



**MERCURY
QUICKTEST PROFESSIONAL™**

VERSION 9.0

Advanced Features User's Guide

MERCURY™

Mercury QuickTest Professional

Advanced Features User's Guide

Version 9.0

MERCURY™

Mercury QuickTest Professional Advanced Features User's Guide, Version 9.0

This manual, and the accompanying software and other documentation, is protected by U.S. and international copyright laws, and may be used only in accordance with the accompanying license agreement. Features of the software, and of other products and services of Mercury Interactive Corporation, may be covered by one or more of the following patents: United States: 5,511,185; 5,657,438; 5,701,139; 5,870,559; 5,958,008; 5,974,572; 6,137,782; 6,138,157; 6,144,962; 6,205,122; 6,237,006; 6,341,310; 6,360,332, 6,449,739; 6,470,383; 6,477,483; 6,549,944; 6,560,564; 6,564,342; 6,587,969; 6,631,408; 6,631,411; 6,633,912; 6,694,288; 6,738,813; 6,738,933; 6,754,701; 6,792,460 and 6,810,494. Australia: 763468 and 762554. Other patents pending. All rights reserved.

Mercury, Mercury Interactive, the Mercury logo, the Mercury Interactive logo, LoadRunner, WinRunner, SiteScope and TestDirector are trademarks of Mercury Interactive Corporation and may be registered in certain jurisdictions. The absence of a trademark from this list does not constitute a waiver of Mercury's intellectual property rights concerning that trademark.

All other company, brand and product names may be trademarks or registered trademarks of their respective holders. Mercury disclaims any responsibility for specifying which marks are owned by which companies or which organizations.

Mercury Interactive Corporation
379 North Whisman Road
Mountain View, CA 94043
Tel: (650) 603-5200
Toll Free: (800) TEST-911
Customer Support: (877) TEST-HLP
Fax: (650) 603-5300

© 1992 - 2006 Mercury Interactive Corporation, All rights reserved

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

Multi-Volume Chapter Summary

QuickTest Professional user documentation is divided into two volumes:

- ▶ The *QuickTest Professional Basic Features User's Guide* introduces QuickTest and describes the basic features that are used in everyday testing.
- ▶ The *QuickTest Professional Advanced Features User's Guide* describes advanced features that you can use when testing applications. It also describes how to work with other Mercury products.

A summary of the various chapters in each guide is provided below:

QuickTest Professional Basic Features User's Guide

PART I: INTRODUCING QUICKTEST PROFESSIONAL

Chapter 1: Introduction	3
Chapter 2: QuickTest at a Glance	15
Chapter 3: Understanding the Test Object Model.....	57

PART II: CREATING TESTS

Chapter 4: Designing Tests.....	75
Chapter 5: Working with the Keyword View	107
Chapter 6: Working with Test Objects	139
Chapter 7: Understanding Checkpoints	209
Chapter 8: Checking Object Property Values.....	217
Chapter 9: Checking Tables	227
Chapter 10: Checking Text	249

Chapter 11: Checking Bitmaps	267
Chapter 12: Checking Databases	279
Chapter 13: Checking XML	295
Chapter 14: Configuring Values	327
Chapter 15: Parameterizing Values	347
Chapter 16: Outputting Values	393
Chapter 17: Working with Actions	455
Chapter 18: Handling Missing Resources	489
Chapter 19: Working with Data Tables	501
Chapter 20: Adding Steps Containing Programming Logic	525

PART III: RUNNING AND DEBUGGING TESTS

Chapter 21: Debugging Tests and Function Libraries	575
Chapter 22: Running Tests	595
Chapter 23: Analyzing Test Results	619

PART IV: CONFIGURING BASIC SETTINGS

Chapter 24: Setting Global Testing Options	693
Chapter 25: Setting Options for Individual Tests	739
Chapter 26: Setting Record and Run Options	773

PART V: WORKING WITH SUPPORTED ENVIRONMENTS

Chapter 27: Working with QuickTest Add-Ins	793
Chapter 28: Testing Web Objects	803

QuickTest Professional Advanced Features User's Guide

PART I: WORKING WITH ADVANCED TESTING FEATURES

Chapter 1: Working with Advanced Action Features	3
Chapter 2: Learning Virtual Objects	33
Chapter 3: Defining and Using Recovery Scenarios	45
Chapter 4: Configuring Object Identification	93
Chapter 5: Working with the Expert View and Function Library Windows ...	123
Chapter 6: Working with User-Defined Functions and Function Libraries	185
Chapter 7: Automating QuickTest Operations	231

PART II: MANAGING AND MERGING OBJECT REPOSITORIES

Chapter 8: Managing Object Repositories	243
Chapter 9: Merging Shared Object Repositories	275

PART III: CONFIGURING ADVANCED SETTINGS

Chapter 10: Configuring Web Event Recording	311
Chapter 11: Customizing the Expert View and Function Library Windows ...	333
Chapter 12: Setting Testing Options During the Run Session	343

PART IV: WORKING WITH OTHER MERCURY PRODUCTS

Chapter 13: Working with WinRunner	351
Chapter 14: Working with Quality Center	361
Chapter 15: Working with Business Process Testing	403
Chapter 16: Working with Mercury Performance Testing and Business Availability Center Products	413

PART V: APPENDIX

Appendix A: Working with QuickTest—Frequently Asked Questions ...	429
---	-----

Table of Contents

Multi-Volume Chapter Summary	iii
QuickTest Professional Basic Features User's Guide.....	iii
QuickTest Professional Advanced Features User's Guide.....	v
Welcome to QuickTest	xiii
Using This Guide.....	xiv
Product Documentation.....	xv
Additional Online Resources.....	xvii
Documentation Updates	xviii
Typographical Conventions.....	xix

PART VI: WORKING WITH ADVANCED TESTING FEATURES

Chapter 1: Working with Advanced Action Features	3
About Working with Advanced Action Features	4
Inserting Calls to Existing Actions	4
Setting Action Parameters	12
Using Action Parameters	16
Setting Action Call Properties	21
Sharing Action Information	26
Understanding Action Syntax in the Expert View.....	29
Exiting an Action.....	31
Chapter 2: Learning Virtual Objects	33
About Learning Virtual Objects	33
Understanding Virtual Objects	35
Understanding the Virtual Object Manager	36
Defining a Virtual Object	37
Removing or Disabling Virtual Object Definitions.....	42
Chapter 3: Defining and Using Recovery Scenarios	45
About Defining and Using Recovery Scenarios.....	45
Deciding When to Use Recovery Scenarios.....	47
Defining Recovery Scenarios	48
Understanding the Recovery Scenario Wizard	52
Managing Recovery Scenarios	79
Setting the Recovery Scenarios List for Your Tests.....	85
Programmatically Controlling the Recovery Mechanism	91

Chapter 4: Configuring Object Identification	93
About Configuring Object Identification	94
Understanding the Object Identification Dialog Box.....	95
Configuring Smart Identification.....	109
Mapping User-Defined Test Object Classes	119
Chapter 5: Working with the Expert View and Function Library	
Windows	123
About Working with the Expert View and Function Library	
Windows	124
Understanding and Using the Expert View	125
Navigating in the Expert View and Function Libraries	136
Understanding Basic VBScript Syntax.....	146
Using Programmatic Descriptions.....	155
Running and Closing Applications Programmatically	165
Using Comments, Control-Flow, and Other VBScript Statements...	167
Retrieving and Setting Test Object Property Values	176
Accessing Run-Time Object Properties and Methods	177
Running DOS Commands.....	179
Enhancing Your Tests and Function Libraries Using the	
Windows API.....	180
Choosing Which Steps to Report During the Run Session	182
Chapter 6: Working with User-Defined Functions and	
Function Libraries	185
About Working with User-Defined Functions and Function	
Libraries.....	186
Managing Function Libraries	188
Working with Associated Function Libraries	200
Using the Function Definition Generator	204
Registering User-Defined Functions as Test Object Methods	219
Additional Tips for Working with User-Defined Functions	225
Executing Externally-Defined Functions from Your Test	228
Chapter 7: Automating QuickTest Operations	231
About Automating QuickTest Operations	232
Deciding When to Use QuickTest Automation Programs	233
Choosing a Language and Development Environment for	
Designing and Running Automation Programs	235
Learning the Basic Elements of a QuickTest Automation Program..	237
Generating Automation Scripts	238
Using the QuickTest Automation Object Model Reference.....	239

PART VII: MANAGING AND MERGING OBJECT REPOSITORIES

Chapter 8: Managing Object Repositories	243
About Managing Object Repositories.....	244
Understanding the Object Repository Manager	246
Working with Object Repositories	253
Modifying Object Repositories.....	258
Working with Repository Parameters	261
Modifying Test Object Details.....	268
Locating Objects.....	270
Performing Merge Operations.....	271
Performing Import and Export Operations.....	272
Chapter 9: Merging Shared Object Repositories.....	275
About Merging Shared Object Repositories	276
Understanding the Object Repository Merge Tool	277
Using Object Repository Merge Tool Commands.....	282
Defining Default Settings	284
Merging Two Object Repositories	288
Updating a Shared Object Repository from Local Object Repositories	290
Viewing Merge Statistics.....	296
Understanding Object Conflicts	297
Resolving Object Conflicts	300
Filtering the Target Repository Pane	302
Synchronizing Object Repository Views	303
Finding Specific Objects	304
Saving the Target Object Repository	305

PART VIII: CONFIGURING ADVANCED SETTINGS

Chapter 10: Configuring Web Event Recording.....	311
About Configuring Web Event Recording	312
Selecting a Standard Event Recording Configuration.....	313
Customizing the Event Recording Configuration	315
Recording Right Mouse Button Clicks	325
Saving and Loading Custom Event Configuration Files.....	329
Resetting Event Recording Configuration Settings.....	331

Chapter 11: Customizing the Expert View and Function Library	
Windows	333
About Customizing the Expert View and Function Library	
Windows	334
Customizing Editor Behavior	335
Customizing Element Appearance	337
Personalizing Editing Commands	339
Chapter 12: Setting Testing Options During the Run Session	343
About Setting Testing Options During the Run Session	343
Setting Testing Options	344
Retrieving Testing Options	346
Controlling the Test Run	346
Adding and Removing Run-Time Settings	347

PART IX: WORKING WITH OTHER MERCURY PRODUCTS

Chapter 13: Working with WinRunner	351
About Working with WinRunner	351
Calling WinRunner Tests	352
Calling WinRunner Functions	356
Chapter 14: Working with Quality Center	361
About Working with Quality Center	362
Connecting to and Disconnecting from Quality Center	363
Saving Tests to a Quality Center Project	374
Opening Tests from a Quality Center Project	375
Working with Template Tests	379
Running a Test Stored in a Quality Center Project from QuickTest	387
Managing Test Versions in QuickTest	389
Setting Preferences for Quality Center Test Runs	398
Chapter 15: Working with Business Process Testing	403
About Working with Business Process Testing	403
Understanding Business Process Testing Roles	404
Understanding Business Process Testing Methodology	408

Chapter 16: Working with Mercury Performance Testing and Business Availability Center Products.....	413
About Working with Mercury Performance Testing and Business Availability Center Products	414
Using QuickTest Performance Testing and Business Availability Center Features	415
Designing QuickTest Tests for Use with LoadRunner or Business Process Monitor	416
Inserting and Running Tests in LoadRunner or Business Process Monitor	417
Measuring Transactions	419
Using Silent Test Runner	423

PART X: APPENDIX

Appendix A: Working with QuickTest—	
Frequently Asked Questions	429
Recording and Running Tests	430
Programming in the Expert View.....	431
Working with Dynamic Content	431
Advanced Web Issues	432
Test Maintenance	434
Testing Localized Applications.....	436
Improving QuickTest Performance	437
Index.....	441

Table of Contents

Welcome to QuickTest

Welcome to QuickTest Professional, the Mercury automated keyword-driven testing solution. QuickTest provides everything you need to quickly create and run tests.

Note: The *QuickTest Professional Basic Features User's Guide* and *QuickTest Professional Advanced Features User's Guide* are available as separate books only in the PDF version. In the context-sensitive Help, the information from both books is combined.

Using This Guide

This guide describes how to use QuickTest to test your applications. It provides step-by-step instructions to help you create, debug, and run tests, and report defects detected during the testing process. This guide contains the following parts:

Part I Working with Advanced Testing Features

Describes how to work with actions, virtual objects, recovery scenarios, configure object identification, and create Smart Identification definitions. It also describes how to work with user-defined functions and function libraries in QuickTest. In addition, it describes several programming techniques to create more powerful scripts, and describes how to enhance your test using the Expert View. It also describes how to automate QuickTest operations.

Part II Managing and Merging Object Repositories

Describes how QuickTest identifies objects in your application and how to manage and merge object repositories.

Part III Configuring Advanced Settings

Describes how to configure Web event recording, customize the Expert View, and set testing options during a run session.

Part IV Working with Other Mercury Products

Describes how you can run tests and call functions in compiled modules from WinRunner, the Mercury enterprise functional testing tool for Microsoft Windows applications. This section also describes how QuickTest can be used with Business Process Testing, and how QuickTest interacts with Mercury Quality Center (formerly TestDirector), the Mercury centralized quality solution. This section also describes considerations for designing QuickTest tests for use with Mercury performance testing and application management products.

Part V Appendix

Provides information on frequently asked questions.

Product Documentation

In addition to this Advanced Features User's Guide, QuickTest Professional comes with the following documentation:

QuickTest Professional Installation Guide explains how to install QuickTest Professional.

What's New (available from **Help > What's New**) describes the newest features, enhancements, and supported environments in this latest version of QuickTest Professional.

QuickTest Professional Basic Features User's Guide provides step-by-step instructions for using QuickTest Professional to test your application or Web site.

QuickTest Professional for Business Process Testing User's Guide provides step-by-step instructions for using QuickTest Professional to create and manage assets for use with Business Process Testing.

QuickTest Professional Tutorial teaches you basic QuickTest skills and shows you how to design tests for your applications.

Readme (available from the QuickTest Professional Start menu program folder) provides the latest news and information about QuickTest Professional.

Printer-Friendly Documentation (available from **Help > Printer-Friendly Documentation**) displays the complete documentation set in Adobe portable document format (PDF). Online books can be read and printed using Adobe Reader, which can be downloaded from the Adobe Web site (<http://www.adobe.com>).

QuickTest Professional Context-Sensitive Help (available from specific dialog boxes and windows) describes QuickTest dialog boxes and windows.

QuickTest Professional Object Model Reference (available from **Help > QuickTest Professional Help**) describes QuickTest Professional test objects, lists the methods and properties associated with each object, and provides syntax information and examples for the methods.

QuickTest Professional Automation Object Model Reference (available from the QuickTest Professional Start menu program folder and from **Help > QuickTest Automation Object Model Reference**) 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. The automation object model assists you in automating test management, by providing objects, methods and properties that enable you to control virtually every QuickTest feature and capability.

VBScript Reference (available from **Help > QuickTest Professional Help**) contains Microsoft VBScript documentation, including VBScript, Script Runtime, and Windows Script Host.

Additional Online Resources

QuickTest Professional includes the following additional online resources:

Mercury Tours sample Web site (available from the QuickTest Professional Start menu program folder and also available from the QuickTest Professional Record and Run Settings dialog box) and the **Mercury Tours** Windows sample flight application (available from the QuickTest Professional Start menu program folder) are the basis for many examples in this book. The URL for the Web site is <http://newtours.mercury.com>.

Knowledge Base (available from **Help > Knowledge Base**) uses your default Web browser to open the Mercury Customer Support knowledge base, which enables you to browse the Mercury and user-contributed knowledge base articles, and add your own articles. The URL for this Web site is <http://support.mercury.com/cgi-bin/portal/CSO/kbBrowse.jsp>.

Customer Support Web Site (available from **Help > Customer Support Web Site**) uses your default Web browser to open the Mercury Customer Support Web site. This site enables you to browse the knowledge base and add your own articles, post to and search user discussion forums, submit support requests, download patches and updated documentation, and more. The URL for this Web site is <http://support.mercury.com>.

Send Feedback (available from **Help > Send Feedback**) enables you to send online feedback about QuickTest Professional to the product team.

Mercury Home Page (available from **Help > Mercury Home Page**) uses your default Web browser to open the Mercury home page. This site provides you with the most up-to-date information on Mercury and its products. This includes new software releases, seminars and trade shows, customer support, educational services, and more. The URL for this Web site is <http://www.mercury.com>.

Mercury Best Practices contain guidelines for planning, creating, deploying, and managing a world-class IT environment. Mercury provides three types of best practices: Process Best Practices, Product Best Practices, and People Best Practices. Licensed customers of Mercury software can read and use the Mercury Best Practices available from the Customer Support site, <http://support.mercury.com>.

Documentation Updates

Mercury is continually updating its product documentation with new information. You can download the latest version of this document from the Customer Support Web site (<http://support.mercury.com>).

To download updated documentation:

- 1** In the Customer Support Web site, click the **Documentation** link.
- 2** Under **Please Select Product**, select **QuickTest Professional**.
Note that if **QuickTest Professional** does not appear in the list, you must add it to your customer profile. Click **My Account** to update your profile.
- 3** Click **Retrieve**. The Documentation page opens and lists the documentation available for the current release and for previous releases. If a document was updated recently, **Updated** appears next to the document name.
- 4** Click a document link to download the documentation.

Typographical Conventions

This book uses the following typographical conventions:

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

Welcome

Part I

Working with Advanced Testing Features

1

Working with Advanced Action Features

You can divide your test into actions to streamline the process of testing your application or Web site. This chapter covers the advanced use of actions in your test. Using basic action-related features is described in Chapter 17, “Working with Actions” in the *QuickTest Professional Basic Features User’s Guide*.

This chapter describes:

- About Working with Advanced Action Features
- Inserting Calls to Existing Actions
- Setting Action Parameters
- Using Action Parameters
- Setting Action Call Properties
- Sharing Action Information
- Understanding Action Syntax in the Expert View
- Exiting an Action

About Working with Advanced Action Features

Actions help divide your test into logical units, such as the main sections of a Web site, or specific activities that you perform in your application.

A test is comprised of calls to actions. When you create a new test, it contains a call to a single action. By creating tests that call multiple actions, you can design tests that are more modular and efficient.

You can pass information between actions in several ways. You can also specify input parameters for actions, so that steps in an action can use values supplied from elsewhere in the test. You can also output values from actions to be used in steps later in the test, or to be passed back to the application that ran the test. For more information, see “Using Action Parameters” on page 16.

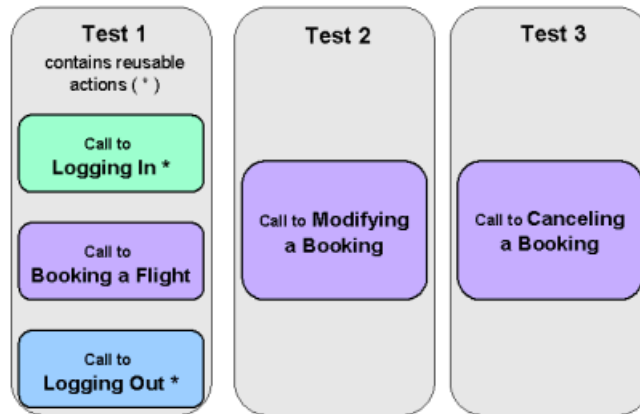
Inserting Calls to Existing Actions

When you plan a suite of tests, you may realize that each test requires some identical activities, such as logging in. Rather than recording the login process three times in three separate tests and enhancing this part of the script (with checkpoints, parameterization, and programming statements) separately for each test, you can create an action that logs into a flight reservation system and store it with one test. Once you are satisfied with the action you created, you can insert calls to the existing action into other tests.

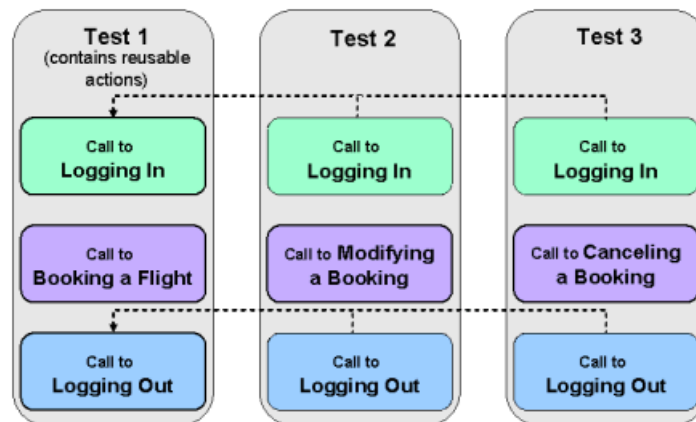
You can insert calls to an existing action by inserting a call to a copy of the action, or by inserting a call to the original action.

For example, suppose you want to record the following three tests in the Mercury Tours site—booking a flight, modifying a reservation, and deleting a reservation. While planning your tests, you realize that for each test, you need to log in and log out of the site, giving a total of five actions for all three tests.

You would initially create three tests with five actions. Test 1 would contain two reusable actions (Logging In and Logging Out). These actions can later be called by Test 2 and Test 3.



You would then finish creating Test 2 and Test 3 by inserting calls to the reusable actions you created in Test 1.



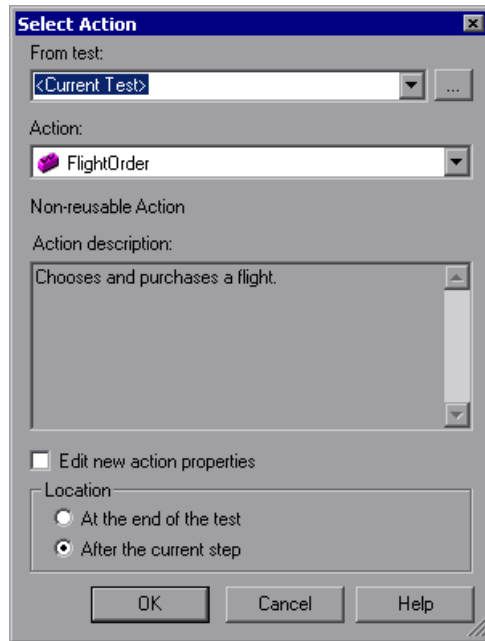
Inserting Calls to Copies of Actions

When you insert a call to a copy of an action into a test, the original action is copied in its entirety, including checkpoints, parameterization, the corresponding action tab in the Data Table, plus any defined action parameters. If the test you are copying has objects in the local object repository, the copied action's local object repository is also copied together with the action.

The action is inserted into the test as an independent, non-reusable action (even if the original action was reusable). 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 non-reusable 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 copied action.

To create a copy of an action and call the copy in your test:

- 1 While recording or editing your test, choose **Insert > Call to Copy of Action**, right-click an action icon and select **Insert Call to Copy of Action**, or right-click any step and select **Action > Insert Call to Copy**. The Select Action dialog box opens.



- 2 Use the **From test** browse button to find the test containing the action you want to copy. The **Action** box displays all local actions (actions that are stored with the test you selected).

Note: You can enter a Quality Center folder or a relative path in the **From test** box. If you enter a relative path, QuickTest searches for the test in the folders listed in the Folders tab of the Options dialog box. For more information, refer to “Setting Folder Testing Options” on page 698 in the *QuickTest Professional Basic Features User’s Guide*.

- 3** In the **Action** list, select the action you want to insert. When you select an action, its type (**Non-reusable** or **Reusable Action**) and description, if one exists, are displayed. This helps you identify the action you want to copy. For more information about action descriptions refer to “Setting General Action Properties” on page 469 in the *QuickTest Professional Basic Features User’s Guide*.
- 4** If you want to modify the copied action’s properties, select the **Edit new action properties** check box. If you select this option, the Action Properties dialog box is displayed when you click **OK**. You can then modify the action properties as described in “Setting Action Call Properties” on page 21.

Note: If you do not select this option, you can modify the action’s properties later by right-clicking the action icon in the Keyword View and selecting **Action Properties**.

- 5** Decide where to insert the call to the copy of the action and select **At the end of the test** or **After the current step**.

For more information about inserting actions within actions, see “Using Action Parameters” on page 16.

Note: If the currently selected step is a reusable action from another test, the call to the copy of the action is added automatically to the end of the test (the **After the current step** option is disabled).

- 6** Click **OK**. The action is inserted into the test as a call to an independent, non-reusable action. You can move your action call to another location in your test by dragging it to the desired location. For more information about moving actions, refer to “Moving Actions and Steps in the Hierarchy” on page 127 in the *QuickTest Professional Basic Features User’s Guide*.

Inserting a Call to an Existing Action

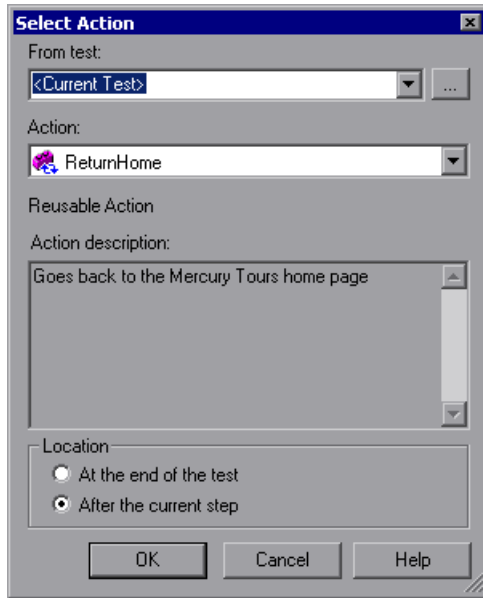
You can insert a call to a reusable action that is stored in your current test (local action), or in any other test (external action). Inserting a call to an existing action is similar to linking to it. You can view the steps of the action in the action view, but you cannot modify them. The called action's local object repository (if it has one) is also read-only. If you call an external action, you can choose, however, whether you want the data from the action's data sheet to be imported as a local, editable copy, or whether you want to use the (read-only) data from the original action.

To modify a called, external action, you must open the test with which the action is stored and make your modifications there. The modifications apply to all tests that call that action. If you chose to use the original action's data when you call an external action, then changes to the original action's data are applied as well.

Tip: You can view the location of the original action in the General tab of the Action Properties dialog box.

To insert a call to an existing action:

- 1 Choose **Insert > Call to Existing Action**, right-click an action icon and select **Insert Call to Existing Action**, or right-click any step and select **Action > Insert Call to Existing**. The Select Action dialog box opens.



- 2 Use the **From test** browse button to find the test that contains the action you want to call. The **Action** box displays all reusable actions in the test you selected.

Note: You can enter a Quality Center folder or a relative path in the **From test** box. If you enter a relative path, QuickTest searches for the test in the folders listed in the Folders tab of the Options dialog box. For more information, refer to “Setting Folder Testing Options” on page 698 in the *QuickTest Professional Basic Features User’s Guide*.

- 3** In the **Action** list, select the action you want to call. When you select an action, its type (**Reusable Action**) and description, if one exists, are displayed. This helps you identify the action you want to call. For more information about action descriptions, refer to “Setting General Action Properties” on page 469 in the *QuickTest Professional Basic Features User’s Guide*.


Tip: External actions that the test calls are also displayed in the list. If the action you want to call is already called from within the selected test, you can select it from the list of actions. This creates another call to the original action.

Note: QuickTest disables the **Action** list if the selected test does not contain any reusable or external actions.

- 4** Decide where to insert the call to the action and select **At the end of the test** or **After the current step**.

Note: If the currently selected step is a reusable action from another test, the call to the action is added automatically to the end of the test (the **After the current step** is disabled).

For more information about inserting actions within actions, see “Using Action Parameters” on page 16.

- 5 Click **OK**. A call to the action  is inserted into the test flow. You can move your action call to another location in your test by dragging it to the desired location. For more information about moving actions, refer to “Moving Actions and Steps in the Hierarchy” on page 127 in the *QuickTest Professional Basic Features User’s Guide*.

Tip: You can create an additional call to any reusable or external action in your test by pressing CTRL while you drag and drop the action to another location at a parallel (sibling) level within your test.

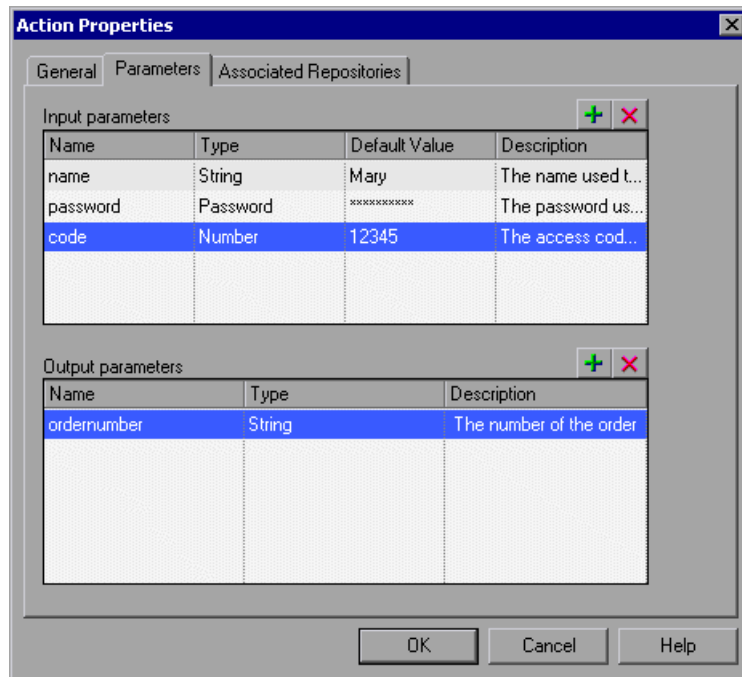
Setting Action Parameters

You can specify input parameters for an action so that steps in the action can use values supplied from elsewhere in the test. Input values for an action parameter can be retrieved from the test (for a top-level action) or from the parameters of the parent action that calls it (for a nested action), or from the output of a previous action call (for a sibling action).

You can specify output parameters for an action, so that it can return values for use later in the test. For example, you can output a parameter value to a parent action so that a later nested action can use the value.

