps and checkpoints manually by opening the appropriate dialog box.
- You can parameterize a step in your test tree while recording or editing your test.

When you parameterize a step, you can parameterize an object property, a method argument, or both.

7.1.2 7.1.2 DataTable Parameters

You can supply the list of possible values for a parameter by creating a Data Table parameter.

Data Table parameters enable you to create a data-driven test that runs several times using the data you supply. In each *iteration*, QuickTest substitutes the constant value with a different value from the Data Table.

7.1.3 7.1.3 Using Environment Variable Parameters

QuickTest can insert a value from the Environment variable list, which is a list of variables and corresponding values that can be accessed from your test. Throughout the test run, the value of an environment variable remains the same, regardless of the number of iterations,

There are three types of environment variables:

7.1.3.1 User-Defined Internal

User defined internal variables are the variables that you define within the test. They are saved with the test and accessible only within the test in which they were defined.

Steps to add or modify environment variable parameters:

- In the **Edit value** section of the Object Properties, Object Repository, Method Arguments, or Checkpoint Properties dialog box, click **Other** as the type of parameter you want to use.
- Click the **Edit Parameter Options** button next to the parameter type box. The Parameter Options dialog box opens.

- Select **Environment** in the **Parameter Types** box.
- Accept the default name or enter a new name to add a new user-defined internal environment parameter, or select an existing environment variable name from the **Name** box. If you select an existing internal parameter, you can modify the value.
- If you created a new parameter or selected an existing user-defined-internal parameter, enter the value for the parameter in the **Value** box.
- Select the **Regular Expression** check box if needed and Enter the regular expression
- Click **OK** to save your changes and close the dialog box.

7.1.3.2 User-Defined External

User defined external variables are the variable that you pre-defined in the active external environment variables file. You can create as many files as you want and select an appropriate file for each test. Note that external environment variable values are designated as read-only within the test.

To define these variables create an external environment variables file

To select the active external environment-variables file:

- Choose Test > Settings to open the Test Settings dialog box.
- Click the Environment tab.
- Select the Load variables and values from external file (reloaded each test run) check box.
- Use the browse button or enter the full path of the external environment- variables file you want to use with your test.

7.1.3.2 Build-in

Built-in variables, such as Test path and Operating system. They are accessible from all tests, and are designated as read-only.

7.2 Create data-driven tests

QuickTest Pro enables you to create and run tests, which are driven by data stored in table.

When you test your application, you may want to check how it performs the same operations with multiple sets of data. For example, suppose you want to check how your application responds to ten separate sets of data. You could record ten separate tests, each with its own set of data. Alternatively, you could create a data-driven test with a loop that runs ten times. In each of the ten iterations, the test is driven by a different set of data.

7.3 Local and Global Data Tables

Each action also has its own sheet in the Data Table so that you can insert data that applies only to that action. When there are parameters in the current action's sheet, you can set QuickTest to run one or more iterations on that action before continuing with the current global iteration of the test.

The Global sheet contains the data that replaces parameters in each iteration of the test. When you run your test, QuickTest inserts or outputs a value from or to the current row of the global data sheet during each global iteration.

7.1.4 7.3.1 Using the Data Driver to Parameterize Your Test

The Data Driver enables you to quickly parameterize several (or all) objects, methods, and/or checkpoints containing the same constant value within a given action. Similar to a 'Find and Replace All' operation versus a step-by-step 'Find and Replace' process, you can choose to replace all occurrences of a selected constant value with a parameter. QuickTest can also show you each occurrence of the constant so that you can decide whether or not to parameterize the value.

To parameterize a value using the Data Driver:

- Display the action you want to parameterize.
- Choose Tools > Data Driver. The Data Driver scans the test for constants (this may take a few moments) and then the Data Driver opens.
The Data Driver displays the Constants list for the action. For each constant value, it displays the number of times the constant value appears in the action.

By default, the list displays only the following constants:

The argument value of each Set method

The argument value of each Select method

The value of the second argument (Value) of each SetTOProperty method

- If you want to parameterize a value that is not currently displayed in the list (such as an object property value), click Add Value. The Add Value dialog box opens.

Enter a constant value in the dialog box and click OK. The constant is added to the list.

- Select the type of parameterization you want to perform:
Step-by-step parameterization—Enables you to view the current values of each step containing the selected value. For each step, you can choose whether or not to parameterize the value and if so, which parameterization options you want to use.

Parameterize all—Enables you to parameterize all occurrences of the selected value throughout the action. You set your parameterization preferences one time and the same options are applied to all occurrences of the value.

If you selected Step-by-step parameterization, click Next. The Parameterize the Selected Step screen opens.

If you selected Parameterize all, the Parameter details area is enabled in the Data Driver main screen. Select your parameterization preferences the same way that you would for an individual step.

- In the Step to parameterize area, the first step with an object property, method argument, or checkpoint value containing the selected value is displayed in the test tree on the left. The parameterization options for the step are displayed on the right.
By default, the value is parameterized as a Data Table variable. Accept the default parameterization settings or set the parameterization options you want to apply to this step.
 - § Click Next to parameterize the selected step and view the next step containing the selected value.
 - § Click Skip if you do not want to parameterize the selected step.
 - § Click Finish to apply the parameterization settings of the current step to all remaining steps containing the selected value.

- If you clicked Next in the previous step, and steps remain that contain the selected value, the Parameterize the Selected Step screen opens displaying the next relevant step. Repeat step previous for each relevant step.

If there are no remaining steps containing the selected value, the Finished screen opens.

- Click Finish. The Data Driver Wizard closes and the Data Driver main screen shows how many occurrences you selected to parameterize and how many remain as constants.

8.0 Working with Data Tables

8.1 Introduction

Purpose: Insert data table parameters into the test to run it several times on different sets of data.

Iteration: Run the test on one set of data

Design-Time Data Table: This is the sheet used to store the data for the test, which is displayed at the bottom of the screen while creating and editing the test. It has the same characteristics of Microsoft excel spreadsheet that means can insert formulas within the cells.

Run-Time Data Table: This is the sheet created by quick test to hold live version of data table when you run your test. It is displayed in the Test Results window.

8.2 Working with Global and Action Sheets

There are two types of sheets within the Data Table—**Global** and **Action**. You can access the different sheets by clicking the appropriate tabs below the Data Table.

- Global Sheet data is available to all actions in your test, which is mainly used to pass parameters from one action to another.
- Action Sheet data available to only one action in your test.

For example, suppose you are creating a test on the sample Mercury Tours Web site. You might create one action for logging in, another for booking flights, and a third for logging out. You may want to create a test in which the user logs onto the site once, and then books flights for five passengers. The data about the passengers is relevant only to the second action, so it should be stored in the action tab corresponding to that action.

8.3 Editing and Saving Data Table

By default, QuickTest automatically saves the test's Data Table as an .x/s file in the test folder. You can save the Data Table in another location and instruct the test to use this Data Table when running a test.

Attach External Data Table: Specify a name and location for the data table in the resources tab of the test setting dialog box.

Usage:

1. Run the same test with different sets of input values

For example, you can test the localization capabilities of your application by running your test with a different Data Table file for each language you want to test.

2. The same input information for different tests

For example, you can test a Web version and a standard Windows version of the same application using different tests, but the same Data Table.

Note: The column names in the external Data Table match the parameter names in the test and that the sheets in the external Data Table match the actions in the test.

Edit information in the table by typing directly into the table.

Import data into Data Table from Microsoft Excel 95, Excel 97, Excel 2000, tabbed text files (.txt), or ASCII format by using **File** à **Import from File** Option.

Note: Microsoft Excel 2002 is not supported.

8.4 Importing Data from a Database

You can import data from a database by selecting a query from Microsoft Query or by manually specifying an SQL statement.

To import data from a database:

1. Right-click on the Data Table sheet to which you want to import the data and select **Sheet > Import > from Database**. The Database Query Wizard opens.
2. Select your database selection preferences and click **Next**. You can choose from the following options:
 - **Create query using Microsoft Query**—Opens Microsoft Query, enabling you to create a new query. Once you finish defining your query, you exit back to QuickTest. This option is only available if you have Microsoft Query installed on your computer.
 - **Specify SQL statement manually**—Opens the **Specify SQL statement** screen in the wizard, which enables you to specify the connection string and an SQL statement.
 - **Maximum number of rows**—Select this check box and enter the maximum number of database rows to import. You can specify a maximum of 32,000 rows.
 - **Show me how to use Microsoft Query**—displays an instruction screen before opening Microsoft Query when you click **next**. (Enabled only when **Create query using Microsoft Query** is selected).
3. If you chose **Create query using Microsoft Query** in the previous step, Microsoft Query opens. Choose a data source and define a query.

Specify the connection string and the SQL statement and click **Finish**.

- **Connection string**—Enter the connection string or click **Create** to open the ODBC Select Data Source dialog box. You can select a *.dsn* file in the ODBC Select Data Source dialog box or create a new *.dsn* file to have it insert the connection string in the box for you.
 - **SQL statement**—Enter the SQL statement.
4. QuickTest takes several seconds to capture the database query and restore the QuickTest window. The resulting data from the database query is displayed in the Data Table.

Note: Contrary to importing an Excel file (**File > Import From File**), existing data in the Data Table is *not* replaced when you import data from a database. If the database you import contains a column with the same name as an existing column, the database column is added as a new column with the column name followed by a sequential number. For example, if your Data Table already contains a column called departures, a database column by the same name would be inserted into the Data Table as departures1.

8.5 Using Formulas in the Data Table

This enables to create contextually relevant data during the test run.

Also formulas can be used as part of a checkpoint to check that objects created on-the-fly (dynamically generated) or other variable objects in the Web page or application have the values expected for a given context.

Note: When ever you compare the values, must be of the same type, i.e. integers, strings, etc.

To do this, you can use the **TEXT** and **VALUE** functions to convert values from one type to another as follows:

- **TEXT(value)** returns the textual equivalent of a numeric value, so that, for example, `TEXT(8.2)="8.2"`.
- **VALUE(string)** returns the numeric value of a string, so that, for example, `VALUE("8.2")=8.2`.

8.6 Using Data Table Scripting Methods

QuickTest provides several Data Table methods that enable you to retrieve information about the run-time Data Table and to set the value of cells in the run-time Data Table.

From a programming perspective, the Data Table is made up of three types of objects—`DataTable`, `Sheet` (sheet), and `Parameter` (column). Each object has several methods and properties that you can use to retrieve or set values.

9.0 Output and Correlation


9.1 About Outputting Values

An *output value* is a step in which one or more values are captured at a specific point in your test or component and stored for the duration of the run session. The values can later be used as input at a different point in the run session.

You can output the property values of any object. You can also output values from text strings, table cells, databases, and XML documents.

When you create output value steps, you can determine where the values are stored during the run session and how they can be used. During the run session, QuickTest retrieves each value at the specified point and stores it in the specified location. When the value is needed later in the run session, QuickTest retrieves it from this location and uses it as required.

Output values are stored only for the duration of the run session. When the run session is repeated, the output values are reset.

Note: You can insert an output value by using commands on the Insert menu or by clicking the arrow beside the **Insert Checkpoint** button  on the Test toolbar. This displays a menu of options that are relevant to the selected step in the test tree.

7.1.5 9.1.1 Creating Page Output Values

You can create a page output value from a Web page property value. When you run the test, QuickTest retrieves the current value of the property and stores it in the run-time Data Table as an output value.

For example, the number of links on a Web page may vary based on the selections a user makes on a form on the previous page. You could make an output value to store the number of links on the page during each test run or iteration.

7.1.6 9.1.2 Creating Text Output Values

You can create a text output value from a text string. When you run the test, QuickTest retrieves the current value of the text string and enters it in the run-time Data Table as an output value.

QuickTest allows you to create text output values by adding one of the following to your test:

- **Text Output Value**—enables you to output the text displayed in a screen or Web page, according to specified criteria. It is supported for all environments.
- **Standard Output Value**—enables you to output an object's text property. This is the preferred way of outputting the text displayed in many Windows applications.
- **Text Area Output Value**—enables you to output the text string displayed within a defined area of a Windows-application screen, according to specified criteria. It is supported for Standard Windows, Visual Basic, and ActiveX environments.

Note: Text Area output values are only supported by the following operating systems: Windows NT, Windows 2000, and Windows XP.

7.1.7 9.1.3 Creating Standard Output Values

You can create a standard output value from an object property value. When you run the test, QuickTest retrieves the current value of the property and enters it in the run-time Data Table as an output value.

7.1.8 9.1.4 Creating Image Output Values

You can create an image output value from the property value of a Web image. When you run the test, QuickTest retrieves the current value of the property and enters it in the run-time Data Table as an output value.

7.1.9 9.1.5 Creating XML Output Values

You can create an XML output value from an element value or attribute of an XML file or Web page/frame. When you run the test, QuickTest retrieves the current value of the element or attribute and enters it in the run-time Data Table as an output value.

When you run your test, you can view summary results of the XML output value in the Test Results window. You can also view detailed results by opening the XML Output Value Results window.

7.1.10 9.1.6 Creating Table Output Values

You can create a table output value from the contents of a table cell. When you run the test, QuickTest retrieves the current value of a table cell and enters it in the run-time Data Table as an output value.

7.1.11 9.1.7 Creating Database Output Values

You can create a database output value from the contents of a database cell. When you run the test, QuickTest retrieves the current value of a database cell and enters it in the run-time Data Table as an output value.

9.2 Capture and Reuse Run Time data

7.1.12 9.2.1 Adding a Standard Output Value

You can create a standard output value from an object property value. When you run the test, QuickTest retrieves the current value of the property and enters it in the run-time Data Table as an output value.

To create a standard output value while recording:

- Choose Insert > Output Value > Standard Output Value .
The mouse pointer turns into a pointing hand.
- Click the object in your application.
If the location you clicked is associated with more than one object, the Select an Object dialog box opens.
- Select the object for which you want to specify an output value.
- Click OK.
The Output Value Properties dialog box opens.
- Specify the settings for the output value.
- Click OK to close the Output Value Properties dialog box.
- An output value step is added to your test tree.

7.1.13 9.2.2 Creating Image Output Values

You can create an image output value from the property value of a Web image. When you run the test, QuickTest retrieves the current value of the property and enters it in the run-time Data Table as an output value.

To create an image output value while recording:

- Choose Insert > Output Value > Standard Output Value .
The mouse pointer turns into a pointing hand.
- Click the image.
- If the location you clicked is associated with more than one object, the Select an Object dialog box opens.
- Select the Image item you want to specify for an output value.
- Click OK.
The Image Output Value Properties dialog box opens.
- Specify the settings for the output value.
- Click OK to close the Image Output Value Properties dialog box.
- An output value step is added to your test.

7.1.14 9.2.3 Creating Table Output Values

You can create a table output value from the contents of a table cell. When you run the test, QuickTest retrieves the current value of a table cell and enters it in the run-time Data Table as an output value.

To create a table output value while recording:

- Choose Insert > Output Value > Standard Output Value. The mouse pointer turns into a pointing hand.
- Click the table. If the location you clicked is associated with more than one object, the Select an Object dialog box opens.
Select a Table item and click OK.
- The Output Value Properties dialog box opens.
- Specify the settings for the output value.
- Click OK to close the Output Value Properties dialog box.
- An output value step is added to your test tree.

Example: Consider the following problem for flight reservation application:

Create a new reservation and verify the reservation details.

Hint: you need order number to verify the reservation details.

Steps to solve the problem:

1. Create two actions à Create New Order and Open Order.
2. Create an output value for the order number field in Create New Order Action.
3. Parameterize Order Number in Open Order Action.

10.0 Alternatives to Standard Recording

10.1 Analog Recording



7.1.15 10.1.1 Analog Recording

Enables you to record the exact mouse and keyboard operations you perform in relation to either the screen or the application window.

In this recording mode, QuickTest records and tracks every movement of the mouse as you drag the mouse around a screen or window.

This mode is useful for recording operations that cannot be recorded at the level of an object, for example, recording a signature produced by dragging the mouse.

7.1.16 10.1.2 Recording in Analog Mode

- Click the Record button  to begin recording.
- Click the Analog Recording button  or choose Test > Analog Recording. The Analog Recording Settings dialog box opens.
- Select from the following options:
 - Record relative to the screen—records mouse movement or keyboard input relative to the coordinates of your screen, regardless of which application(s) are open.
 - Record relative to the following window—QuickTest records any mouse movement or keyboard input relative to the coordinates of the specified window.
- If you choose to Record relative to the following window, click the pointing hand and click anywhere in the window on which you want to record in analog mode.
- Click Start Analog Record. Perform the operations you want to record in analog mode.
- When you are finished and want to return to normal recording mode, click the Analog Recording button or choose Test > Analog Recording to turn off the option.