For each input or output action parameter, you define a name, a type, and optionally, a description. You can also specify a default value for each action input parameter, or you can use the default value that QuickTest provides for the parameter value type that you choose. The default value is saved with the action and is used by the action if a value is not defined for a parameter in the action call. You can define, modify, and delete input and output parameters in the Parameters tab of the Action Properties dialog box (**Edit > Action > Action Properties** or right-click an action and choose **Action Properties**).

For more information on using action parameters, see “Using Action Parameters” on page 16 and “Guidelines for Working with Action Parameters” on page 19.



To add a new input or output action parameter:



- 1 Click the **Add** button above the **Input parameters** or **Output parameters** lists to add a new parameter to the appropriate list. A row for the new parameter is added to the relevant list.
- 2 Click in the **Name** box and enter a name for the parameter.

- 3** Select the value type for the parameter in the **Type** box. You can select one of the following types:

 - ▶ **String**—A character string enclosed within a pair of quotation marks, for example, “New York”. If you enter a value and do not include the quotation marks, QuickTest adds them automatically when the value is inserted in the script during the test run. The default value is an empty string.
 - ▶ **Boolean**—A true or false value. If you select a **Boolean** value type, you can click in the **Default Value** column and click the arrow to select a **True** or **False** value. The default value is **True**.
 - ▶ **Date**—A date string, for example, 3/2/2005. If you select a **Date** value type, you can click in the **Default Value** column and click the arrow to open a calendar from which you can select a date. The default value is today’s date.
 - ▶ **Number**—Any number. The default value is 0.
 - ▶ **Password**—An encrypted password value. If you select a **Password** value type, the password characters are masked when you enter the password in the **Default Value** field. In the action, however, the value appears encrypted. The default value is an empty string, which also appears as an encrypted value in the actual action.
 - ▶ **Any**—A variant value type, which accepts any of the above value types. Note that if you select the **Any** value type, you must specify the value in the format that is required in the location where you intend to use the value. For example, if you intend to use the value later as a string, you must enclose it in quotation marks. When you specify a value of **Any** type, QuickTest checks whether it is a number. If the value is not a number, QuickTest automatically encloses it in quotation marks. If you are editing an existing value, QuickTest automatically encloses it in quotation marks if the previous value had quotation marks. The default value is an empty string.
- 4** If you are defining an input action parameter, click in the **Default Value** box and enter a default value for the parameter. Alternatively, you can leave the default value provided by QuickTest for the parameter value type. The default value is required so that you can run the action without receiving parameter values from elsewhere in the test.

- 5 (Optional) Click in the **Description** box and enter a description of the parameter, for example, the purpose of the parameter in the action. QuickTest displays this description together with the name of the parameter in any dialog box in which you can choose an action parameter, including the Output Options, Parameter Options, and Value Configuration Options dialog boxes.

To modify an existing action parameter:

- 1 Select the parameter you want to modify from the **Input parameters** or **Output parameters** list.
- 2 Modify the values as necessary in the edit boxes of the parameter row.

To delete an existing action parameter:

- 1 Select the parameter you want to delete from the **Input parameters** or **Output parameters** list.
- 2 Click the **Delete** button. The parameter is removed from the list.



Note: When you delete an action parameter, make sure that you also delete any steps that use the action parameter.

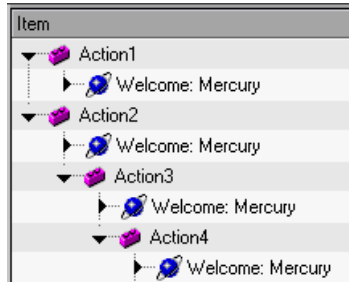
Using Action Parameters

Action parameters enable you to transfer input values from your test to a top-level action, from a parent action to a nested action, or from an action to a sibling action that occurs later in the test. Action parameters also enable you to transfer output values from a step in an action to its parent action, or from a top-level action back to the script or application that ran (called) your test. For example, you can output a value from a step in a nested action and store it in an output action parameter, and then use that value as input in a later step in the calling parent action.

You can use action parameters in any step in your action (including function calls). You define the parameters that an action can receive and the output values that it can return in the Parameters tab of the Action Properties dialog box (**Edit > Action > Action Properties** or right-click an action and choose **Action Properties**). You specify the actual values that are provided to these parameters and the locations in which the output values are stored using the Parameter Values tab in the Action Call Properties dialog box (opened by right-clicking an action and choosing **Action Call Properties**).


You can specify input parameters for an action so it can receive input values from elsewhere in the test. Input values for an action parameter can be retrieved from the test (for a top-level action), from the parameters of the parent action that calls it (for a nested action), or from the output of a previous action call (for a sibling action). You can also specify output parameters for an action, so that it can output values for use later in the test, or pass values back to the application that ran (called) the test.

For example, suppose you want to take a value from the external application that runs (calls) your test and use it in an action within your test. In the test below, you would need to pass the input test parameter from the external application through Action2 and Action3 to the required step in Action4.



You would do this as follows:

- 1 Define the input test parameter (**File > Settings > Parameters** tab) with the value that you want to use later in the test.
- 2 Define an input action parameter for Action2 (**Edit > Action > Action Properties > Parameters** tab) with the same value type as the input test parameter.
- 3 Parameterize the input action parameter value (**Edit > Action > Action Call Properties > Parameter Values** tab) using the input test parameter value you specified above.
- 4 Define an input action parameter for Action3 (**Edit > Action > Action Properties > Parameters** tab) with the same value type as the input test parameter.
- 5 Parameterize the input action parameter value.
 - Choose **Edit > Action > Action Call Properties > Parameter Values** tab and select the input action parameter value you specified for Action2.
 - Use the **Parameter** utility object to specify the action parameter as the *parameter* argument for the **RunAction** statement in the Expert View. For more information, see “Calling Actions with Parameters” on page 29.
- 6 Define an input action parameter for Action4 (**Edit > Action > Action Properties > Parameters** tab) with the same value type as the input test parameter.

- 7 Parameterize the input action parameter value.
 - ▶ Choose **Edit > Action > Action Call Properties > Parameter Values** tab and select the input action parameter value you specified for Action3.
 - ▶ Use the **Parameter** utility object to specify the action parameter as the *parameter* argument for the **RunAction** statement in the Expert View. For more information, see “Calling Actions with Parameters” on page 29.
- 8 Parameterize the value in the required step in Action4.
 - ▶ Click the parameterization icon  and specify the parameter in the Value Configuration Options dialog box using the input action parameter you specified for Action 4.
 - ▶ Use the **Parameter** utility object in the Expert View to specify the value to use for the step. For more information, refer to “Using Action Parameters in Steps in the Expert View” on page 360 in the *QuickTest Professional Basic Features User's Guide*.

An action’s parameters are stored with the action and are the same for all calls to that action. If you modify an action parameter’s name, type, or description, and then view the action properties for a call to that same action in a different part of the test, you will see that the action parameter has changed.

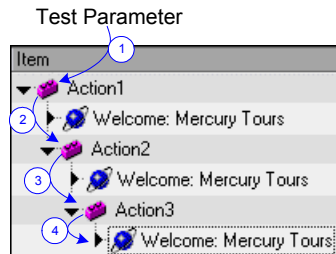
The actual value specified for an input action parameter and the location specified for action output parameter can be different for each call to the action. When you insert a call to a copy of an action, the copy of the action is inserted with the action parameters and action call parameter values that were defined for the action you copied. When you split an action, the action parameters are copied to both actions. The action call values for the second action are taken from the default values of that action’s parameters.

For information on defining action parameters and the values used in action calls, see “Setting Action Parameters” on page 12, and “Setting Action Call Parameter Values” on page 23.

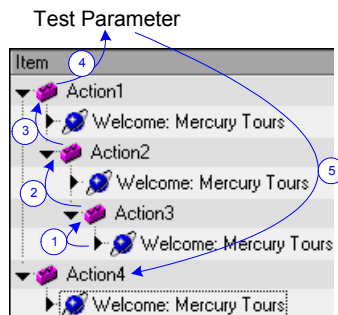
Guidelines for Working with Action Parameters

Consider the following guidelines when working with action parameters:

- ▶ Input action parameter values can be used only within the steps of the current action. You can use an action input value from another action (or from the test) only if you pass the value from action to action down the test hierarchy to the action in which you want to use it. For example: Test -> Action1 -> Action2 -> Action3 -> (Action3) Step 1

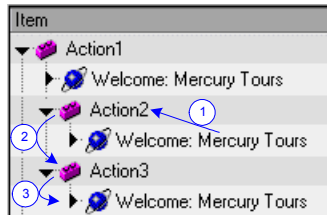


- ▶ Output action parameter values can be retrieved from a previous action at the same hierarchical level, from a parent action, or from the current action. You can use an action output value from one action within the step of another action if:
 - ▶ You pass the value from action to action up the test hierarchy to the action in which you want to use it. For example:
(Action3) Step 1 -> Action3 -> Action2 -> Action1 -> Test -> Action4



In this example, any step in Action 1, Action 2, or Action 3 can potentially use the output value from (Action3) Step 1, even though the example shows that the output value is used by steps in Action4.

- ▶ You pass the value from a previous action to the sibling action in which you want to use it. For example:
(Action2) Step 1 -> Action2 -> Action3 -> (Action3) Step 1



In this example, any step in Action 2 or Action 3 can potentially use the output value from (Action2) Step 1, even though the example shows that the output value is used by (Action3) Step 1.

- ▶ In subsequent steps of a calling action, you can use any type of action output value as a variable, if the value was retrieved from the called action. For example, if ActionA calls ActionB and specifies MyBVar as the variable in which to store ActionB's output parameter, then steps in ActionA after the call to ActionB can use the MyBVar as a value (just as you would use any other variable).

Setting Action Call Properties

The Action Call Properties dialog box controls the way the action behaves in a specific call to the action. It enables you to specify how many times QuickTest should run the called action (according to the number of rows in the Data Table), and also to specify the initial value for any input action parameters and the location in which you want to store the values of any output action parameters.

Note: The following sections describe how to define action call properties using the Action Call Properties dialog box. You can also define actions calls and action call parameters in the Expert View. For more information, see “Understanding Action Syntax in the Expert View” on page 29.

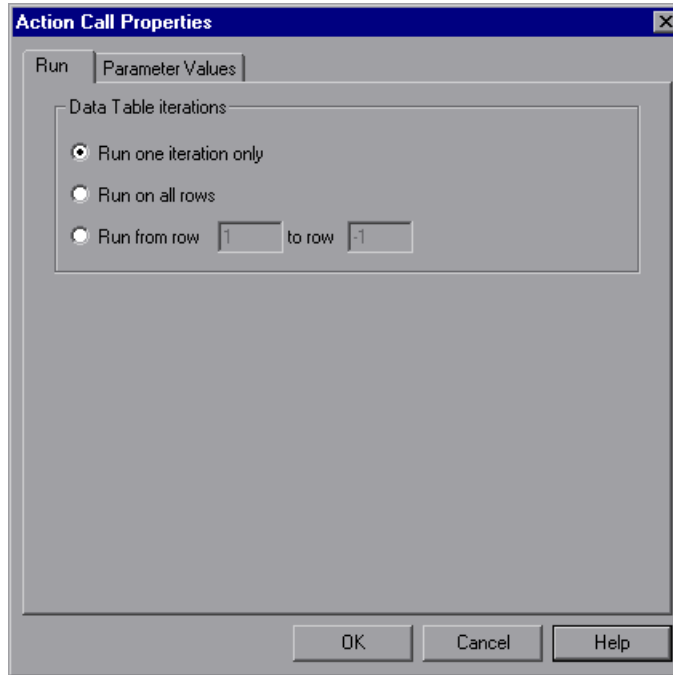
You can open the Action Call Properties dialog box while recording or editing your test by:

- ▶ Choosing **Edit > Action > Action Call Properties** from the Keyword View when an action node is highlighted.
- ▶ Right-clicking an action node in the Keyword View and selecting **Action Call Properties**.

The Action Call Properties dialog box enables you to set options that apply only to a specific action call. The dialog box contains both the Run tab and the Parameter Values tab.

Setting the Run Properties for an Action

You can use the Run tab of the Action Call Properties dialog box to instruct QuickTest to run only one iteration on the called action, to run iterations on all rows in the Data Table, or to run iterations only for a certain row range in the Data Table.



The Run tab includes the following options:

Option	Description
Run one iteration only	Runs the called action only once, using the first row in the action's data sheet.
Run on all rows	Runs the called action with the number of iterations according to the number of rows in the action's Data Table.
Run from row __ to row __	Runs the called action with the number of iterations according to the specified row range.

Notes:

If you run multiple iterations on an action, the action must begin and end at the same point in the application, so that the application is in the proper location and state to run the next iteration of the action.

The Run tab of the Action Call Properties dialog box applies to individual action calls and refers to the rows in the action's data sheet. You can set the Run properties for an entire test (setting iterations for rows on the Global data sheet) from the Run tab in the Test Settings dialog box. For more information, refer to Chapter 25, "Setting Options for Individual Tests" in the *QuickTest Professional Basic Features User's Guide*.

Setting Action Call Parameter Values

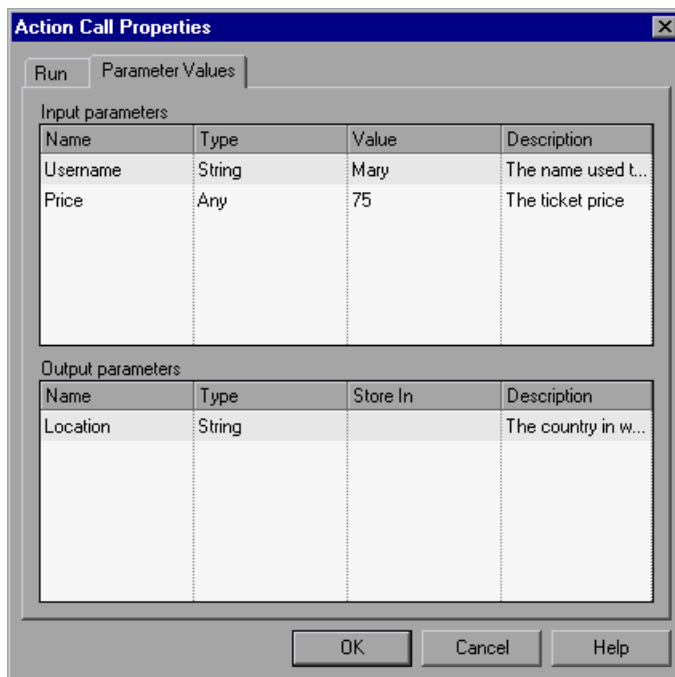
You use the Parameter Values tab of the Action Call Properties dialog box to specify the values of input action parameters used by the called action and to specify the locations in which you want to store output action parameter values. You can also parameterize the value used for a particular input action parameter using any available parameter type.

The actual input and output action parameters that an action can receive or return, and their types, are defined in the Action Properties dialog box.

Note: Specifying input and output parameter values in action calls is optional.

If you do not set a value for an input action parameter, the default value that is specified in the Action Properties dialog box is used.


If you do not define a storage location for an output parameter value, the calling action still has access to the output parameter data generated by the actions it calls. However, specifying a storage location can make your action call statements more readable.



For more information about defining input and output action parameters, see “Setting Action Call Properties” on page 21. For general information about using action parameters, see “Using Action Parameters” on page 16.

To specify the value for an input action parameter:


- 1 In the **Input parameters** area, click in the **Value** box for the parameter and enter a value. For a description of the different options available for each value type, see the definitions included in “Setting Action Parameters” on page 12.

Alternatively, you can click the parameterization button  in the **Value** box to open the Value Configuration Options dialog box in which you can parameterize the value. You can parameterize the value using a test or action parameter (test parameter for a top-level action, or action parameter for a nested or sibling action), Data Table parameter, environment parameter, or random number parameter. For more information, refer to Chapter 15, “Parameterizing Values” in the *QuickTest Professional Basic Features User’s Guide*.

- 2 Repeat this procedure for any additional input action parameter values you want to set.

To specify a location in which to store an output action parameter value:

- 1 In the **Output parameters** area, click in the **Store In** box for the parameter and enter a variable name.

Alternatively, you can click the output storage button  in the **Store In** box to open the Storage Location Options dialog box in which you can specify a location for storing the output value. You can select to store the value in a test parameter, the calling action parameter, a Data Table parameter, or an environment parameter. For more information, refer to “Sharing Action Information” on page 26 and “Storing Return Values and Action Output Parameter Values” on page 544 in the *QuickTest Professional Basic Features User’s Guide*.

- 2 Repeat this procedure for each output action parameter value in the list.

Sharing Action Information

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

- ▶ Store values in the output action parameters of a called action and use those values in steps that are performed after the action call within the calling action, or in steps within sibling actions. For more information, refer to “Storing Values in Test and Action Parameters” on page 398 in the *QuickTest Professional Basic Features User’s Guide*.
- ▶ Store values from one action in the global Data Table and use these values as Data Table parameters in other actions. For more information, see “Sharing Values via the Global Data Table,” below.
- ▶ Set a value from one action as a user-defined environment variable and then use the environment variable in other actions. For more information, see “Sharing Values Using Environment Variables” on page 27.
- ▶ Add values to a Dictionary object in one action and retrieve the values in other actions. For more information, see “Sharing Values Using the Dictionary Object” on page 28.

Sharing Values via the Global Data Table

You can share a value that is generated in one action with other actions in your test by storing the value in the global Data Table. Other actions can then use the value in the Data Table as an input parameter. You can store a value in the Data Table by outputting the value to the global Data Table or by using **DataTable**, **Sheet** and **Parameter** objects and methods in the Expert View to add or modify a value.

For example, suppose you are testing a flight reservation application. When a user logs into the application, his or her full name is displayed on the top of the page. Later, when the user chooses to purchase the tickets, the user must enter the name that is listed on his or her credit card. Suppose your test contains three actions—Login, SelectFlight, and PurchaseTickets and the test is set to run multiple iterations with a different login name for each iteration. In the Login action, you can create a text output value to store the displayed name of the user. In the PurchaseTickets action, you can parameterize the value that is set in the Credit Card Owner edit box using the Data Table column containing the user’s full name.

For more information on output values, refer to Chapter 16, “Outputting Values” in the *QuickTest Professional Basic Features User’s Guide*. For more information on parameterization, refer to Chapter 15, “Parameterizing Values” in the *QuickTest Professional Basic Features User’s Guide*. For more information about DataTable objects and methods, refer to Chapter 19, “Working with Data Tables” in the *QuickTest Professional Basic Features User’s Guide*, and to the *QuickTest Professional Object Model Reference*.

Sharing Values Using Environment Variables

If you don’t need to run multiple iterations of your test or you want the value you are sharing to stay constant for all iterations, you can use an internal, user-defined environment variable that can be accessed by all local actions in your test.

For example, suppose you want to test that your flight reservation application correctly checks the credit card expiration date that the user enters. The application should request a different credit card if the expiration date that was entered is earlier than the scheduled flight departure date. In the SelectFlight action, you can store the value entered in the departure date edit box in an environment variable. In the PurchaseTickets action, you can compare the value of the expiration date edit box with the value stored in your environment variable.

For more information on environment variables, refer to Chapter 15, “Parameterizing Values” in the *QuickTest Professional Basic Features User’s Guide*. For information on the **Environment** object, refer to the *QuickTest Professional Object Model Reference*.

Sharing Values Using the Dictionary Object

As an alternative to using environment variables to share values between actions as described above, you can use the Dictionary object. The Dictionary object enables you to assign values to variables that are accessible from all actions (local and external) called in the test in which the Dictionary object is created.

To use the Dictionary object, you must first add a reserved object to the registry (in **HKEY_CURRENT_USER\Software\Mercury Interactive\QuickTest Professional\MicTest\ReservedObjects**) with ProgID = "Scripting.Dictionary". For example:

```
HKEY_CURRENT_USER\Software\Mercury Interactive\QuickTest  
Professional\MicTest\ReservedObjects\GlobalDictionary
```

After you have added the reserved Dictionary object to the registry and restarted QuickTest, you can add and remove values to the Dictionary in one action and retrieve the values in another action from the same test.

For example, if you want to access the departure date set in the SelectFlight action from the PurchaseTickets action, you can add the value of the DepartDate WebEdit object to the dictionary in the SelectFlight action as follows:

```
GlobalDictionary.RemoveAll  
GlobalDictionary.Add "DateCheck", DepartDate
```

Then you can retrieve the date from the PurchaseTickets action as follows:

```
Dim CompareDate  
CompareDate=GlobalDictionary("DateCheck")
```

For more information about the Dictionary object, refer to the VBScript Reference documentation (**Help > QuickTest Professional Help > VBScript Reference > Script Runtime**).

Understanding Action Syntax in the Expert View

An action call in the expert view can define the action iterations, input parameter values, output parameter storage locations, and an action return values.

Calling Actions Using Basic Syntax

In the Expert View, a call to an action with no parameters is displayed within the calling action with the following basic syntax:

RunAction *ActionName*, *IterationQuantity*

For example, to call the **Select Flight** action and run it one iteration:

RunAction "Select Flight", oneIteration

For example, to call the **Select Flight** action and run it as many iterations as there are rows in the Data Table:

RunAction "Select Flight", allIterations

For example, to call the **Select Flight** action and run it four iterations (for the first four rows of the Data Table):

RunAction "Select Flight", "1 - 4"

Calling Actions with Parameters

If the action you are calling has input and/or output parameters defined, you can also supply the values for the input parameters and the storage location of the output parameters as arguments of the **RunAction** statement. Input parameters are listed before output parameters.

For an input parameter, you can specify either a fixed value or you can specify the name of another defined parameter (Data Table parameter, environment parameter, or an action input parameter of the calling action) from which the argument should take its value.

For an output parameter, you can specify either a variable in which you want to store the value or the name of a defined parameter (Data Table parameter, environment parameter, or an action output parameter of the calling action).

An action call with parameters has the following syntax:

RunAction *ActionName*, *IterationQuantity*, *Parameters*

For example, suppose you call Action2 from Action1, and Action2 has one input and one output parameter defined.

The following statement supplies a string value of MyValue for the input parameter and stores the resulting value of the output parameter in a variable called MyVariable.

```
RunAction "Action2", oneliteration, "MyValue", MyVariable
```

The following statement uses the value defined for Action1's Axn1_In input action parameter as the value for the input parameter, and stores the resulting value of the output parameter in Action1's Data Table sheet in a column called Column1_out.

```
RunAction "Action2", oneliteration, Parameter("Axn1_In"),  
    DataTable("Column1_out", dtLocalSheet)
```

In the following example, the first statement calls Action2 using its default input parameter value. The second statement uses the value defined for Action2's Axn2_out output action parameter as the value for the call to Action 3's input parameter, and stores the resulting value of the output parameter in Action1's Axn1_out so that the output value is available at the parent action level.

```
RunAction "Action2", oneliteration  
RunAction "Action3", oneliteration, Parameter("Action2", "Axn2_out"),  
    Parameter("Axn1_out")
```

Note that the Action2 output parameter is available for use in the call to Action3, even though no storage location is specified in the call to Action2.

Storing Action Return Values

If the action called by the **RunAction** statement includes an **ExitAction** statement, the **RunAction** statement can return the value of the **ExitAction**'s *RetVal* argument. Note that this return value is a return value of the action call itself and is independent of any values returned by specific output parameters of the action call.

To store the return value of an action call, use the syntax:

MyRetVal=**RunAction** (*ActionName*, *IterationQuantity*, *Parameters*)

For more information about the Expert View, see Chapter 5, “Working with the Expert View and Function Library Windows.” For more information on the **RunAction** statement, refer to the *QuickTest Professional Object Model Reference*.

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. You may want to use this option to return the current value of the action to the value at a specific point in the run or based on the result of a conditional statement. 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.

You can view the exit action node in the Test Results tree. If your exit action statement returns a value, the value is displayed in the action, iteration, or test summary, as applicable.

For more information about these functions, refer to the *QuickTest Professional Object Model Reference*. For more information about the Test Results, refer to Chapter 23, “Analyzing Test Results” in the *QuickTest Professional Basic Features User’s Guide*.

2

Learning Virtual Objects

You can teach QuickTest 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 QuickTest.

This chapter describes:

- ▶ About Learning Virtual Objects
- ▶ Understanding Virtual Objects
- ▶ Understanding the Virtual Object Manager
- ▶ Defining a Virtual Object
- ▶ Removing or Disabling Virtual Object Definitions

About Learning Virtual Objects

Your application may contain objects that behave like standard objects but are not recognized by QuickTest. You can define these objects as virtual objects and map them to standard classes, such as a button or a check box. QuickTest emulates the user's action on the virtual object during the run session. In the test results, the virtual object is displayed as though it is a standard class object.

For example, suppose you want to record a test on a Web page containing a bitmap that the user clicks. The bitmap contains several different hyperlink areas, and each area opens a different destination page. When you record a test, the Web site matches the coordinates of the click on the bitmap and opens the destination page.

To enable QuickTest to click at the required coordinates during a run session, you can define a virtual object for an area of the bitmap, which includes those coordinates, and map it to the button class. When you run a test, QuickTest clicks the bitmap in the area defined as a virtual object so that the Web site opens the correct destination page.

You define a virtual object using the Virtual Object Wizard (**Tools > Virtual Objects > New Virtual Object**). The wizard prompts you to select the standard object class to which you want to map the virtual object. You then mark the boundaries of the virtual object using a crosshairs pointer. Next, you select a test object as the parent of the virtual object. Finally, you specify a name and a collection for the virtual object. A virtual object *collection* is a group of virtual objects that is stored in the Virtual Object Manager under a descriptive name.

Note: QuickTest does not support virtual objects for analog or low-level recording. For additional information about low-level recording, see “Recording and Running Tests” on page 430.

Understanding Virtual Objects

QuickTest identifies a virtual object according to its boundaries. Marking an object's boundaries specifies its size and position on a Web page or application window. When you assign a test object as the parent of your virtual object, you specify that the coordinates of the virtual object boundaries are relative to that parent object. When you record a test, QuickTest recognizes the virtual object within the parent object and adds it as a test object in the object repository so that QuickTest can identify the object during the run session. QuickTest also recognizes the virtual object as a test object when you add it manually to the object repository.

Note: During a run session, make sure that the application window is the same size and in the same location as it was during recording, otherwise the coordinates of the virtual object relative to its parent object may be different, and this may affect the success of the run session.

You can disable recognition of virtual objects without deleting them from the Virtual Object Manager. For additional information, see “Removing or Disabling Virtual Object Definitions” on page 42.

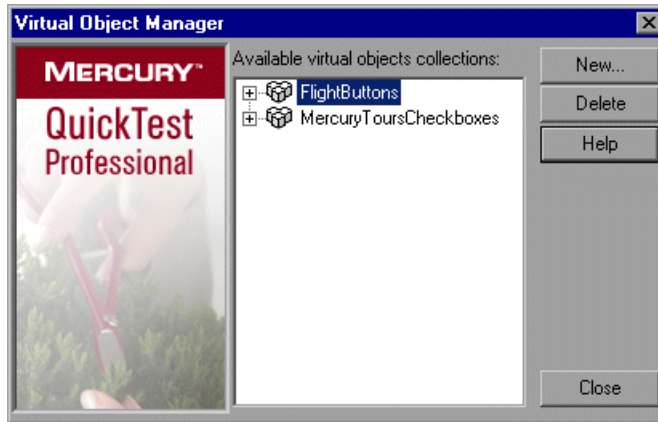
Notes:

You can use virtual objects only when recording and running a test. You cannot insert any type of checkpoint on a virtual object, or use the Object Spy to view its properties.

In order to perform an operation in the Active Screen on a marked virtual object, you must first record it, so that its properties are saved in the test object description in the object repository. If you perform an operation in the Active Screen on a virtual object that has not yet been recorded, QuickTest treats it as a standard object.

Understanding the Virtual Object Manager

The Virtual Object Manager contains all the virtual object collections defined on your computer. From the Virtual Object Manager, you can define and delete virtual objects and collections.



Available virtual object collections list—Displays the virtual object collections defined on your computer and the virtual objects contained in each one. Use the + and - signs next to a collection to view or hide the virtual objects defined in that collection.

New—Opens the Virtual Object Wizard, which guides you through the process of defining a new virtual object for a new or existing collection. For more information, see “Defining a Virtual Object” on page 37.

Delete—Deletes the selected virtual object or virtual object collection. For more information, see “Removing or Disabling Virtual Object Definitions” on page 42.

Note: The virtual object collections displayed in the Virtual Object Manager are stored on your computer and not with the tests that contain virtual object steps. This means that if you use a virtual object in a test step, the object will be recognized during the run session only if it is run on a computer containing the appropriate virtual object definition. To copy your virtual object collection definitions to another computer, copy the contents of your <QuickTest installation folder>\dat\VoTemplate folder (or individual .vot collection files within this folder) to the same folder on the destination computer.

Defining a Virtual Object

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 name. You can also group your virtual objects logically by assigning them to collections.

Note: You can define virtual objects only for objects on which you can click or double-click and that record a **Click** or **DbClick** step. Otherwise, the virtual object is ignored. For example, if you define a virtual object over the WinList object, the **Select** operation is recorded, and the virtual object is ignored.

To define a virtual object:

- 1** With QuickTest 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.
- 2** In QuickTest, choose **Tools > Virtual Objects > New Virtual Object**. Alternatively, from the Virtual Object Manager, click **New**. The Virtual Object Wizard opens.



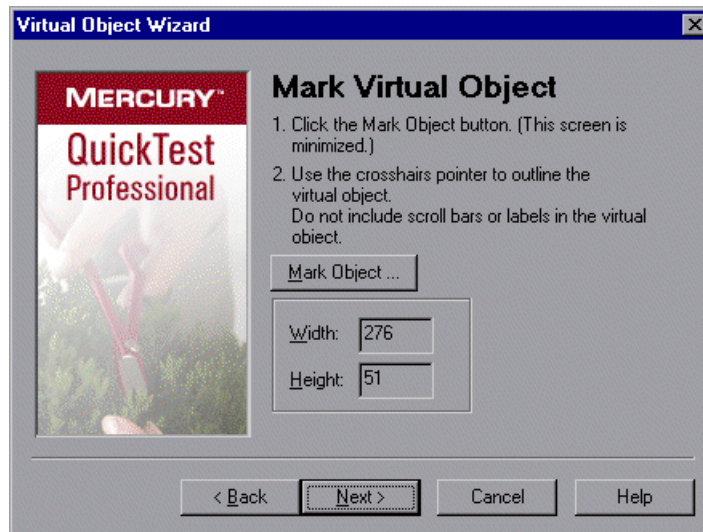
Click **Next**.