Notes:

When you record in analog mode relative to the screen, the test run will fail if your screen resolution or the screen location on which you recorded your analog steps has changed from the time you recorded.

All of your keyboard input, mouse movements, and clicks are recorded and saved in an external file. When QuickTest runs the test, the external data file is called. It tracks every movement and click of the mouse to replicate exactly the operations you recorded.

If you chose to Record relative to the screen, QuickTest inserts the RunAnalog step under a Desktop parent item.

For example: Desktop.RunAnalog "Track9"

If you chose to Record relative to the following window, QuickTest inserts the RunAnalog step under a Window parent item. For example:

Window("Microsoft Internet").RunAnalog "Track8"

The track file called by the RunAnalog method contains all your analog data and is stored with the action.

Tip: To stop an analog step in the middle of a test run, click Ctrl + Esc, then click Stop in the test toolbar.



10.2 Low-Level Recording

Enables you to record on any object in your application whether or not QuickTest recognizes the specific object or the specific operation.

This mode records at the object level and records all run-time objects as Window or WinObject test objects.

Use low-level recording for recording tests in an environment or on an object not recognized by QuickTest or if the exact coordinates of the object are important for your test

7.1.17 10.2.1 Recording in Low-Level mode

- Click the Record button  to begin a recording session.
- Click the Low Level Recording button  or choose Test > Low Level Recording.
- When you are finished and want to return to normal recording mode, click the Low Level Recording button or choose Test > Low Level Recording to turn off the option.

Notes:

In low-level recording and your entire keyboard input and mouse clicks are recorded based on mouse coordinates. When test is run, the cursor retraces the recorded clicks.

Suppose you type the word mercury into a user name edit box and then click the Tab key while in normal recording mode. Your script are displayed as follows:

```
Browser("Mercury Tours").Page("Mercury Tours").WebEdit("username").Set "mercury"
```

If you perform the same action while in low-level recording mode, the script is recorded as follows

```
Window("Microsoft Internet").WinObject("Internet Explorer_Se").Click 29,297
```

```
Window("Microsoft Internet").WinObject("Internet Explorer_Se").Type "mercury" + micTab
```


10.3 Configuring Web Event Recording

The Web Event Recording Configuration dialog box offers three standard event-configuration levels. By default, Quick Test uses the Basic recording -configuration level.

If Quick Test does not record all the events you need, you may require a higher event-configuration level.

Level	Description
Basic (Default)	<ul style="list-style-type: none"> •Always records click events on standard Web objects such as images, buttons, and radio buttons. •Always records the submit event within forms. •Records click events on other objects with a handler or behavior connected. Records the mouse over event on images and image maps only if the event following the mouse over is performed on the same object and is dependent on the mouse over event.
Medium	Records click events on the <DIV>, , and <TD> HTML tag objects, in addition to the objects recorded in the basic level.
High	Records mouse over, mouse down, and double-click events on objects with handlers or behaviors attached, in addition to the objects recorded in the basic level.

7.1.18 10.3.1 To set Web Event Recording Configuration:

- Choose Tools > Web Event Recording Configuration. The Web Event Recording Configuration dialog box opens. Use the slider to select your preferred standard event-recording configuration.
- Click OK.

10.4 Define a Virtual Object

You can teach Quick Test to recognize any area of your application as an object by defining it as a virtual object.

Virtual objects enable you to record and run tests on objects that are not normally recognized by Quick Test.

Using the Virtual Object Wizard, you can map a virtual object to a standard object class, specify the boundaries and the parent of the virtual object, and assign it a logical name.

7.1.19 10.4.1 To define a virtual object:

- With Quick Test open (but not in record mode), open your Web site or application and display the object containing the area you want to define as a virtual object.
- In Quick Test, choose Tools > Virtual Objects > New Virtual Object. The Virtual Object Wizard opens. Click Next.
- Select a standard class to which you want to map your virtual object.

- Click Mark Object.
The Quick Test window and the Virtual Object Wizard are minimized. Use the crosshairs pointer to mark the area of the virtual object. You can use the arrow keys while holding down the left mouse button to make precise adjustments to the area you define with the crosshairs. Click Next.
- Click an object in the object tree to assign it as the parent of the virtual object.
- In the Identify object-using box, select how you want Quick Test to identify and map the virtual object.

If you want Quick Test to identify all occurrences of the virtual object, select parent only. Quick Test identifies the virtual object using its direct parent only, regardless of the entire parent hierarchy.

If you want Quick Test to identify the virtual object in one occurrence only, select entire parent hierarchy. Quick Test identifies the virtual object only if it has the exact parent hierarchy. Click Next.

- Specify a name and a collection for the virtual object. Choose from the list of collections or create a new one by entering a new name in the Collection name box.
- To add the virtual object to the Virtual Object Manager and close the wizard, select No and then click Finish.

11.0 Introduction to the Expert View

11.1 Object Model in the Expert View

The Expert View provides an alternative to the Tree View for testers who are familiar with VBScript. In the Expert View, you can view the recorded test in VBScript and enhance it with programming.

The Expert View displays the steps you executed while recording your test in VBScript. After you record your test, you can increase its power and flexibility by adding recordable and non-recordable VBScript statements. You can add statements that perform operations on objects or retrieve information from your site. For example, you can add a step that checks that an object exists, or you can retrieve the return value of a method.

The objects in QuickTest are divided by environment. QuickTest environments include standard Windows objects, Visual Basic objects, ActiveX objects, Web objects, Multimedia objects, as well as objects from other environments available as external add-ins.

Most objects have corresponding methods. For example, the **Back** method is associated with the **Browser** object.

In the following example, the user enters mercury in the User Name edit box while recording. The following line is recorded in the Expert View:

```
Browser ("Mercury_Tours"). Page ("Mercury_Tours"). WebEdit ("username").Set "mercury"
```

When running the test, the **Set** method inserts the mercury text into the WebEdit object.

In the following example, the user selects **Paris** from the **Departure City** list box while recording. The following line is recorded in the Expert View:

```
Browser("Mercury Tours").Page("Find Flights").WebList("depart").Select "Paris"
```

When the test runs, the **Select** method selects **Paris** in the WebList object.

11.2 Using QuickTest Professional's online books

QuickTest Professional documentation helps you take full advantage of the capabilities of this powerful, easy-to-use, testing tool.

- QuickTest Professional User's Guide—provides step-by-step instructions on how to use QuickTest to test your applications. It describes many useful features, options, and testing procedures with guidelines and helpful examples.
- QuickTest Professional Installation Guide—explains how to install QuickTest Professional.

- QuickTest Professional Tutorial—teaches you basic QuickTest skills and shows you how to start testing your applications.
- QuickTest Professional Object Model Reference—provides access to the QuickTest Professional VBScript methods, including a description of each object, a list of the methods associated with each object, description syntax, and an example of usage for each object and method.
- QuickTest Professional Shortcut Key Reference Card—provides a list of commands that you can execute using shortcut keys.
- QuickTest Professional Automation Object Model Reference — (available from the QuickTest Professional Start menu program folder and from the QuickTest Professional **Help** menu) provides syntax, descriptive information, and examples for the automation objects, methods, and properties. It also contains a detailed overview to help you get started writing QuickTest automation scripts.

12.0 Working in the Expert View

12.1 VBScript Language Overview

7.1.20 12.1.1 VBScript Data Types

VBScript has only one data type called a Variant. A Variant is a special kind of data type that can contain different kinds of information, depending on how it is used.

Subtype	Description
Empty	Variant is uninitialized. Value is 0 for numeric variables or a zero-length string ("") for string variables.
Null	Variant intentionally contains no valid data.
Boolean	Contains either True or False.
Byte	Contains integer in the range 0 to 255.
Integer	Contains integer in the range -32,768 to 32,767.
Currency	-922,337,203,685,477.5808 to 922,337,203,685,477.5807.
Long	Contains integer in the range -2,147,483,648 to 2,147,483,647.
Single	Contains a single-precision, floating-point number in the range -3.402823E38 to -1.401298E-45 for negative values; 1.401298E-45 to 3.402823E38 for positive values.
Double	Contains a double-precision, floating-point number in the range -1.79769313486232E308 to -4.94065645841247E-324 for negative values; 4.94065645841247E-324 to 1.79769313486232E308 for positive values.
Date (Time)	Contains a number that represents a date between January 1, 100 to December 31, 9999.
String	Contains a variable-length string that can be up to approximately 2 billion characters in length.
Object	Contains an object.
Error	Contains an error number.

7.1.21 12.1.2 VBScript Variables

12.1.2.1 Declaring Variables

You declare variables explicitly in your script using the Dim statement, the Public statement, and the Private statement.

Variables declared with Dim at the script level are available to all procedures within the script. At the procedure level, variables are available only within the procedure.

Public statement variables are available to all procedures in all scripts.

Private statement variables are available only to the script in which they are declared.

12.1.2.2 Variable Naming Restrictions

Variable names follow the standard rules for naming anything in VBScript. A variable name:

Must begin with an alphabetic character.

Cannot contain an embedded period.

Must not exceed 255 characters.

Must be unique in the scope in which it is declared.

12.1.2.3 Scope and Lifetime of Variables

A variable's scope is determined by where you declare it. When you declare a variable within a procedure, only code within that procedure can access or change the value of that variable. It has local scope and is a procedure-level variable. If you declare a variable outside a procedure, you make it recognizable to all the procedures in your script. This is a script-level variable, and it has script-level scope.

The lifetime of a variable depends on how long it exists. The lifetime of a script-level variable extends from the time it is declared until the time the script is finished running. At procedure level, a variable exists only as long as you are in the procedure. When the procedure exits, the variable is destroyed. Local variables are ideal as temporary storage space when a procedure is executing. You can have local variables of the same name in several different procedures because each is recognized only by the procedure in which it is declared.

7.1.22 12.1.3 VBScript Constants

You create user-defined constants in VBScript using the [Const](#) statement. Using the Const statement, you can create string or numeric constants with meaningful names and assign them literal values. For example:

```
Const MyString = "This is my string."
```

```
Const MyAge = 49
```

7.1.23 12.1.4 VBScript Operators

12.1.4.1 Arithmetic Operators

Description	Symbol
Exponentiation	^
Unary negation	-
Multiplication	*
Division	/
Integer division	\
Modulus arithmetic	Mod
Addition	+
Subtraction	-
String concatenation	&

12.1.4.2 Comparison Operators

Description	Symbol
Equality	=
Inequality	<>
Less than	<
Greater than	>
Less than or equal to	<=
Greater than or equal to	>=
Object equivalence	Is

12.1.4.3 Logical Operators

Description	Symbol
Logical negation	Not
Logical conjunction	And
Logical disjunction	Or
Logical exclusion	Xor
Logical equivalence	Eqv
Logical implication	Imp

7.1.24 12.1.5 Using Conditional Statements

The following conditional statements are available in VBScript:

12.1.5.1 If...Then...Else statement.

Example:

```
If value = 0 Then
    AlertLabel.ForeColor = vbRed
Else
    AlertLabel.Forecolor = vbBlack
End If
```

12.1.5.2 Select Case statement.

Example:

```
Select Case strMyText
    Case "MasterCard"
        DisplayMCLogo
        ValidateMCAccount
    Case "Visa"
        DisplayVisaLogo
        ValidateVisaAccount
    Case Else
```


DisplayUnknownImage

PromptAgain

End Select

7.1.25 12.1.6 Looping Through Code

The following looping statements are available in VBScript:

12.1.6.1 Do...Loop:

Loops while or until a condition is True.

Repeats a block of statements while a condition is True or until a condition becomes True.

Do [{While | Until} condition]

[statements]

[Exit Do]

[statements]

Loop

Or, you can use this syntax:

Do

[statements]

[Exit Do]

[statements]

Loop [{While | Until} condition]

12.1.6.2 While...Wend:

Loops while a condition is True.

Executes a series of statements as long as a given condition is True.

While [condition](#)

Version [[statements](#)]

Wend

12.1.6.3 For...Next:

Uses a counter to run statements a specified number of times.

Repeats a group of statements a specified number of times.

For counter = start To end [Step step]

[statements]

[Exit For]

[statements]

Next

For Each...Next: Repeats a group of statements for each item in a collection or each element of an array.

Repeats a group of statements for each element in an array or collection.

For Each element In group

[statements]

[Exit For]

[statements]

Next [element]

7.1.26 12.1.7 VBScript Procedures

12.1.7.1 Sub Procedures

A Sub procedure is a series of VBScript statements (enclosed by Sub and End Sub statements) that perform actions but don't return a value.

Example:

```
Sub ConvertTemp()
```

```
    temp = InputBox("Please enter the temperature in degrees F.", 1)
```

```
    MsgBox "The temperature is " & Celsius(temp) & " degrees C."
```

```
End Sub
```

12.1.7.2 Function Procedures

A Function procedure is a series of VBScript statements enclosed by the Function and End Function statements.

Example:

```
Function Celsius(fDegrees)
```

```
    Celsius = (fDegrees - 32) * 5 / 9
```

```
End Function
```

The easiest way to create a test is to begin by recording typical business processes that you perform on your application or Web site. Then, to increase your test's power and flexibility, you can add programming statements to the recorded framework. Programming statements can contain:

- *recordable* test object methods: operations that a user can perform on an application or Web site.
- *non-recordable* test object methods: operations that users cannot perform on an application or Web site. You use these methods to retrieve or set information, or to perform operations triggered by an event.
- run-time methods of the object being tested.
- various VBScript programming commands that affect the way the test runs, such as conditions and loops. These are often used to control the logical flow of a test.
- supplemental statements, such as comments, to make your test easier to read, and messages that appear in the test results, to alert you to a specified condition

Note: *Test object* methods are defined in QuickTest; *run-time* methods are defined within the object you are testing, and therefore are retrieved from them.

You can incorporate decision-making into your test and define messages for the test results by using the appropriate dialog boxes.

In addition, you can improve the readability of your test using **With** statements. You can instruct QuickTest to automatically generate **With** statements as you record. But even after your basic test is recorded, you can convert its statements, in the Expert View, to **With** statements—by selecting a menu command.

12.2 Working with the Data Table Object

7.1.27 12.2.1 AddSheet Method

Adds the specified sheet to the run-time Data Table

Syntax: **DataTable.AddSheet**(*SheetName*)

Example:

```
Variable=DataTable.AddSheet ("MySheet").AddParameter("Time", "8:00")
```

7.1.28 12.2.2 DeleteSheet Method

Deletes the specified sheet from the run-time Data Table.

Syntax: **DataTable.DeleteSheet** *SheetID*

Example: `DataTable.DeleteSheet "MySheet"`

7.1.29 12.2.3 Export Method

Saves a copy of the run-time Data Table in the specified location.

Syntax: **DataTable.Export**(*FileName*)

Example: `DataTable.Export ("C:\flights.xls")`

7.1.30 12.2.4 ExportSheet Method

Exports a specified sheet of the run-time Data Table to the specified file.

Syntax: **DataTable.ExportSheet**(*FileName*, *DTSheet*)

Example: `DataTable.ExportSheet "C:\name.xls" ,1`

7.1.31 12.2.5 GetCurrentRow Method

Returns the current (active) row in the run-time global data sheet.

7.1.32 12.2.6 GetRowCount Method

Returns the total number of rows in the longest column in the global data sheet or in the specified data sheet of the run-time Data Table.

Example:

```
rowcount = DataTable.GetSheet("MySheet").GetRowCount
```

7.1.33 12.2.7 GetSheet Method

Returns the specified sheet from the run-time Data Table.

Example: `MyParam=DataTable.GetSheet ("MySheet").AddParameter("Time", "8:00")`

7.1.34 12.2.8 GetSheetCount Method

Returns the total number of sheets in the run-time Data Table.

7.1.35 12.2.9 Import Method

Imports the specified Microsoft Excel file to the run-time Data Table.

Syntax: **DataTable.Import**(*FileName*)

Example: DataTable.Import ("C:\flights.xls")

7.1.36 12.2.10 ImportSheet Method

Imports a sheet of a specified file to a specified sheet in the run-time Data Table.

Syntax: **DataTable.ImportSheet**(*FileName*, *SheetSource*, *SheetDest*)

Example: DataTable.ImportSheet "C:\name.xls" ,1 ,"name"

7.1.37 12.2.11 SetCurrentRow Method

Sets the specified row as the current (active) row in the run-time Data Table.

Example: DataTable.SetCurrentRow (2)

7.1.38 12.2.12 SetNextRow Method

Sets the row after the current (active) row as the new current row in the run-time Data Table.

7.1.39 12.2.13 SetPrevRow Method

Sets the row above the current (active) row as the new current (active) row in the run-time Data Table.

7.1.40 12.2.14 GlobalSheet Property

Returns the Global sheet of the run-time Data Table.

Example:

DataTable.GlobalSheet.AddParameter "Time", "5:45"

7.1.41 12.2.15 LocalSheet Property

Returns the current (active) local sheet of the run-time Data Table.

Example:

MyParam=DataTable.LocalSheet.AddParameter("Time", "5:45")

7.1.42 12.2.17 RawValue Property

Retrieves the *raw value* of the cell in the specified parameter and the current row of the run-time Data Table.

Syntax: **DataTable.RawValue** *ParameterID* [, *SheetID*]

SheetID can be the sheet name, index or dtLocalSheet, or dtGlobalSheet.

7.1.43 12.2.18 Value Property

Retrieves or sets the value of the cell in the specified parameter and the current row of the run-time Data Table.

Syntax: `DataTable.Value(ParameterID [, SheetID])`

12.3 Working with TextUtil Object.**7.1.44 12.3.1 GetText Method**

Returns the text from the specified window handle area.

Syntax

TextUtil.GetText(*hWnd* [, *Left*, *Top*, *Right*, *Bottom*])

7.1.45 12.3.2 GetTextLocation Method

Checks whether a specified text string is contained in a specified window area.

Syntax

TextUtil.GetTextLocation(*TextToFind*, *hWnd*, *Left*, *Top*, *Right*, *Bottom* [, *MatchWholeWordOnly*])

12.4 Working with Reporter Objects**7.1.46 12.4.1 ReportEvent Method**

Reports an event to the Test Report.

Syntax

Reporter.ReportEvent *EventStatus*, *ReportStepName*, *Details* [, *in*]

EventStatus – micPass, micFail, micDone, micWarning

7.1.47 12.4.2 Filter Property

Retrieves or sets the current mode for displaying events in the Test Results.

Syntax

To retrieve mode setting: *CurrentMode* = **Reporter.Filter**

To set the mode: **Reporter.Filter** = *NewMode*

Mode - 0 or rfEnableAll, 1 or rfEnableErrorsAndWarnings, 2 or rfEnableErrorsOnly, 3 or rfDisableAll

7.1.48 12.4.3 ReportPath Property

Retrieves the folder path in which the current test's Test Results are stored.

Syntax *Path* = Reporter.ReportPath

13.0 Object Recognition and Smart Identification

13.1 Object Repository Custom Configuration

You use the main screen of the Object Identification dialog box to set mandatory and assistive recording properties, to select the ordinal identifier, and to specify whether you want to enable the Smart Identification mechanism for each test object. From the Object Identification dialog box, you can also define user-defined object classes and map them to Standard Windows object classes, and you can configure the smart identification mechanism for any object.

To configure mandatory and assistive properties for a test object class:

- Choose **Tools > Object Identification**. The Object Identification dialog box opens.
- Select the appropriate environment in the **Environment** list. The test object classes associated with the selected environment are displayed in the **Test object classes** list.
- Select the properties you want to include in the Mandatory Properties list and/or clear the properties you want to remove from the list.
- Enter a valid property in the format `attribute/<PropertyName>` and click **OK**. The new property is added to the **Mandatory Properties** list. For example, to add a property called MyColor, enter `attribute/MyColor`.
- Click **OK** to close the Add/Remove Properties dialog box. The updated set of mandatory properties is displayed in the **Mandatory Properties** list.
- In the **Assistive Properties** list, click **Add/Remove**. The Add/Remove Properties dialog box for assistive properties opens.
- Select the properties you want to include in the assistive properties list and/or clear the properties you want to remove from the list.
- Enter a valid property in the format `attribute/<PropertyName>` and click **OK**. The new property is added to the **Mandatory Properties** list. For example, to add a property called MyColor, enter `attribute/MyColor`.
- Click **OK** to close the Add/Remove Properties dialog box. The properties are displayed in the Assistive Properties list.
- Use the up and down arrows to set your preferred order for the assistive properties. When you record a test and assistive properties are necessary to create a unique object description, QuickTest adds the assistive properties to the description one at a time until it has enough information to create a unique description, according to the order you set in the Assistive Properties list.

13.2 Introduction to Smart Identification

When QuickTest uses the recorded description to identify an object, it searches for an object that matches every one of the property values in the description. In most cases, this description is the simplest way to identify the object and unless the main properties of the object change, this method will work.

If QuickTest is unable to find any object that matches the recorded object description, or if it finds more than one object that fits the description, then QuickTest ignores the recorded description, and uses the Smart Identification mechanism to try to identify the object.

While the Smart Identification mechanism is more complex, it is more flexible, and thus, if configured logically, a Smart Identification definition can probably help QuickTest identify an object, if it is present, even when the recorded description fails.

The Smart Identification mechanism uses two types of properties:

13.2.1 13.2.1 Base filter properties

The most fundamental properties of a particular test object class; those whose values cannot be changed without changing the essence of the original object. For example, if a Web link's tag was changed from <A to any other value, you could no longer call it the same object.

13.2.2 13.2.2 Optional filter properties

Other properties that can help identify objects of a particular class as they are unlikely to change on a regular basis, but which can be ignored if they are no longer applicable.

13.3 Understanding the Smart Identification Process

If QuickTest activates the Smart Identification mechanism during a test run (because it was unable to identify an object based on its recorded description), it follows the following process to identify the object:

1. QuickTest “forgets” the recorded test object description and creates a new *object candidate* list containing the objects (within the object's parent object) that match all of the properties defined in the base filter property list.
2. From that list of objects, QuickTest filters out any object that does not match the first property listed in the Optional Filter Properties list. The remaining objects become the new object candidate list.
3. QuickTest evaluates the new object candidate list:
If the new object candidate list still has more than one object, QuickTest uses the new (smaller) object candidate list to repeat step 2 for the next optional filter property in the list. If the new object candidate list is empty, QuickTest ignores this optional filter property, returns to the previous object candidate list, and repeats step 2 for the next optional filter property in the list. If the object candidate list contains exactly one object, then QuickTest concludes that it has identified the object and performs the statement containing the object.
4. QuickTest continues the process described in steps 2 and 3 until it either identifies one object, or runs out of optional filter properties to use.
If, after completing the Smart Identification elimination process, QuickTest still cannot identify the object, then QuickTest uses the recorded description plus the ordinal identifier to identify the object

13.4 Smart Identification Configuration

1. Choose **Tools > Object Identification**. The Object Identification dialog box opens.
2. Select the appropriate environment in the **Environment** list. The test object classes associated with the selected environment are displayed in the **Test object classes** list.
3. Select the test object class you want to configure.
4. Click the **Configure** button next to the **Enable Smart Identification** check box. The **Configure** button is enabled only when the **Enable Smart Identification** option is selected. The Smart Identification Properties dialog box opens.

5. In the **Base Filter Properties** list, click **Add/Remove**. The Add/Remove Properties dialog box for base filter properties opens.
6. Select the properties you want to include in the **Base Filter Properties** list and/or clear the properties you want to remove from the list.
7. Click **OK** to close the Add/Remove Properties dialog box. The updated set of base filter properties is displayed in the **Base Filter Properties** list.
8. In the **Optional Filter Properties** list, click **Add/Remove**. The Add/Remove Properties dialog box for optional filter properties opens.
9. Select the properties you want to include in the **Optional Filter Properties** list and/or clear the properties you want to remove from the list.
10. Click **OK** to close the Add/Remove Properties dialog box. The properties are displayed in the **Optional Filter Properties** list.
11. Use the up and down arrows to set your preferred order for the optional filter properties. When QuickTest uses the Smart Identification mechanism, it checks the remaining object candidates against the optional properties one-by-one according to the order you set in the **Optional Properties** list until it filters the object candidates down to one object.

14.0 Enhance Test Cases with Descriptive Programming

14.1 Interact with Test Objects not stored in the Object Repository

You can also instruct QuickTest to perform methods on objects without referring to the Object Repository, without referring to the object's logical name. To do this, you provide QuickTest with a list of properties and values that QuickTest can use to identify the object or objects on which you want to perform a method.

Such a **programmatic description** can be very useful if you want to perform an operation on an object that is not stored in the object repository. You can also use programmatic descriptions in order to perform the same operation on several objects with certain identical properties, or in order to perform an operation on an object whose properties match a description that you determine dynamically during the test run.

There are two types of programmatic descriptions.

- You can either list the set of properties and values that describe the object directly in a test statement,
- or you can add a collection of properties and values to a description object, and then enter the description object name in the statement.

14.1.1 14.1.1 Entering Programmatic Description Directly into Test Statements

You can describe an object directly in a test statement by specifying *property:=value* pairs describing the object instead of specifying an object's logical name.

The general syntax is:

```
TestObject("PropertyName1:=PropertyValue1", "...", "PropertyNameX:=PropertyXValueX")
```

TestObject—the test object class.

PropertyName:=PropertyValue—the test object property and its value. Each property:=value pair should be separated by commas and quotation marks.

For Example: `Window("Text:=Myfile.txt - Notepad").Move 50, 50`

If you want to use the same programmatic description several times in one test, you may want to assign the object you create to a variable.

For Example:

```
Set MyWin = Window("Text:=Myfile.txt - Notepad")
MyWin.Move 50, 50
```

14.1.2 14.1.2 Using Description Objects for Programmatic Descriptions

You can use the **Description** object to return a **Properties** collection object containing a set of **Property** objects. A **Property** object consists of a property name and value. You can then specify the returned **Properties** object in place of a logical name in a test statement.

To create the **Properties** collection, you enter a **Description.Create** statement using the following syntax:

```
Set MyDescription = Description.Create()
```

Once you have created a **Properties** object (such as *MyDescription* in the example above), you can enter statements to add, edit, remove, and retrieve properties and values to or from the **Properties** object during the test run. This enables you to determine which, and how many properties to include in the object description in a dynamic way during the test run.

Once you have filled the **Properties** collection with a set of Property objects (properties and values), you can specify the **Properties** object in place of a logical name in a test statement.

For example, instead of entering:

```
Window("Error").WinButton("text:=OK", "width:=50").Click
```

You can enter:

```
Set MyDescription = Description.Create()
MyDescription("text").Value = "OK"
MyDescription("width").Value = 50
Window("Error").WinButton(MyDescription).Click
```

14.1.3 14.1.3 Retrieving ChildObjects

You can use the **ChildObjects** method to retrieve all objects located inside a specified parent object, or only those child objects that fit a certain programmatic description. In order to retrieve this subset of child objects, you first create a description object and add the set of properties and values that you want your child object collection to match using the **Description** object.

Note: You must use the **Description** object to create the programmatic description for the **ChildObjects** description argument. You cannot enter the programmatic description directly into the argument using the *property:=value* syntax.

Once you have "built" a description in your description object, use the following syntax to retrieve child objects that match the description:

```
Set MySubSet=TestObject.ChildObjects(MyDescription)
```

For example, the statements below instruct QuickTest to select all of the check boxes on the Itinerary Web page:

```
Set MyDescription = Description.Create()
MyDescription("html tag").Value = "INPUT"
MyDescription("type").Value = "checkbox"
Set Checkboxes = Browser("Itinerary").Page("Itinerary").ChildObjects(MyDescription)
NoOfChildObjs = Checkboxes.Count
For Counter=0 to NoOfChildObjs-1
    Checkboxes(Counter).Set "ON"
Next
```

14.1.4 14.1.4 Using Programmatic Descriptions for the WebElement Object

The **WebElement** object enables you to perform methods on Web objects that may not fit into any other Mercury test object class. The **WebElement** test object is never recorded, but you can use a programmatic description with the **WebElement** object to perform methods on any Web object in your Web site.

For example, when you run the statement below:

```
Browser("Mercury Tours").Page("Mercury Tours").WebElement("Name:=UserName",
"Index:=0").Click
```

or

```
set WebObjDesc = Description.Create()
```

```
WebObjDesc("Name").Value = "UserName"
```

```
WebObjDesc("Index").Value = "0"
```

```
Browser("Mercury Tours").Page("Mercury Tours").WebElement(WebObjDesc).Click
```

QuickTest clicks on the first Web object in the Mercury Tours page with the name UserName.

14.1.5 14.1.5 Using the Index Property in Programmatic Descriptions

The *index* property can sometimes be a useful test object property for uniquely identifying an object. The *index* test object property identifies an object based on the order in which it appears within the source code, where the first occurrence is 0.

Note that *index* property values are object-specific. Thus, if you use an index value of 3 to describe a **WebEdit** test object, QuickTest searches for the fourth **WebEdit** object in the page.

If you use an index value of 3 to describe a **WebElement** object, however, QuickTest searches for the fourth Web object on the page regardless of the type, because the **WebElement** object applies to all Web objects.

For example, suppose you have a page with the following objects:

an image with the name "Apple"

an image with the name "UserName"

a WebEdit object with the name "UserName"

an image with the name "Password"

a WebEdit object with the name "Password"

The description below refers to the third item in the list above, as it is the first WebEdit object on the page with the name UserName.

```
WebEdit("Name:=UserName", "Index:=0")
```

The following description, however, refers to the second item in the list above, as that is the first object of any type (WebElement) with the name UserName.

```
WebElement("Name:=UserName", "Index:=0")
```

14.2 Access Dynamic Objects during run-time

14.2.1 14.2.1 Retrieving Run-Time Object Properties

You can use the Object property to access the native properties of any run-time object. For example, you can retrieve the current value of the ActiveX calendar's internal Day property as follows:

```
Dim MyDay
```

```
Set MyDay=Browser("index").Page("Untitled").ActiveX("MSCAL.Calendar.7").Object.Day
```

14.2.2 14.2.2 Activating Run-Time Object Methods

You can use the **Object** property to activate the internal methods of any run-time object. For example, you can activate the edit box's native **focus** method as follows:

```
Dim MyWebEdit
```

```
Set MyWebEdit=Browser("Mercury Tours").Page("Mercury Tours").WebEdit("username").Object
```

```
MyWebEdit.focus
```

15.0 Enhance Test Cases with User-Defined Functions

15.1 Utilize external Windows API functions in Test Cases

15.1.1 Extern Object

Enables you to declare calls to external procedures from an external dynamic-link library(DLL)

Associated Methods

- **Declare Method** : Declares references to external procedures in a dynamic-link library (DLL).

Once you use the Declare method for a method, you can use the Extern object to call the declared method.

Syntax

Extern.Declare(*RetType*, *MethodName*, *LibName*, *Alias* [, *ArgType(s)*])

<i>RetType</i>	String	Data type of the value returned by the method.
<i>MethodName</i>	String	Any valid procedure name.
<i>LibName</i>	String	Name of the DLL or code resource that contains the declared procedure.
<i>Alias</i>	String	Name of the procedure in the DLL or code resource. Note: DLL entry points are case sensitive. Note: If <i>Alias</i> is an empty string, <i>MethodName</i> is used as the <i>Alias</i> .
<i>ArgType(s)</i>	String	A list of data types representing the data types of the arguments that are passed to the procedure when it is called. Note: For out arguments, use the micByRef flag.

Declare Data Types

The following data types can be used for the Extern.Declare *RetType* and *ArgType(s)* arguments.

Type	Description
micVoid	=0 ; void (RetType only)
micInteger	=2 ; int
micLong	=3 ; long
micFloat	=4 ; float
micDouble	=5 ; double
micString	=8 ; CHAR*
micDispatch	=9 ; IDispatch*

micWideString	=18 ; WChar*
micChar	=19 ; char
micUnknown	=20 ; IUnknown
micHwnd	=21 ; HWND
micVPtr	=22 ; void*
micShort	=23 ; short
micWord	=24 ; WORD
micDWord	=25; DWORD
micByte	=26 ; BYTE
micWParam	=27 ; WPARAM
micLParam	=28 ; LPARAM
micLResult	=29 ; LRESULT
micByRef	=0X4000 ; out
micUnsigned	=0X8000 ; unsigned
micUChar	=micChar micUnsigned ; unsigned char
micULong	=micLong micUnsigned ; ; unsigned long
micUShort	=micShort micUnsigned ; ; unsigned short
micUInteger	=micInteger micUnsigned ; ; unsigned int

Example

The following example uses the Extern.Declare and Extern.<declared method>methods to change the title of the Notepad window.

```
'Declare FindWindow method
```

```
Extern.Declare micHwnd, "FindWindow", "user32.dll", "FindWindowA", micString, micString
```

```
'Declare SetWindowText method
```

```
Extern.Declare micLong, "SetWindowText", "user32.dll", "SetWindowTextA", micHwnd, micString
```

```
'Get HWND of the Notepad window
```

```
hwnd = Extern.FindWindow("Notepad", vbNullString)
```

```
if hwnd = 0 then
```

```
    MsgBox "Notepad window not found"
```

```
end if
```

```
'Change the title of the notepad window
```

```
res = Extern.SetWindowText(hwnd, "Textpad")
```

15.2 Create QuickTest user-defined functions

In addition to the test objects, methods, and functions supported by the QuickTest Test Object Model, you can define your own VBScript functions subroutines, classes, modules, etc., and then call them from your test.