- 3 Select a standard class to which you want to map your virtual object.



If you select the list class, specify the number of rows in the virtual object. For the table class, select the number of rows and columns. Click **Next**.

- 4 Click **Mark Object**.

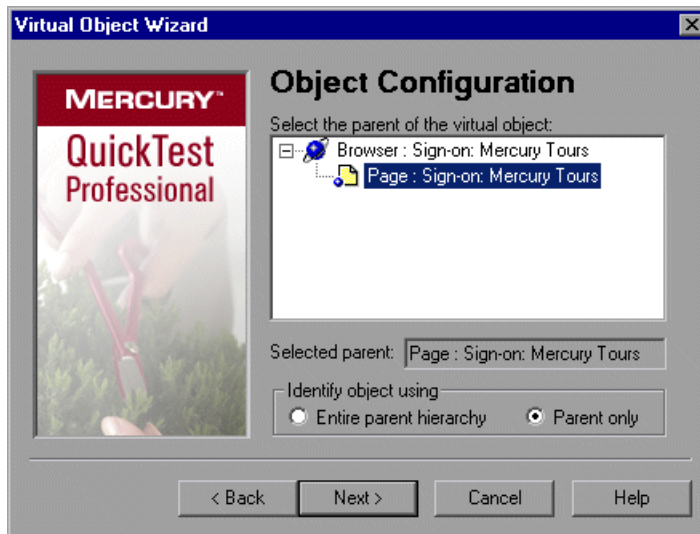


The QuickTest window and the Virtual Object Wizard are minimized. Use the crosshairs pointer to mark the area of the virtual object. You can use the arrow keys while holding down the left mouse button to make precise adjustments to the area you define with the crosshairs.

Click **Next**.

Note: The virtual object should not overlap other virtual objects in your application or Web page. If the virtual object overlaps another virtual object, QuickTest may not record or run tests correctly on the virtual objects.

- 5 Click an object in the object tree to assign it as the parent of the virtual object.



The coordinates of the virtual object outline are relative to the parent object you select.

- 6** In the **Identify object using** box, select how you want QuickTest to identify and map the virtual object.
- ▶ If you want QuickTest to identify all occurrences of the virtual object, select **parent only**. QuickTest identifies the virtual object using its direct parent only, regardless of the entire parent hierarchy. For example, if the virtual object was defined using `Browser("A").Page("B").Image("C")`, QuickTest will recognize the virtual object even if the hierarchy changes to `Browser("X").Page("Y").Image("C")`.
 - ▶ If you want QuickTest to identify the virtual object in one occurrence only, select **entire parent hierarchy**. QuickTest identifies the virtual object only if it has the exact parent hierarchy. For example, if the virtual object was defined using `Browser("A").Page("B").Image("C")`, QuickTest will not recognize it if the hierarchy changes to `Browser("X").Page("B").Image("C")`.

Click **Next**.

- 7** 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.



8 Perform one of the following:

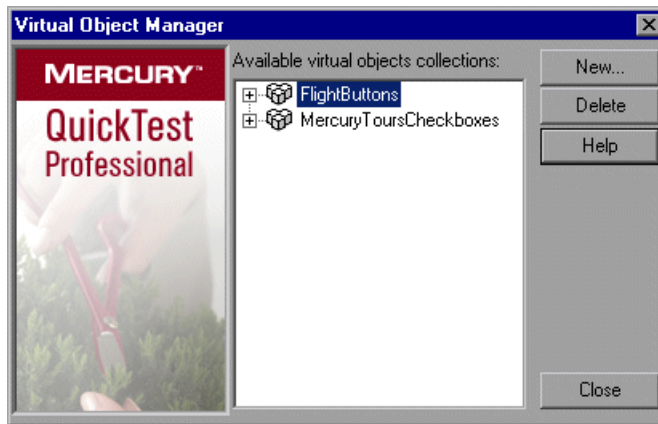
- ▶ To add the virtual object to the Virtual Object Manager and close the wizard, select **No** and then click **Finish**.
- ▶ To add the virtual object to the Virtual Object Manager and define another virtual object, select **Yes** and then click **Next**. The wizard returns to the Map to a Standard Class screen, where you can define the next virtual object.

Removing or Disabling Virtual Object Definitions

You can remove virtual objects from your test by deleting them or by disabling recognition of these objects while recording.

To delete a virtual object:

- 1** Choose **Tools > Virtual Objects > Virtual Object Manager**. The Virtual Object Manager opens.



- 2** In the list of available virtual object collections, click the plus sign next to the collection to display the virtual object you want to delete. Select the virtual object, and click **Delete**.

To delete an entire collection, select it and click **Delete**.

3 Click Close.

Tip: Click **New** in the Virtual Object Manager to open the Virtual Object Wizard, where you can define a new virtual object.

To disable recognition of virtual objects while recording:

- 1** Choose **Tools > Options** or click the **Options** toolbar button. The Options dialog box opens.
- 2** In the General tab, select the **Disable recognition of virtual objects while recording** check box.
- 3** Click **OK**.

Note: When you want QuickTest to recognize virtual objects during recording, ensure that the **Disable recognition of virtual objects while recording** check box in the General tab of the Options dialog box is cleared. For more information, refer to “Setting General Testing Options” on page 696 in the *QuickTest Professional Basic Features User’s Guide*.

3

Defining and Using Recovery Scenarios

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

This chapter describes:

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

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 unattended—the test is suspended until you perform the operation needed to recover. For information on when to use recovery scenarios, see “Deciding When to Use Recovery Scenarios” on page 47.

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.

A recovery scenario consists of the following:

- ▶ **Trigger Event**—The event that interrupts your run session. For example, a window that may pop up on screen, or a QuickTest run error.
- ▶ **Recovery Operation(s)**—The operation(s) that need to be performed in order to continue running the test. For example, clicking an **OK** button in a pop-up window, or restarting Microsoft Windows.
- ▶ **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 QuickTest should continue, if at all. For example, you may want to restart a test from the beginning, or skip a step entirely and continue with the next step in the test.

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 the recovery scenario with that test. A test can have any number of recovery scenarios associated with it. You can prioritize the scenarios associated with your test to ensure that trigger events are recognized and handled in the required order. For more information, see “Adding Recovery Scenarios to Your Test” on page 85.

When you run a test 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. For more information, see “Programmatically Controlling the Recovery Mechanism” on page 91.

Note: If you choose **On error** in the **Activate recovery scenarios** box in the Recovery tab of the Test Settings dialog box, the recovery mechanism does not handle triggers that occur in the last step of a test. 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, you can do this by adding an extra step to the end of your test.

Deciding When to Use Recovery Scenarios

If you can predict that a certain event may happen at a specific point in your test, it is highly recommended to handle that event directly within your test 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. 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, since you cannot know at what point the network will return the printer error. You could try to handle this event in your test 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 may have progressed several steps before the error is displayed. Therefore, for this type of event, only a recovery scenario can handle it.

For more information on optional steps, refer to “Using Optional Steps” on page 613 in the *QuickTest Professional Basic Features User’s Guide*. For more information on inserting programming statements such as **If** statements, refer to Chapter 20, “Adding Steps Containing Programming Logic” in the *QuickTest Professional Basic Features User’s Guide*.

Defining Recovery Scenarios

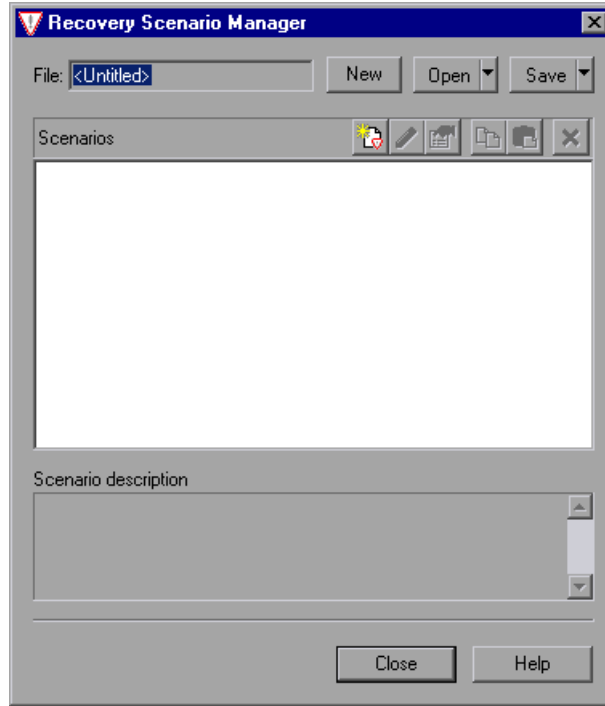
The Recovery Scenario Manager dialog box enables you to create recovery scenarios and save them in recovery files. You create recovery scenarios using the Recovery Scenario Wizard, which leads you through the process of defining each of the stages of the recovery scenario. You then save the recovery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

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 **Resources > Recovery Scenario Manager**. The Recovery Scenario Manager dialog box opens.



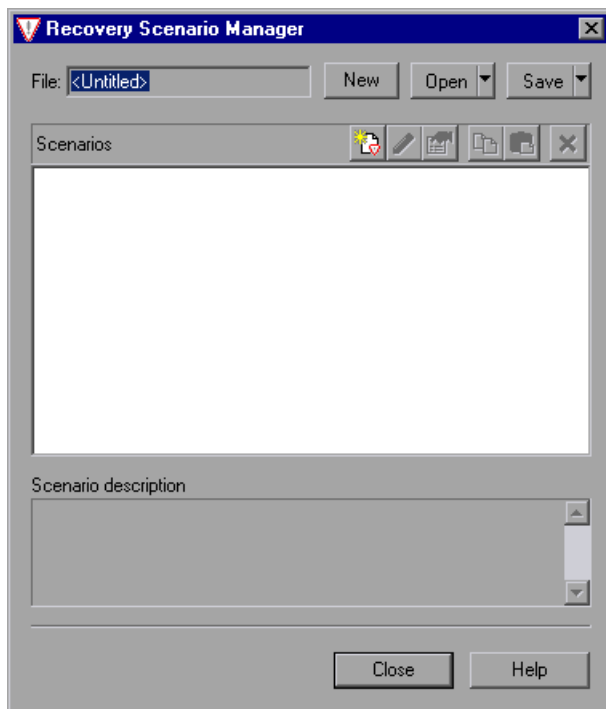
- 2 By default, the Recovery Scenario Manager dialog box opens with a new recovery file. You can either use this new file, or click the **Open** button to choose an existing recovery file. Alternatively, you can click the arrow next to the **Open** button to select a recently-used recovery file from 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.










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.



The Recovery Scenario Manager dialog box contains the following toolbar buttons:

Option	Description
	Creates a new recovery file. For more information, see “Creating a Recovery File” on page 48.
	Opens an existing recovery file. You can also click the arrow to select a recovery file from the list of recently-used recovery files.
	Saves the current recovery file. For more information, see “Saving the Recovery Scenario in a Recovery File” on page 78.
	Opens the Recovery Scenario Wizard, in which you define a new recovery scenario. For more information, see “Understanding the Recovery Scenario Wizard” on page 52.
	Opens the Recovery Scenario Wizard for the selected recovery scenario, in which you can modify the recovery scenario settings. For more information, see “Modifying Recovery Scenarios” on page 82.
	Displays summary properties for the selected recovery scenario in read-only format. For more information, see “Viewing Recovery Scenario Properties” on page 81.
	Copies a recovery scenario from the open recovery file to the Clipboard. This enables you to paste a recovery scenario into another recovery file. For more information, see “Copying Recovery Scenarios between Recovery Scenario Files” on page 83.
	Pastes a recovery scenario from the Clipboard into the open recovery file. For more information, see “Copying Recovery Scenarios between Recovery Scenario Files” on page 83.
	Deletes a recovery scenario. For more information, see “Deleting Recovery Scenarios” on page 82.

Note: Each recovery scenario is represented by an icon that indicates its type. For more information, see “Managing Recovery Scenarios” on page 79.

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:

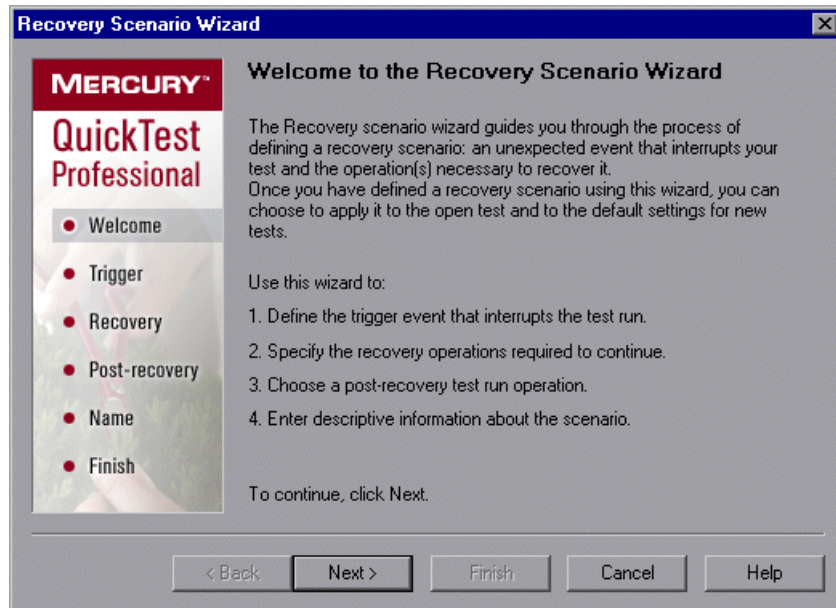
- ▶ defining the trigger event that interrupts the run session
- ▶ specifying the recovery operation(s) required to continue
- ▶ choosing a post-recovery test run operation
- ▶ specifying a name and description for the recovery scenario
- ▶ specifying whether to associate the recovery scenario to the 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 (**Resources > Recovery Scenario Manager**).

Welcome to the Recovery Scenario Wizard Screen

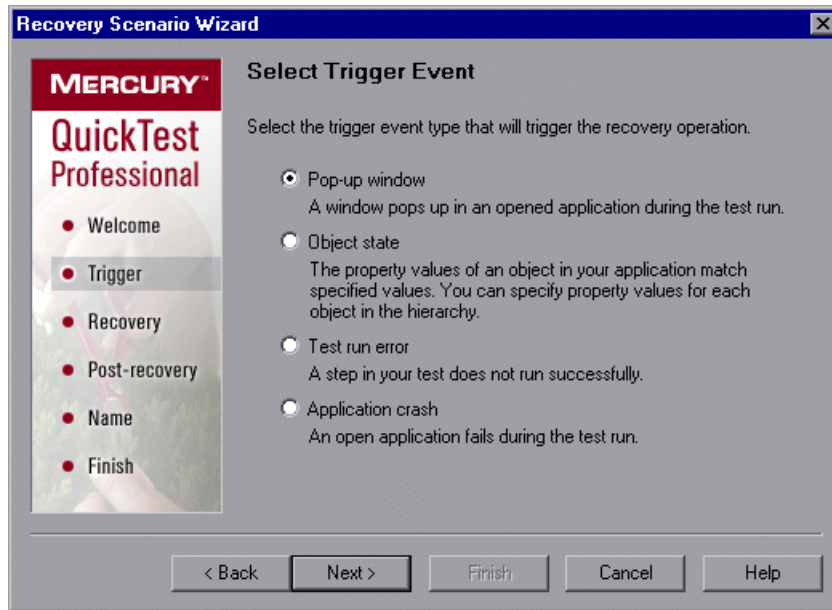
The Welcome to the Recovery Scenario Wizard screen provides general information about the different options in the Recovery Scenario Wizard, and provides an overview of the stages involved in defining a recovery scenario.



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

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:

- **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.

- ▶ **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. 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.

- ▶ **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.

- ▶ **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.

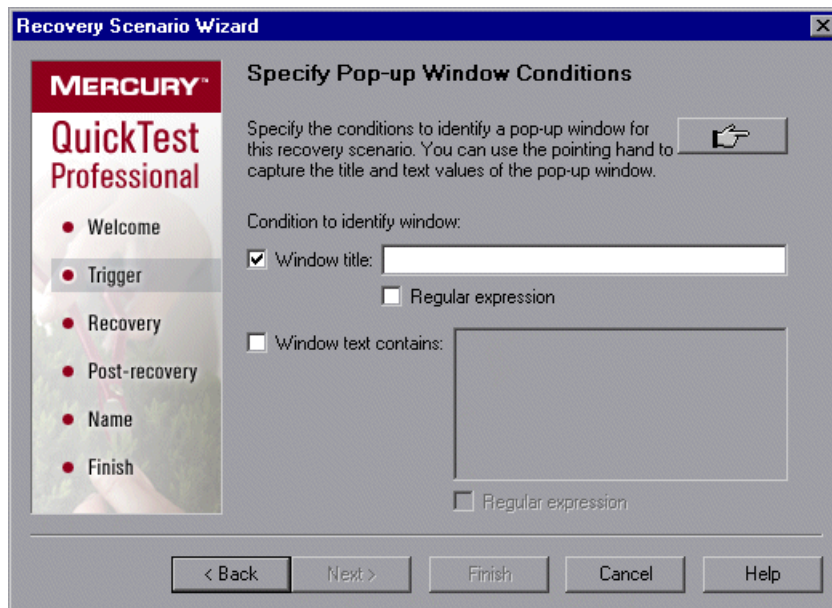
Notes:

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

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.



Perform one of the following to specify how the pop-up window should be identified:

- ▶ Choose whether you want to identify the pop-up window according to its **Window title** and/or **Window text** and then enter the text used to identify the pop-up window. You can 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. For information on regular expressions, refer to “Understanding and Using Regular Expressions” on page 334 in the *QuickTest Professional Basic Features User's Guide*.
- ▶ Click the pointing hand and then click the pop-up window to capture the window title and textual content of the window.

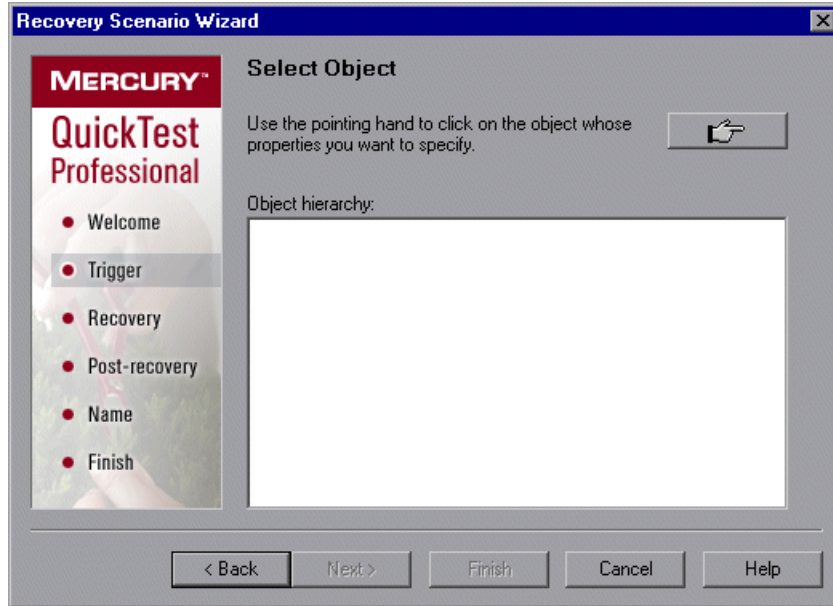
Note: Using the first option (**Window title** and/or **Window text**) instructs QuickTest to identify any pop-up window that contains the relevant title and/or text. Using the second option (pointing hand) instructs QuickTest to identify only pop-up windows that match the object property values of the window you select.

Tip: Hold the left CTRL key to change the window focus or perform operations such as a right-click or mouseover to display a context menu. If the window containing the object you want to select is minimized, you can display it by holding the left CTRL key, right-clicking the application in the Windows task bar, and choosing **Restore** from the context menu.

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

Select Object Screen

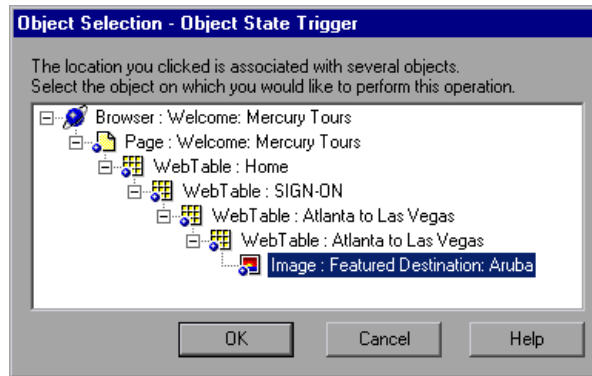
If you chose an **Object state** trigger in the Select Trigger Event screen, the Select Object screen opens.



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

Tip: Hold the left CTRL key to change the window focus or perform operations such as a right-click or mouseover to display a context menu. If the window containing the object you want to select is minimized, you can display it by holding the left CTRL key, right-clicking the application in the Windows task bar, and choosing **Restore** from the context menu.

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



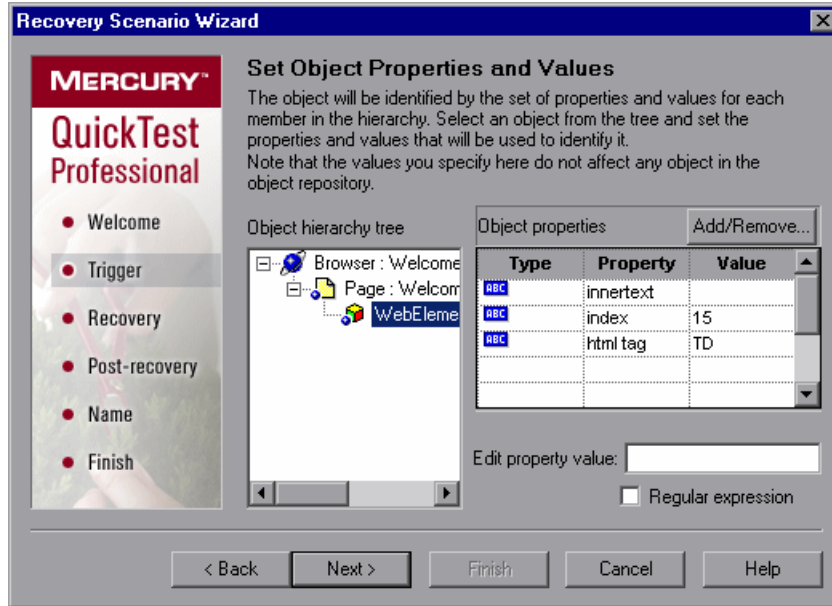
Select the object whose properties you want to specify and click **OK**. The selected object and its parents are displayed in the Select Object screen.

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.

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.



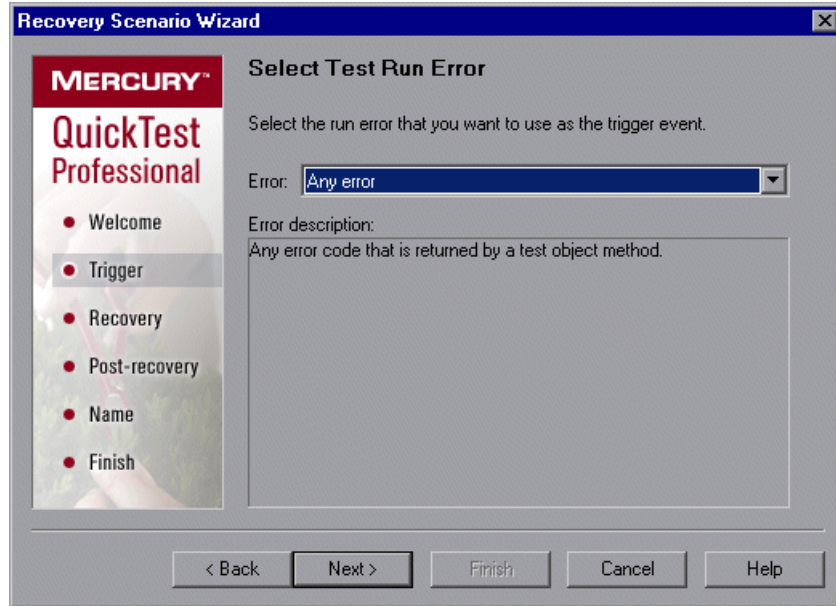
For each object in the hierarchy, in the **Edit property value** box, you can modify the property values used to identify the object. You can also click the **Add/Remove** button to add or remove object properties from the list of property values to check. Note that an object is identified only by its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

Select the **Regular expression** check box if you want to use regular expressions in the property value. For information on regular expressions, refer to “Understanding and Using Regular Expressions” on page 334 in the *QuickTest Professional Basic Features User’s Guide*.

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

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:

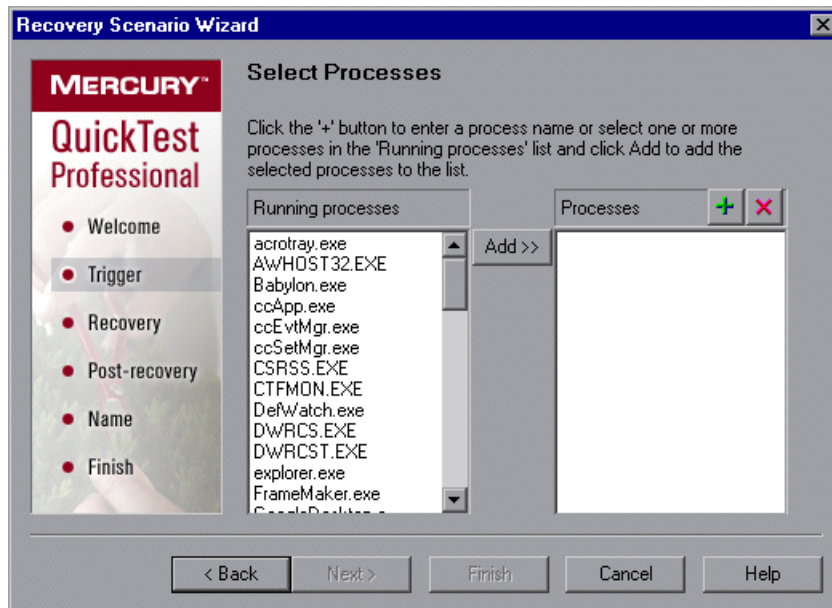
- **Any error**—Any error code that is returned by a test object method.
- **Item in list or menu is not unique**—Occurs when more than one item in the list, menu, or tree has the name specified in the method argument.
- **Item in list or menu not found**—Occurs when QuickTest cannot identify the list, 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.
- **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.

- ▶ **Object is disabled**—Occurs when QuickTest cannot perform the step because the object specified in the step is currently disabled.
- ▶ **Object not found**—Occurs when no object within the specified parent object matches the test object description for the object.
- ▶ **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.

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.



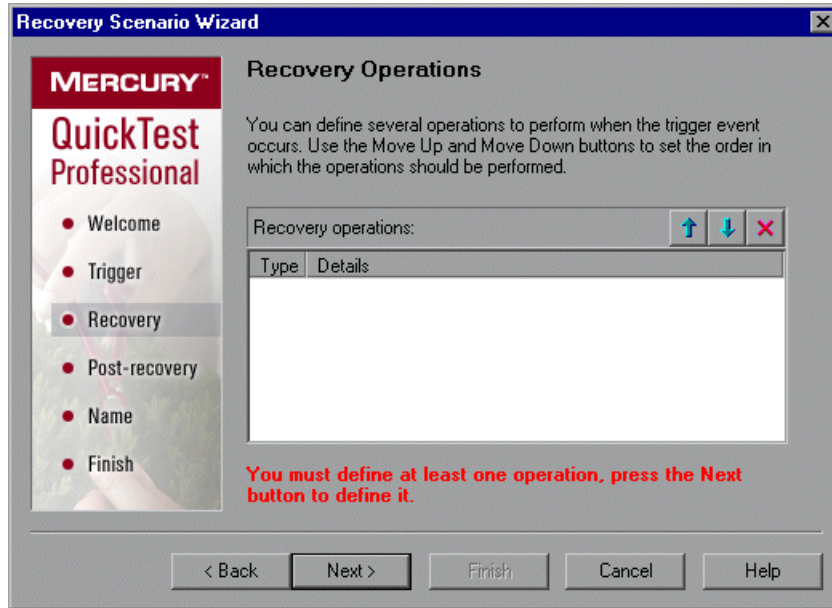
To remove a process from the **Processes** list, select it and click the **Remove Process** button.

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.

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.

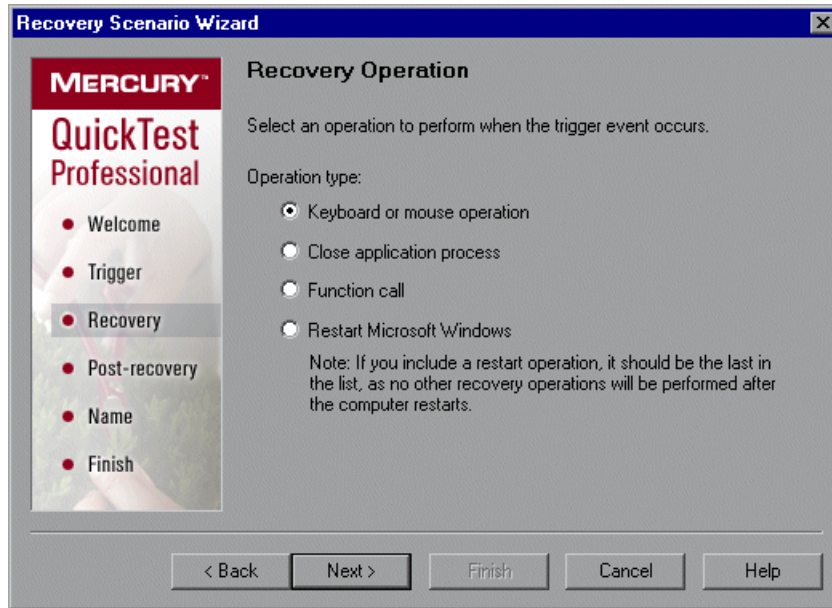
Note: If you define a **Restart Microsoft Windows** recovery operation, it is always inserted as the last recovery operation, and you cannot change its position in the list.