Vbscript function libraries can be written in any text editor. We have to save the file with .vbs extension. Syntax for writing vbscript function in vbscript file is as follows.

```
[Public [Default] | Private] Function name [(arglist)]
    [statements]
    [name = expression]
[Exit Function]
    [statements]
    [name = expression]
End Function
```

Arguments

Public Indicates that the Function procedure is accessible to all other procedures in all scripts.

Default Used only with the Public keyword in a Class block to indicate that the Function procedure is the default method for the class. An error occurs if more than one Default procedure is specified in a class.

Private Indicates that the Function procedure is accessible only to other procedures in the script where it is declared or if the function is a member of a class, and that the Function procedure is accessible only to other procedures in that class.

name Name of the Function; follows standard variable naming conventions.

arglist List of variables representing arguments that are passed to the Function procedure when it is called. Commas separate multiple variables.

statements Any group of statements to be executed within the body of the Function procedure.

expression Return value of the Function.

The *arglist* argument has the following syntax and parts:

```
[ByVal | ByRef] varname[( )]
```

Arguments

ByVal Indicates that the argument is passed by value.

ByRef Indicates that the argument is passed by reference.

varname Name of the variable representing the argument; follows standard variable naming conventions.

You can call these user defined functions from QuickTest as follows:

- Specify the default library files for all new tests in the Test Settings dialog box (**Test > Settings > Resources tab**).
- call a function contained in a VBScript file directly from any action using the **ExecuteFile** function.

To execute an externally-defined function:

- Create a VBScript file using standard VBScript syntax
- Store the file in any folder that you can access from the machine running your test.
- Add an **ExecuteFile** statement to an action in your test or in an associated library file using the following syntax:

ExecuteFile *FileName*

where *FileName* is the absolute or relative path of your VBScript file.

- Use the functions, subroutines, classes, etc., from the specified VBScript file as necessary in your action or within other functions in an associated library file.

16.0 Database Verification

16.1 Review of database concepts

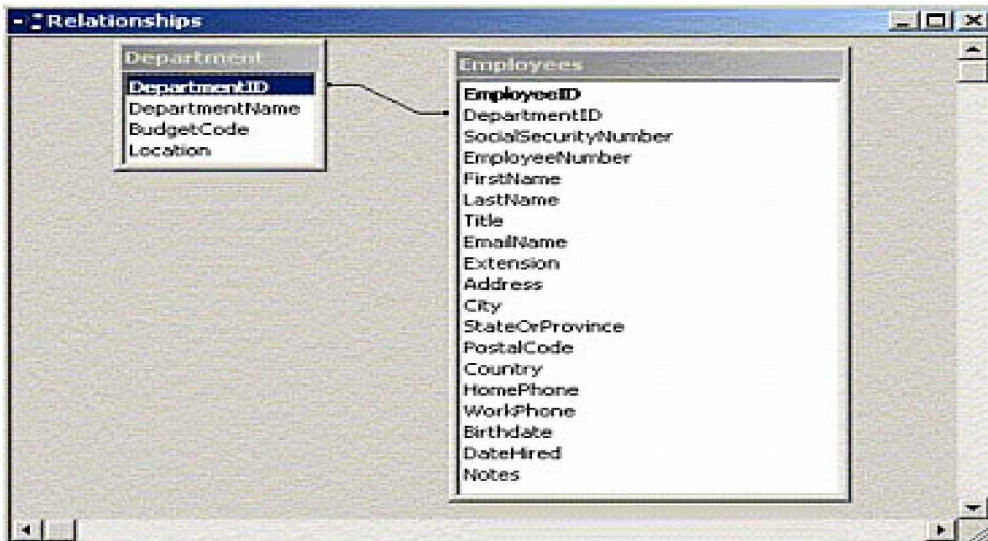
A *relational database* is a structured collection of information that is related to a particular subject or purpose, such as an inventory database or a human resources database. You use databases to manage information. Information, such as product name, cost, and on-hand inventory, is stored in a database.

Within the database, you organize the data into storage containers called tables. *Tables* are made up of columns and rows. Columns represent individual *fields* in a table. Rows represent *records* of data in a table.

Each field in the table contains one piece of information. In an employee table, for example, one column contains the employee name, another contains the employee phone number, and the address, city, state, zip, and salary are all stored in their own columns. Each record represents one set of related information. For example, an employee table might store information about one employee per row. The number of rows in a table represents the total number of table records.

16.1.1 Understanding relational tables

In a database, you can organize data in multiple tables. For example, if you manage a database for the Human Resource department, you might have one table that lists all the employees information and another table that lists all the departments:



Because you have multiple departments for employees, but you would not store the information about the departments in every employee row for several reasons:

- The department information is the same for each employee in a given department, however, repeating the department information for each employee is redundant. Storing redundant data takes up more disk space.
- If the department information changes, you can update one occurrence. All references to that department are updated automatically.

Storing multiple occurrences of the same data is rarely a good thing. Good relational database design separates application entities into their own tables. Key values from one table are often stored in a related table rather than repeating the information. The key value is used to join the data between the tables to return the complete set of data required.

16.1.2 About SQL

SQL (Structured Query Language) is a language that lets you communicate with databases. For example, you can use SQL to retrieve data from a database, add data to a database, delete or update records in a database, change columns in multiple rows, add columns to tables, and add and delete tables.

16.1.3 Using SQL to interact with a database

Unlike other computer languages, SQL is made up of a small number of language elements that let you interact efficiently with a database. Some of the more frequently used elements include these SQL commands:

Command	Description
SELECT	Use to retrieve (query) information in a database.
INSERT	Use to add records to a database.
UPDATE	Use to update information in a database.
DELETE	Use to delete information in a database.

16.1.4 Connection String

For connecting to database we must provide the information to connect to the database. There are two ways to provide this information.

The first way is through the use of a System DSN. A System DSN is a file containing information about a particular database. This information includes the physical location of the database on the computer, what type of database it is (SQL, Access, etc.), and other pertinent information. (*DSN stands for Data Source Name*)

the other way is to use what is called a **DSN-less connection**. This approach uglies up your connection string a bit, because you need to supply all of the important information in the connection string, since there isn't a DSN which holds that information. Access databases are the ones that typically use DSN-less connections.

Here is an example of a DSN-less connection string.

```
objConn.ConnectionString = "DBQ=C:\WebShare\MyDatabase.mdb;DRIVER={MS
Access (*.mdb)}"
```

In QuickTest proYou can use database checkpoints in your test to check databases accessed by your Web site or application and to detect defects. You define a query on your database, and then you create a database checkpoint that checks the results of the query.

There are two ways to define a database query:

Use Microsoft Query. You can install Microsoft Query from the *custom installation* of Microsoft Office.

Manually define an SQL statement.

In QuickTest, you create a database checkpoint based on the results of the query you defined on a database. QuickTest captures the current information about the database and saves this information as *expected data*. A database checkpoint is inserted into the test script. This checkpoint is displayed in your test script as a **DbTable.Check CheckPoint** statement .

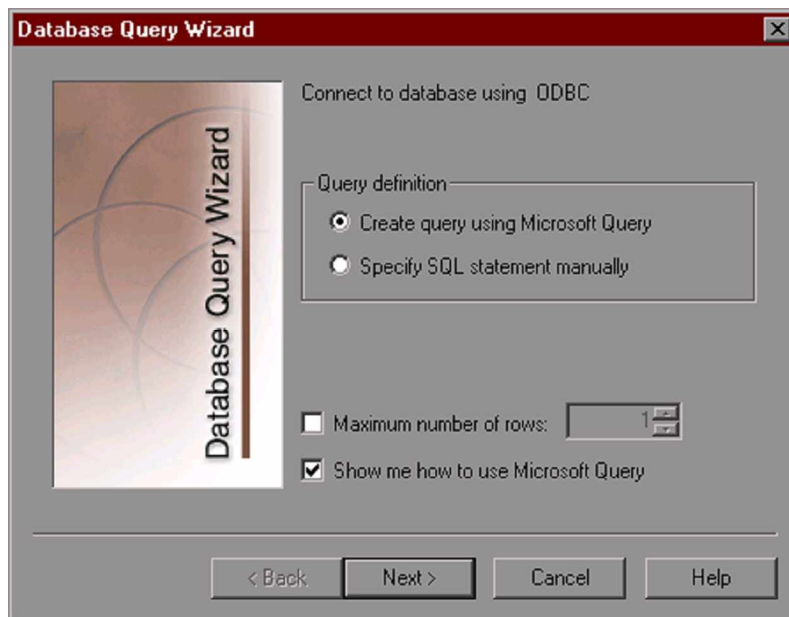
When you run the test, the database checkpoint compares the current state of the database to the expected data defined in the Checkpoint Properties dialog box. If the expected data and the current results do not match, the database checkpoint fails.

16.2 How to add a database checkpoint in QuickTest

You can define the query for your checkpoint using Microsoft Query or by manually entering a database connection and SQL statement.

To create a database checkpoint:

1 Choose Insert > Checkpoint > Database Checkpoint. The Database Query Wizard opens.



2 Select your database selection preferences and click **Next**. You can choose from the following options:

- Ø **Create query using Microsoft Query**—Opens Microsoft Query, enabling you to create a new query. Once you finish defining your query, you return to QuickTest. This option is available only if you have Microsoft Query installed on your computer.

- Ø **Specify SQL statement manually**—Opens the **Specify SQL statement** screen in the wizard, which enables you to specify the connection string and an SQL statement. For additional information, see step 3.
- Ø **Maximum number of rows**—Select this check box if you would like to limit the number of rows and enter the maximum number of database rows to check. You can specify a maximum of 32,000 rows.
- Ø **Show me how to use Microsoft Query**—Displays an instruction screen when you click **Next** before opening Microsoft Query. (Enabled only when **Create query using Microsoft Query** is selected).

3 If you chose **Create query using Microsoft Query** in the previous step, Microsoft Query opens. Choose a data source and define a query. If you chose **Specify SQL statement** in the previous step, the **Specify SQL statement** screen opens. Specify the connection string and the SQL statement, and click **Finish**.

4 The Checkpoint Properties dialog box opens. Select the checks to perform on the result set. You can also modify the expected data in the result set.

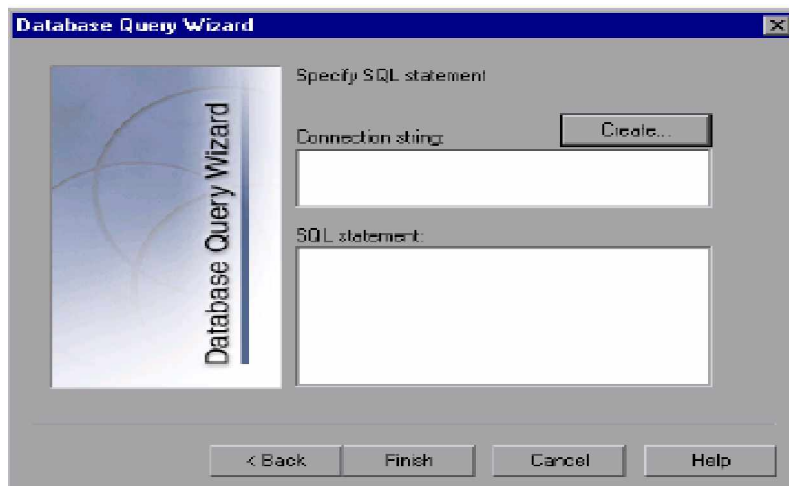
5 Click **OK** to close the dialog box. A checkpoint statement is added for the selected object in the Keyword View and Expert View.

1.3 Specifying SQL Statements

You can manually specify the database connection string and the SQL statement.

1.3.1 To specify SQL statements:

1 Choose **Specify SQL statement** in the Database Query Wizard screen. The following screen opens:



2 Specify the connection string and the SQL statement, and click **Finish**.

- Ø **Connection string**—Enter the connection string, or click **Create** to open the ODBC Select Data Source dialog box. You can select a **.dsn** file in the ODBC Select Data Source dialog box or create a new **.dsn** file to have the Database Query Wizard insert the connection string in the box for you.

- Ø **SQL statement**—Enter the SQL statement. QuickTest takes several seconds to capture the database query and restore the QuickTest window.

3 Return to step 4 in the previous procedure to continue creating your database checkpoint in QuickTest.

1.4 Creating a Query in Microsoft Query

You can use Microsoft Query to choose a data source and define a query within the data source.

1.4.1 To choose a data source and define a query in Microsoft Query:

1 When Microsoft Query opens during the insert database checkpoint process, choose a new or an existing data source.

2 Define a query.

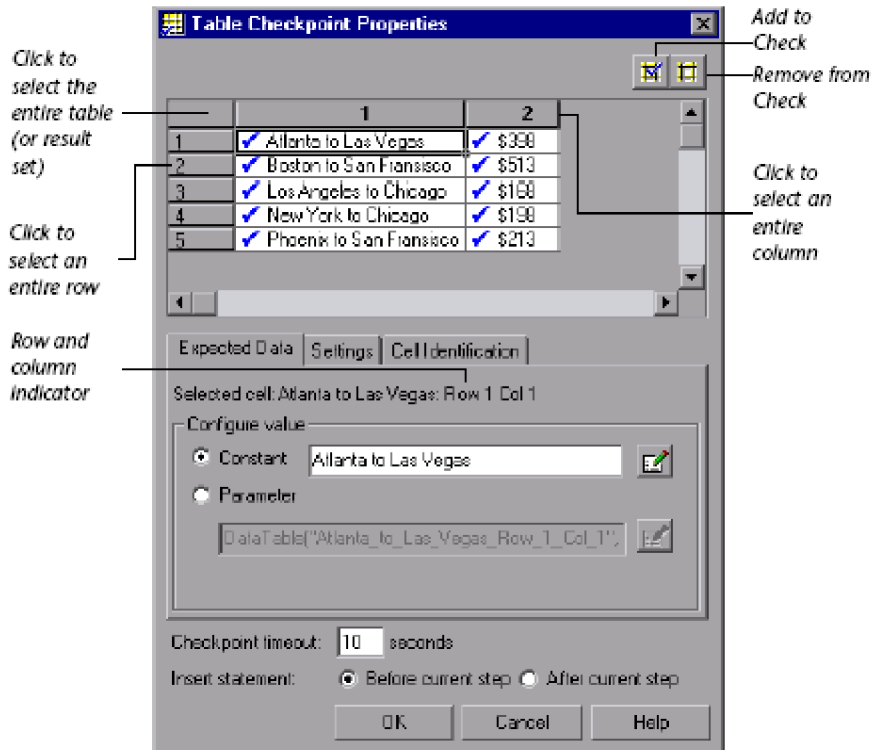
3 When you are done, in the Finish screen of the Query Wizard, select **Exit and return to QuickTest Professional** and click **Finish** to exit Microsoft Query. Alternatively, click **View data or edit query in Microsoft Query** and click **Finish**. After viewing or editing the data, choose

File > Exit and return to QuickTest Professional to close Microsoft Query and return to QuickTest.

4 Return to step 4 on above page to continue creating your database checkpoint in QuickTest.

1.5 Understanding the Table/Database Checkpoint Properties Dialog Box

The Table/Database Checkpoint Properties dialog box enables you to specify which cell contents of your table or database to check and which verification method and type to use. You can also edit or parameterize the expected data for the cells included in the check.



The top part of the Table/Database Checkpoint Properties dialog box displays the data that was captured for the checkpoint. You use this area to specify which cells you want to check.

Below the expected data, the Database Checkpoint dialog box contains the following three tabs:

- Ø **Expected Data**—Enables you to set each checked cell as a constant or parameterized value. For example, you can instruct QuickTest to use a value from the Data Table as the expected value for a particular cell.
- Ø **Settings**—Enables you to set the criteria for a successful match between the expected and actual values. For example, you can instruct QuickTest to treat the value as a number so that 45 or 45.00 are treated as the same value, or you can instruct QuickTest to ignore spaces when comparing the values.
- Ø **Cell Identification**—Enables you to instruct QuickTest how to locate the cells to be checked. For example, suppose you want to check the data that is displayed in the first row and second column in the Table/Database Checkpoint Properties dialog box. However, you know that each time you run your test or component, it is possible that the rows may be in a different order, depending on the sorting that was performed in a previous step. Therefore, rather than finding the data based on row and column numbers, you may want QuickTest to identify the cell based on the column name and the row containing a known value in a *key column*.

The bottom part of the Table/Database Checkpoint Properties dialog box contains the following options:

- Ø **Checkpoint timeout**—Specifies the time interval (in seconds) during which QuickTest attempts to perform the checkpoint successfully. QuickTest continues to perform the checkpoint until it passes or until the timeout occurs. If the checkpoint does not pass before the timeout occurs, the checkpoint fails.

For example, suppose it takes some time for data to load in a table. Increasing the checkpoint timeout value in this case can ensure that the data has sufficient time to load and therefore enables the checkpoint to pass before the end of the timeout period is reached.

You can see information about the checkpoint timeout, including the time interval used by QuickTest to perform the checkpoint, in the Test Results window.

Note: The **Checkpoint timeout** option is available only when creating a table checkpoint. It is not available when creating a database checkpoint.

- Ø **Insert statement**—Specifies when to perform the checkpoint in the test or component. Choose **Before current step** if you want to check the table or database content before the highlighted step is performed. Choose **After current step** if you want to check the table or database content after the highlighted step is performed.

Note: The **Insert statement** option is not available when adding a checkpoint during recording or when modifying an existing checkpoint. It is available when adding a new checkpoint to an existing test or component.

1.6 Modifying a Database Checkpoint

You can make the following changes to an existing database checkpoint:

- Ø Modify the SQL query definition.
- Ø Modify the expected data, verification type, or method.

1.6.1 To modify the SQL query definition:

1 In the Keyword View, right-click the database object that you want to modify.

2 Select **Object Properties**.

3 Modify the SQL and connection string properties as necessary and click **OK**.

1.6.2 To modify the expected data in a database checkpoint:

1 In the Keyword View or Expert View, right-click the database checkpoint that you want to modify.

2 Select Checkpoint Properties. The Checkpoint Properties dialog box opens. Modify the settings as described “Understanding the Table/Database Checkpoint Properties Dialog Box” above.

17.0 Recovery Manager and Scenarios

You can instruct QuickTest to recover from unexpected events and errors that occur in your testing environment during a run session.

This section describes:

- Ø About Defining and Using Recovery Scenarios
- Ø Deciding When to Use Recovery Scenarios
- Ø Defining Recovery Scenarios
- Ø Understanding the Recovery Scenario Wizard
- Ø Managing Recovery Scenarios
- Ø Setting the Recovery Scenarios List for Your Tests or Components
- Ø Programmatically Controlling the Recovery Mechanism

2.1 About Defining and Using Recovery Scenarios

Unexpected events, errors, and application crashes during a run session can disrupt your run session and distort results. This is a problem particularly when running tests or components unattended—the test or component is suspended until you perform the operation needed to recover.

The Recovery Scenario Manager provides a wizard that guides you through the process of defining a *recovery scenario*—a definition of an unexpected event and the operation(s) necessary to recover the run session. For example, you can instruct QuickTest to detect a Printer out of paper message and recover the run session by clicking the OK button to close the message and continue the test or component.

A recovery scenario consists of the following:

- Ø Trigger Event—The event that interrupts your run session. For example, a window that may pop up on screen, or a QuickTest run error.
- Ø Recovery Operation(s)—The operation(s) that need to be performed in order to continue running the test or component. For example, clicking an OK button in a pop-up window, or restarting Microsoft Windows.
- Ø Post-Recovery Test Run Option—The instructions on how QuickTest should proceed once the recovery operations have been performed, and from which point in the test or component QuickTest should continue, if at all. For example, you may want to restart a test or component from the beginning, or skip a step entirely and continue with the next step in the test or component.

Recovery scenarios are saved in recovery scenario files. A recovery scenario file is a logical collection of recovery scenarios, grouped according to your own specific requirements.

To instruct QuickTest to perform a recovery scenario during a run session, you must first associate it with that test or component. A test or component can have any number of recovery scenarios

associated with it. You can prioritize the scenarios associated with your test or component to ensure that trigger events are recognized and handled in the required order.

When you run a test or component for which you have defined recovery scenarios and an error occurs, QuickTest looks for the defined trigger event(s) that caused the error. If a trigger event has occurred, QuickTest performs the corresponding recovery and post-recovery operations.

You can also control and activate your recovery scenarios during the run session by inserting Recovery statements into your test or component.

Note: If you choose On error in the Activate recovery scenarios box in the Recovery tab of the Test Settings dialog box or Business Component Settings dialog box, the recovery mechanism does not handle triggers that occur in the last step of a test or component. If you chose this option and need to recover from an unexpected event or error that may occur in the last step of

a test or component, you can do this by adding an extra step to the end of your test or component.

2.2 Deciding When to Use Recovery Scenarios

If you can predict that a certain event may happen at a specific point in your test or component, it is highly recommended to handle that event directly within your test or component by adding steps such as If statements or optional steps, rather than depending on a recovery scenario. For

example, if you know that an Overwrite File message box may open when a Save button is clicked during a run session, you can handle this event with an If statement that clicks OK if the message box opens or by adding an optional step that clicks OK in the message box. Handling an event directly within your test enables you to handle errors more specifically than

recovery scenarios, which by nature are designed to handle a more generic set of unpredictable events. It also enables you to control the timing of the corrective operation with minimal resource usage and maximum performance. By default, recovery operations are activated only occur after a step returns an error, which can potentially occur several steps after the one that actually caused the error. The alternative, checking for trigger events after every step, may slow performance.

You should use recovery scenarios only for unpredictable events, or events that you cannot synchronize with a specific step in your test or component. For example, a recovery scenario can handle a printer error by clicking the default button in the Printer Error message box. You cannot handle this error directly in your test or component, since you cannot know at what point the network will return the printer error. You could try to handle this event in your test or component by adding an If statement immediately after the step that sent a file to the printer, but if the network takes time to return the printer error, your test or component may have progressed several steps before the error is displayed. Therefore, for this type of event, only a

recovery scenario can handle it.

2.3 Defining Recovery Scenarios

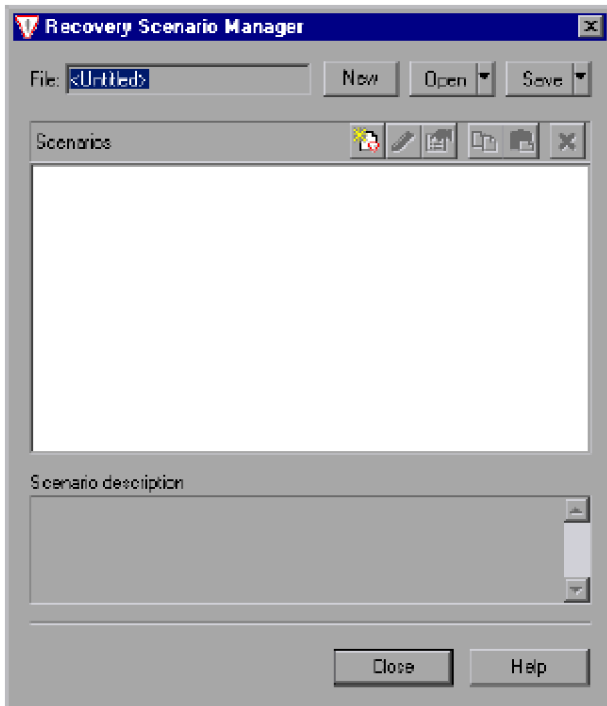
The Recovery Scenario Manager dialog box enables you to create recovery scenarios and save them in recovery files. You create recovery scenarios using the Recovery Scenario Wizard, which leads you through the process of defining each of the stages of the recovery scenario. You then save the recovery scenarios in a recovery file, and associate them with specific tests or components.

2.3.1 Creating a Recovery File

You save your recovery scenarios in a recovery file. A recovery file is a convenient way to organize and store multiple recovery scenarios together. You can create a new recovery file or edit an existing one.

To create a recovery file:

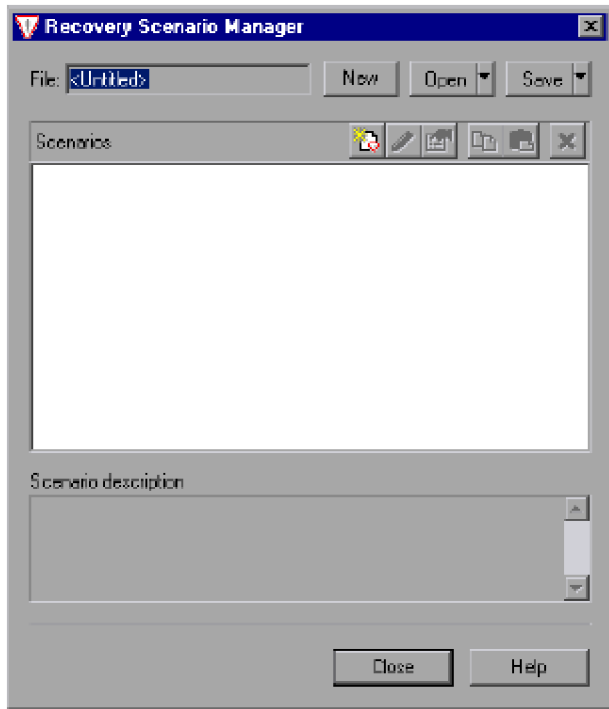
1 Choose Tools > Recovery Scenario Manager. The Recovery Scenario Manager dialog box opens.



2 By default, the Recovery Scenario Manager dialog box opens with a new recovery file. You can either use this new file, or click the Open button to choose an existing recovery file. Alternatively, you can click the arrow next to the Open button to select a recently-used recovery file from the list. You can now create recovery scenarios using the Recovery Scenario Wizard and save them in your recovery file, as described in the following sections.

2.3.2 Understanding the Recovery Scenario Manager Dialog Box

The Recovery Scenario Manager dialog box enables you to create and edit recovery files, and create and manage recovery scenarios. The Recovery Scenario Manager dialog box displays the name of the currently open recovery file, a list of the scenario(s) saved in the recovery file, and a description of each scenario.



2.4 Understanding the Recovery Scenario Wizard

The Recovery Scenario Wizard leads you, step-by-step, through the process of creating a recovery scenario. The Recovery Scenario Wizard contains five main steps:

- Ø defining the trigger event that interrupts the run session
- Ø specifying the recovery operation(s) required to continue
- Ø choosing a post-recovery test run operation
- Ø specifying a name and description for the recovery scenario
- Ø specifying whether to associate the recovery scenario to the current test and/or to all new tests

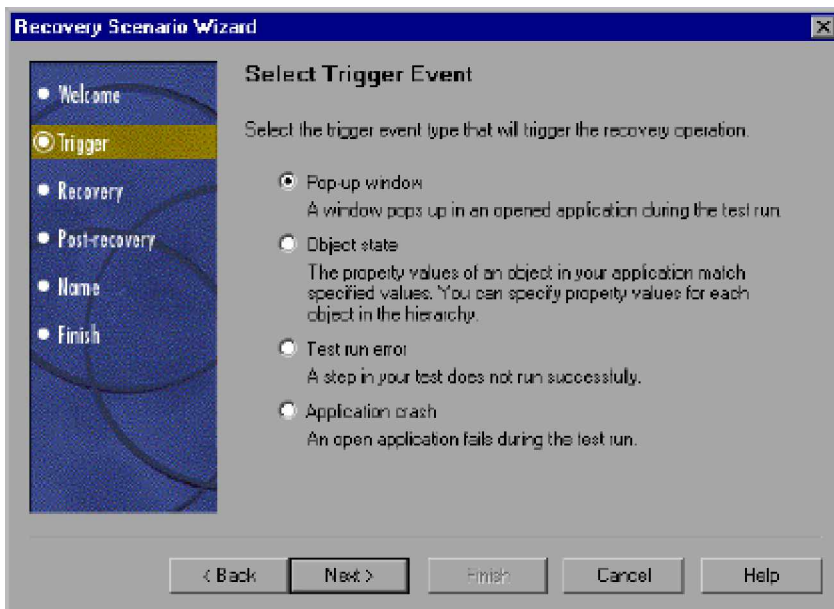
You open the Recovery Scenario Wizard by clicking the **New Scenario** button in the Recovery Scenario Manager dialog box. **Welcome to the Recovery Scenario Wizard Screen** The Welcome to the Recovery Scenario Wizard screen provides general information about the different options in the Recovery Scenario Wizard, and provides an overview of the stages involved in defining a recovery scenario.



Click **Next** to continue to the Select Trigger Event screen.

2.4.1 Select Trigger Event Screen

The Select Trigger Event screen enables you to define the event type that triggers the recovery scenario, and the way in which QuickTest recognizes the event.



Select a type of trigger and click **Next**. The next screen displayed in the wizard depends on which of the following trigger types you select:

- Ø **Pop-up window**—QuickTest detects a pop-up window and identifies it according to the window title and textual content. For example, a message box may open during a run session, indicating that the printer is out of paper. QuickTest can detect this window and activate a defined recovery scenario in order to continue the run session. Select this option and click **Next** to continue to the Specify Pop-up Window Conditions screen.
- Ø **Object state**—QuickTest detects a specific test object state and identifies it according to its property values and the property values of all its ancestors. **Part III • Creating Tests 418** Note that an object is identified only by its property values, and not by its class. For example, a

specific button in a dialog box may be disabled when a specific process is open. QuickTest can detect the object property state of the button that occurs when this problematic process is open and activate a defined recovery scenario to close the process and continue the run session. Select this option and click **Next** to continue to the Select Object screen.

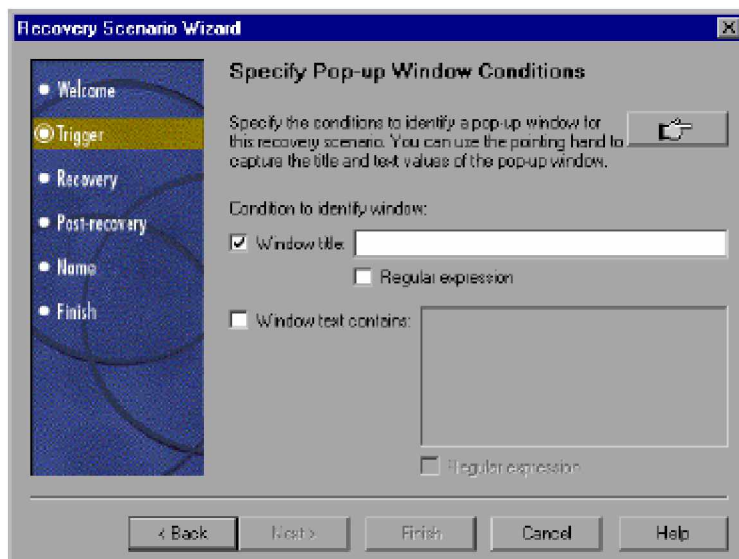
- Ø **Test run error**—QuickTest detects a run error and identifies it by a failed return value from a method. For example, QuickTest may not be able to identify a menu item specified in the method argument, due to the fact that the menu item is not available at a specific point during the run session. QuickTest can detect this run error and activate a defined recovery scenario in order to continue the run session. Select this option and click **Next** to continue to the Select Test Run Error screen.
- Ø **Application crash**—QuickTest detects an application crash and identifies it according to a predefined list of applications. For example, a secondary application may crash when a certain step is performed in the run session. You want to be sure that the run session does not fail because of this crash, which may indicate a different problem with your application. QuickTest can detect this application crash and activate a defined recovery scenario to continue the run session. Select this option and click **Next** to continue to the Recovery Operations screen.

Notes: The set of recovery operations is performed for each occurrence of the trigger event criteria. For example, suppose you define a specific object state, and two objects match this state, the set of replay operations is performed two times, once for each object that matches the specified state. The recovery mechanism does not handle triggers that occur in the last step

of a test or component. If you need to recover from an unexpected event or error that may occur in the last step of a test or component, you can do this by adding an extra step to the end of your test or component.