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

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

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:

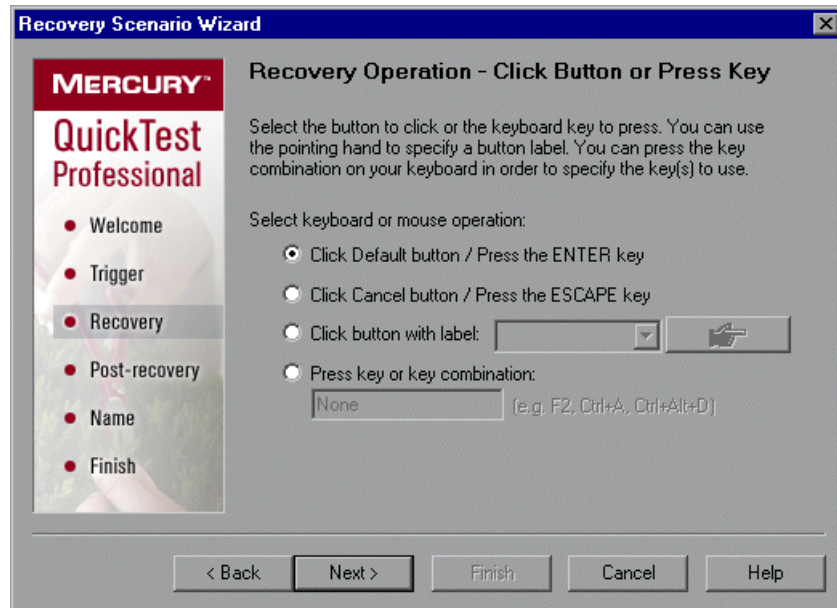
- ▶ **Keyboard or mouse operation**—QuickTest simulates a click on a button in a window or a press of a keyboard key. Select this option and click **Next** to continue to the Recovery Operation – Click Button or Press Key screen.
- ▶ **Close application process**—QuickTest closes specified processes. Select this option and click **Next** to continue to the Recovery Operation – Close Processes screen.

- ▶ **Function call**—QuickTest calls a VBScript function. Select this option and click **Next** to continue to the Recovery Operation – Function Call screen.
- ▶ **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 associated with this recovery scenario is saved before you run it. You must also configure the computer on which the test is run to automatically log in on restart.

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:

- ▶ **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.
- ▶ **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.
- ▶ **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 left CTRL key to change the window focus or perform operations such as a right-click or mouseover to display a context menu. If the window containing the object you want to select is minimized, you can display it by holding the left CTRL key, right-clicking the application in the Windows task bar, and choosing **Restore** from the context menu.

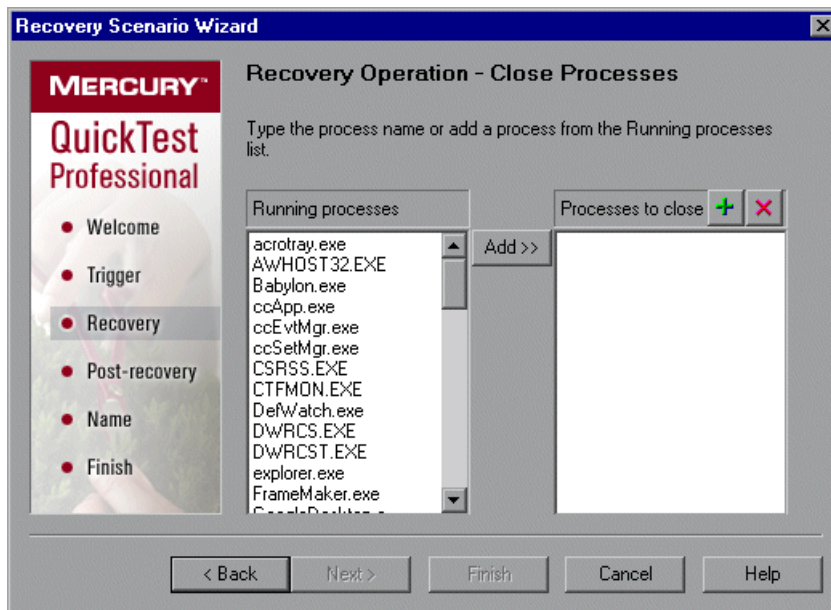
All button labels in the selected window are displayed in the list box. Select the required button from the list.

- ▶ **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.

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.

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.



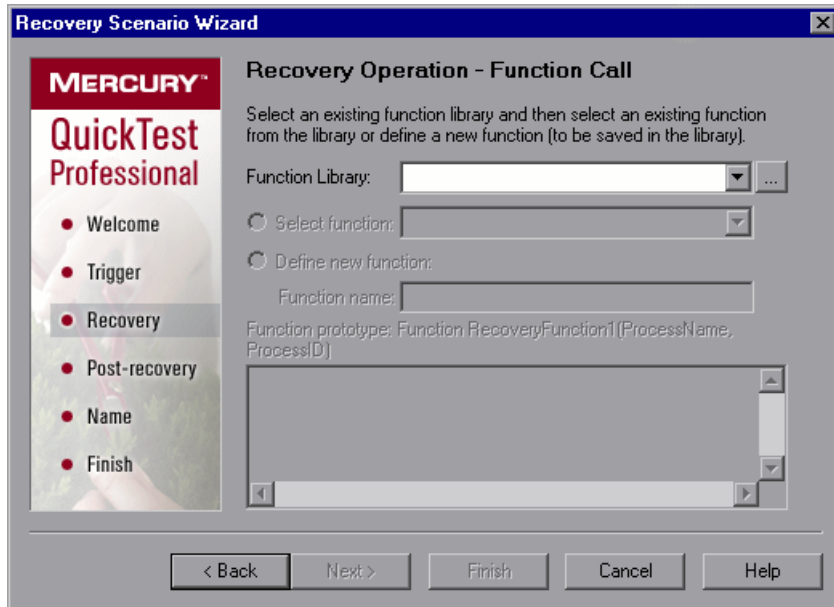
To remove a process from the **Processes to close** list, select it and click the **Remove Process** button.

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.

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 function library in the **Function Library** box. Alternatively, click the browse button to navigate to an existing function library.

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

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

- **Select function**—Choose an existing function from the function library 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.  
)
```

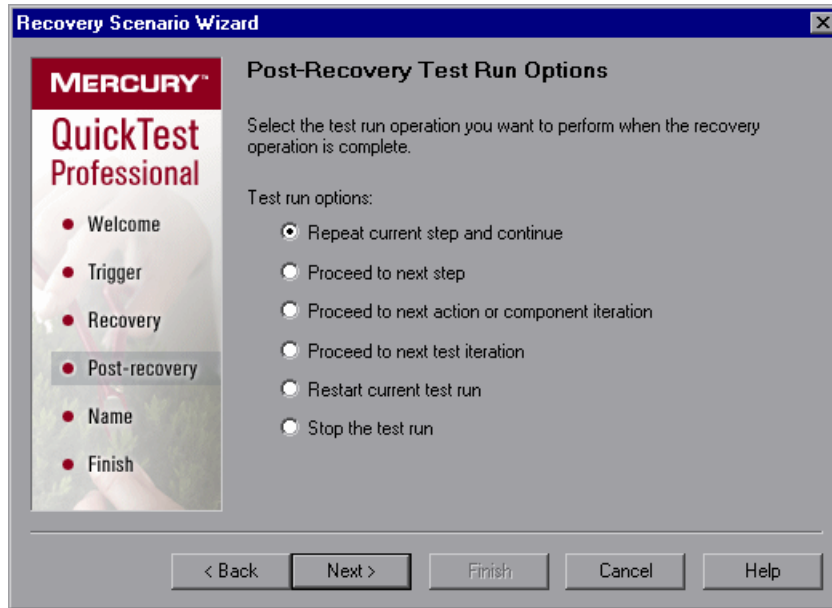
- **Define new function**—Create a new function by specifying a unique name for it, and defining the function in the **Function Name** box according to the displayed function prototype. The new function is added to the function library you selected.

Note: If more than one scenario uses a function with the same name from different function libraries, 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.

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:

► **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. For more information, see “Enabling and Disabling Recovery Scenarios” on page 89.

➤ **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.

➤ **Proceed to next action or component 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).

➤ **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).

➤ **Restart current test run**

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

➤ **Stop the test run**

Stops running the test.

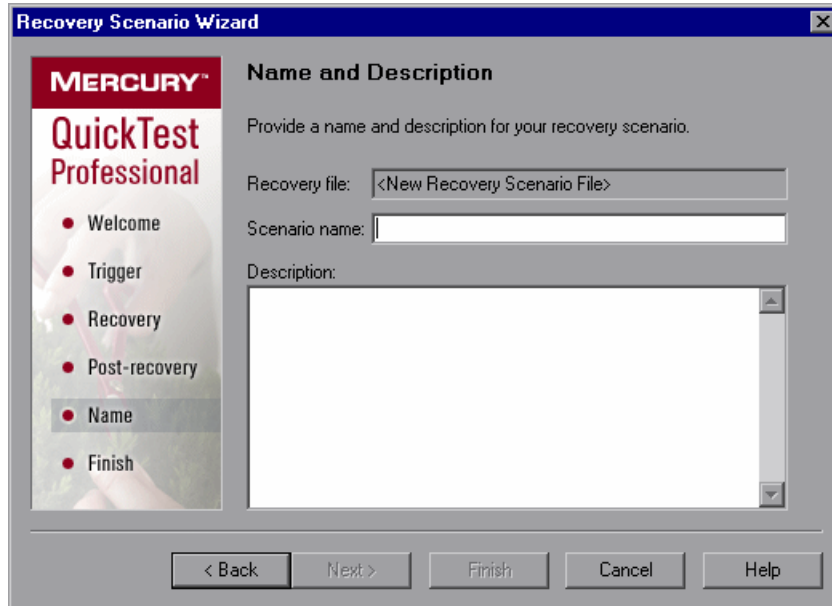
Note: If you chose **Restart Microsoft Windows** as a recovery operation, you can choose from only the last two test run options listed above.

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

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.

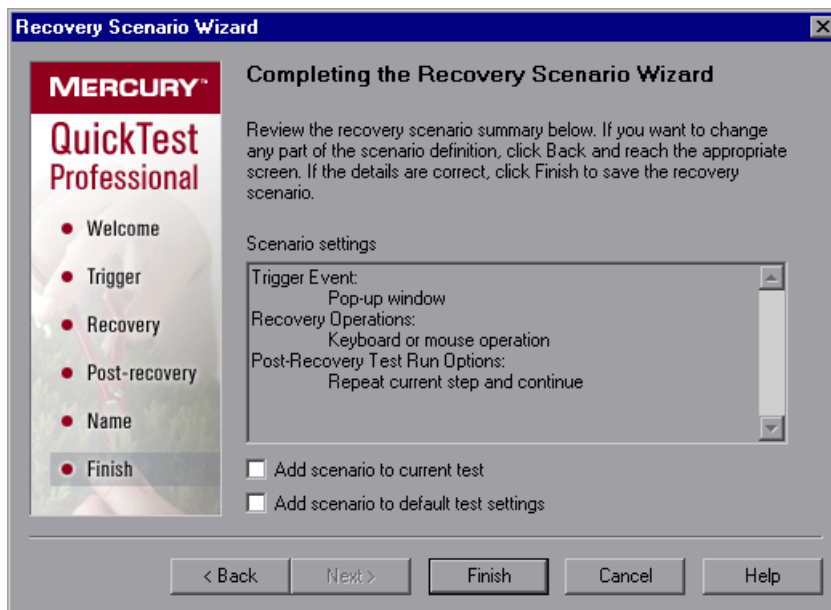


Enter a name and a textual description for your recovery scenario, and click **Next** to continue to the Completing the Recovery Scenario Wizard screen.

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 and/or to add it to the default settings for all new tests.



You can select the **Add scenario to current test** check box to associate this recovery scenario with the current test. When you click **Finish**, QuickTest adds the recovery scenario to the **Scenarios** list in the Recovery tab of the Test Settings dialog box.

You can 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. For more information, refer to “Defining Recovery Scenario Settings for Your Test” on page 768 in the *QuickTest Professional Basic Features User’s Guide*.

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.

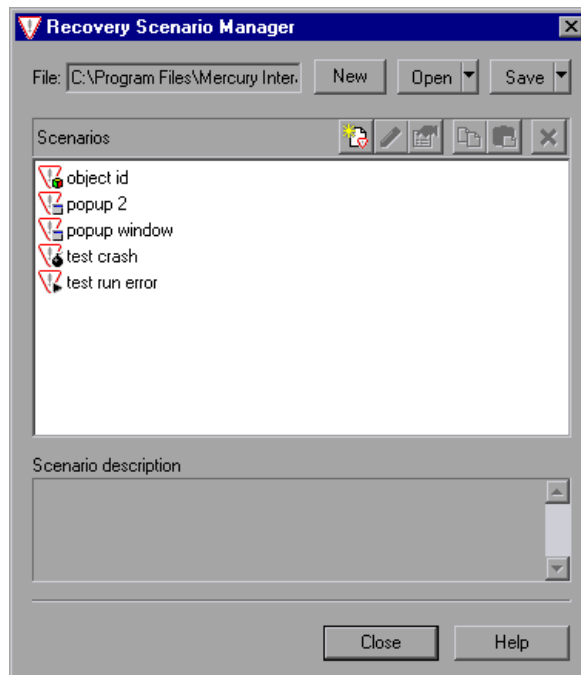
- 2** Choose the folder in which you want to save the file.

- 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
	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 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:

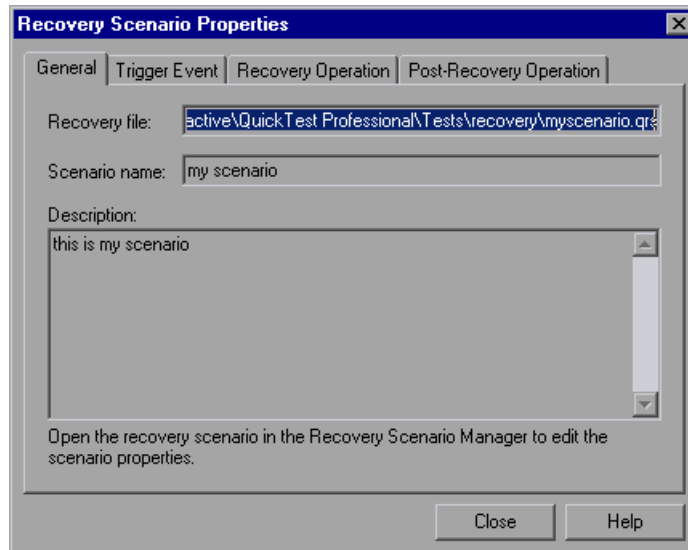
- Viewing Recovery Scenario Properties
- Modifying Recovery Scenarios
- Deleting Recovery Scenarios
- Copying Recovery Scenarios between Recovery Scenario Files

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:

- **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.
- **Trigger Event tab**—Displays the settings for the trigger event defined for the recovery scenario.

- ▶ **Recovery Operation tab**—Displays the recovery operation(s) defined for the recovery scenario.
- ▶ **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. For information on the Recovery Scenario Wizard options, see “Defining Recovery Scenarios,” on page 48.

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, 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

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. You can also define which recovery scenarios will be used as the default scenarios for all new tests.

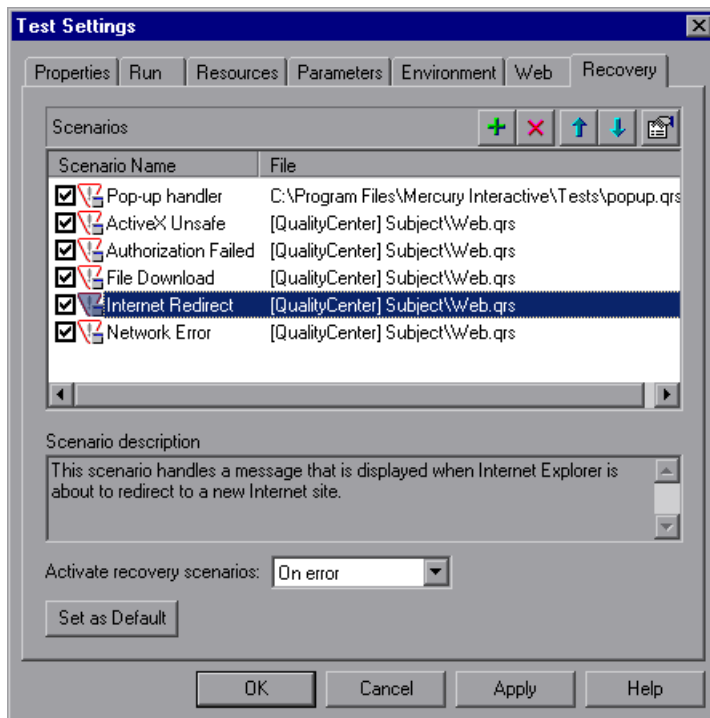
Adding Recovery Scenarios to Your Test

After you have created recovery scenarios, you can associate one or more scenarios with a test 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 lists all the recovery scenarios associated with the current test.

Tip: When a trigger event occurs, QuickTest checks for applicable recovery scenarios in the order in which they are displayed in the Recovery tab. You can change this order as described in “Setting Recovery Scenario Priorities” on page 88.

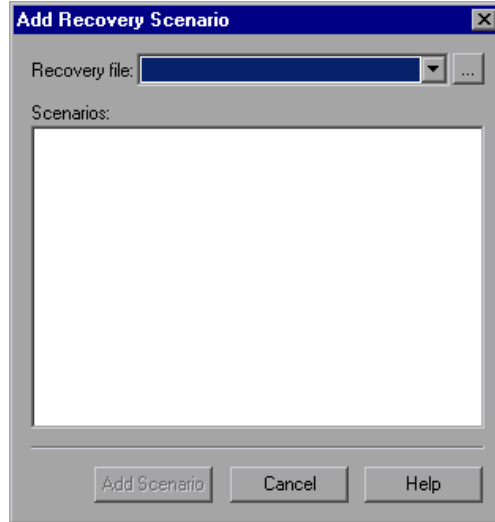
To add a recovery scenario to a test:

- 1 Choose **File > Settings**. The Test 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. 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 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.

Viewing Recovery Scenario Properties

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

Note: You modify recovery scenario settings from the Recovery Scenario Manager dialog box. For more information, see “Modifying Recovery Scenarios” on page 82.


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. For more information, see “Viewing Recovery Scenario Properties” on page 81.

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.

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

You can remove the association between a specific scenario and a test using the Recovery tab of the Test Settings dialog box. After you remove a scenario from a test, 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:

- 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.



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. When you disable a specific scenario, it remains associated with the test, but is not performed by QuickTest during the run session. You can enable the scenario at a later time.

You can also specify the conditions for which the recovery scenario is to be activated.

To enable/disable specific recovery scenarios:

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

To define when the recovery mechanism is activated:

- Select one of the following options in the **Activate recovery scenarios** box:
 - **On every step**—The recovery mechanism is activated after every step.
 - **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.

- **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 programmatically during the run session. For more information, see “Programmatically Controlling the Recovery Mechanism” on page 91.

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.

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.

For more information on the Recovery object and its methods, refer to the *QuickTest Professional Object Model Reference*.

4

Configuring Object Identification

When you record an operation on an object or add an object to the object repository, QuickTest learns a set of properties and values that uniquely describe the object within the object hierarchy. In most cases, this description is sufficient to enable QuickTest to identify the object during the run session.

If you find that the description QuickTest uses for a certain object class is not the most logical one for the objects in your application, or if you expect that the values of the properties in the object description may change frequently, you can configure the way that QuickTest learns and identifies objects. You can also map user-defined objects to standard test object classes and configure the way QuickTest learns objects from your user-defined object classes.

This chapter describes:

- ▶ About Configuring Object Identification
- ▶ Understanding the Object Identification Dialog Box
- ▶ Configuring Smart Identification
- ▶ Mapping User-Defined Test Object Classes

About Configuring Object Identification

QuickTest has a predefined set of properties that it learns for each test object. If these mandatory property values are not sufficient to uniquely identify an object you record or add, QuickTest can add some assistive properties and/or an ordinal identifier to create a unique description.

Mandatory properties are properties that QuickTest always learns for a particular test object class.

Assistive properties are properties that QuickTest learns only if the mandatory properties that QuickTest learns for a particular object in your application are not sufficient to create a unique description. If several assistive properties are defined for an object class, then QuickTest learns one assistive property at a time, and stops as soon as it creates a unique description for the object. If QuickTest does learn assistive properties, those properties are added to the test object description.

Note: If the combination of all defined mandatory and assistive properties is not sufficient to create a unique test object description, QuickTest also learns the value for the selected ordinal identifier. For more information, see “Selecting an Ordinal Identifier” on page 102.

When you run a test, QuickTest searches for the object that matches the description it learned (without the ordinal identifier). If it cannot find any object that matches the description, or if it finds more than one object that matches, QuickTest uses the Smart Identification mechanism (if enabled) to identify the object. In many cases, a Smart Identification definition can help QuickTest identify an object, if it is present, even when the learned description fails due to changes in one or more property values. The test object description is used together with the ordinal identifier only in cases where the Smart Identification mechanism does not succeed in narrowing down the object candidates to a single object.

You use the Object Identification dialog box (**Tools > Object Identification**) to configure the mandatory, assistive, and ordinal identifier properties that QuickTest uses to learn descriptions of the objects in your application, and to enable and configure the Smart Identification mechanism.

The Object Identification dialog box also enables you to configure new user-defined classes and map them to an existing test object class so that QuickTest can recognize objects from your user-defined classes when you run your test.

Understanding the Object Identification Dialog Box

You use the main screen of the Object Identification dialog box to set mandatory and assistive 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 displayed in the **Test Object classes** list for a selected environment.

Notes:

Any changes you make in the Object Identification dialog box have no effect on objects already added to the object repository.

The learned and Smart Identification properties of certain test objects cannot be configured, for example, the WinMenu, VbLabel, VbObject, and VbToolbar objects. These objects are therefore not included in the **Test Object classes** list for the selected environment.

For more information, see:

- “Configuring Mandatory and Assistive Recording Properties” on page 96
- “Selecting an Ordinal Identifier” on page 102
- “Enabling and Disabling Smart Identification” on page 107
- “Restoring Default Object Identification Settings for Test Objects” on page 108
- “Generating Automation Scripts for Your Object Identification Settings” on page 108
- “Configuring Smart Identification” on page 109
- “Mapping User-Defined Test Object Classes” on page 119

Configuring Mandatory and Assistive Recording Properties

If you find that the description QuickTest uses for a certain object class is not the most logical one for the objects in your application, or if you expect that the values of the properties currently used in the object description may change, you can modify the mandatory and assistive properties that QuickTest learns when you learn an object of a given class.

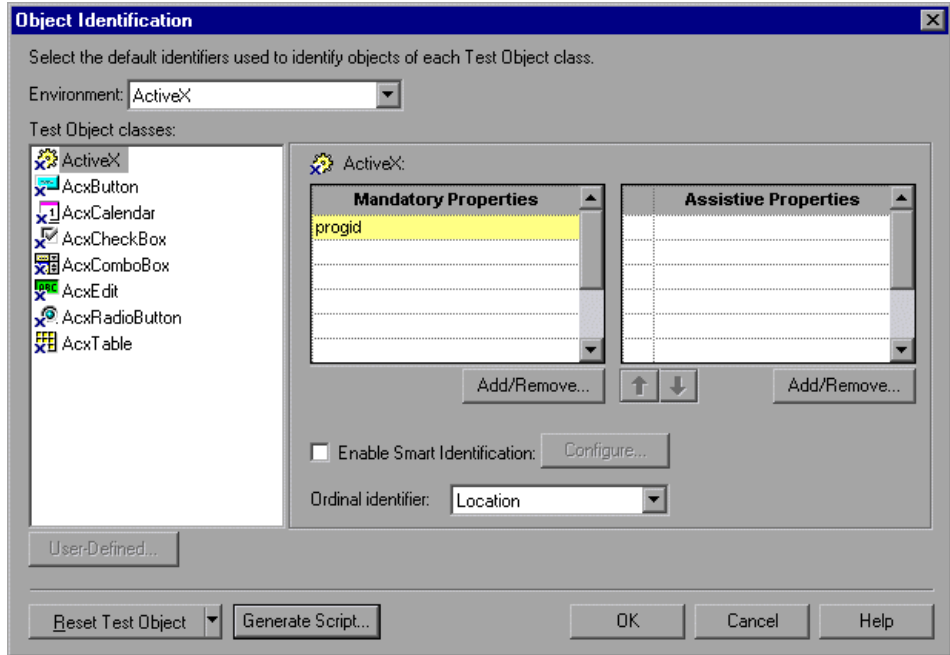
During the run session, QuickTest looks for objects that match all properties in the test object description—it does not distinguish between properties that were learned as mandatory properties and those that were learned as assistive properties.

For example, the default mandatory properties for a Web Image object are the **alt**, **html tag**, and **image type** properties. There are no default assistive properties defined. Suppose your Web site contains several space holders for different collections of rotating advertisements. You want to record a test that clicks on the images in each one of these space holders. However, since each advertisement image has a different **alt** value, one **alt** value would be recorded when you create the test, and most likely another **alt** value will be captured when you run the test, causing the run to fail. In this case, you could remove the **alt** property from the Web Image mandatory properties list. Instead, since each advertisement image displayed in a certain space holder in your site has the same value for the image **name** property, you could add the **name** property to the mandatory properties to enable QuickTest to uniquely identify the object.

Also, suppose that whenever a Web image is displayed more than once on a page (for example, a logo displayed on the top and bottom of a page), the Web designer adds a special **ID** property to the Image tag. The mandatory properties are sufficient to create a unique description for images that are displayed only once on the page, but you also want QuickTest to learn the **ID** property for images that are displayed more than once on a page. To do this, you add the **ID** property as an assistive property, so that QuickTest learns the **ID** property only when it is necessary for creating a unique test object description.

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

- 1 Choose **Tools > Object Identification**. The Object Identification dialog box opens.

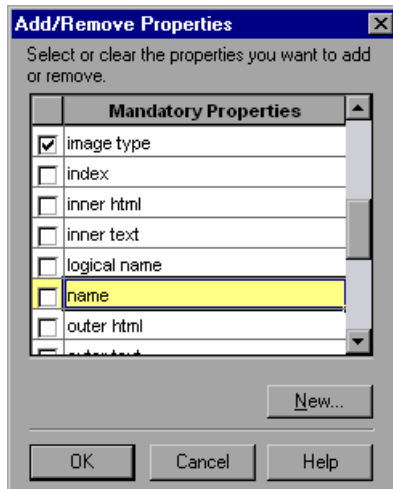


- 2 Select the appropriate environment in the **Environment** list. The test object classes associated with the selected environment are displayed alphabetically in the **Test Object classes** list. (In Standard Windows, the user-defined objects appear at the bottom of the list.)

Note: The environments included in the Environment list correspond to the loaded add-in environments. For more information on loading add-ins, refer to Chapter 27, “Working with QuickTest Add-Ins.” in the *QuickTest Professional Basic Features User’s Guide*.

- 3 In the **Test Object classes** list, select the test object class you want to configure.

- 4 In the **Mandatory Properties** list, click **Add/Remove**. The Add/Remove Properties dialog box for mandatory properties opens.



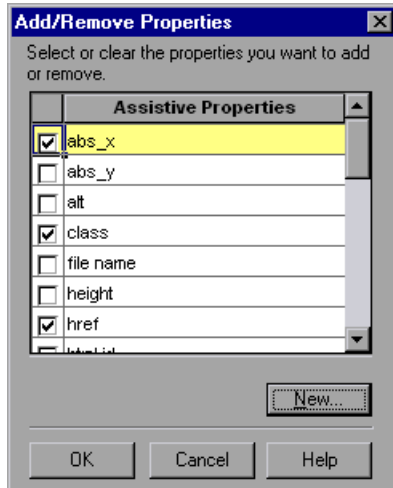
- 5 Select the properties you want to include in the Mandatory Properties list and/or clear the properties you want to remove from the list.

Note: You cannot include the same property in both the mandatory and assistive property lists.

You can specify a new property by clicking **New** and specifying a valid property name in the displayed dialog box.

Tip: You can also add property names to the set of available properties for Web objects using the `attribute/<PropertyName>` notation. To do this, click **New**. The New Property dialog box opens. Enter a valid property using 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`.

- 6 Click **OK** to close the Add/Remove Properties dialog box. The updated set of mandatory properties is displayed in the **Mandatory Properties** list.
- 7 In the **Assistive Properties** list, click **Add/Remove**. The Add/Remove Properties dialog box for assistive properties opens.



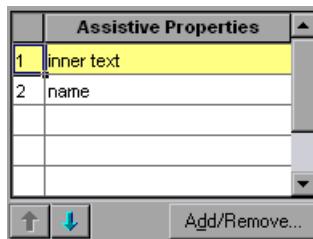
- 8 Select the properties you want to include in the assistive properties list and/or clear the properties you want to remove from the list.

Note: You cannot include the same property in both the mandatory and assistive property lists.

You can specify a new property by clicking **New** and specifying a valid property name in the displayed dialog box.

Tip: You can also add property names to the set of available properties for Web objects using the `attribute/<PropertyName>` notation. To do this, click **New**. The New Property dialog box opens. Enter a valid property in the format `attribute/<PropertyName>` and click **OK**. The new property is added to the **Assistive Properties** list. For example, to add a property called MyColor, enter `attribute/MyColor`.

- 9 Click **OK** to close the Add/Remove Properties dialog box. The properties are displayed in the Assistive Properties list.



- 10 Use the up and down arrows to set your preferred order for the assistive properties. When you learn an object, 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.

Selecting an Ordinal Identifier

In addition to learning the mandatory and assistive properties specified in the Object Identification dialog box, QuickTest can also learn a backup ordinal identifier for each test object. The ordinal identifier assigns the object a numerical value that indicates its order relative to other objects with an otherwise identical description (objects that have the same values for all properties specified in the mandatory and assistive property lists). This ordered value enables QuickTest to create a unique description when the mandatory and assistive properties are not sufficient to do so.

Because the assigned ordinal property value is a relative value and is accurate only in relation to the other objects displayed when QuickTest learns an object, changes in the layout or composition of your application page or screen could cause this value to change, even though the object itself has not changed in any way. For this reason, QuickTest learns a value for this backup ordinal identifier only when it cannot create a unique description using all available mandatory and assistive properties.

In addition, even if QuickTest learns an ordinal identifier, it will use it during the run session only if the learned description and the Smart Identification mechanism are not sufficient to identify the object in your application. If QuickTest can use other test object properties to identify the object during a run session, the ordinal identifier is ignored.

QuickTest can use the following types of ordinal identifiers to identify an object:

- ▶ **Index**—Indicates the order in which the object appears in the application code relative to other objects with an otherwise identical description. For more information, see “Identifying an Object Using the Index Property” on page 103.
- ▶ **Location**—Indicates the order in which the object appears within the parent window, frame, or dialog box relative to other objects with an otherwise identical description. For more information, see “Identifying an Object Using the Location Property” on page 104.
- ▶ **CreationTime** (Browser object only)—Indicates the order in which the browser was opened relative to other open browsers with an otherwise identical description. For more information, see “Identifying an Object Using the CreationTime Property” on page 105.

By default, an ordinal identifier type exists for each test object class. To modify the default ordinal identifier, you can select the desired type from the **Ordinal identifier** box.



Tip: While recording, if QuickTest successfully creates a unique test object description using the mandatory and assistive properties, it does not learn an ordinal identifier value. You can add an ordinal identifier to an object's test object properties at a later time using the **Add/Remove** option from the Object Properties or Object Repository dialog box. For more information, refer to Chapter 6, "Working with Test Objects" in the *QuickTest Professional Basic Features User's Guide*.

Identifying an Object Using the Index Property

While learning an object, QuickTest can assign a value to the test object's **Index** property to uniquely identify the object. The value is based on the order in which the object appears within the source code. The first occurrence is 0.

Index property values are object-specific. Therefore, if you use `Index:=3` to describe a WebEdit test object, QuickTest searches for the fourth WebEdit object in the page. However, if you use `Index:=3` to describe a WebElement object, 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 a page contains 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 following statement refers to the third item in the list, as this is the first WebEdit object on the page with the name `UserName`:

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

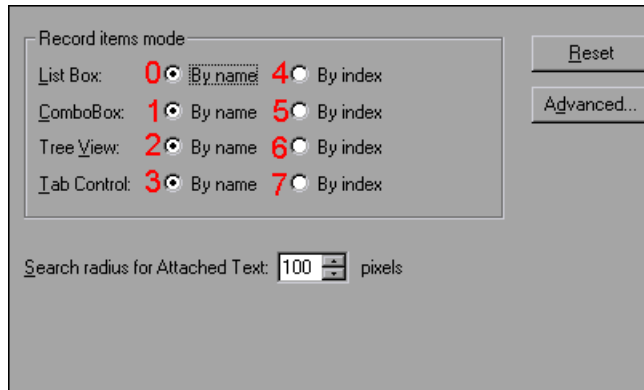
In contrast, the following statement refers to the second item in the list, as that is the first object of any type (`WebElement`) with the name `UserName`:

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

Identifying an Object Using the Location Property

While learning an object, QuickTest can assign a value to the test object's **Location** property to uniquely identify the object. The value is based on the order in which the object appears within the window, frame, or dialog box, in relation to other objects with identical properties. The first occurrence of the object is 0. Values are assigned in columns from top to bottom, and left to right.

In the following example, the radio buttons in the dialog box are numbered according to their location property.



Location property values are object-specific. Therefore, if you use `Location:=3` to describe a `WinButton` test object, QuickTest searches from top to bottom, and left to right for the fourth `WinButton` object in the page. However, if you use `Location:=3` to describe a `WinObject` object, QuickTest searches from top to bottom, and left to right for the fourth standard object on the page—regardless of the type—because the `WinObject` object applies to all standard objects.

For example, suppose a dialog box contains the following objects:

- a button object with the name OK
- a button object with the name Add/Remove
- a check box object with the name Add/Remove
- a button object with the name Help
- a check box object with the name Check spelling

The following statement refers to the third item in the list, as this is the first check box object on the page with the name Add/Remove.