2.4.1.1 Specify Pop-up Window Conditions Screen

If you chose a **Pop-up window** trigger in the Select Trigger Event screen, the Specify Pop-up Window Conditions screen opens.



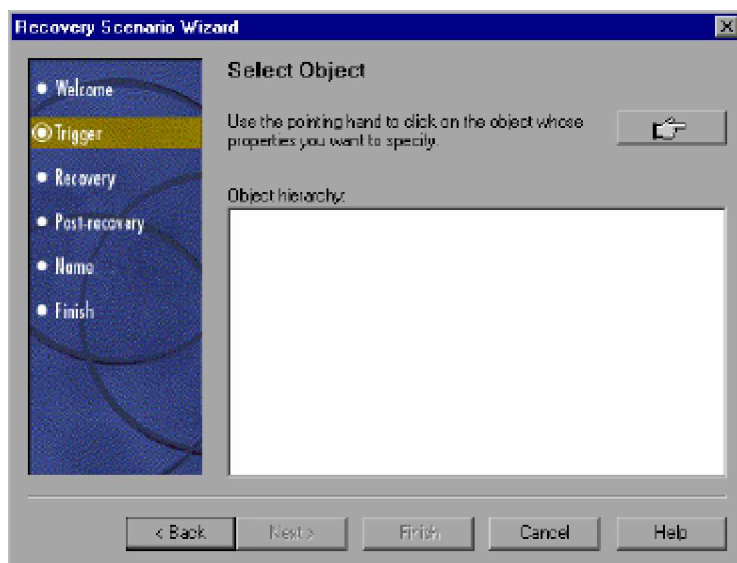
Click the pointing hand and then click the pop-up window to capture the window title and textual content of the window.

Tip: Hold the CTRL key to change the window focus or perform operations such as a right-click or mouseover to display a context menu. Note that pressing the CTRL key does not enable you to select an application from the Windows taskbar, therefore you must make sure that the window you want to access is not minimized. You can choose whether you want to identify the pop-up window according to its **Window title** and/or **Window text**. You can also use regular expressions

in the window title or textual content by selecting the relevant **Regular expression** check box and then entering the regular expression in the relevant location. Click **Next** to continue to the Recovery Operations screen.

2.4.1.2 Select Object Screen

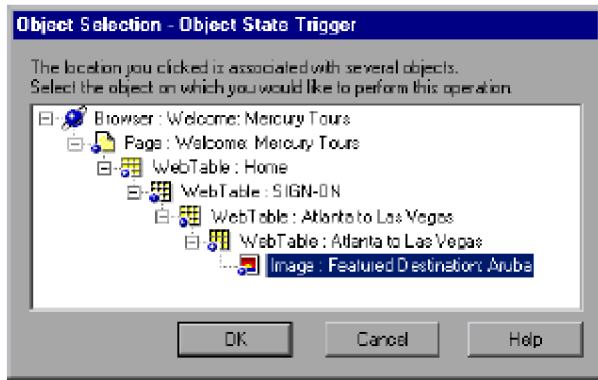
If you chose an **Object state** trigger in the Select Trigger Event screen, the Select Object screen opens.



Click the pointing hand and then click the object whose properties you want to specify.

Tip: Hold the CTRL key to change the window focus or perform operations such as a right-click or mouseover to display a context menu. Note that pressing the CTRL key does not enable you to select an application from the Windows taskbar, therefore you must make sure that the window you want to access is not minimized.

If the location you click is associated with more than one object, the Object Selection–Object State Trigger dialog box opens.

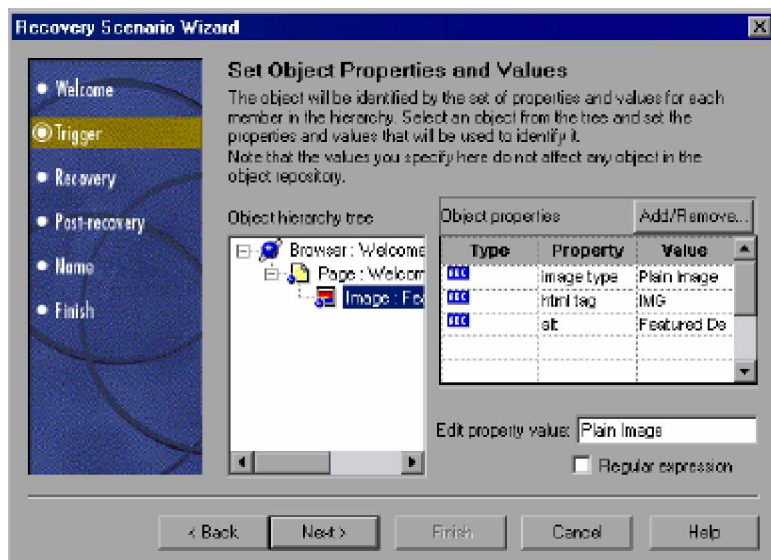


Select the object whose properties you want to specify and click **OK**. The selected object and its parents are displayed in the Select Object screen.

Note: The hierarchical object selection tree also enables you to select an object that QuickTest would not ordinarily record (a non-parent object), such as a web table. Click **Next** to continue to the Set Object Properties and Values screen.

2.4.1.3 Set Object Properties and Values Screen

After you select the object whose properties you want to specify in the Select Object screen, the Set Object Properties and Values screen opens.

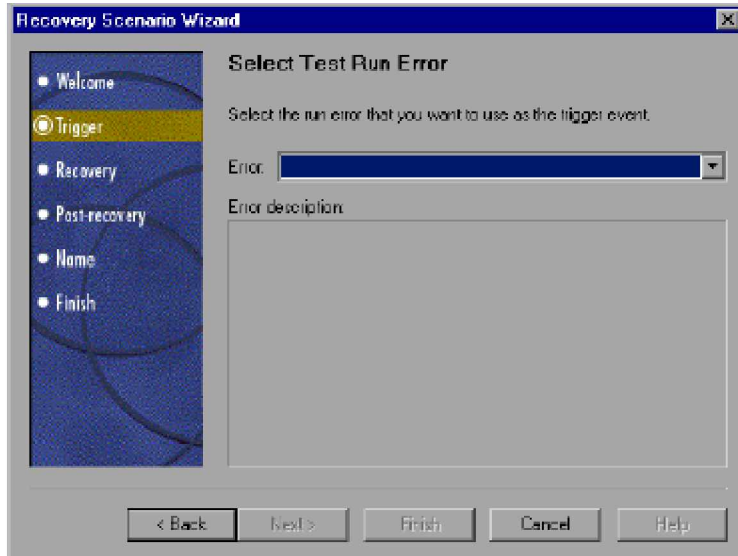


For each object in the hierarchy, in the **Edit property value** box, you can modify the property values used to identify the object. You can also click the **Add/Remove** button to add or remove object properties from the list of property values to check. Note that an object is identified only by its property values, and not by its class.

Select the **Regular expression** check box if you want to use regular expressions in the property value. Click **Next** to continue to the Recovery Operations screen.

2.4.1.4 Select Test Run Error Screen

If you chose a **Test run error** trigger in the Select Trigger Event screen, the Select Test Run Error screen opens.

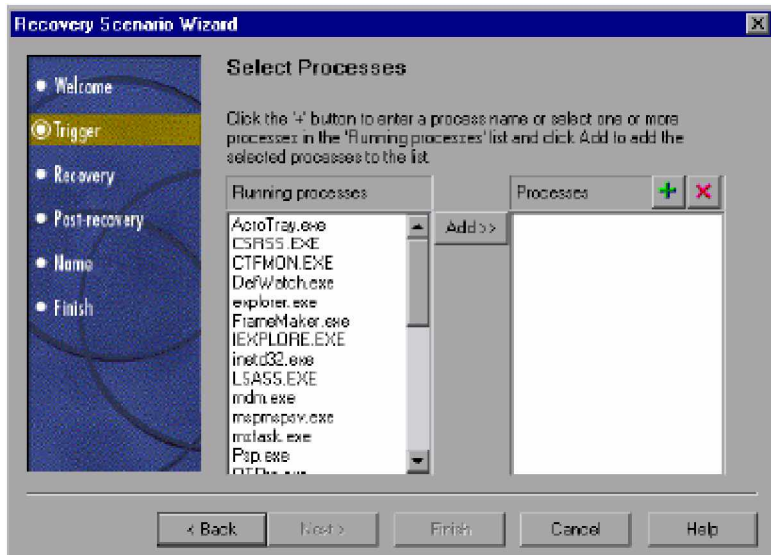


In the **Error** list, choose the run error that you want to use as the trigger event:

- Ø **Any error**—Any error code that is returned by a step.
- Ø **Item in list or menu is not unique**—Occurs when more than one item in the list, menu, or tree has the name specified in the method argument.
- Ø **Item in list or menu not found**—Occurs when QuickTest cannot identify the list, menu, or tree item specified in the method argument. This may be due to the fact that the item is not currently available or that its name has changed.
- Ø **More than one object responds to the physical description**—Occurs when more than one object in your application has the same property values as those specified in the test object description for the object specified in the step.
- Ø **Object is disabled**—Occurs when QuickTest cannot perform the step because the object specified in the step is currently disabled.
- Ø **Object not found**—Occurs when no object within the specified parent object matches the test object description for the object.
- Ø **Object not visible**—Occurs when QuickTest cannot perform the step because the object specified in the step is not currently visible on the screen. Click **Next** to continue to the Recovery Operations screen.

2.4.1.5 Select Processes Screen

If you chose an **Application crash** trigger in the Select Trigger Event screen, the Select Processes screen opens.



The **Running processes** list displays all application processes that are currently running. The **Processes** list displays the application processes that will trigger the recovery scenario if they crash.

You can add application processes to the **Processes** list by typing them in the **Processes** list or by selecting them from the **Running processes** list.

To add a process from the **Running processes** list, double-click a process in the **Running processes** list or select it and click the **Add** button. You can select multiple processes using standard Windows multiple selection techniques (CTRL and SHIFT keys).

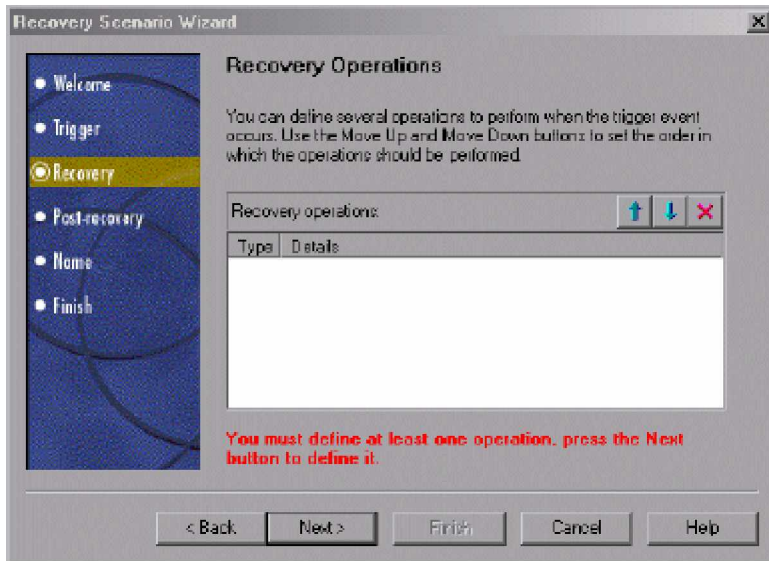
To add a process directly to the **Processes** list, click the **Add New Process** button to enter the name of any process you want to add to the list. To remove a process from the **Processes** list, select it and click the **Remove Process** button.

Tip: You can modify the name of a process by selecting it in the **Processes** list and clicking the process name to edit it.

Click **Next** to continue to the Recovery Operations screen.

2.4.2 Recovery Operations Screen

The Recovery Operations screen enables you to manage the collection of recovery operations in the recovery scenario. Recovery operations are operations that QuickTest performs sequentially when it recognizes the trigger event.



You must define at least one recovery operation. To define a recovery operation and add it to the **Recovery operations** list, click **Next** to continue to the Recovery Operation screen.

If you define two or more recovery operations, you can select a recovery operation and use the **Move Up** or **Move Down** buttons to change the order in which QuickTest performs the recovery operations. You can also select a recovery operation and click the **Remove** button to delete a recovery operation from the recovery scenario.

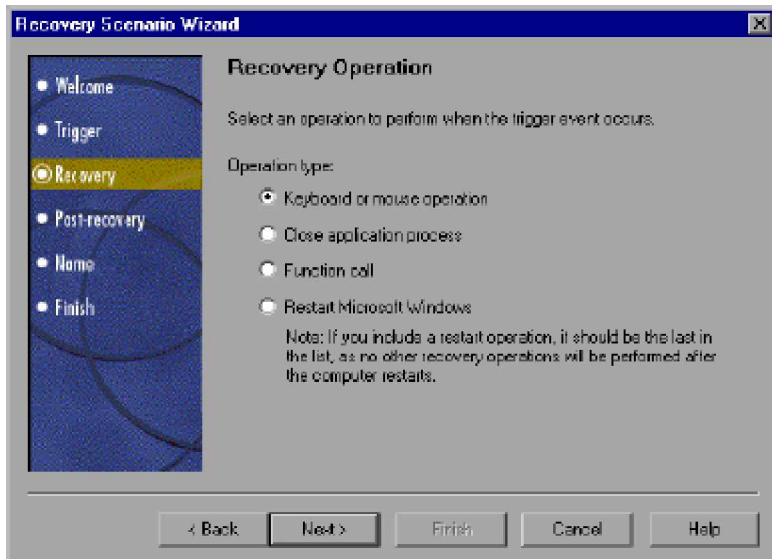
Note: If you define a **Restart Microsoft Windows** recovery operation, it is always inserted as the last recovery operation, and you cannot change its position in the list.

After you have defined at least one recovery operation, the **Add another recovery operation** check box is displayed.

- Ø Select the check box and click **Next** to define another recovery operation.
- Ø Clear the check box and click **Next** to continue to the Post-Recovery Test Run Options screen.

2.4.2.1 Recovery Operation Screen

The Recovery Operation screen enables you to specify the operation(s) QuickTest performs after it detects the trigger event.



Select a type of recovery operation and click **Next**. The next screen displayed in the wizard depends on which recovery operation type you select.

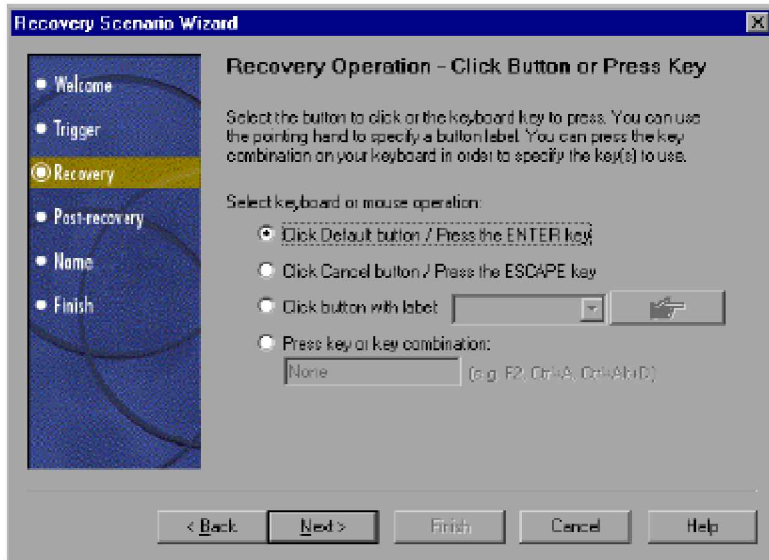
You can define the following types of recovery operations:

- Ø **Keyboard or mouse operation**—QuickTest simulates a click on a button in a window or a press of a keyboard key. Select this option and click **Next** to continue to the Recovery Operation – Click Button or Press Key screen.
- Ø **Close application process**—QuickTest closes specified processes. Select this option and click **Next** to continue to the Recovery Operation – Close Processes screen.
- Ø **Function call**—QuickTest calls a VBScript function. Select this option and click **Next** to continue to the Recovery Operation – Function screen.
- Ø **Restart Microsoft Windows**—QuickTest restarts Microsoft Windows. Select this option and click **Next** to continue to the Recovery Operations screen.

Note: If you use the Restart Microsoft Windows recovery operation, you must ensure that any test or component associated with this recovery scenario is saved before you run it. You must also configure the computer on which the test or component is run to auto login on restart.