```
WinCheckBox("Name:=Add/Remove", "Location:=0")
```

In contrast, the following statement, refers to the second item in the list, as that is the first object of any type (WinObject) with the name Add/Remove.

```
WinObject("Name:=Add/Remove", "Location:=0")
```

Identifying an Object Using the CreationTime Property

While learning a browser object, if QuickTest is unable to uniquely identify the object according to its test object description, it assigns a value to the **CreationTime** test object property. This value indicates the order in which the browser was opened relative to other open browsers with an otherwise identical description. The first browser that opens receives the value `CreationTime = 0`.

During the run session, if QuickTest is unable to identify a browser object based solely on its test object description, it examines the order in which the browsers were opened, and then uses the **CreationTime** property to identify the correct one.

For example, if you record a test on three otherwise identical browsers that are opened at 9:01 pm, 9:03 pm, and 9:05 pm, QuickTest assigns the `CreationTime` values, as follows: `CreationTime = 0` to the 9:01 am browser, `CreationTime = 1` to the 9:03 am browser, and `CreationTime = 2` to the 9:06 am browser.

At 10:30 pm, when you run your test, suppose the browsers are opened at 10:31 pm, 10:33 pm, and 10:34 pm. QuickTest identifies the browsers, as follows: the 10:31 pm browser is identified with the browser test object with `CreationTime = 0`, 10:33 pm browser is identified with the test object with `CreationTime = 1`, 10:34 pm browser is identified with the test object with `CreationTime = 2`.

If there are several open browsers, the one with the lowest `CreationTime` is the first one that was opened and the one with the highest `CreationTime` is the last one that was opened. For example, if there are three or more browsers open, the one with `CreationTime = 2` is the third browser that was opened. If seven browsers are opened during a recording session, the browser with `CreationTime = 6` is the last browser opened.

If a step was recorded on a browser with a specific `CreationTime` value, but during a run session there is no open browser with that `CreationTime` value, the step will run on the browser that has the highest `CreationTime` value. For example, if a step was recorded on a browser with `CreationTime = 6`, but during the run session there are only two open browsers, with `CreationTime = 0` and `CreationTime = 1`, then the step runs on the last browser opened, which in this example is the browser with `CreationTime = 1`.

Note: It is possible that at a particular time during a session, the available `CreationTime` values may not be sequential. For example, if you open six browsers during a record or run session, and then during that session, you close the second and fourth browsers (`CreationTime` values 1 and 3), then at the end of the session, the open browsers will be those with `CreationTime` values 0, 2, 4, and 5).

Enabling and Disabling Smart Identification

Selecting the **Enable Smart Identification** check box for a particular test object class instructs QuickTest to learn the property values of all properties specified as the object's base and/or optional filter properties in the Smart Identification Properties dialog box.

By default, some test objects already have Smart Identification configurations and others do not. Those with default configurations also have the **Enable Smart Identification** check box selected by default.

You should enable the Smart Identification mechanism only for test object classes that have defined Smart Identification configuration. However, even if you define a Smart Identification configuration for a test object class, you may not always want to learn the Smart Identification property values. If you do not want to learn the Smart Identification properties, clear the **Enable Smart Identification** check box.

Note: Even if you choose to learn Smart Identification properties for an object, you can disable use of the Smart Identification mechanism for a specific object in the Object Properties or Object Repository dialog box. You can also disable use of the mechanism for an entire test in the Run tab of the Test Settings dialog box. For more information, refer to Chapter 6, “Working with Test Objects,” and “Defining Run Settings for Your Test” on page 747 in the *QuickTest Professional Basic Features User's Guide*.

However, if you do not learn Smart Identification properties, you cannot enable the Smart Identification mechanism for an object later.

For more information on the Smart Identification mechanism, see “Configuring Smart Identification” on page 109.

Restoring Default Object Identification Settings for Test Objects

You can restore the default settings for object identification and Smart Identification property settings for all loaded environments, for the current environment only, or for a selected test object.

Only built-in object properties can be reset. User-defined objects will be deleted when resetting the Standard Windows environment.

Note: Only currently loaded environments are listed in the Environments box in the Object Identification dialog box.

By default, the **Reset Test Object** button is displayed, but you can click the down arrow to select one of the following options:

- ▶ **Reset Test Object**—Resets the settings for the selected test object to the system default.
- ▶ **Reset Environment**—Resets the settings for all the test objects in the current environment to the system default.
- ▶ **Reset All**—Resets the settings for all currently loaded environments to the system default.

Generating Automation Scripts for Your Object Identification Settings

You can click the **Generate Script** button to generate an automation script containing the current object identification settings. For more information, see “Automating QuickTest Operations” on page 231, or refer to the *QuickTest Automation Object Model Reference* (**Help > QuickTest Automation Object Model Reference**).

Configuring Smart Identification

Configuring Smart Identification properties enables you to help QuickTest identify objects in your application, even if some of the properties in the object's learned description have changed.

When QuickTest uses the learned description to identify an object, it searches for an object that matches all 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 learned object description, or if it finds more than one object that fits the description, then QuickTest ignores the learned 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. Therefore, if configured logically, a Smart Identification definition can probably help QuickTest identify an object, if it is present, even when the learned description fails.

The Smart Identification mechanism uses two types of properties:

- ▶ **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.
- ▶ **Optional Filter Properties**—Other properties that can help identify objects of a particular class. These properties are unlikely to change on a regular basis, but can be ignored if they are no longer applicable.

Understanding the Smart Identification Process

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

- 1** QuickTest “forgets” the learned 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 Properties list.
- 2** QuickTest filters out any object in the object candidate list 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 learned description plus the ordinal identifier to identify the object.

If the combined learned description and ordinal identifier are not sufficient to identify the object, then QuickTest stops the run session and displays a Run Error message.

Reviewing Smart Identification Information in the Test Results

If the learned description does not enable QuickTest to identify a specified object in a step, and a Smart Identification definition is defined (and enabled) for the object, then QuickTest tries to identify the object using the Smart Identification mechanism.

If QuickTest successfully uses Smart Identification to find an object after no object matches the learned description, the Test Results receive a warning status and indicate that the Smart Identification mechanism was used.

If the Smart Identification mechanism cannot successfully identify the object, QuickTest uses the learned description plus the ordinal identifier to identify the object. If the object is still not identified, the test fails and a normal failed step is displayed in the results.

For more information, refer to “Analyzing Smart Identification Information in the Test Results” on page 670 in the *QuickTest Professional Basic Features User’s Guide*.

Walking Through a Smart Identification Example

The following example walks you through the object identification process for an object.

Suppose you have the following statement in your test:

```
Browser("Mercury Tours").Page("Mercury Tours").Image("Login").Click 22,17
```

When you created your test, QuickTest learned the following object description for the Login image:

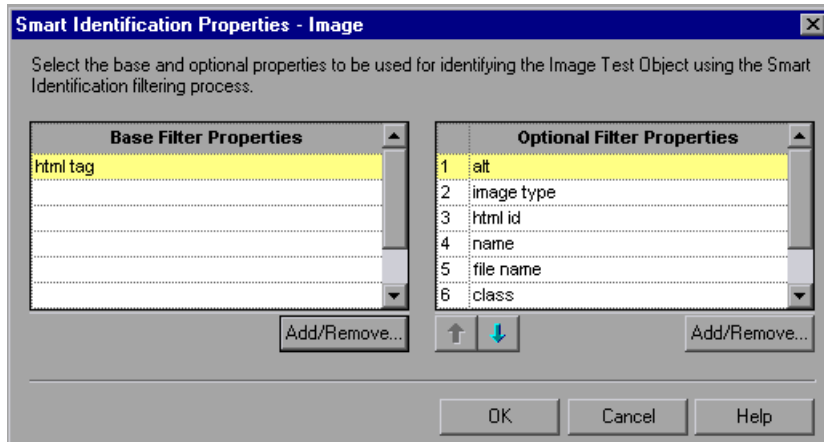
Name	Value
[-] Description properties	
image type	Image Button
html tag	INPUT
alt	Login

However, at some point after you created your test, a second login button (for logging into the VIP section of the Web site) was added to the page, so the Web designer changed the original Login button’s **alt** tag to: basic login.

The default description for Web Image objects (**alt**, **html tag**, **image type**) works for most images in your site, but it no longer works for the Login image, because that image's **alt** property no longer matches the learned description. Therefore, when you run your test, QuickTest is unable to identify the Login button based on the learned description. However, QuickTest succeeds in identifying the Login button using its Smart Identification definition.

The explanation below describes the process that QuickTest uses to find the Login object using Smart Identification:

- 1 According to the Smart Identification definition for Web image objects, QuickTest learned the values of the following properties when you recorded the click on the Login image:



The learned values are as follows:

Base Filter Properties:

Property	Value
html tag	INPUT

Optional Filter Properties:

Property	Value
alt	Login
image type	Image Button
name	login
file name	login.gif
class	<null>
visible	1

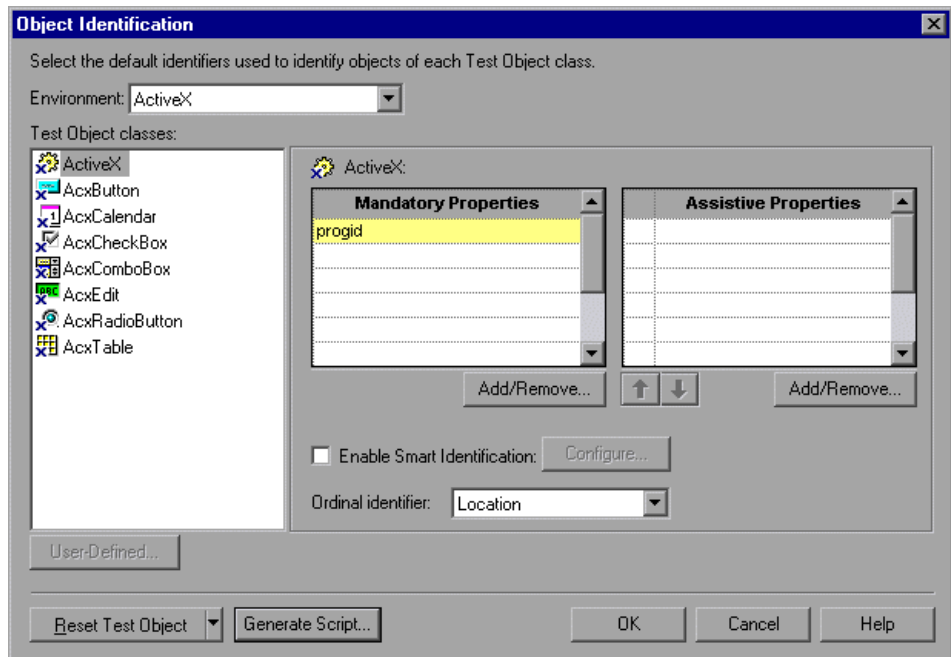
- 2** QuickTest begins the Smart Identification process by identifying the five objects on the Mercury Tours page that match the base filter properties definition (**html tag** = INPUT and **image type** = Image Button). QuickTest considers these to be the object candidates and begins checking the object candidates against the **Optional Filter Properties** list.
- 3** QuickTest checks the **alt** property of each of the object candidates, but none have the **alt** value: Login, so QuickTest ignores this property and moves on to the next one.
- 4** QuickTest checks the **name** property of each of the object candidates, and finds that two of the objects (both the basic and VIP Login buttons) have the name: login. QuickTest filters out the other three objects from the list, and these two login buttons become the new object candidates.
- 5** QuickTest checks the **file name** property of the two remaining object candidates. Only one of them has the file name login.gif, so QuickTest correctly concludes that it has found the Login button and clicks it.

Step-by-Step Instructions for Configuring a Smart Identification Definition

You use the Smart Identification Properties dialog box, accessible from the Object Identification dialog box, to configure the Smart Identification definition for a test object class.

To configure Smart Identification properties:

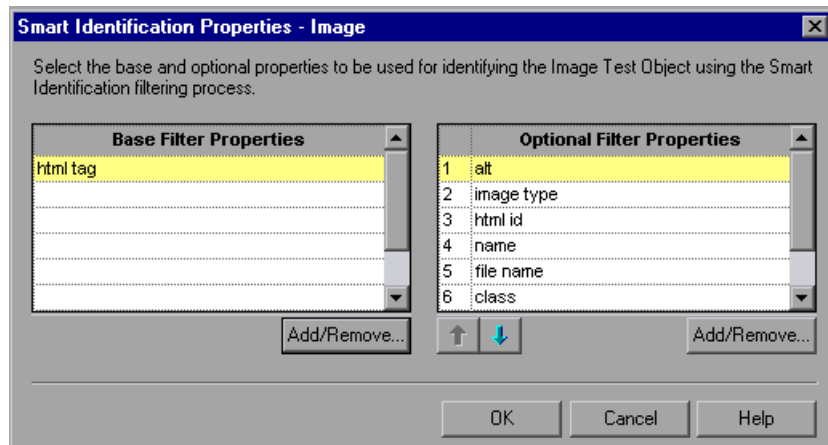
- 1 Choose **Tools > Object Identification**. The Object Identification dialog box opens.



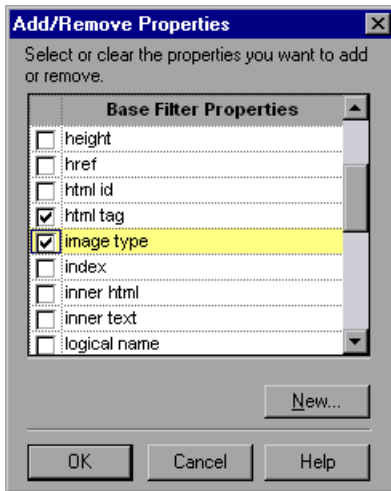
- 2 Select the appropriate environment in the **Environment** list. The test object classes associated with the selected environment are displayed in the **Test object classes** list.

Note: The environments included in the Environment list are those that correspond to the loaded add-in environments. For more information on loading add-ins, refer to Chapter 27, “Working with QuickTest Add-Ins.” in the *QuickTest Professional Basic Features User’s Guide*.

- 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.



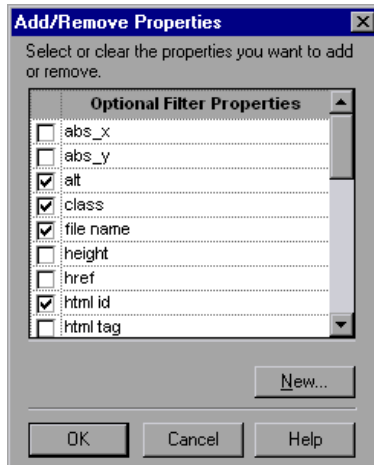
- 6 Select the properties you want to include in the **Base Filter Properties** list and/or clear the properties you want to remove from the list.

Note: You cannot include the same property in both the base and optional property lists.

You can specify a new property by clicking **New** and specifying a valid property name in the displayed dialog box.

Tip: You can also add property names to the set of available properties for Web objects using the `attribute/<PropertyName>` notation. To do this, click **New**. The New Property dialog box opens. Enter a valid property in the format `attribute/<PropertyName>` and click **OK**. The new property is added to the **Base Filter Properties** list. For example, to add a property called MyColor, enter `attribute/MyColor`.

- 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.



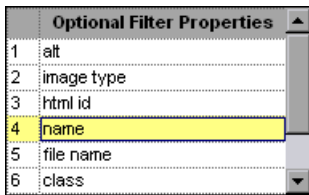
- 9 Select the properties you want to include in the **Optional Filter Properties** list and/or clear the properties you want to remove from the list.

Note: You cannot include the same property in both the base and optional property lists.

You can specify a new property by clicking **New** and specifying a valid property name in the displayed dialog box.

Tip: You can also add property names to the set of available properties for Web objects using the `attribute/<PropertyName>` notation. To do this, click **New**. The New Property dialog box opens. Enter a valid property in the format `attribute/<PropertyName>` and click **OK**. The new property is added to the **Optional Filter Properties** list. For example, to add a property called `MyColor`, enter `attribute/MyColor`.

- 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 Filter Properties** list until it filters the object candidates down to one object.

Mapping User-Defined Test Object Classes

The Object Mapping dialog box enables you to map an object of an unidentified or custom class to a Standard Windows class. For example, if your application has a button that cannot be identified, this button is learned as a generic WinObject. You can teach QuickTest to identify your object as if it belonged to a standard Windows button class. Then, when you click the button while recording, QuickTest records the operation in the same way as a click on a standard Windows button. When you map an unidentified or custom object to a standard object, your object is added to the list of Standard Windows test object classes as a user-defined test object. You can configure the object identification settings for a user defined object class just as you would any other object class.

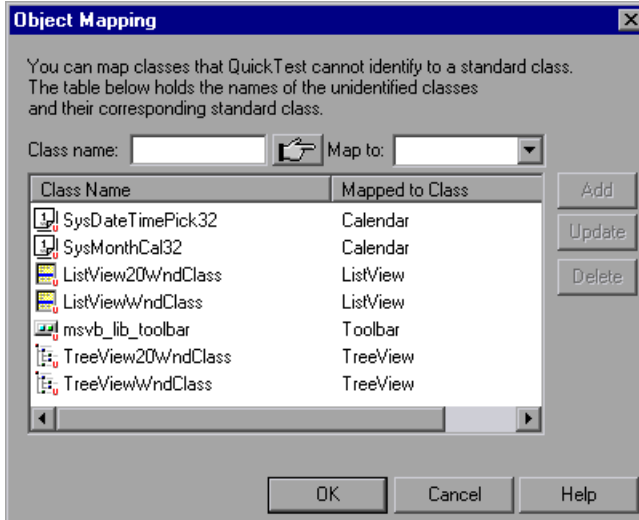
You should map an object that cannot be identified only to a standard Windows class with comparable behavior. For example, do not map an object that behaves like a button to the edit class.

Note: You can define user-defined classes only when **Standard Windows** is selected in the **Environment** box.

To map an unidentified or custom class to a standard Windows class:

- 1** Choose **Tools > Object Identification**. The Object Identification dialog box opens.
- 2** Select **Standard Windows** in the **Environment** box. The **User-Defined** button becomes enabled.

- 3 Click **User-Defined**. The Object Mapping dialog box opens.



- 4 Click the pointing hand and then click the object whose class you want to add as a user-defined class. The name of the user-defined object is displayed in the **Class Name** box.

Tip: Hold the left CTRL key to change the window focus or perform operations such as a right-click or mouseover to display a context menu. If the window containing the object you want to select is minimized, you can display it by holding the left CTRL key, right-clicking the application in the Windows task bar, and choosing **Restore** from the context menu.

- 5 In the **Map to** box, select the standard object class to which you want to map your user-defined object class and click **Add**. The class name and mapping is added to the object mapping list.
- 6 If you want to map additional objects to standard classes, repeat steps 4-5 for each object.

- 7** Click **OK**. The Object Mapping dialog box closes and your object is added to the list of Standard Windows test object classes as a user-defined test object. Note that your object has an icon with a red U in the lower-right corner, identifying it as a user-defined class.
- 8** Configure the object identification settings for your user defined object class just as you would any other object class. For more information, see “Configuring Mandatory and Assistive Recording Properties,” on page 96, and “Configuring Smart Identification,” on page 109.

To modify an existing mapping:

- 1** In the Object Mapping dialog box, select the class you want to modify from the object mapping list. The class name and current mapping are displayed in the Class name and Map to boxes.
- 2** Select the standard object class to which you want to map the selected user-defined class and click **Update**. The class name and mapping is updated in the object mapping list.
- 3** Click **OK** to close the Object Mapping dialog box.

To delete an existing mapping:

- 1** In the Object Mapping dialog box, select the class you want to delete from the object mapping list.
- 2** Click **Delete**. The class name and mapping is deleted from the object mapping list in the Object Mapping dialog box.
- 3** Click **OK**. The Object Mapping dialog box closes and the class name is deleted from the Standard Windows test object classes list in the Object Identification dialog box.

5

Working with the Expert View and Function Library Windows

In QuickTest, tests are composed of statements coded in Microsoft's VBScript programming language. The Expert View provides an alternative to the Keyword View for testers who are familiar with VBScript. You can also create function libraries in QuickTest using VBScript.

This chapter explains how to work in the Expert View, provides a brief introduction to VBScript, and shows you how to enhance your tests and function libraries using a few simple programming techniques.

This chapter describes:

- About Working with the Expert View and Function Library Windows
- Understanding and Using the Expert View
- Navigating in the Expert View and Function Libraries
- Understanding Basic VBScript Syntax
- Using Programmatic Descriptions
- Running and Closing Applications Programmatically
- Using Comments, Control-Flow, and Other VBScript Statements
- Retrieving and Setting Test Object Property Values
- Accessing Run-Time Object Properties and Methods
- Running DOS Commands
- Enhancing Your Tests and Function Libraries Using the Windows API
- Choosing Which Steps to Report During the Run Session

About Working with the Expert View and Function Library Windows

In the Expert View, you can view an action in VBScript. If you are familiar with VBScript, you can add and update statements and enhance your tests and function libraries with programming. You can also create and work with function libraries using the Function Library window.

When you record steps, the Expert View displays the steps as VBScript statements. After you record your test, you can increase its power and flexibility by adding recordable and non-recordable VBScript statements.

To learn about working with VBScript, you can view the VBScript documentation directly from the QuickTest Help menu (**Help > QuickTest Professional Help > VBScript Reference**).

You can add statements that perform operations on objects or retrieve information from your application. For example, you can add a step that checks that an object exists, or you can retrieve the return value of a method.

You can add steps to your test or function library either manually or using the Step Generator. For more information on using the Step Generator, refer to “Inserting Steps Using the Step Generator” on page 527 in the *QuickTest Professional Basic Features User’s Guide*.

You can print the test displayed in the Expert View or a function library at any time. You can also include additional information in the printout. For more information on printing from the Expert View, refer to “Printing a Test” on page 103 in the *QuickTest Professional Basic Features User’s Guide*. For more information on printing a function library, see “Printing a Function Library” on page 198.

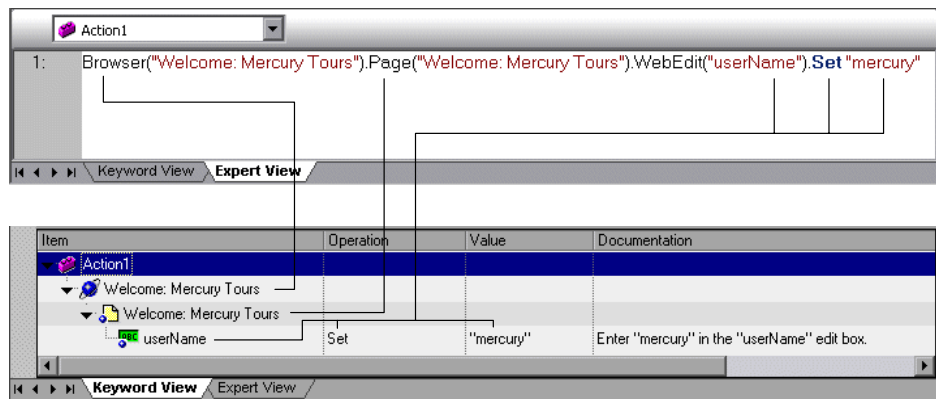
Understanding and Using the Expert View

If you prefer to work with VBScript statements, you can choose to work with your tests in the Expert View, as an alternative to using the Keyword View. You can move between the two views as you wish, by selecting the Expert View or Keyword View tab at the bottom of the Test pane in the QuickTest window.

The Expert View displays the same steps and objects as the Keyword View, but in a different format:

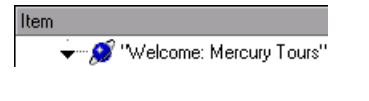

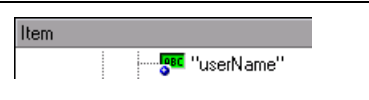


- ▶ In the Keyword View, QuickTest displays information about each step and shows the object hierarchy in an icon-based table. For more information, refer to Chapter 5, “Working with the Keyword View” in the *QuickTest Professional Basic Features User's Guide*.
- ▶ In the Expert View, QuickTest displays each step as a VBScript line or statement. In object-based steps, the VBScript statement defines the object hierarchy.

The following diagram shows how the same object hierarchy is displayed in the Expert View and in the Keyword View:



Each line of VBScript in the Expert View represents a step in the test. The example above represents a step in a test in which the user inserts the name mercury into an edit box. The hierarchy of the step enables you to see the name of the site, the name of the page, the type and name of the object in the page, and the name of the method performed on the object.

The table below explains how the different parts of the same step are represented in the Keyword View and the Expert View:

Keyword View	Expert View	Explanation
	Browser ("Welcome: Mercury Tours")	The name of the browser test object is Welcome: Mercury Tours.
	Page ("Welcome: Mercury Tours")	The name of the current page is Welcome: Mercury Tours.
	WebEdit ("userName")	The object type is WebEdit; the name of the edit box on which the operation is performed is userName.
	Set	The method performed on the edit box is Set .
	"mercury"	The value inserted into the username edit box is mercury.

In the Expert View, an object's description is displayed in parentheses following the object type. For all objects stored in the object repository, the object name is a sufficient object description. In the following example, the object type is Browser, and the object name is Welcome: Mercury Tours:

Browser ("Welcome: Mercury Tours")

Tip: Test object and method names are not case sensitive.

The objects in the object hierarchy are separated by a dot. In the following example, Browser and Page are two separate objects in the same hierarchy:

Browser("Welcome: Mercury Tours").Page("Welcome: Mercury Tours")

The operation (method) performed on the object is always displayed at the end of the statement, followed by any values associated with the operation. In the following example, the word mercury is inserted in the `userName` edit box using the **Set** method:

```
Browser("Welcome: Mercury Tours").Page("Welcome: Mercury Tours").  
  WebEdit("userName").Set "mercury"
```

When you record your test, QuickTest records the operations you perform on your application in terms of the objects in it.

The objects in QuickTest are divided by environment. QuickTest environments include standard Windows objects, Visual Basic objects, ActiveX objects, and Web 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.

For a complete list of objects and their associated methods and properties, choose **Help > QuickTest Professional Help**, and open the **Object Model Reference** from the Contents tab.

For more information about adding steps that use methods to perform operations, see “Generating Statements in the Expert View or a Function Library” on page 130.

For more information about using VBScript, see “Understanding Basic VBScript Syntax” on page 146.


Understanding Checkpoint and Output Statements

In QuickTest, you can create checkpoints and output values on pages, text strings, tables, and other objects. When you create a checkpoint or output value in the Keyword View, QuickTest creates a corresponding line in VBScript in the Expert View. It uses the **Check** method to perform the checkpoint, and the **Output** method to perform the output value step.

For example, in the following statement QuickTest performs a check on the words New York:

```
Browser("Mercury Tours").Page("Flight Confirmation").Check
    Checkpoint("New York")
```

The corresponding step in the Keyword View is displayed as follows:

	Operation	Value	Documentation
 "Flight Confirmation:"	Check	Checkpoint("New York")	Check whether text in the "Flight Confirmation:" Web page

Notes:

The details about a checkpoint are set in the relevant Checkpoint Properties dialog box and are stored with the object it checks. The details about an output value step are set in the relevant Output Value Properties dialog box and are stored with the object whose values it outputs. The statement displayed in the Expert View is a reference to the stored information. Therefore, you cannot insert a checkpoint or output value statement in the Expert View manually and you cannot copy a **Checkpoint** or **Output** statement from the Expert View to another test.

For more information on inserting and modifying checkpoints, refer to Chapter 7, “Understanding Checkpoints” in the *QuickTest Professional Basic Features User’s Guide*. For more information on inserting and modifying output values, refer to Chapter 16, “Outputting Values” in the *QuickTest Professional Basic Features User’s Guide*.

Understanding Parameter Indications

You can use QuickTest to enhance your tests by parameterizing values. A parameter is a variable that is assigned a value from an external data source or generator.

When you create a parameter in the Keyword View, QuickTest creates a corresponding line in VBScript in the Expert View.

For example, if you define the value of a method argument as a Data Table parameter, QuickTest retrieves the value from the Data Table using the following syntax:

Object_Hierarchy.Method **DataTable** (*parameterID*, *sheetID*)

Item	Description
<i>Object_Hierarchy</i>	The hierarchical definition of the test object, consisting of one or more objects separated by a dot.
<i>Method</i>	The name of the method that QuickTest executes on the parameterized object.
<i>DataTable</i>	The reserved object representing the Data Table.
<i>parameterID</i>	The name of the column in the Data Table from which to take the value.
<i>sheetID</i>	The name of the sheet in which the value is stored. If the parameter is a global parameter, dtGlobalSheet is the sheet ID.

For example, suppose you are recording a test on the Mercury Tours site, and you select San Francisco as your destination. The following statement is inserted into your test in the Expert View:

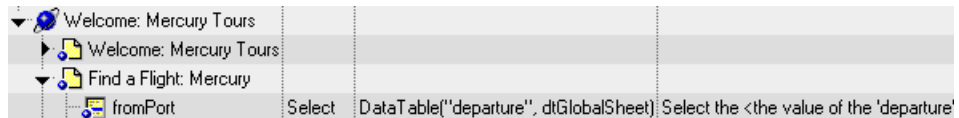
```
Browser("Welcome: Mercury").Page("Find a Flight:").WebList("toPort").
    Select "San Francisco"
```

Now suppose you parameterize the destination value, and you create a **Destination** column in the Data Table. The previous statement is modified to the following:

```
Browser("Welcome: Mercury").Page("Find a Flight:").WebList("toPort").  
    Select DataTable("Destination",dtGlobalSheet)
```

In this example, **Select** is the method name, **DataTable** is the object that represents the Data Table, **Destination** is the name of the column in the Data Table, and **dtGlobalSheet** indicates the Global sheet in the Data Table.

In the Keyword View, this step is displayed as follows:



For more information on using and defining parameter values, refer to Chapter 15, “Parameterizing Values” in the *QuickTest Professional Basic Features User’s Guide*.

Generating Statements in the Expert View or a Function Library

You can generate statements in the following ways:

- ▶ You can use the Step Generator to add steps that use methods and functions. For more information, refer to “Inserting Steps Using the Step Generator” on page 527 in the *QuickTest Professional Basic Features User’s Guide*.
- ▶ You can manually insert VBScript statements that use methods to perform operations. QuickTest includes IntelliSense, a statement completion feature that helps you select the test object, method, property, or collection for your statement and to view the relevant syntax as you type in the Expert View or a function library. For more information, see “Generating a Statement for an Object,” below.
- ▶ When you start to type a VBScript keyword in the Expert View or a function library, QuickTest automatically adds the relevant syntax or blocks to your script, if the **Auto-expand VBScript syntax** option is enabled. For more information, see “Automatically Completing VBScript Syntax” on page 135.

Generating a Statement for an Object

When you type in the Expert View or a function library, IntelliSense (the statement completion feature included with QuickTest) enables you to select the test object, method, property, or collection for your statement from a drop-down list and view the relevant syntax.

The **Statement Completion** option is enabled by default. You can disable or enable this option in the Editor Options dialog box. For additional information, see Chapter 11, “Customizing the Expert View and Function Library Windows.”

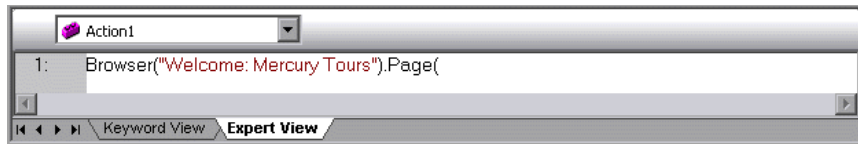
When the **Statement Completion** option is enabled:

- ▶ If you type an object followed by an open parenthesis (, for example, Page(, QuickTest displays a list of all test objects of this type in the object repository. If there is only one object of this type in the object repository, QuickTest automatically enters its name in quotes after the open parenthesis.
- ▶ If you type a period after a test object in a statement, QuickTest displays a list of the relevant test objects, methods, properties, collections, and registered functions that you can add after the object you typed.
- ▶ If you type the name of a method or property, QuickTest displays a list of available methods and properties. Pressing CTRL+SPACE automatically completes the word if there is only one option, or highlights the first method or property (alphabetically) that matches the text you typed.
- ▶ If you type the name of a method or property, QuickTest displays the syntax for it, including its mandatory and optional arguments. When you add a step that uses a method or property, you must define a value for each mandatory argument associated with the method or property.
- ▶ If you press CTRL+SPACE, QuickTest displays a list of the relevant test objects, methods, properties, collections, VBScript functions, user-defined functions, VBScript constants, and utility objects that you can add. This list is displayed even if you typed an object that has not yet been added to the object repository. If the test contains a function, or is associated with a function library, the functions are also displayed in the list.

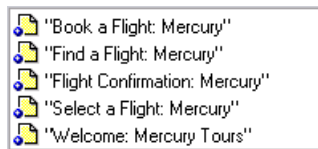
- ▶ If you use the **Object** property in your statement, if the object data is currently available in the Active screen or the open application, QuickTest displays native methods and properties of any run-time object in your application. For more information on the **Object** property, see “Accessing Run-Time Object Properties and Methods” on page 177.

To generate a statement using statement completion in the Expert View or a function library:

- 1** Confirm that the **Statement completion** option is selected (**Tools > View Options > General** tab).
- 2** Perform one of the following:
 - ▶ If you are working in a function library, skip to step 4.
 - ▶ If you are working in the Expert View, type an object followed by an open parenthesis (.



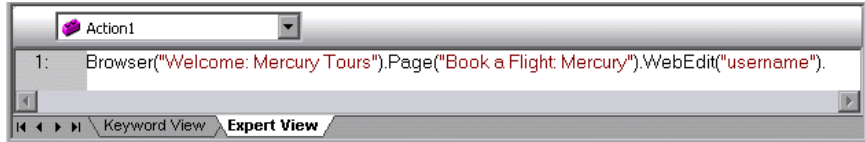
If there is only one object of this type in the object repository, QuickTest automatically enters its name in quotes after the open parenthesis. If more than one object of this type exists in the object repository, QuickTest displays them in a list.



- 3** Double-click an object in the list or use the arrow keys to choose an object and press ENTER. QuickTest inserts the object into the statement.

4 Perform one of the following:

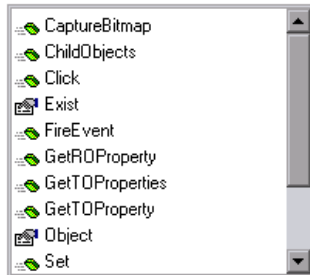
- ▶ If you are working in the Expert View, type a period (.) after the object on which you want to perform the method.



- ▶ If you are working in a function library, type the full hierarchy of an object, for example:

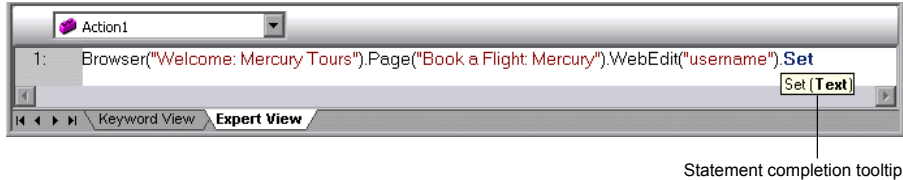
Browser("Welcome: Mercury Tours").Page("Book a Flight:
Mercury").WebEdit("username").

- 5** Type a period (.) after the object description, for example ("username"). QuickTest displays a list of the available methods and properties for the object.



Tip: You can press CTRL+SPACE or choose **Edit > Advanced > Complete Word** after a period, or after you have begun to type a method or property name. QuickTest automatically completes the method or property name if only one method or property matches the text you typed. If more than one method or property matches the text, the first method or property (alphabetically) that matches the text you typed is highlighted.

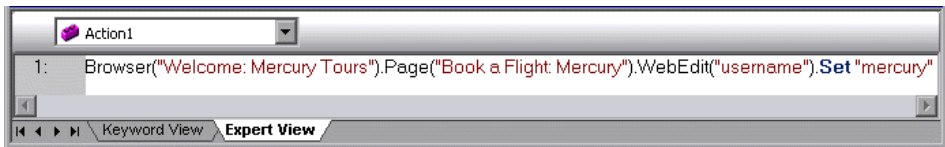
- 6 Double-click a method or property in the list or use the arrow keys to choose a method or property and press ENTER. QuickTest inserts the method or property into the statement. If the method or property contains arguments, QuickTest displays the syntax of the method or property in a tooltip, as shown in this example from the Expert View.



In the above example, the **Set** method has one argument, called **Text**. The argument name represents the text to insert in the box.

Tip: You can also place the cursor on any method or function that contains arguments and press CTRL+SHIFT+SPACE or choose **Edit > Advanced > Argument Info** to display the statement completion (argument syntax) tooltip for that item.

- 7 Enter the method arguments after the method according to the displayed syntax.



Note: After you have added a step in the Expert View, you can view the new step in the Keyword View. If the statement that you added in the Expert View contains syntax errors, QuickTest displays the errors in the Information pane when you select the Keyword View. For more information, see “Handling VBScript Syntax Errors” on page 153.

For more details and examples of any QuickTest method, refer to the *QuickTest Professional Object Model Reference*.

For more information about VBScript syntax, see “Understanding Basic VBScript Syntax” on page 146.

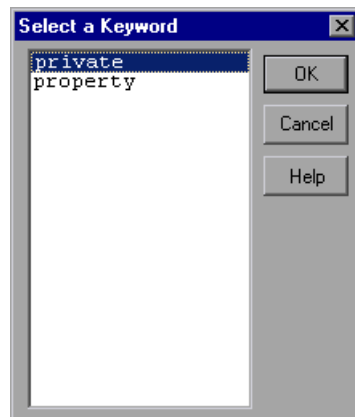
Automatically Completing VBScript Syntax

When the **Auto-expand VBScript syntax** option is enabled and you start to type a VBScript keyword in the Expert View or a function library, QuickTest automatically recognizes the first two characters of the keyword and adds the relevant VBScript syntax or blocks to the script. For example, if you enter the letters `if` and then enter a space at the beginning of a line, QuickTest automatically enters:

```
If Then  
End If
```

The **Auto-expand VBScript syntax** option is enabled by default. You can disable or enable this option in the Editor Options dialog box. For more information, see “Customizing Editor Behavior” on page 335.

If you enter two characters that are the initial characters of multiple keywords, the Select a Keyword dialog box is displayed and you can select the keyword you want. For example, if you enter the letters `pr` and then enter a space, the Select a Keyword dialog box opens containing the keywords `private` and `property`.



You can then select a keyword from the list and click **OK**. QuickTest automatically enters the relevant VBScript syntax or block in the script.

For more information on VBScript syntax, see “Understanding Basic VBScript Syntax” on page 146.

Navigating in the Expert View and Function Libraries

You can use the Go To dialog box or bookmarks to jump to a specific line in the Expert View or a function library. You can also find specific text strings in the Expert View or a function library, and, if desired, replace them with different strings. These options make it easier to navigate through sections of a long action or function.

Note: When working with tests, the Expert View displays only one action. The navigation features described in this section are available only for the currently selected action and not for the entire test.

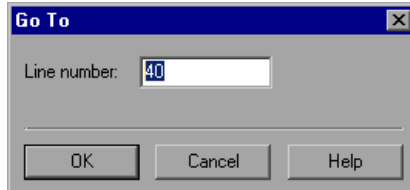
Using the Go To Dialog Box

You can use the Go To dialog box to navigate to a specific line in an action or in a function library.

Tip: By default, line numbers are displayed in the Expert View and in function libraries. If they are not displayed, you can select the **Show line numbers** option in the **Tools > View Options > General** tab. For more information on the Editor options, see Chapter 11, “Customizing the Expert View and Function Library Windows.”

To navigate to a line in the Expert View or a function library using the Go To dialog box:

- 1 Click the Expert View tab or activate a function library.
- 2 Click the **Go To** button, or choose **Edit > Go To**. The Go To dialog box opens.



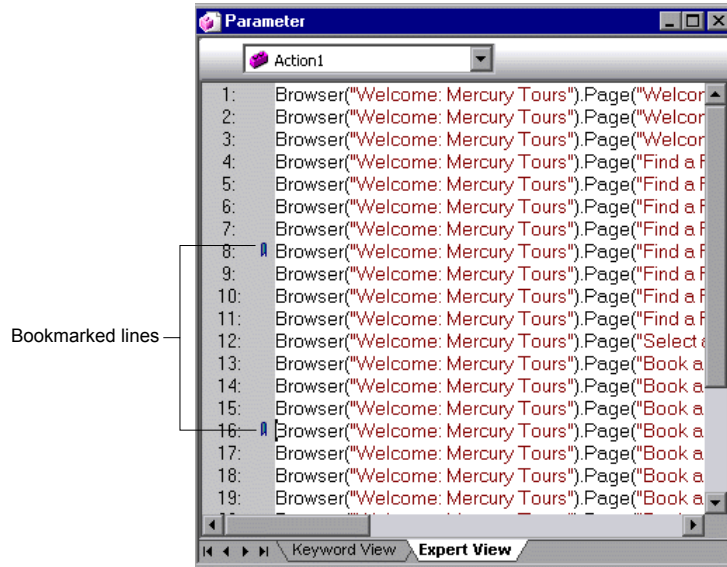
- 3 Enter the line to which you want to navigate in the **Line number** box and click **OK**. The cursor moves to the beginning of the line you specify.

Working with Bookmarks

You can use bookmarks to mark important sections in your action or function library so that you can easily navigate between the various parts. In tests, bookmarks apply only within a specific action; they are not preserved when you navigate between actions and they are not saved with the test or function library.

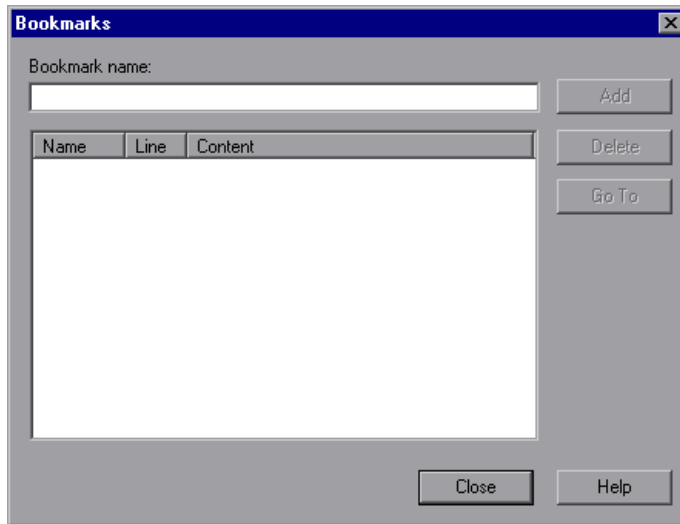
When you assign a bookmark, an icon is added to the left of the selected line in the Expert View or function library. You can then use the **Go To** button in the Bookmarks dialog box to jump to the bookmarked rows.


Bookmarks look the same in tests and in function libraries. In the following example, two bookmarks have been added to an action in a test.



To set bookmarks:

- 1 Click the **Expert View** tab or activate a function library.
- 2 Click in the line to which you want to assign a bookmark.
- 3 Choose **Edit > Bookmarks**. The Bookmarks dialog box opens.



- 4 In the **Bookmark name** field, enter a unique name for the bookmark and click **Add**. The bookmark is added to the Bookmarks dialog box, together with the line number at which it is located and the textual content of the line. In addition, a bookmark icon  is added to the left of the selected line in the Expert View or function library.
- 5 To delete a bookmark, select it in the list and click **Delete**.

To navigate to a specific bookmark:

- 1** Click the **Expert View** tab or activate a function library.
- 2** Choose **Edit > Bookmarks**. The Bookmarks dialog box opens.
- 3** Select a bookmark from the list and click the **Go To** button. QuickTest jumps to the appropriate line in the current action or function library.

Tip: By default, line numbers are displayed in the Expert View and in function libraries. If they are not displayed, you can select the **Show line numbers** option in the **Tools > View Options > General** tab. For more information on the Editor options, see Chapter 11, “Customizing the Expert View and Function Library Windows.”

Finding Text Strings

You can specify text strings to locate in the current action in the Expert View or in a function library. You can also search for strings in the Edit HTML Source and Edit HTML Tags dialog boxes, and in the “With” Generation Results dialog box. You can either search for literal text or use regular expressions for a more advanced search. You can also use other options to further fine-tune your search results.

To find a text string:

- 1 In the Expert View or function library, perform one of the following:



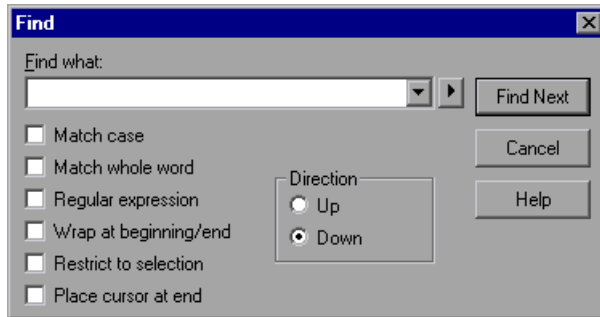
- ▶ Click the **Find** button.
- ▶ Choose **Edit > Find**.


Tip: In the Expert View, you can also perform one of the following:

Choose **Edit > Advanced > Apply “With” to Script**, and then press CTRL+F.

In the Page Checkpoint Properties dialog box, click **Edit HTML Source** or **Edit HTML Tags**, and then right-click and choose **Find** in the displayed dialog box.

The Find dialog box opens.



- 2 In the **Find what** box, enter the text string you want to locate.
- 3 If you want to use regular expressions in the string you specify, click the arrow button  and select a regular expression. When you select a regular expression from the list, it is automatically inserted in the **Find what** box at the cursor location. For more information, see “Using Regular Expressions in the Find and Replace Dialog Boxes” on page 145.

- 4 Select any of the following options to help fine-tune your search:
 - **Match case**—Distinguishes between upper-case and lower-case characters in the search. When **Match case** is selected, QuickTest finds only those occurrences in which the capitalization matches the text you entered in the **Find what** box exactly.
 - **Match whole word**—Searches for occurrences that are only whole words and not part of longer words.
 - **Regular expression**—Treats the specified text string as a regular expression. This option is automatically selected when you select a regular expression from the list.
 - **Wrap at beginning/end**—Continues the search from the beginning or end of the action, dialog box, or function library text when either the beginning or end is reached, depending on the selected search direction.
 - **Restrict to selection**—Searches only within the selected part of the action, dialog box, or function library text.
 - **Place cursor at end**—Places the cursor at the end of the highlighted occurrence when the search string is located.
- 5 Specify the direction in which you want to search, from the current cursor location in the action, dialog box, or function library: **Up** or **Down**
- 6 Click **Find Next** to highlight the next occurrence of the specified string in the current action or dialog box, or in the active function library.

Replacing Text Strings

You can specify text strings to locate in the current action in the Expert View or function library, and specify the text strings you want to use to replace them. You can also search and replace strings in the Edit HTML Source and Edit HTML Tags dialog boxes. You can either find and replace literal text or use regular expressions for a more advanced process. You can also use other options to further fine-tune your find and replace process.

To replace a text string:

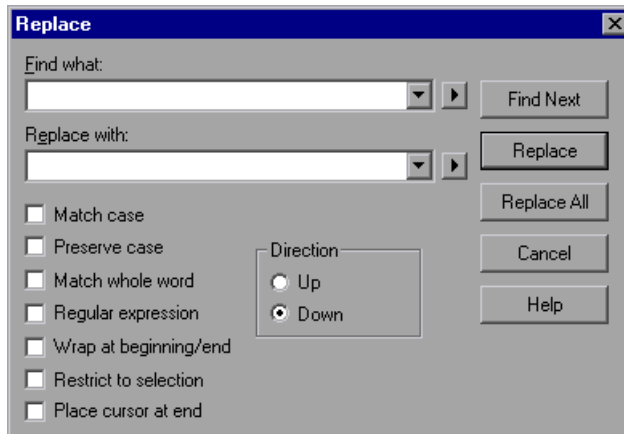
- 1 In the Expert View or function library, perform one of the following:




- ▶ Click the **Replace** button.
- ▶ Choose **Edit > Replace**.

Tip: (For tests only) In the Page Checkpoint Properties dialog box, click **Edit HTML Source** or **Edit HTML Tags**, and then right-click and choose **Replace** in the displayed dialog box.

The Replace dialog box opens.




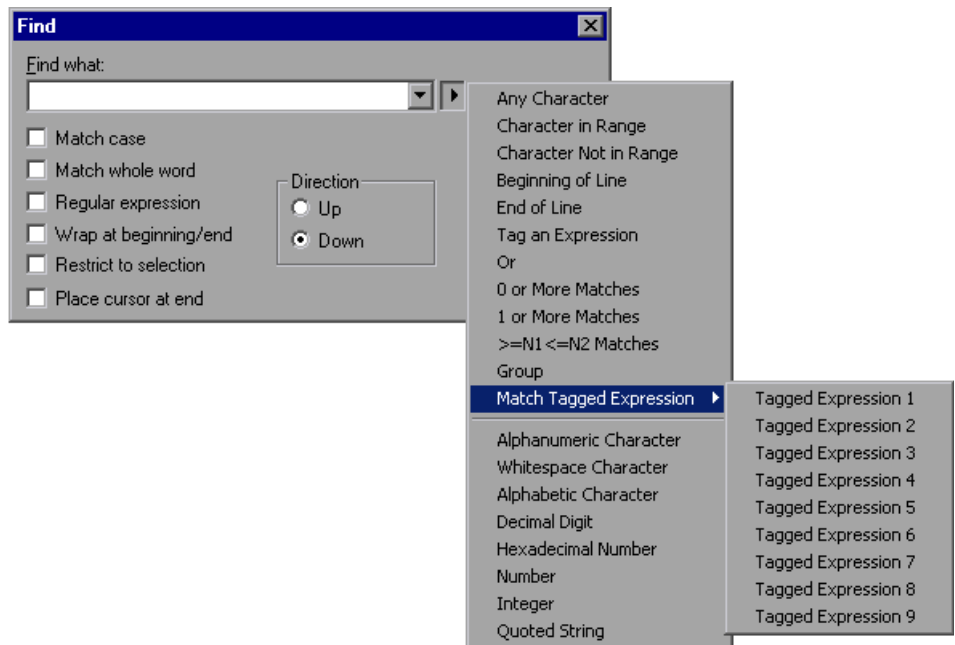
- 2 In the **Find what** box, enter the text string you want to locate.
- 3 In the **Replace with** box, enter the text string you want to use to replace the found text.
- 4 If you want to use regular expressions in the **Find what** or **Replace with** string, click the arrow button  and select a regular expression. When you select a regular expression from the list, it is automatically inserted in the **Find what** or **Replace with** box at the cursor location. For more information, see “Using Regular Expressions in the Find and Replace Dialog Boxes” on page 145.

- 5 Select any of the following options to help fine-tune your search:
 - ▶ **Match case**—Distinguishes between upper-case and lower-case characters in the search. When **Match case** is selected, QuickTest finds only those occurrences in which the capitalization exactly matches the text you entered in the **Find what** box.
 - ▶ **Preserve case**—Checks each occurrence of the **Find what** string for all lowercase, all uppercase, sentence caps or mixed case. The **Replace with** string is converted to the same case as the occurrence found, except when the occurrence found is mixed case. In this case, the **Replace with** string is used without modification.
 - ▶ **Match whole word**—Searches for occurrences that are whole words only and not part of longer words.
 - ▶ **Regular expression**—Treats the specified text string as a regular expression. This option is automatically selected when you select a regular expression from the list.
 - ▶ **Wrap at beginning/end**—Continues the search from the beginning or end of the action, dialog box, or function library text when either the beginning or end is reached, depending on the selected search direction.
 - ▶ **Restrict to selection**—Searches only within the selected part of the action, dialog box, or function library text.
 - ▶ **Place cursor at end**—Places the cursor at the end of the highlighted occurrence when the search string is located.
- 6 Click **Find Next** to highlight the next occurrence of the specified text string in the current action or dialog box, or in the active function library.
- 7 Click **Replace** to replace the highlighted text with the text in the **Replace with** box, or click **Replace All** to replace all occurrences specified in the **Find what** box with the text in the **Replace with** box in the current action or dialog box, or in the active function library.

Using Regular Expressions in the Find and Replace Dialog Boxes

You can use regular expressions in the **Find what** and **Replace with** strings to enhance your search. For a general understanding of regular expressions, refer to “Understanding and Using Regular Expressions” on page 334 in the *QuickTest Professional Basic Features User’s Guide*. Note that there are differences in the expressions supported by the Find and Replace dialog boxes and the expressions supported in other parts of QuickTest.

You display the regular expressions available for selection by clicking the arrow button  in the Find or Replace dialog boxes.



You can select from a predefined list of regular expressions. You can also use tagged expressions. When you use regular expressions to search for a string, you may want the string to change depending on what was already found.

For example, you can search for **(save\n)\1**, which will find any occurrence of **save** followed by any number, immediately followed by **save**, as well as the same number that was already found (meaning that it will find **save6save6** but not **save6save7**).

You can also use tagged expressions to insert parts of what is found into the replace string. For example, you can search for **save(\:n)** and replace it with **open\1**. This will find **save** followed by any number, and replace it with **open** and the number that was found.

Select **Tag an Expression** from the regular expressions list to insert parentheses "(" to indicate a tagged expression in the search string.

Select **Match Tagged Expression** and then select the specific tag group number to specify the tagged expression you want to use, in the format '\' followed by a tag group number 1-9. (Count the left parentheses '(' in the search string to determine a tagged expression number. The first (left-most) tagged expression is "\1" and the last is "\9".)

Understanding Basic VBScript Syntax

VBScript is an easy-to-learn, yet powerful scripting language. You can use VBScript to develop scripts to perform both simple and complex object-based tasks, even if you have no previous programming experience.

This section provides some basic guidelines to help you use VBScript statements to enhance your QuickTest test or function library. For more detailed information about using VBScript, you can view the VBScript documentation from the QuickTest Help menu (**Help > QuickTest Professional Help > VBScript Reference**).

Each VBScript statement has its own specific syntax rules. If you do not follow these rules, errors will be generated when you run the problematic step. Additionally, if you try to move to the Keyword View from the Expert View, QuickTest lists any syntax errors found in the document in the Information pane. You cannot switch to the Keyword View without fixing or eliminating the syntax errors. For more information, see “Handling VBScript Syntax Errors” on page 153.



Tip: You can check the syntax of the current document at any time by clicking the **Check Syntax** button, or by choosing **Tools > Check Syntax**. If a test is open, the syntax of all the actions is checked. If a function library is open, the syntax of the library script is checked.

When working in the Expert View or in a function library, you should consider the following general VBScript syntax rules and guidelines:

- **Case-sensitivity**—By default, VBScript is not case sensitive and does not differentiate between upper-case and lower-case spelling of words, for example, in variables, object and method names, or constants.

For example, the two statements below are identical in VBScript:

```
Browser("Mercury").Page("Find a Flight:").WebList("toDay").Select "31"
browser("mercury").page("find a flight:").weblist("today").select "31"
```

- **Text strings**—When you enter a value as a text string, you must add quotation marks before and after the string. For example, in the above segment of script, the names of the Web site, Web page, and edit box are all text strings surrounded by quotation marks.

Note that the value 31 is also surrounded by quotation marks, because it is a text string that represents a number and not a numeric value.

In the following example, only the property name (first argument) is a text string and is in quotation marks. The second argument (the value of the property) is a variable and therefore does not have quotation marks. The third argument (specifying the timeout) is a numeric value, which also does not need quotation marks.