2.4.2.2 Recovery Operation – Click Button or Press Key Screen

If you chose a **Keyboard or mouse operation** recovery operation in the Recovery Operation screen, the Recovery Operation – Click Button or Press Key screen opens.



Specify the keyboard or mouse operation that you want QuickTest to perform when it detects the trigger event:

- Ø **Click Default button / Press the ENTER key**—Instructs QuickTest to click the default button or press the ENTER key in the displayed window when the trigger occurs.
- Ø **Click Cancel button / Press the ESCAPE key**—Instructs QuickTest to click the **Cancel** button or press the ESCAPE key in the displayed window when the trigger occurs.
- Ø **Click button with label**—Instructs QuickTest to click the button with the specified label in the displayed window when the trigger occurs. If you select this option, click the pointing hand and then click anywhere in the trigger window.

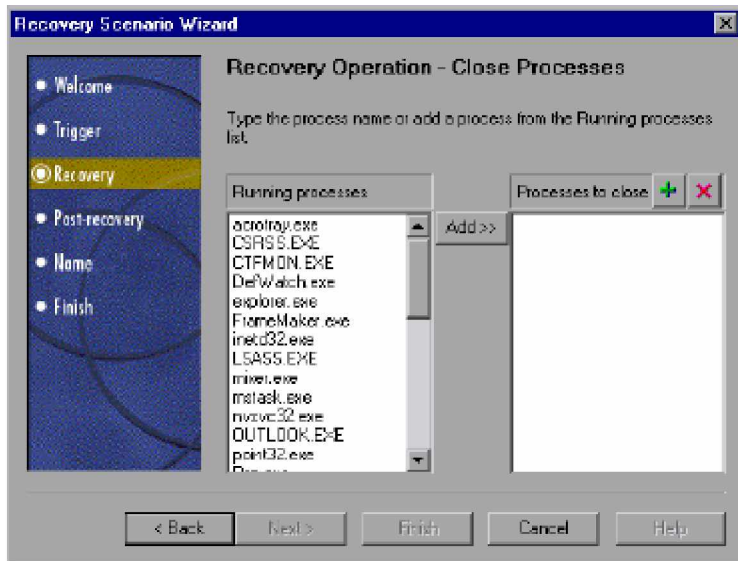
Tip: Hold the CTRL key to change the window focus or perform operations such as a right-click or mouseover to display a context menu. Note that pressing the CTRL key does not enable you to select an application from the Windows taskbar, therefore you must make sure that the window you want to access is not minimized.

All button labels in the selected window are displayed in the list box. Select the required button from the list.

- Ø **Press key or key combination**—Instructs QuickTest to press the specified keyboard key or key combination in the displayed window when the trigger occurs. If you select this option, click in the edit box and then press the key or key combination on your keyboard that you want to specify. Click **Next**. The Recovery Operations screen reopens, showing the keyboard or mouse recovery operation that you defined.

2.4.2.3 Recovery Operation – Close Processes Screen

If you chose a **Close application process** recovery operation in the Recovery Operation screen, the Recovery Operation – Close Processes screen opens.



The **Running processes** list displays all application processes that are currently running. The **Processes to close** list displays the application processes that will be closed when the trigger is activated. You can add application processes to the **Processes to close** list by typing them in the **Processes to close** list or by selecting them from the **Running processes** list.

To add a process from the **Running processes** list, double-click a process in the **Running processes** list or select it and click the **Add** button. You can select multiple processes using standard Windows multiple selection techniques (CTRL and SHIFT keys).

To add a process directly to the **Processes to close** list, click the **Add New Process** button to enter the name of any process you want to add to the list. To remove a process from the **Processes to close** list, select it and click the **Remove Process** button.

Tip: You can modify the name of a process by selecting it in the **Processes to close** list and clicking the process name to edit it. Click **Next**. The Recovery Operations screen reopens, showing the close processes recovery operation that you defined.

2.4.2.4 Recovery Operation – Function Call Screen

If you chose a **Function call** recovery operation in the Recovery Operation screen, the Recovery Operation – Function Call screen opens. Select a recently specified library file in the **Library file** box. Alternatively, click the browse button to navigate to an existing library file.

Note: QuickTest automatically associates the library file you select with your test or component. Therefore, you do not need to associate the library file with your test or component in the Resources tab of the Test Settings dialog box or Business Component Settings dialog box, respectively.

After you select a library file, choose one of the following options:

- Ø **Select function**—Choose an existing function from the library file you selected.

Note: Only functions that match the prototype syntax for the trigger type selected in the Select Trigger Event screen are displayed.

Following is the prototype for each trigger type:

Test run error trigger

OnRunStep

```
(
[in] Object as Object: The object of the current step.
[in] Method as String: The method of the current step.
[in] Arguments as Array: The actual method's arguments.
[in] Result as Integer: The actual method's result.
)
```

Pop-up window and Object state triggers

OnObject

```
(
[in] Object as Object: The detected object.
)
```

Application crash trigger

OnProcess

```
(
[in] ProcessName as String: The detected process's Name.
[in] ProcessId as Integer: The detected process' ID.
)
```

- Ø **Define new function**—Create a new function by specifying a unique name for it, and defining the function in the **Function Name** box according to the displayed function prototype. The new function is added to the library file you selected.

Note: If more than one scenario use a function with the same name from different library files, the recovery process may fail. In this case, information regarding the recovery failure is displayed during the run session.

Click **Next**. The Recovery Operations screen reopens, showing the function operation that you defined.

2.4.3 Post-Recovery Test Run Options Screen

When you clear the **Add another recovery operation** check box in the Recovery Operations screen and click **Next**, the Post-Recovery Test Run Options screen opens.

Post-recovery test run options specify how to continue the run session after QuickTest has identified the event and performed all of the specified recovery operations.



QuickTest can perform one of the following run session options after it performs the recovery operations you defined:

Ø **Repeat current step and continue**

The current step is the step that QuickTest was running when the recovery scenario was triggered. If you are using the **On error** activation option for recovery scenarios, the step that returns the error is often one or more steps later than the step that caused the trigger event to occur. Thus, in most cases, repeating the current step does not repeat the trigger event.

Ø **Proceed to next step**

Skips the step that QuickTest was running when the recovery scenario was triggered. Keep in mind that skipping a step that performs operations on your application may cause subsequent steps to fail.

Ø **Proceed to next action iteration**

Stops performing steps in the current action iteration and begins the next action iteration from the beginning (or the next action if no additional iterations of the current action are required).

Ø **Proceed to next test iteration**

Stops performing steps in the current action and begins the next test iteration from the beginning (or stops running the test if no additional iterations of the current action are required).

Ø **Restart current test run**

Stops performing steps and re-runs the test or component from the beginning.

Ø **Stop the test run**

Stops running the test or component.

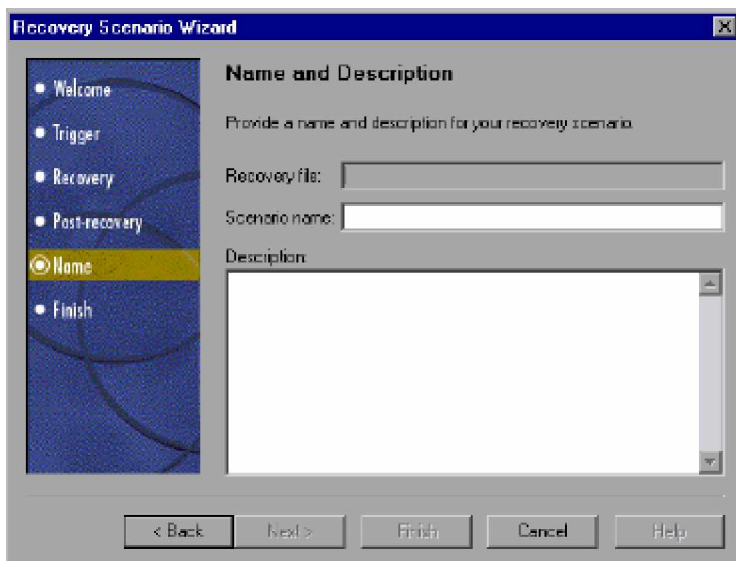
Note: If you chose **Restart Microsoft Windows** as a recovery operation, you can choose from only the last two test run options listed above.

Select a test run option and click **Next** to continue to the Name and Description screen.

2.4.4 Name and Description Screen

After you specify a test run option in the Post-Recovery Test Run Options screen, and click **Next**, the Name and Description screen opens.

In the Name and Description screen, you specify a name by which to identify your recovery scenario. You can also add descriptive information regarding the scenario.

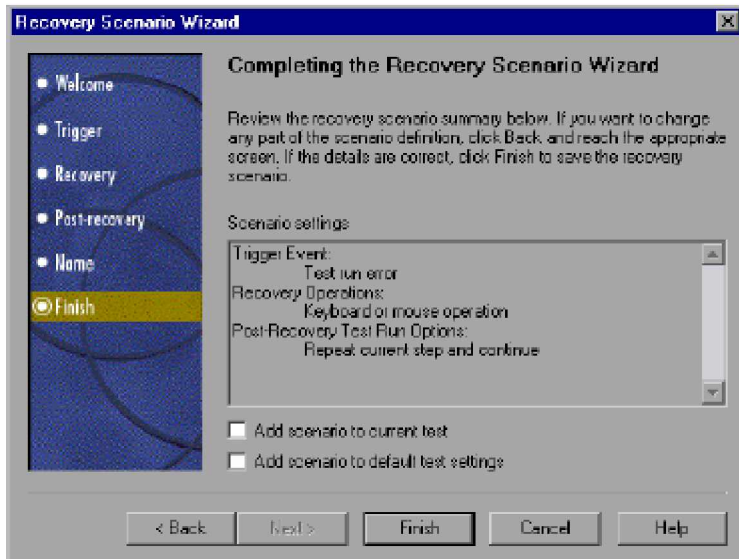


Enter a name and a textual description for your recovery scenario, and click **Next** to continue to the Completing the Recovery Scenario Wizard screen.

2.4.5 Completing the Recovery Scenario Wizard Screen

After you specify a recovery scenario name and description in the Name and Description screen and click **Next**, the Completing the Recovery Scenario Wizard screen opens.

In the Completing the Recovery Scenario Wizard screen, you can review a summary of the scenario settings you defined. You can also specify whether to automatically associate the recovery scenario with the current test or component, and/or to add it to the default settings for all new tests.



Select the **Add scenario to current test** check box to associate this recovery scenario with the current test or component. When you click **Finish**, QuickTest adds the recovery scenario to the **Scenarios** list in the Recovery tab of the Test Settings dialog box or Business Component Settings dialog box, respectively.

Select the **Add scenario to default test settings** check box to make this recovery scenario a default scenario for all new tests. The next time you create a test; this scenario will be listed in the **Scenarios** list in the Recovery tab of the Test Settings dialog box.

Note: You can remove scenarios from the default scenarios list.

Note: You define the default recovery scenarios for all new components in the component template.

Click **Finish** to complete the recovery scenario definition.

Saving the Recovery Scenario in a Recovery File

After you create or modify a recovery scenario in a recovery file using the Recovery Scenario Wizard, you need to save the recovery file.

To save a new or modified recovery file:

- 1 Click the **Save** button. If you added or modified scenarios in an existing recovery file, the recovery file and its scenarios are saved. If you are using a new recovery file, the Save Attachment dialog box opens.

Tip: You can also click the arrow to the right of the **Save** button and select **Save As** to save the recovery file under a different name.

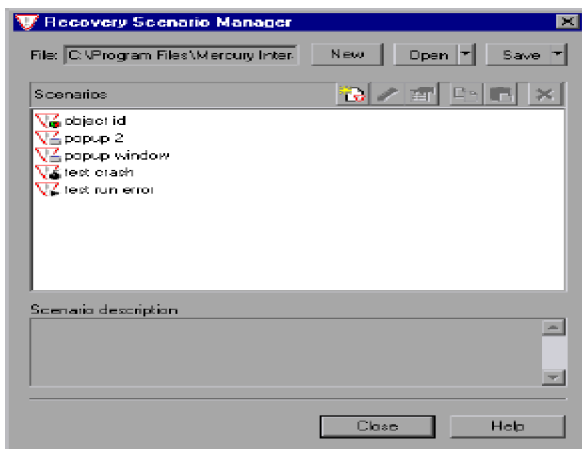
2 Choose the folder in which you want to save the file.

3 Type a name for the file in the **File name** box. The recovery file is saved in the specified location with the file extension **.qrs**.

Tip: If you have not yet saved the recovery file, and you click the **Close** button in the Recovery Scenario Manager dialog box, QuickTest prompts you to save the recovery file. Click **Yes**, and proceed with step 2 above. If you added or modified scenarios in an existing recovery file, and you click **Yes** to the message prompt, the recovery file and its scenarios are saved.





Managing Recovery Scenarios

Once you have created recovery scenarios, you can use the Recovery Scenario Manager to manage them.



The Recovery Scenario Manager contains the following recovery scenario icons:

Icon Description

Icon	Description
	Indicates that the recovery scenario is triggered when a window pops up in an open application during the run session.
	Indicates that the recovery scenario is triggered when the property values of an object in an application match specified values.
	Indicates that the recovery scenario is triggered when a step in the test or component does not run successfully.
	Indicates that the recovery scenario is triggered when an open application fails during the run session.

The Recovery Scenario Manager enables you to manage existing scenarios by:

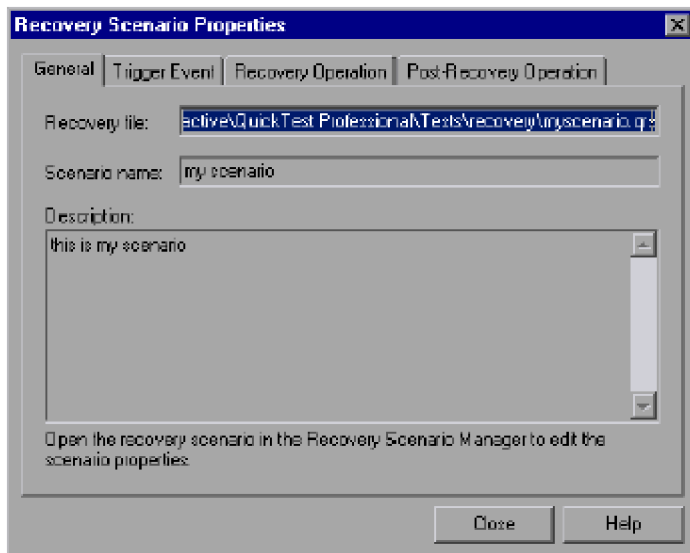
- Ø Viewing Recovery Scenario Properties
- Ø Modifying Recovery Scenarios
- Ø Deleting Recovery Scenarios
- Ø Copying Recovery Scenarios between Recovery Scenario Files

Viewing Recovery Scenario Properties

You can view properties for any defined recovery scenario.

To view recovery scenario properties:

- 1 In the **Scenarios** box, select the recovery scenario whose properties you want to view.
- 2 Click the **Properties** button. Alternatively, you can double-click a scenario in the **Scenarios** box. The Recovery Scenario Properties dialog box opens.



The Recovery Scenario Properties dialog box displays the following read-only information about the selected scenario:

- Ø **General tab**—Displays the name and description defined for the recovery scenario, plus the name and path of the recovery file in which the scenario is saved.
- Ø **Trigger Event tab**—Displays the settings for the trigger event defined for the recovery scenario.
- Ø **Recovery operation tab**—Displays the recovery operation(s) defined for the recovery scenario.
- Ø **Post-Recovery Operation tab**—Displays the post-recovery operation defined for the recovery scenario.

Modifying Recovery Scenarios

You can modify the settings for an existing recovery scenario.

To modify a recovery scenario:

- 1 In the **Scenarios** box, select the scenario that you want to modify.
- 2 Click the **Edit** button. The Recovery Scenario Wizard opens, with the settings you defined for the selected recovery scenario.
- 3 Navigate through the Recovery Scenario Wizard and modify the details as needed.

Note: Modifications you make are not saved until you click **Save** in the Recovery Scenario Manager dialog box. If you have not yet saved your modifications, and you click the **Close** button in the Recovery Scenario Manager dialog box, QuickTest prompts you to save the recovery file. Click **Yes** to save your changes.

Deleting Recovery Scenarios

You can delete an existing recovery scenario if you no longer need it. When you delete a recovery scenario from the Recovery Scenario Manager, the corresponding information is deleted from the recovery scenario file.

Note: If a deleted recovery scenario is associated with a test or component, QuickTest ignores it during the run session.

To delete a recovery scenario:

- 1 In the **Scenarios** box, select the scenario that you want to delete.
- 2 Click the **Delete** button. The recovery scenario is deleted from the Recovery Scenario Manager dialog box.

Note: The scenario is not actually deleted until you click **Save** in the Recovery Scenario Manager dialog box. If you have not yet saved the deletion, and you click the **Close** button in the Recovery Scenario Manager dialog box, QuickTest prompts you to save the recovery file. Click **Yes** to save the recovery scenario file and delete the scenarios.

Copying Recovery Scenarios between Recovery Scenario Files

You can copy recovery scenarios from one recovery scenario file to another.

To copy a recovery scenario from one recovery scenario file to another:

- 1 In the **Scenarios** box, select the recovery scenario that you want to copy.
- 2 Click the **Copy** button. The scenario is copied to the Clipboard.

3 Click the **Open** button and select the recovery scenario file to which you want to copy the scenario, or click the **New** button to create a new recovery scenario file in which to copy the scenario.

4 Click the **Paste** button. The scenario is copied to the new recovery scenario file.

Notes: If a scenario with the same name already exists in the recovery scenario file, you can choose whether you want to replace it with the new scenario you have just copied. Modifications you make are not saved until you click **Save** in the Recovery Scenario Manager dialog box. If you have not yet saved your modifications, and you click the **Close** button in the Recovery Scenario Manager dialog box, QuickTest prompts you to save the recovery file. Click **Yes** to save your changes.

Setting the Recovery Scenarios List for Your Tests or Components

After you have created recovery scenarios, you associate them with selected tests or components so that QuickTest will perform the appropriate scenario(s) during the run sessions if a trigger event occurs. You can prioritize the scenarios and set the order in which QuickTest applies the scenarios during the run session. You can also choose to disable specific scenarios, or all scenarios, that are associated with a test or component. You can also define which recovery scenarios will be used as the default scenarios for all new tests.

Note: You define the default recovery scenarios for all new components in the component template.

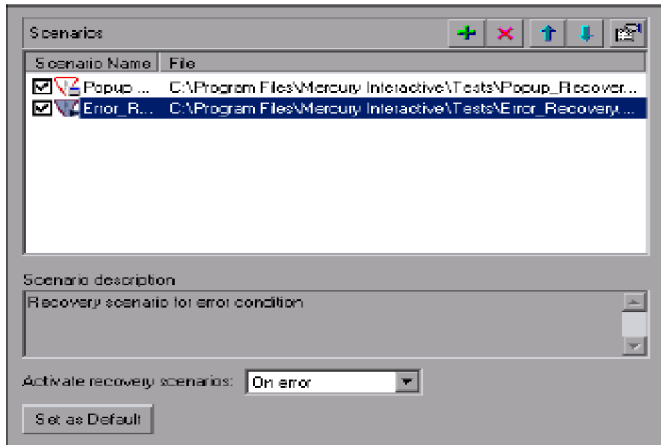
Adding Recovery Scenarios to Your Test or Component

After you have created recovery scenarios, you can associate one or more scenarios with a test or component in order to instruct QuickTest to perform the recovery scenario(s) during the run session if a trigger event occurs. The Recovery tab of the Test Settings dialog box or Business Component Settings dialog box lists all the recovery scenarios associated with the current test or component.

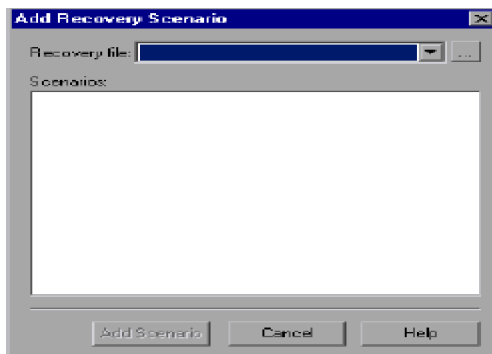
Tip: When a trigger event occurs, QuickTest checks for applicable recovery scenarios in the order in which they are displayed in the Recovery tab.

To add a recovery scenario to a test or component:

1 Choose **Test > Settings** or **Component > Settings**. The Test Settings dialog box or Business Component Settings dialog box opens. Select the Recovery tab.



2 Click the **Add** button. The Add Recovery Scenario dialog box opens.



3 In the **Recovery file** box, select the recovery file containing the recovery scenario(s) you want to associate with the test or component. Alternatively, click the browse button to navigate to the recovery file you want to select. The **Scenarios** box displays the names of the scenarios saved in the selected file.

4 In the **Scenarios** box, select the scenario(s) that you want to associate with the test or component and click **Add Scenario**. The Add Recovery Scenario dialog box closes and the selected scenarios are added to the **Scenarios** list in the Recovery tab.

Tip: You can edit a recovery scenario file path by clicking the path once to highlight it, and then clicking it again to enter edit mode. For example, you may want to modify an absolute file path to be a relative file path. If you modify a recovery scenario file path, you must ensure that the recovery scenario is defined in the new path location before running your test or component.

Viewing Recovery Scenario Properties

You can view properties for any recovery scenario associated with your test or component.

Note: You modify recovery scenario settings from the Recovery Scenario Manager dialog box.

To view recovery scenario properties:

- 1 In the **Scenarios** box, select the recovery scenario whose properties you want to view.
- 2 Click the **Properties** button. Alternatively, you can double-click a scenario in the **Scenarios** box. The Recovery Scenario Properties dialog box opens, displaying read-only information regarding the settings for the selected scenario.

Setting Recovery Scenario Priorities

You can specify the order in which QuickTest performs associated scenarios during a run session. When a trigger event occurs, QuickTest checks for applicable recovery scenarios in the order in which they are displayed in the Recovery tab of the Test Settings dialog box or Business Component Settings dialog box.

To set recovery scenario priorities:

- 1 In the **Scenarios** box, select the scenario whose priority you want to change.
- 2 Click the **Up** or **Down** button. The selected scenario's priority changes according to your selection.
- 3 Repeat steps 1-2 for each scenario whose priority you want to change.

Removing Recovery Scenarios from Your Test or Component

You can remove the association between a specific scenario and a test or component using the Recovery tab of the Test Settings dialog box or Business Component Settings dialog box. After you remove a scenario from a test or component, the scenario itself still exists, but QuickTest will no longer perform the scenario during a run session.

To remove a recovery scenario from your test or component:

- 1 In the **Scenarios** box, select the scenario you want to remove.
- 2 Click the **Remove** button. The selected scenario is no longer associated with the test or component.

Enabling and Disabling Recovery Scenarios

You can enable or disable specific scenarios and determine when QuickTest activates the recovery scenario mechanism in the Recovery tab of the Test Settings dialog box or Business Component Settings dialog box. When you disable a specific scenario, it remains associated with the test or component, but is not performed by QuickTest during the run session. You can enable the scenario at a later time.

To enable/disable specific recovery scenarios:

- Ø Select the check box to the left of one or more individual scenarios to enable them.
- Ø Clear the check box to the left of one or more individual scenarios to disable them.

To define when the recovery mechanism is activated:

- Ø Select one of the following options in the **Activate recovery scenarios** box:
- Ø **On every step**—The recovery mechanism is activated after every step.
- Ø **On error**—The recovery mechanism is activated only after steps that return an error return value. Note that the step that returns an error is often not the same as the step that causes the exception event to occur.

For example, a step that selects a check box may cause a pop-up dialog box to open. Although the pop-up dialog box is defined as a trigger event, QuickTest moves to the next step because it successfully performed the check box selection step. The next several steps could potentially perform checkpoints, functions or other conditional or looping statements that do not require performing operations on your application. It may only be ten statements later that a step instructs

QuickTest to perform an operation on the application that it cannot perform due to the pop-up dialog box. In this case, it is this tenth step that returns an error and triggers the recovery mechanism to close the dialog box. After the recovery operation is completed, the current step is this tenth step, and not the step that caused the trigger event.

- Ø **Never**—The recovery mechanism is disabled.

Note: Choosing **On every step** may result in slower performance during the run session.

Tip: You can also enable or disable specific scenarios or all scenarios associated with a test or component programmatically during the run session.

Setting Default Recovery Scenario Settings for All New Tests

You can click the **Set as Default** button in the Recovery tab of the Test Settings dialog box to set the current list of recovery scenarios to be the default scenarios for all new tests. Any future changes you make to the current recovery scenario list only affect the current test, and do not change the default list that you defined. You define the default recovery scenarios for all new components in the component template.

Programmatically Controlling the Recovery Mechanism

You can use the Recovery object to control the recovery mechanism programmatically during the run session. For example, you can enable or disable the entire recovery mechanism or specific recovery scenarios for certain parts of a run session, retrieve status information about specific recovery scenarios, and explicitly activate the recovery mechanism at a certain point in the run session.

By default, QuickTest checks for recovery triggers when an error is returned during the run session. You can use the Recovery object's **Activate** method to force QuickTest to check for triggers after a specific step in the run session. For example, suppose you know that an object property checkpoint will fail if certain processes are open when the checkpoint is performed. You

want to be sure that the pass or fail of the checkpoint is not affected by these open processes, which may indicate a different problem with your application.

However, a failed checkpoint does not result in a run error. So by default, the recovery mechanism would not be activated by the object state. You can define a recovery scenario that looks for and closes specified open processes when an object's properties have a certain state. This state shows the object's property values as they would be if the problematic processes were open. You can instruct QuickTest to activate the recovery mechanism if the checkpoint fails so that QuickTest will check for and close any problematic open processes and then try to perform the checkpoint again. This ensures that when the checkpoint is performed the second time it is not affected by the open processes.

18.0 Scripting in Real Time Environments

18.1 QuickTest Pro Coding Standards & Best Practices

18.1.1 Introduction:

These standards are for testers who are using QuickTest Pro to develop their automated test scripts. The purpose of this document is to provide testers with general guidelines and rules for building repeatable, accurate, maintainable, and portable test scripts. This standard document covers areas such as Variable Naming, Commenting Code, Formatting of Code etc.

18.2 Naming Conventions

18.2.1 Local scope variables

Variables represent values that can be changed within a procedure or function. Local scope variables are placeholders that reside within a function- or a script-body.

Prefixes for Variable Data Types

Data Type	Prefix	Example
Boolean	bln	BlnLoggedIn
Date	d	dTomorrow
Variant	v	vTblValue
Integer	int	IntAge
Object	o	oUserTable
String	str	strName
Arrays	a	Used in another prefix, as in astrEmployee

18.2.2 Global scope variables

The values of global variables can be used and changed all over the project within all scripts and libraries.

Syntax

"g" + [Prefix]+[ShortDescription]

Letter “g” indicates that the scope of the variable is global. *[Prefix]* is a lowercase letter that represents the type of the global variable. The rules for *[Prefix]* are the same as for “Local scope variables”.

Examples

gintNumOfPersons
gstrPersonLastname

18.2.3 Constants

Constants are “variables” that cannot be changed within a function- or script-body. The value will always be the same during script-execution. Constant names should be descriptive and not ambiguous. Constant names should be in upper case, with proper words separated by the underscore character.

Examples

ID_OS_VERSION
MY_TEMP_URL

18.2.4 Functions/Actions

The Function/Action name should be descriptive of its primary purpose. The Function/Action name must always correspond to what the Function/Action is actually doing. If name of the function is "OpenLogFile("log file name") then it is expected that the function only opens log file.

Examples

CountItemInString()

18.2.5 Reusable Actions

Quick Test Professional reusable actions names should start with an “r” to reflect that the action is reusable.

18.2.6 Scripts

must capitalize the first letter of each word in the test script name . The exception being the product name, which should always be capitalized.

Examples

IB_Auditlog_Save

18.2.7 Function Libraries

Functional library names should reflect the product name, underscore, functionality that is being tested, underscore and the words “lib” at the end.

Example:

IB_Log_Lib

IB = Intranet Browser product

Log = log functionality in the application

Lib = Functional Library

18.2.8 Object Repository Files

Object GUI files names should reflect the product name being tested.

Example:

IB_GUI.tsr

IB = Intranet Browser product

18.3 Coding Rules

18.3.1 Commenting Code

Headers - Every test script, functional library, or actions should contain a header comment that displays the name, creator, date, description, assumptions, preconditions, and post conditions.

Test Script Header Comment.

```
*****
' SCRIPT NAME      : CL11_FeatureAuditAnnotateSDOC
' CREATED BY      : Intertester
' DATE CREATED    : 16/Mar/2001
' DESCRIPTION     : This Script Checks That Feature Audit for Annotate is not
'                   transferred to the auditlog file when a sealed Word document
'                   is annotated by inserting comment
' Modification Log :
' ID   Date   Name   Purpose
' ---  ----  -
*****
```

Function Header Comment.

```
*****
' FUNCTION NAME    : CheckClientFeatureAudited
' CREATED BY      : Intertester
' DATE CREATED    : 09/March/2001
' DESCRIPTION     : This Function checks that required client feature audit is transferred to
auditlog file.
*****
```

```
' Parameters      : strFilePath - Path of decrypted auditlog file.
'
'                 strFeature - Expected feature to be audited.
'                 strPublisherID - Publisher ID of sealed content.
' Return Values   : Returns the number of times the client feature is audited in the auditlog file.
' Preconditions   : Audit log files exist
' Postcondition:  : Feature audit transfer to audit log file
' Modification Log :
' ID   Date   Name   Purpose
' ---  -----  -----
'
'*****
```

Commenting single lines / paragraphs – The need to comment an individual line of code is rare. Two possible reasons for commenting a single line of code are:

The line is complicated enough to need an explanation.

The single line once had an error and you want to record the change. When modifying a single line of code, the original line should be commented out using the author's initials and date in brackets, separated by a single dash.

For example (in VBScript), the following shows a code change by SMB (initials):

```
'[SMB-03/09/01]Browser("e.POWER").Page("Worklist").WebRadioGroup("item_select").Select
"122|0|23"
Browser("e.POWER").Page("Worklist").WebRadioGroup("item_select").Select strItemID '[SMB-
03/09/01]
```

Use end of line comments to annotate data declarations.

Example:

```
Dim intEmpAge      'Stores Employee Age.
Dim strEmpName     'Stores Employee Name.
```

Such data annotations include: units of numeric data, range of allowable values, limitations on input data, and the meaning of enumerated or constant codes.

Focus paragraph comments on the *why* rather than the *how*. Simply put, a reader should be able to discern from quality code how an operation is performed. Why that operation is performed will be less apparent.

When commenting individual lines or paragraphs, indent the comment and align it with its corresponding code. Also, precede the comment line by a single blank line.

Commenting Control Structures

Place a comment before each block of statements, if, case, or loop. Use the comment to clarify the purpose of the control structure.

Example:

```
'Loop through the each row in the data table.
```

```
For intCount=1 to datatable.GetRowCount
```

```
    'Check whether content in both columns are equal.
```

```

If datatable("Name1", dtGlobalSheet)= datatable("Name2", dtGlobalSheet) then
    reporter.ReportEvent micPass,"content of both column are equal in row" & intCount
End If
Next

```

For long code blocks, comment the end of each control structure. This makes script code which spans many lines easier to follow.

Example:

```

For intCount=1 to datatable.GetRowCount
    'Check whether content in both columns are equal.
    If datatable("Name1", dtGlobalSheet)= datatable("Name2", dtGlobalSheet) then
        reporter.ReportEvent micPass,"content of both column are equal in row" & intCount
    End If
Next 'Loop through the each row in the data table.

```

18.3.2 Formatting Code

The following are the directives that must be complied with while formatting the code

- Do not place multiple statements on a single line.
Placing multiple statements in a single line increase complexity. Do not do it.
- Do not exceed 90 Characters on a line
- Make sure that not more than 90 characters are placed on a single line.
- Use Indentation to show organizational structure
- Do proper indentation through out your coding. Flow control statements should be indented one tab length for easier readability.

Example:

```

For intCount = 0 to 10
    MyVar = MyVar + 1
    If MyVar = 20
        MyVar =0
    End If
Next

```

18.3.3 Using Shared Object Repository

Use shared Object Repository instead of Object Repository per Action. This will an easy way for handling changes in the object repository.

18.3.4 Using Relative paths

Use relative paths while calling Shared object repository files, VbScript function library files and Reusable actions.

18.3.5 Using Global Variables

Declare and Assign, all the variables that can be used in many scripts (For example: Username, Password, PrinterName etc) in a Reusable Action or VBScript file and use these variables in Other Actions. Using this method we can modify the value of variable in single global file instead of making modification in many script files.