```
Browser("Mercury").Page("Find a Flight:").WaitProperty("items count",  
    Total_Items, 2000)
```

- ▶ **Variables**—You can specify variables to store strings, integers, arrays and objects. Using variables helps to make your script more readable and flexible. For more information, see “Using Variables,” below.
- ▶ **Parentheses**—To achieve the desired result and to avoid errors, it is important that you use parentheses () correctly in your statements. For more information, see “Using Parentheses” on page 150.
- ▶ **Indentation**—You can indent or outdent your script to reflect the logical structure and nesting of the statements. For more information, see “Formatting VB Script Text” on page 151.
- ▶ **Comments**—You can add comments to your statements using an apostrophe ('), either at the beginning of a separate line, or at the end of a statement. It is recommended that you add comments wherever possible, to make your scripts easier to understand and maintain. For more information, see “Formatting VB Script Text” on page 151, and “Inserting Comments” on page 167.
- ▶ **Spaces**—You can add extra blank spaces to your script to improve clarity. These spaces are ignored by VBScript.

For more information on using specific VBScript statements to enhance your tests or function libraries, see “Using Comments, Control-Flow, and Other VBScript Statements” on page 167.

Using Variables

You can specify variables to store test objects or simple values in your test or function library. When using a variable for a test object, you can use the variable instead of the entire object hierarchy in other statements. Using variables in this way makes your statements easier to read and to maintain.

To specify a variable to store an object, use the **Set** statement, with the following syntax:

Set *ObjectVar* = *ObjectHierarchy*

In the example below, the **Set** statement specifies the variable `UserEditBox` to store the full `Browser > Page > WebEdit` object hierarchy for the **username** edit box. The **Set** method then enters the value `John` into the **username** edit box, using the `UserEditBox` variable:

```
Set UserEditBox = Browser("Mercury Tours").Page("Mercury Tours").
    WebEdit("username")
UserEditBox.Set "John"
```

Note: Do not use the **Set** statement to specify a variable containing a simple value (such as a string or a number). The example below shows how to define a variable for a simple value:

```
MyVar = Browser("Mercury Tours").Page("Mercury Tours").
    WebEdit("username").GetTOPProperty("type")
```

You can also use the **Dim** statement to declare variables of other types, including strings, integers, and arrays. This statement is not mandatory, but you can use it to improve the structure of your test or function library. In the following example, the **Dim** statement is used to declare the `passengers` variable, which can then be used in different statements within the current action or function library:

```
Dim passengers
passengers = Browser("Mercury Tours").Page("Find Flights").
    WebEdit("numpassengers").GetROProperty("value")
```

Using Parentheses

When programming in VBScript, it is important that you follow the rules for using or not using parentheses () in your statements.

You must use parentheses around method arguments if you are calling a method that returns a value and you are using the return value.

For example, use parentheses around method arguments if you are returning a value to a variable, if you are using the method in an **If** statement, or if you are using the **Call** keyword to call an action or function. You also need to add parentheses around the name of a checkpoint if you want to retrieve its return value.

Tip: If you receive an **Expected end of statement** error message when running a step in your test or function library, it may indicate that you need to add parentheses around the arguments of the step's method.

Following are several examples showing when to use or not use parentheses.

The following example requires parentheses around the method arguments for the **ChildItem** method because it returns a value to a variable.

```
Set WebEditObj = Browser("Mercury Tours").Page("Method of Payment").  
    WebTable("FirstName").ChildItem (8, 2, "WebEdit", 0)  
WebEditObj.Set "Example"
```

The following example requires parentheses around the method arguments because **Call** is being used.

```
Call RunAction("BookFlight", oneliteration)
```

or

```
Call MyFunction("Hello World")
```

...

...

The following example requires parentheses around the **WaitProperty** method arguments because the method is used in an **If** statement.

```
If Browser("index").Page("index").Link("All kind of").  
    WaitProperty("attribute/readyState", "complete", 4) Then  
    Browser("index").Page("index").Link("All kind of").Click  
End If
```

The following example requires parentheses around the **Check** method arguments, since it returns the value of the checkpoint.

```
a = Browser("MyBrowser").Page("MyPage").Check (CheckPoint("MyProperty"))
```

The following example does not require parentheses around the **Click** method arguments because it does not return a value.

```
Browser("Mercury Tours").Page("Method of Payment").WebTable("FirstName").  
    Click 3,4
```

Formatting VB Script Text

When working in the Expert View or in a function library, it is important to follow accepted VBScript practices for comments and indentation.

Use comments to explain sections of a script. This improves readability and make tests and function libraries easier to maintain and update. For more information, see “Inserting Comments” on page 167.

Use indentation to reflect the logical structure and nesting of your statements.

- ▶ **Adding Comments**—You can add comments to your statements by adding an apostrophe ('), either at the beginning of a separate line, or at the end of a statement.

Tips:



You can comment a statement by clicking anywhere in the statement and clicking the **Comment Block** button.

You can comment a selected block of text by clicking the **Comment Block** button, or by choosing **Edit > Advanced > Comment Block**. Each line in the block will be preceded by an apostrophe.

-
- ▶ **Removing Comments**—You can remove comments from your statements by deleting the apostrophe ('), either at the beginning of a separate line, or at the end of a statement.



Tip: You can remove the comments from a selected block or line of text by clicking the **Uncomment Block** button, or by choosing **Edit > Advanced > Uncomment Block**.



- ▶ **Indenting Statements**—You can indent your statements by selecting the statements and clicking the **Indent** button. Alternatively, you can select text and choose **Edit > Advanced > Indent** or press the TAB key. The text is indented according to the tab spacing selected in the Editor Options dialog box, as described in “Customizing Editor Behavior” on page 335.

Note: The **Indent selected text when using the Tab key** check box must be selected in the Editor Options dialog box, otherwise pressing the TAB key will delete the selected text.



- **Outdenting Statements**—You can outdent your statements by selecting the statement and clicking the **Outdent** button. Alternatively, you can choose **Edit > Advanced > Outdent** or you can delete the space at the beginning of the statements.

For more detailed information about formatting in VBScript, you can view the VBScript documentation from the QuickTest Help menu (**Help > QuickTest Professional Help > VBScript Reference**).

Handling VBScript Syntax Errors

When you select the Keyword View tab from the Expert View, QuickTest attempts to display the updated information in the Keyword View. If a new or updated VBScript statement contains syntax errors, the text **Error** flashes in red at the right of the status bar, and an error message is displayed in the status bar informing you that you should view the Information pane for information about syntax errors in the script. QuickTest is unable to display the document in the Keyword View until you have fixed all the syntax errors.

You can view a description of each of the VBScript errors in the VBScript Reference. For more information, choose **Help > QuickTest Professional Help > VBScript Reference > VBScript > Reference > Errors > VBScript Syntax Errors**.

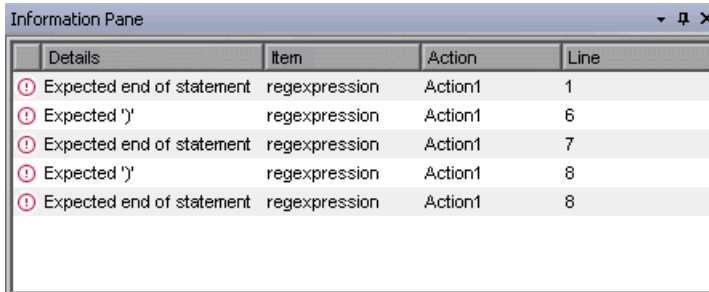
Tips:



You can check the syntax of the current document at any time by clicking the **Check Syntax** button, or by choosing **Tools > Check Syntax**. If a test is open, the syntax of all the actions is checked. If a function library is open, the syntax of the library script is checked.

The Microsoft VBScript Language Reference defines VBScript syntax errors as: “errors that result when the structure of one of your VBScript statements violates one or more of the grammatical rules of the VBScript scripting language”. To learn about working with VBScript, you can view the VBScript Reference from the QuickTest Help menu (**Help > QuickTest Professional Help > VBScript Reference**).

The Information pane lists the syntax errors found in your document, and enables you to locate each syntax error so that you can correct it.



The Information pane shows the following information for each syntax error:

Pane Element	Description
Details	The description of the syntax error. For example, if you opened a conditional block with an If statement but did not close it with an End If statement, the description is Expected 'End If' . Note: In certain cases, QuickTest is unable to identify the exact error and displays a number of possible error conditions, for example: Expected 'End Sub' , or 'End Function' , or 'End Property' . Check the statement at the specified line to clarify which error is relevant in your case.
Item	The name of the test or function library containing the problematic statement.
Action	The name of the action containing the problematic statement. This column is not relevant for function libraries that are associated with business components (via application areas).
Line	The line containing the syntax error. Lines are numbered from the beginning of each action or function library.

Using the Information Pane

- ▶ Hold your mouse over the description of a syntax error to display the currently incorrect syntax.
- ▶ To navigate to the line containing a specific syntax error, double-click the syntax error in the Information pane.
- ▶ You can resize the columns in the Information pane to make the information more readable by dragging the column headers.
- ▶ You can sort the details in the Information pane in ascending or descending order by clicking the column header.
- ▶ You can press F1 on an error in the Information pane to display information about VBScript syntax errors.

Using Programmatic Descriptions

When you record an operation on an object, QuickTest adds the appropriate test object to the object repository. Once the object exists in the object repository, you can add statements in the Expert View to perform additional methods on that object. To add these statements, you usually enter the name (not case sensitive) of each of the objects in the object's hierarchy as the object description, and then add the appropriate method.

For example, in the statement below, `username` is the name of an edit box. The edit box is located on a page with the name `Mercury Tours` and the page was recorded in a browser with the name `Mercury Tours`.

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

Because each object in the object repository has a unique name, the object name is all you need to specify. During the run session, QuickTest finds the object in the object repository based on its name and parent objects, and uses the stored test object description for that test object to identify the object in your Web site or application.

You can also instruct QuickTest to perform methods on objects without referring to the object repository or to the object's 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 run session.

For example, suppose you are testing a Web site that generates a list of potential employers based on biographical information you provide, and offers to send your resume to the employer names you select from the list. You want your test to select all the employers displayed in the list, but when you design your test, you do not know how many check boxes will be displayed on the page, and you cannot, of course, know the exact object description of each check box. In this situation, you can use a programmatic description to instruct QuickTest to perform a Set "ON" method for all objects that fit the description: HTML TAG = input, TYPE = check box.

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.

Entering programmatic descriptions directly into your statements may be the easier method for basic object description needs. However, in most cases, the Description object method is more powerful and more efficient.

Entering Programmatic Descriptions Directly into Statements

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

The general syntax is:

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

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.

Note that you can enter a variable name as the property value if you want to find an object based on property values you retrieve during a run session.

Note: QuickTest evaluates all property values in programmatic descriptions as regular expressions. Therefore, if you want to enter a value that contains a special regular expression character (such as *, ?, or +), use the \ (backslash) character to instruct QuickTest to treat the special characters as literal characters. For more information on regular expressions, refer to “Understanding and Using Regular Expressions” on page 334 in the *QuickTest Professional Basic Features User’s Guide*.

The statement below specifies a WebEdit test object in the Mercury Tours page with the Name author and an index of 3. During the run session, QuickTest finds the WebEdit object with matching property values and enters the text Mark Twain.

```
Browser("Mercury Tours").Page("Mercury Tours").WebEdit("Name:=Author",  
    "Index:=3").Set "Mark Twain"
```

Note: When using programmatic descriptions from a specific point within a test object hierarchy, you must continue to use programmatic descriptions from that point onward within the same statement. If you specify a test object by its object repository name after other objects in the hierarchy have been specified using programmatic descriptions, QuickTest cannot identify the object.

For example, you can use the following statement since it uses programmatic descriptions throughout the entire test object hierarchy:

```
Browser("Title:=Mercury Tours").Page("Title:=Mercury Tours").  
    WebEdit("Name:=Author", "Index:=3").Set "Mark Twain"
```

You can also use the statement below, since it uses programmatic descriptions from a certain point in the description (starting from the Page object description):

```
Browser("Mercury Tours").Page("Title:=Mercury Tours").  
    WebEdit("Name:=Author", "Index:=3").Set "Mark Twain"
```

However, you cannot use the following statement, since it uses programmatic descriptions for the Browser and Page objects but then attempts to use an object repository name for the WebEdit test object:

```
Browser("Title:=Mercury Tours").Page("Title:=Mercury Tours").  
    WebEdit("Author").Set "Mark Twain"
```

QuickTest tries to locate the WebEdit object based on its name, but cannot locate it in the repository because the parent objects were specified using programmatic descriptions.

For more information on working with test objects, refer to Chapter 6, “Working with Test Objects” in the *QuickTest Professional Basic Features User’s Guide*.

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

For example, instead of entering:

```
Window("Text:=Myfile.txt - Notepad").Move 50, 50
Window("Text:=Myfile.txt - Notepad").WinEdit("AttachedText:=Find what:").
    Set "hello"
Window("Text:=Myfile.txt - Notepad").WinButton("Caption:=Find next").Click
```

You can enter:

```
Set MyWin = Window("Text:=Myfile.txt - Notepad")
MyWin.Move 50, 50
MyWin.WinEdit("AttachedText:=Find what:").Set "hello"
MyWin.WinButton("Caption:=Find next").Click
```

Alternatively, you can use a **With** statement:

```
With Window("Text:=Myfile.txt - Notepad")
    .Move 50, 50
    .WinEdit("AttachedText:=Find what:").Set "hello"
    .WinButton("Caption:=Find next").Click
End With
```

For more information about the **With** statement, see “With Statement” on page 174.

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** collection in place of an object name in a statement. (Each property object contains a property name and value pair.)

Note: By default, the value of all **Property** objects added to a **Properties** collection are treated as regular expressions. Therefore, if you want to enter a value that contains a special regular expression character (such as *, ?, +), use the \ (backslash) character to instruct QuickTest to treat the special characters as literal characters. For more information about regular expressions, refer to “Understanding and Using Regular Expressions” on page 334 in the *QuickTest Professional Basic Features User’s Guide*.

You can set the **RegularExpression** property to **False** in order to specify a value as a literal value for a specific **Property** object in the collection. For more information, refer to the **Utility** section of the *QuickTest Professional Object Model Reference*.

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 run session. This enables you to determine which, and how many properties to include in the object description in a dynamic way during the run session.

After you fill the **Properties** collection with a set of **Property** objects (properties and values), you can specify the **Properties** object in place of an object 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
```

Tip: When creating a programmatic description for an ActiveX test object and the relevant run-time object is windowless (has no window handle associated with it), you must add the **windowless** property to the description and set its value to **True**.

For example:

```
Set ButDesc = Description.Create  
ButDesc("ProgId").Value = "Forms.CommandButton.1"  
ButDesc("Caption").Value = "OK"  
ButDesc("Windowless").Value = True  
Window("Form1").AcxButton(ButDesc).Click
```

Note: When using programmatic descriptions from a specific point within a test object hierarchy, you must continue to use programmatic descriptions from that point onward within the same statement. If you specify a test object by its object repository name after other objects in the hierarchy have been described using programmatic descriptions, QuickTest cannot identify the object.

For example, you can use `Browser(Desc1).Page(Desc1).Link(desc3)`, since it uses programmatic descriptions throughout the entire test object hierarchy.

You can also use `Browser("Index").Page(Desc1).Link(desc3)`, since it uses programmatic descriptions from a certain point in the description (starting from the Page object description).

However, you cannot use `Browser(Desc1).Page(Desc1).Link("Example1")`, since it uses programmatic descriptions for the Browser and Page objects but then attempts to use an object repository name for the Link test object (QuickTest tries to locate the Link object based on its name, but cannot locate it in the repository because the parent objects were specified using programmatic descriptions).

When working with **Properties** objects, you can use variable names for the properties or values in order to generate the object description based on properties and values you retrieve during a run session.

You can create several **Properties** objects in your test if you want to use programmatic descriptions for several objects.

For more information on the **Description** and **Properties** objects and their associated methods, refer to the *QuickTest Professional Object Model Reference*.

Retrieving Child Objects

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
```

For more information about the **ChildObjects** method, refer to the *QuickTest Professional Object Model Reference*.

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`.

For more information about the WebElement object, refer to the *QuickTest Professional Object Model Reference*.

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.

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")
```

Note: If there is only one object, using *index=0* will not retrieve it. You should not include the **index** property in the object description.

Running and Closing Applications Programmatically

In addition to using the Record and Run dialog box to instruct QuickTest to open a new browser or application when a test run begins, and/or opening the application you want to test manually, you can also insert statements into your test that open and close the applications you want to test.

You can run any application from a specified location using a **SystemUtil.Run** statement. This is especially useful if your test includes more than one application, and you selected the **Record and run test on any application** check box in the Record and Run Settings dialog box. You can specify an application and pass any supported parameters, or you can specify a file name and the associated application starts with the specified file open.

You can close most applications using the **Close** method.

For example, you could use the following statements to open a file named **type.txt** in the default text application (Notepad), type **happy days**, save the file using shortcut keys, and then close the application:

```
SystemUtil.Run "C:\type.txt", "", "", ""  
Window("Text:=type.txt - Notepad").Type "happy days"  
Window("Text:=type.txt - Notepad").Type micAltDwn & "F" & micAltUp  
Window("Text:=type.txt - Notepad").Type micLShiftDwn & "S" & micLShiftUp  
Window("Text:=type.txt - Notepad").Close
```

Notes:

When you specify an application to open using the Record and Run Settings dialog box, QuickTest does not add a **SystemUtil.Run** statement to your test.

The **InvokeApplication** method can open only executable files and is used primarily for backward compatibility.

For more information, refer to the *QuickTest Professional Object Model Reference*.

Using Comments, Control-Flow, and Other VBScript Statements

QuickTest enables you to incorporate decision-making into your test or function library by adding conditional statements that control the logical flow of your test or function library. In addition, you can define messages in your test that QuickTest sends to your test results. To improve the readability of your tests and function libraries, you can also add comments to them.

For information on how to use these programming concepts in the Keyword View, refer to Chapter 20, “Adding Steps Containing Programming Logic” in the *QuickTest Professional Basic Features User’s Guide*.

Note: The **VBScript Reference** (available from **Help > QuickTest Professional Help**) contains Microsoft VBScript documentation, including VBScript, Script Runtime, and Windows Script Host.

Inserting Comments

A comment is a line or part of a line in a script that is preceded by an apostrophe ('). When you run a test, QuickTest does not process comments. Use comments to explain sections of a script in order to improve readability and to make tests and function libraries easier to update.

The following example shows how a comment describes the purpose of the statement below it:

```
'Sets the word "mercury" into the "username" edit box.  
Browser("Mercury Tours").Page("Mercury Tours").WebEdit("username").  
    Set "mercury"
```

By default, comments are displayed in green in the Expert View and in function libraries. You can customize the appearance of comments in the Editor Options dialog box. For more information, see “Customizing Element Appearance” on page 337.

Tips:



You can comment a block of text by choosing **Edit > Advanced > Comment Block** or by clicking the **Comment Block** button.



To remove the comment, choose **Edit > Advanced > Uncomment Block** or click the **Uncomment Block** button.

Note: You can also add a comment line using VBScript’s **Rem** statement. For additional information, refer to the Microsoft VBScript Language Reference (choose **Help > QuickTest Professional Help > VBScript Reference > VBScript**).

Performing Calculations

You can write statements that perform simple calculations using mathematical operators. For example, you can use a multiplication operator to multiply the values displayed in two text boxes in your site. VBScript supports the following mathematical operators:

Operator	Description
+	addition
-	subtraction
-	negation (a negative number—unary operator)
*	multiplication
/	division
^	exponent

In the following example, the multiplication operator is used to calculate the maximum luggage weight of the passengers at 100 pounds each:

'Retrieves the number of passengers from the edit box using the GetROProperty method

```
passenger = Browser("Mercury_Tours").Page("Find_Flights").
    WebEdit("numPassengers").GetROProperty("value")
```

'Multiplies the number of passengers by 100

```
weight = passenger * 100
```

'Inserts the maximum weight into a message box.

```
msgbox("The maximum weight for the party is "& weight &"pounds.")
```

For...Next Statement

A **For...Next** loop instructs QuickTest to execute one or more statements a specified number of times. It has the following syntax:

```
For counter = start to end [Step step]
    statement
Next
```

Item	Description
<i>counter</i>	The variable used as a counter for the number of iterations.
<i>start</i>	The start number of the counter.
<i>end</i>	The last number of the counter.
<i>step</i>	The number to increment at the end of each loop. Default = 1. Optional.
<i>statement</i>	A statement, or series of statements, to be executed during the loop.

In the following example, QuickTest calculates the factorial value of the number of passengers using the **For** statement:

```
passengers = Browser("Mercury Tours").Page("Find Flights").
    WebEdit("numPassengers").GetROProperty("value")
total = 1
For i=1 To passengers
    total = total * i
Next
MsgBox "!" & passengers & "=" & total
```


For...Each Statement

A **For...Each** loop instructs QuickTest to execute one or more statements for each element in an array or an object collection. It has the following syntax:

```
For Each item In array
    statement
Next
```

Item	Description
<i>item</i>	A variable representing the element in the array.
<i>array</i>	The name of the array.
<i>statement</i>	A statement, or series of statements, to be executed during the loop.

The following example uses a **For...Each** loop to display each of the values in an array:

```
MyArray = Array("one", "two", "three", "four", "five")
For Each element In MyArray
    msgbox element
Next
```

Do...Loop Statement

The **Do...Loop** statement instructs QuickTest to execute a statement or series of statements while a condition is true or until a condition becomes true. It has the following syntax:

```
Do [{while} {until} condition]
    statement
Loop
```

Item	Description
<i>condition</i>	A condition to be fulfilled.
<i>statement</i>	A statement or series of statements to be executed during the loop.

In the following example, QuickTest calculates the factorial value of the number of passengers using the **Do...Loop**:

```

passengers = Browser("Mercury Tours").Page("Find Flights").
    WebEdit("numPassengers").GetROProperty("value")
total = 1
i = 1
Do while i <= passengers
    total = total * i
    i = i + 1
Loop
MsgBox "!" & passengers & "=" & total
    
```

While...Wend Statement

A **While...Wend** statement instructs QuickTest to execute a statement or series of statements while a condition is true. It has the following syntax:

```

While condition
    statement
Wend
    
```

Item	Description
<i>condition</i>	A condition to be fulfilled.
<i>statement</i>	A statement or series of statements to be executed during the loop.

In the following example, QuickTest performs a loop using the **While** statement while the number of passengers is fewer than ten. Within each loop, QuickTest increments the number of passengers by one:

```

passengers = Browser("Mercury Tours").Page("Find Flights").
    WebEdit("numpassengers").GetROProperty("value")
While passengers < 10
    passengers = passengers + 1
Wend
msgbox("The number of passengers in the party is " & passengers)
    
```

If...Then...Else Statement

The **If...Then...Else** statement instructs QuickTest to execute a statement or a series of statements based on specified conditions. If a condition is not fulfilled, the next **Elseif** condition or **Else** statement is examined. It has the following syntax:

```
If condition Then
    statement
Elseif condition2 Then
    statement
Else
    statement
End If
```

Item	Description
<i>condition</i>	Condition to be fulfilled.
<i>statement</i>	Statement to be executed.

In the following example, if the number of passengers is fewer than four, QuickTest closes the browser:

```
passengers = Browser("Mercury Tours").Page("Find Flights").
    WebEdit("numpassengers").GetROProperty("value")
If (passengers < 4) Then
    Browser("Mercury Tours").Close
Else
    Browser("Mercury Tours").Page("Find Flights").Image("continue").Click 69,5
End If
```

The following example, uses **If**, **ElseIf**, and **Else** statements to check whether a value is equal to 1, 2, or a different value:

```
value = 2
If value = 1 Then
    msgbox "one"
ElseIf value = 2 Then
    msgbox "two"
Else
    msgbox "not one or two"
End If
```

With Statement

With statements make your script more concise and easier to read and write or edit by grouping consecutive statements with the same parent hierarchy.

Note: Applying **With** statements to your script has no effect on the run session itself, only on the way your script appears in the Expert View.

The **With** statement has the following syntax:

```
With object
    statements
End With
```

Item	Description
<i>object</i>	An object or a function that returns an object.
<i>statements</i>	One or more statements to be executed on an object.

For example, you could replace this script:

```
Window("Flight Reservation").WinComboBox("Fly From:").Select "London"
Window("Flight Reservation").WinComboBox("Fly To:").Select "Los Angeles"
Window("Flight Reservation").WinButton("FLIGHT").Click
Window("Flight Reservation").Dialog("Flights Table").WinList("From").
    Select "19097 LON "
Window("Flight Reservation").Dialog("Flights Table").WinButton("OK").Click
```

with the following:

```
With Window("Flight Reservation")
    .WinComboBox("Fly From:").Select "London"
    .WinComboBox("Fly To:").Select "Los Angeles"
    .WinButton("FLIGHT").Click
With .Dialog("Flights Table")
    .WinList("From").Select "19097 LON "
    .WinButton("OK").Click
End With 'Dialog("Flights Table")
End With 'Window("Flight Reservation")
```

Note that entering **With** statements in the Expert View does not affect the Keyword View in any way.

Note: In addition to entering **With** statements manually, you can also instruct QuickTest to automatically generate **With** statements as you record or to generate **With** statements for an existing test. For more information, refer to “Generating With Statements for Your Test” on page 556 in the *QuickTest Professional Basic Features User’s Guide*.

Retrieving and Setting Test Object Property Values

Test object properties are the set of properties defined by QuickTest for each object. You can set and retrieve a test object's property values, and you can retrieve the values of test object properties from a run-time object.

When you run your test, QuickTest creates a temporary version of the test object that is stored in the test object repository. You can use the **GetTOProperty**, **GetTOProperties**, and **SetTOProperty** methods in your test or function library to set and retrieve the test object property values of the test object.

The **GetTOProperty** and **GetTOProperties** methods enable you to retrieve a specific property value or all the properties and values that QuickTest uses to identify an object.

The **SetTOProperty** method enables you to modify a property value that QuickTest uses to identify an object.

Note: Because QuickTest refers to the temporary version of the test object during the run session, any changes you make using the **SetTOProperty** method apply only during the course of the run session, and do not affect the values stored in the test object repository.

For example, the following statements would set the **Submit** button's name value to my button, and then retrieve the value my button to the **ButtonName** variable:

```
Browser("QA Home Page").Page("QA Home Page").  
    WebButton("Submit").SetTOProperty "Name", "my button"  
  
ButtonName=Browser("QA Home Page").Page("QA Home Page").  
    WebButton("Submit").GetTOProperty("Name")
```

You use the **GetROProperty** method to retrieve the current value of a test object property from a run-time object in your application.

For example, you can retrieve the target value of a link during the run session as follows:

```
link_href = Browser("Mercury Technologies").Page("Mercury Technologies").  
    Link("Jobs").GetROProperty("href")
```

Tip: If you do not know the test object properties of objects in your Web site or application, you can view them using the Object Spy. For information on the Object Spy, refer to Chapter 3, “Understanding the Test Object Model” in the *QuickTest Professional Basic Features User’s Guide*.

For a list and description of test object properties supported by each object, and for additional information about the **GetROProperty**, **GetTOProperty**, **GetTOProperties**, and **SetTOProperty** methods, refer to the *QuickTest Professional Object Model Reference*.

Accessing Run-Time Object Properties and Methods

If the test object methods and properties available for a particular test object do not provide the functionality you need, you can access the native methods and properties of any run-time object in your application using the **Object** property.

You can use QuickTest’s statement completion feature with object properties to view a list of the available native methods and properties of an object. For more information about the statement completion option, see “Generating Statements in the Expert View or a Function Library” on page 130.

Tip: If the object is a Web object, you can also reference its native properties in programmatic descriptions using the attribute/property notation. For more information, see “Accessing User-Defined Properties of Web Objects” on page 178.

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
```

For more information on the **Object** property, refer to the *QuickTest Professional Object Model Reference*.

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 native **focus** method of the edit box as follows:

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

For more information on the **Object** property, refer to the *QuickTest Professional Object Model Reference*.

Accessing User-Defined Properties of Web Objects

You can use the **attribute/<property name>** notation to access native properties of Web objects and use these properties to identify such objects with programmatic descriptions.

For example, suppose a Web page has the same company logo image in two places on the page:

```
<IMG src="logo.gif" LogoID="122">
<IMG src="logo.gif" LogoID="123">
```


You could identify the image that you want to click using a programmatic description by including the user-defined property LogoID in the description as follows:

```
Browser("Mercury Tours").Page("Find Flights").Image("src=logo.gif",  
"attribute/LogoID=123").Click 68, 12
```

For more information on programmatic descriptions, see “Using Programmatic Descriptions” on page 155.

Running DOS Commands

You can run standard DOS commands in your QuickTest test or function using the VBScript Windows Scripting Host Shell object (WScript.shell). For example, you can open a DOS command window, change the path to C:\, and execute the **DIR** command using the following statements:

```
Dim oShell  
Set oShell = CreateObject ("WScript.shell")  
oShell.run "cmd /K CD C:\ & Dir"  
Set oShell = Nothing
```

For more information, refer to the Microsoft VBScript Language Reference (choose **Help > QuickTest Professional Help > VBScript Reference > VBScript**).

Enhancing Your Tests and Function Libraries Using the Windows API

Using the Windows API, you can extend testing abilities and add usability and flexibility to your tests and function libraries. The Windows operating system provides a large number of functions to help you control and manage Windows operations. You can use these functions to obtain additional functionality.

The Windows API is documented in the Microsoft MSDN Web site, which can be found at: http://msdn.microsoft.com/library/en-us/winprog/winprog/windows_api_start_page.asp?frame=true

A reference to specific API functions can be found at: http://msdn.microsoft.com/library/en-us/winprog/winprog/windows_api_reference.asp?frame=true

To use Windows API functions:

- 1** In MSDN, locate the function you want to use in your test or function library.
- 2** Read its documentation and understand all required parameters and return value(s).
- 3** Note the location of the API function. API functions are located inside Windows DLLs. The name of the DLL in which the requested function is located is usually identical to the Import Library section in the function's documentation. For example, if the documentation refers to **User32.lib**, the function is located in a DLL named **User32.dll**, typically located in your System32 library.
- 4** Use the QuickTest **Extern** object to declare an external function. For more information, refer to the *QuickTest Professional Object Model Reference*.

The following example declares a call to a function called **GetForegroundWindow**, located in **user32.dll**:

```
extern.declare micHwnd, "GetForegroundWindow", "user32.dll",  
"GetForegroundWindow"
```

- 5** Call the declared function, passing any required arguments, for example, `hwnd = extern.GetForegroundWindow()`.

In this example, the foreground window's handle is retrieved. You can enhance your test or function library if the foreground window is not in the object repository or cannot be determined beforehand (for example, a window with a dynamic title). You may want to use this handle as part of a programmatic description of the window, for example:

```
Window("HWND:="&hWnd).Close
```

In some cases, you may have to use predefined constant values as function arguments. Since these constants are not defined in the context of your test or function, you need to find their numerical value in order to pass them to the called function. The numerical values of these constants are usually declared in the function's header file. A reference to header files can also be found in each function's documentation under the Header section. If you have Microsoft Visual Studio installed on your computer, you can typically find header files under X:\Program Files\Microsoft Visual Studio\VC98\Include.

For example, the **GetWindow** API function expects to receive a numerical value that represents the relationship between the specified window and the window whose handle is to be retrieved. In the MSDN documentation, you can find the constants: GW_CHILD, GW_ENABLEDPOPUP, GW_HWNDFIRST, GW_HWNDLAST, GW_HWNDNEXT, GW_HWNDPREV and GW_HWNDPREV. If you open the **WINUSER.H** file, mentioned in the **GetWindow** documentation, you will find the following flag values:

```
/*
 * GetWindow() Constants
 */
#define GW_HWNDFIRST0
#define GW_HWNDLAST 1
#define GW_HWNDNEXT2
#define GW_HWNDPREV 3
#define GW_OWNER 4
#define GW_CHILD 5
#define GW_ENABLEDPOPUP 6
#define GW_MAX 6
```

Example

The following example retrieves a specific menu item's value in the Notepad application.

```
' Constant Values:
const MF_BYPOSITION = 1024
' API Functions Declarations
Extern.Declare micHwnd,"GetMenu","user32.dll","GetMenu",micHwnd
Extern.Declare
micInteger,"GetMenuItemCount","user32.dll","GetMenuItemCount",micHwnd
Extern.Declare
micHwnd,"GetSubMenu","user32.dll","GetSubMenu",micHwnd,micInteger
Extern.Declare
micInteger,"GetMenuString","user32.dll","GetMenuString",micHwnd,micInteger,
    micString+micByRef,micInteger,micInteger
' Notepad.exe
hwin = Window("Notepad").GetROProperty ("hwnd")' Get Window's handle
MsgBox hwin
men_hwnd = Extern.GetMenu(hwin)' Get window's main menu's handle
MsgBox men_hwnd
' Use API Functions
item_cnt = Extern.GetMenuItemCount(men_hwnd)
MsgBox item_cnt
hSubm = Extern.GetSubMenu(men_hwnd,0)
MsgBox hSubm
rc = Extern.GetMenuString(hSubm,0,value,64 ,MF_BYPOSITION)
MsgBox value
```

Choosing Which Steps to Report During the Run Session

You can use the **Report.Filter** method to determine which steps or types of steps are included in the Test Results. You can completely disable or enable reporting of steps following the statement, or you can indicate that you only want subsequent failed or failed and warning steps to be included in the report. You can also use the **Report.Filter** method to retrieve the current report mode.

The following report modes are available:

Mode	Description
0 or <code>rfEnableAll</code>	All events are displayed in the Test Results. Default.
1 or <code>rfEnableErrorsAndWarnings</code>	Only events with a warning or fail status are displayed in the Test Results.
2 or <code>rfEnableErrorsOnly</code>	Only events with a fail status are displayed in the Test Results.
3 or <code>rfDisableAll</code>	No events are displayed in the Test Results.

To disable reporting of subsequent steps, enter the following statement:

```
Reporter.Filter = rfDisableAll
```

To re-enable reporting of subsequent steps, enter:

```
Reporter.Filter = rfEnableAll
```

To instruct QuickTest to include only subsequent failed steps in the Test Results, enter:

```
Reporter.Filter = rfEnableErrorsOnly
```

To instruct QuickTest to include only subsequent failed or warning steps in the Test Results, enter:

```
Reporter.Filter = rfEnableErrorsAndWarnings
```

To retrieve the current report mode, enter:

```
MyVar=Reporter.Filter
```

For more information, refer to the *QuickTest Professional Object Model Reference*.

6

Working with User-Defined Functions and Function Libraries

In addition to the test objects, methods, and built-in functions supported by the QuickTest Test Object Model, you can define your own function libraries containing VBScript functions, subroutines, modules, and so forth, and then call their functions from your test.

This chapter describes:

- ▶ About Working with User-Defined Functions and Function Libraries
- ▶ Managing Function Libraries
- ▶ Working with Associated Function Libraries
- ▶ Using the Function Definition Generator
- ▶ Registering User-Defined Functions as Test Object Methods
- ▶ Additional Tips for Working with User-Defined Functions
- ▶ Executing Externally-Defined Functions from Your Test

About Working with User-Defined Functions and Function Libraries

If you have segments of code that you need to use several times in your test(s), you may want to create a user-defined function. A user-defined function encapsulates an activity (or a group of steps that require programming) into a keyword (or operation). By using user-defined functions, your tests are shorter, and easier to design, read, and maintain. You can then call user-defined functions from an action by inserting the relevant keywords (or operations) into that action.

You can register a user-defined function as a method for a QuickTest test object. A registered method can either override the functionality of an existing test object method for the duration of a run session, or be registered as a new method for a test object class. For more information about registering user-defined functions, see “Using the Function Definition Generator” on page 204 and “Registering User-Defined Functions as Test Object Methods” on page 219.

Note: When you create a user-defined function, do not give it the same name as a built-in function (for example, **GetLastError**, **MsgBox**, or **Print**). Built-in functions take priority over user-defined functions, so if you call a user-defined function that has the same name as a built-in function, the built-in function is called instead. For a list of built-in functions, refer to the **Built-in functions** list in the Step Generator (**Insert > Step Generator**).

Using QuickTest, you can define and store your user-defined functions either in a function library (saved as a **.qfl** file, by default) or directly in an action within a test. A function library is a Visual Basic script containing VBScript functions, subroutines, modules, and so forth. You can also use QuickTest to modify and debug any existing function libraries (such as **.vbs** or **.txt** files). For information on using VBScript, see “Handling VBScript Syntax Errors” on page 153 and “Understanding Basic VBScript Syntax” on page 146.)

When you store a function in a function library and associate the function library with a test, the test can call the public functions in that function library. For more information, see “Working with Associated Function Libraries” on page 200. Functions that are stored in an associated function library can be accessed from the Step Generator and the **Operation** column in the Keyword View, as well as being entered manually in the Expert View.

When you store a function in a test action, it can be called only from within that action—the function cannot be called from any other action or test. This is useful if you do not want the function to be available outside of a specific action.

You can also define private functions and store them in a function library. Private functions are functions that can be called only by other functions within the same function library. This is useful if you to reuse segments of code in your public functions.

You can define functions manually or using the Function Definition Generator, which creates the basic function definition for you automatically. Even if you prefer to define functions manually, you may still want to use the Function Definition Generator to view the syntax required to add header information, register a function to a test object, or set the function as the default method for the test object. For more information, see “Using the Function Definition Generator” on page 204.

Managing Function Libraries

You can create function libraries in QuickTest and call their functions from an action in your test. A function library is a separate QuickTest document containing VBScript functions, subroutines, modules, and so forth. Each function library opens in a separate window, enabling you to open and work on one or several function libraries at the same time. After you finish editing a function library, you can close it, leaving your QuickTest session open. You can also close all open function libraries simultaneously.

By implementing user-defined functions in function libraries and associating them with your test, you and other users can choose functions that perform complex operations, such as adding if/then statements and loops to test steps, or working with utility objects—without adding the code directly to the test. In addition, you save time and resources by implementing and using reusable functions.

QuickTest provides tools that enable you to edit and debug any function library, even if it was created using an external editor. For example, QuickTest can check the syntax of your functions, and the function library window provides the same editing features that are available in the Expert View. For more information on the options available in the Expert View, see Chapter 5, “Working with the Expert View and Function Library Windows.”

Note: In QuickTest, when you open a test, QuickTest creates a local copy of the external resources that are saved to your Quality Center project. Therefore, if another user modifies an external resource saved in your Quality Center project, such as a function library, or if you modify a resource using an external editor (not QuickTest)—the changes will not be implemented in the test until the test is closed and reopened.

In contrast with this, any changes you apply to external resources saved in the file system, such as function libraries, are implemented immediately, as these files are accessed directly and are not saved as local copies when you open your test.

Creating a Function Library

You can create a new function library at any time.

To create a new function library in QuickTest:

Perform one of the following:

- ▶ Choose **File > New > Function Library**
- ▶ Click the **New** button down arrow and choose **Function Library**



A new function library opens.

You can now add content to your function library and/or save it. When you add content to your function library, QuickTest applies the same formatting it applies to content in the Expert View. You can modify the formatting, if needed. For more information, see “Customizing the Expert View and Function Library Windows” on page 333.

Saving a Function Library

After you create or edit a function library in QuickTest, you can save it to your Quality Center project or to the file system.

Tips:

- ▶ When you modify a function library, an asterisk (*) is displayed in the title bar until the function library is saved.
 - ▶ To save all open documents, choose **File > Save All**. QuickTest prompts you to specify a location in which to save any new files that have not yet been saved.
 - ▶ To save multiple documents, choose **Window > Windows**. In the Window dialog box, select the documents you want to save and click the **Save** button. QuickTest prompts you for the save location for any new files that have not yet been saved.
 - ▶ You can also choose **File > Save As** to save the active function library under a different name or using a different path.
-

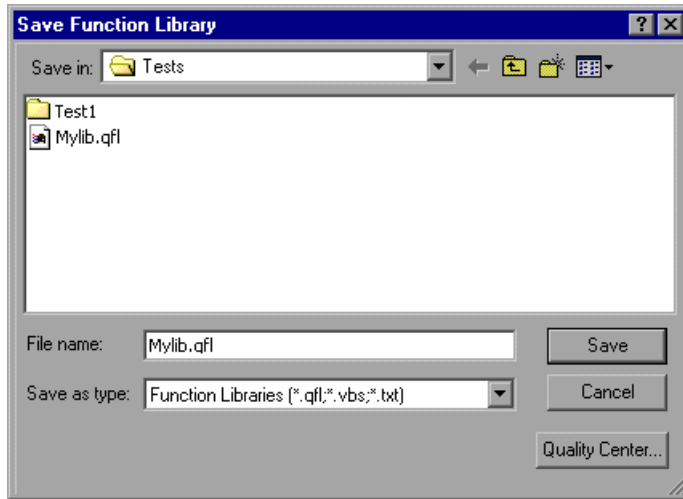
To save a function library:

- 1 Make sure that the function library you want to save is the active document. (You can click the function library's tab to bring it into focus.)
- 2 Perform one of the following:



- Click the **Save** button
- Choose **File > Save**
- Right-click the function library document's tab and choose **Save**

If the function library was previously saved, QuickTest saves it with your changes. Otherwise, if this is the first time you are saving this function library, the Save Function Library dialog box opens.



- 3 Save the function library to your Quality Center project or to the file system. (If the function library will be used in a business process test, you must save it to your Quality Center project.)

Note: If you are connected to Quality Center, the dialog box that opens is different from the standard file system dialog box. You can switch between the two dialog box versions by clicking the **File System** and **Quality Center** buttons in the relevant **Save** dialog box.

- ▶ To save the function library to your Quality Center project, in the Test Plan Tree box, choose the folder in which you want to save the function library. In the **Attachment name** box, type a name for the function library and click **OK**.
- ▶ To save the function library to the file system, in the Save Function Library dialog box, type a name for the function library in the **File name** box and click **Save**.

QuickTest saves the function library with a **.qfl** extension (unless you specify a different extension, such as **.vbs** or **.txt**, or remove the extension altogether).

Opening a Function Library

In QuickTest, you can open any function library that is saved in the file system or your Quality Center project—even if another document is already open in QuickTest. You can only open a function library if you have read or read-write permissions for the file.

You can choose to open a function library in edit mode or read-only mode:

- ▶ **Edit mode**—Enables you to view and modify the function library. While the function library is open on your computer, other users can view the file in read-only mode, but they cannot modify it.
- ▶ **Read-only mode**—Enables you to view the function library but not modify it. By default, when you open a function library that is currently open on another computer, it opens in read-only mode. You can also choose to open a function library in read-only mode if you want to review it, but you do not want to prevent another user from modifying it.

Tip: You can also navigate directly from a function in your document to its function definition in another function library. For more information, see “Navigating to a Specific Function in a Function Library” on page 195.

To open an existing function library:

Perform one of the following:

- ▶ Choose **File > Open > Function Library**
- ▶ Click the **Open** button down arrow and choose **Function Library**

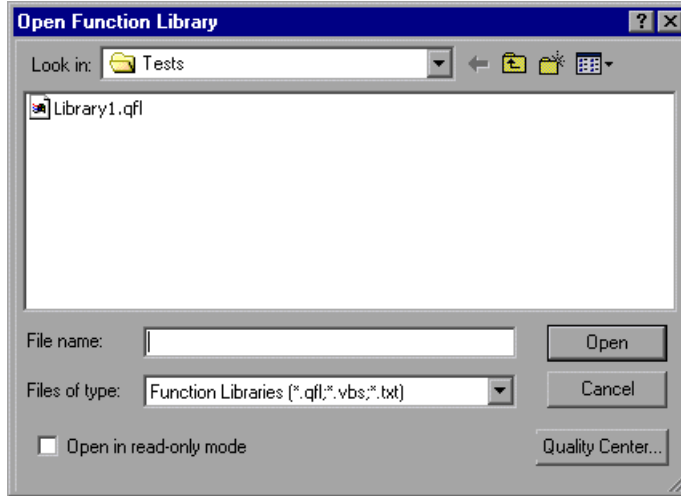


Tips:

If the function library was recently created or opened, you can choose it from the recent files list in the **File** menu.

If the function library is associated with the open test, you can choose it from **Resources > Associated Function Libraries**. (If you choose a function library that is stored in a Quality Center project, QuickTest must be connected to that project to open the associated function library.)

The Open Function Library dialog box opens.



Note: If you are connected to Quality Center, the dialog box that opens is different from the standard file system dialog box. You can switch between the two dialog box versions by clicking the **File System** and **Quality Center** buttons in the relevant **Open** dialog box.

Tip: You can open the function library in read-only mode by selecting the **Open in read-only mode** check box.

Browse to and select a function library, and click **Open**. QuickTest opens the specified function library in a new window. You can now view and modify its content. For more information, see “Editing a Function Library” on page 195 and “Debugging a Function Library” on page 197.

Navigating Between Open QuickTest Documents

You can open multiple function libraries while a test is open, and you can navigate between all of your open documents.

To navigate between open QuickTest documents:

Perform one of the following:

- Click the tab for the required document in the Document pane



Tip: If not all tabs are displayed due to lack of space, use the left and right scroll arrows in the Document pane to display the required document's tab.

- Press CTRL+TAB on your keyboard to scroll between open documents
- Choose the required document from the Window menu
- Choose **Window > Windows**, select the required document in the Windows dialog box, and click the **Activate** button

Note: You can also choose **Resources > Associated Function Libraries** and choose the required function library from the list. This also opens closed function libraries that are associated with your test.

Navigating to a Specific Function in a Function Library

After you insert a call to a function, you can navigate directly to its definition in the source document. The function definition can be located either in the same document (test or function library) or in another function library that is associated with your test. If the document containing the function definition is already open, QuickTest activates the window (brings the window into focus). If the document is closed, QuickTest opens it.

To navigate to a function's definition:

- 1** In the Expert View or function library, click in the step containing the relevant function.
- 2** Perform one of the following:
 - ▶ Choose **Edit > Advanced > Go to Function Definition**
 - ▶ Right-click the step and choose **Go to Function Definition** from the context menu

QuickTest activates the relevant document (if the function definition is located in another function library) and positions the cursor at the beginning of the function definition.

Editing a Function Library

You can edit a function library at any time using the QuickTest editing features that are available in the Expert View.

You can drag and drop a function (or part of it) from one document to another. (To do so, you must first separate the tabbed documents into separate document panes by clicking the **Restore Down** button (located below the QuickTest window's **Restore Down / Maximize** button).)

You can add steps to your function library manually or using the Step Generator. The Step Generator enables you to add steps that contain reserved objects (the objects that QuickTest supplies for enhancement purposes, such as utility objects), VBScript functions (such as **MsgBox**), utility statements (such as **Wait**), and user-defined functions that are defined in the same function library. IntelliSense is available for all functions defined in your action or for public functions defined in associated function libraries.

Note: In function libraries, IntelliSense does not enable you to view test object names or collections because function libraries are not connected to object repositories.



You can instruct QuickTest to check syntax by clicking the **Check Syntax** button, or by choosing **Tools > Check Syntax**.

Tips:

For information on using VBScript, see “Understanding Basic VBScript Syntax” on page 146.

To check the syntax for all function libraries associated with your test, click the **Check Syntax** button in the Resources tab of the Test Settings dialog box (**File > Settings**). For more information, refer to “Defining Resource Settings for Your Test” on page 751 in the *QuickTest Professional Basic Features User’s Guide*.

Editing a Read-Only Function Library

If you open a function library in read-only mode and then decide to modify it, you can convert the function library to an editable file—as long as the function library is not locked by another user. For more information on the options available when opening a function library, see “Opening a Function Library” on page 191.

Note: During a debug session, all documents (such as tests and function libraries) are read-only. To edit a document during a debug session, you must first stop the debug session.

To edit a read-only function library:



Choose **File > Enable Editing** or click the **Enable Editing** button. You can now edit the function library.

Debugging a Function Library

Before you can debug a function library, you must first associate it with a test and then insert a call to at least one of its functions. For example, you can use the Debug Viewer to view, set, or modify the current value of objects or variables in your function library. You can step into functions (including user-defined functions), set breakpoints, stop at breakpoints, view expressions, and so forth. You can begin debugging from a specific step, or you can instruct QuickTest to pause at a specific step. For more information, refer to “Debugging Tests and Function Libraries” on page 575 in the *QuickTest Professional Basic Features User’s Guide*.

Note: During a debug session, all documents are read-only and cannot be edited. To edit a document during a debug session, you must first stop the debug session.

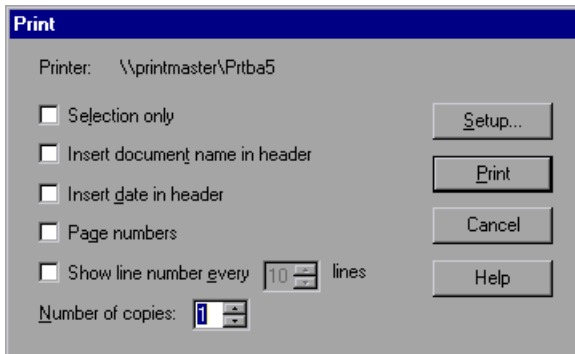
Printing a Function Library

You can print a function library at any time. You can also include additional information in the printout.

To print from the function library:



- 1 Click the **Print** button or choose **File > Print**. The Print dialog box opens.



- 2 Specify the print options that you want to use:
 - ▶ **Printer**—Displays the printer to which the print job will be sent. You can change the printer by clicking the **Setup** button.
 - ▶ **Selection only**—Prints only the text that is currently selected (highlighted) in the function library.
 - ▶ **Insert document name in header**—Includes the name of the function library at the top of the printout.
 - ▶ **Insert date in header**—Includes today's date at the top of the printout. The date format is taken from your Windows regional settings.
 - ▶ **Page numbers**—Includes page numbers on the bottom of the printout (for example, page 1 of 3).
 - ▶ **Show line numbers every __ lines**—Displays line numbers to the left of the script lines, as specified.
 - ▶ **Number of copies**—Specifies the number of times to print the document.
- 3 If you want to print to a different printer or change your printer preferences, click **Setup** to display the Print Setup dialog box.
- 4 Click **Print** to print according to your selections.

Closing a Function Library

You can close an individual function library, or if you have several function libraries open, you can close some or all of them simultaneously. If any of the function libraries are not saved, QuickTest prompts you to save them.

To close an individual function library:

Perform one of the following:

- ▶ Make sure that the function library you want to save is the active document—you can click the function library's tab to bring it into focus—and choose **File > Close**
- ▶ Right-click the function library document's tab and choose **Close**
- ▶ Click the **Close** button in the top right corner of the function library window
- ▶ Choose **Window > Windows**. In the Windows dialog box, select the function library to close if it is not already selected, and click the **Close Window(s)** button



To close several function libraries:


Choose **Window > Windows**. In the Windows dialog box, select the function libraries you want to close and click the **Close Window(s)** button.

To close all open function libraries:

Choose **File > Close All Function Libraries**, or **Window > Close All Function Libraries**.

Working with Associated Function Libraries

In QuickTest, you can create function libraries containing functions, subroutines, modules, and so forth, and then associate the files with your test. This enables you to insert a call to a public function or subroutine in the associated function library from that test. (Public functions stored in function libraries can be called from any associated test, whereas private functions can be called only from within the same function library.)

If a test can no longer access a function that was used in a step (for example, if the function was deleted from the associated function library), the  icon is displayed adjacent to the step in the Keyword View. When you run the test, an error will occur when it reaches the step using the nonexistent function.

Note: Any text file written in standard VBScript syntax can be used as a function library.

You can specify the default function libraries for all new tests in the Test Settings dialog box (**File > Settings > Resources** tab). After a test is created, the list of default function libraries is integrated into the test. Therefore any changes to the default function libraries list in the Test Settings dialog box do not affect existing tests.

You can edit the list of associated function libraries for an existing test in the Test Settings dialog box. For more information, refer to “Defining Resource Settings for Your Test” on page 751 in the *QuickTest Professional Basic Features User’s Guide*.

Notes:

- ▶ In addition to the functions available in the associated function libraries, you can also call a function contained in any function library (or VBScript file) directly from any action using the **ExecuteFile** function. You can also insert **ExecuteFile** statements within an associated function library. For more information, see “Executing Externally-Defined Functions from Your Test” on page 228.
- ▶ You cannot debug a file that is called using an **ExecuteFile** statement, or any of the functions contained in the file. In addition, when debugging a test that contains an **ExecuteFile** statement, the execution marker may not be correctly displayed.

Working with Associated Function Libraries in Quality Center

You can associate a function library with your test, regardless of whether the function library is stored in the file system or your Quality Center project. However, if you are planning on using the function library in a business process test, you must save it in your Quality Center project.

When working with Quality Center and associated function libraries, you must save the associated function library as an attachment in your Quality Center project *before* you specify the associated file in the Resources tab of the Test Settings dialog box. You can add a new or existing function library to your Quality Center project.

If you add an existing function library from the file system to a Quality Center project, you are actually adding a copy of that file to the project. Therefore, if you later make modifications to either of these function libraries (in the file system or in your Quality Center project), the other function library remains unaffected.

Associating Function Libraries with a Test

You can associate an open function library with the currently open test.

You can also associate function libraries with the currently open test using the associated function libraries list. For more information, see “Modifying Function Library Associations” on page 203.


To associate a function library with a test:


- 1** Make sure that the test with which you want to associate the function library is open in QuickTest.
- 2** Create or open a function library in QuickTest. (Before continuing to the next step, make sure that the function library you want to associate with the test is the active document—you can click the function library’s tab to bring it into focus.) For more information, see “Managing Function Libraries” on page 188.
- 3** Save the function library either in your Quality Center project as an attachment or in the file system. For more information, see “Saving a Function Library” on page 189.
- 4** In QuickTest, choose **File > Associate Library '<Function Library>' with '<Test>'** or right-click in the in the function library and choose **Associate Library '<Function Library>' with '<Test>'**. QuickTest associates the function library with the open test.

Modifying Function Library Associations

You can modify the list of associated function libraries for a test. You can add or remove function libraries from the list, and change their priorities.


To modify function library associations in your test:

- 1 In the Test Settings dialog box, click the **Resources** tab.
- 2  In the associated function libraries list, click the **Add** button. QuickTest displays a browse button enabling you to browse to a function library in the file system. If you are connected to a Quality Center project, QuickTest also adds [QualityCenter] to the file path, indicating that you can browse to a function library either in your Quality Center project or in the file system.

 **Tip:** If you want to add a file from your Quality Center project but are not connected to Quality Center, press and hold the **SHIFT** key and click the **Add** button. QuickTest adds [QualityCenter], and you can enter the path manually. If you do, make sure there is a space after [QualityCenter]. For example: [QualityCenter] Subject\Tests

Note that QuickTest searches Quality Center project folders only when you are connected to the corresponding Quality Center project.

- 3 Select the function library you want to associate with your test and click **Open** or **OK** (depending on whether you are selecting it from the file system or your Quality Center project).

 **Tip:** You can remove an associated function library from the list by selecting it and clicking the **Remove** button. You can also prioritize associated function libraries by using the **Up** and **Down** arrows.

For more information, refer to “Defining Resource Settings for Your Test” on page 751 in the *QuickTest Professional Basic Features User’s Guide*.

Using the Function Definition Generator

QuickTest provides a Function Definition Generator, which enables you to generate definitions for new user-defined functions and add header information to them. You can then register these functions to a test object, if needed. You fill in the required information and the Function Definition Generator creates the basic function definition for you. After you define the function definition, you can insert the definition in your function library and associate it with your test, or you can insert the definition directly in a test script in the Expert View. Finally, you complete the function by adding its content (code).

Note: If you insert the function directly in the Expert View, the test will be able to access the function anywhere within the specific action.

If you register the function to a test object, it can be called by that test object, and is displayed in the list of available operations for that test object.

If you do not register the function to a test object, it becomes a global operation and is displayed in the list of operations in the **Operation** box in the Step Generator, and in the **Operation** column in the Keyword View, and when using IntelliSense. If you register a function, you can define it as the default operation that is displayed in the Step Generator or the Keyword View when the test object to which it is registered is selected.

Finally, you can document your user-defined function by defining the tooltip that displays when the cursor is positioned over the operation in the Step Generator, in the Keyword View, and when using IntelliSense. You can also add a sentence that describes what the step that includes the user-defined function actually does. This sentence is then displayed in the Keyword View in the **Step documentation** box of the Step Generator and in the **Documentation** column.

As you add information to the Function Definition Generator, the **Preview** area displays the emerging function definition. After you finish defining the function, you insert the definition in the active QuickTest document. If you insert it in a function library, the function will be accessible to any associated test. If you insert the function directly in a test in the Expert View, it can be called only from within the specific action. Finally, you add the content (code) of the function.

The following section provides an overview of the steps you perform when using the Function Definition Generator to create a function.

To use the Function Definition Generator:

- 1** Open the Function Definition Generator, as described in “Opening the Function Definition Generator” on page 206.
- 2** Define the function, as described in “Defining the Function Definition” on page 208.
- 3** Register the function to a test object, if needed, as described in “Registering a Function Using the Function Generator” on page 209.

By default, functions that are not registered to a test object are automatically defined as global functions that can be called by selecting the **Functions** category in the Step Generator, the **Operation** item in the Keyword View, or when using IntelliSense. Note that if you register the function to a test object, you can also define the function (operation) as the default operation for that selected test object.

- 4** Add arguments to the function, as described in “Specifying Arguments for the Function” on page 213.
- 5** Document the function by adding header information to it, as described in “Documenting the Function” on page 214.
- 6** Preview the function before finalizing it, as described in “Previewing the Function” on page 216.
- 7** Generate another function definition, if needed, as described in “Generating Another User-Defined Function” on page 216.

- 8 Finalize each function by inserting it in your active document and adding content to it, as described in “Finalizing the User-Defined Function” on page 217.

Note: Each of the steps listed in this section assumes that you have performed the previous steps.

Opening the Function Definition Generator

You open the Function Definition Generator from QuickTest.

To open the Function Definition Generator:

- 1 Make sure that the function library or test in which you want to insert the function definition is the active document. (You can click the document’s tab to bring it into focus.) This is because the Function Definition Generator inserts the function in the currently active document after you finish defining it.



- 2 Choose **Insert > Function Definition Generator** or click the **Function Definition Generator** button. The Function Definition Generator opens.

Function Definition Generator

Function definition

Name:

Type:

Scope:

Arguments:

Name	Pass Mode

Register to a test object

Test object:

Operation:

Register as default operation

Additional information

Description:

Documentation:

Preview

```
Public Function
  'TODO: add function body here
End Function
```

Insert another function definition

OK Cancel Help

After you open the Function Definition Generator, you can begin to define a new function.

Defining the Function Definition

After you open the Function Definition Generator, you can begin defining a function.

For example, if you want to define a function that verifies the value of a specified property, you might name it `VerifyProperty` and define it as a public function so that it can be called from any associated test. (If you define it as private, the function can only be called from elsewhere in the same function library. Private functions cannot be registered to a test object.)

To define a function:

- 1 In the **Name** box, enter a name for the new function. The name should clearly indicate what the operation does so that it can be easily selected from the Step Generator or the Keyword View. Function names cannot contain non-English letters or characters. In addition, function names must begin with a letter and cannot contain spaces or any of the following characters:
! @ # \$ % ^ & * () + = [] \ { } | ; ' : "" , / < > ?

Note: Do not give the user-defined function the same name as a built-in function (for example, `GetLastError`, `MsgBox`, or `Print`). Built-in functions take priority over user-defined functions, so if you call a user-defined function that has the same name as a built-in function, the built-in function is called instead. For a list of built-in functions, refer to the **Built-in functions** list in the Step Generator (**Insert > Step Generator**).

- 2 From the **Type** list, choose **Function** or **Sub**, according to whether you want to define a function or a subroutine.

- 3 From the **Scope** list, choose the scope of the function—either **Public** (to enable the function to be called by any test that is associated with this function library), or **Private** (to enable the function to be called only from elsewhere in the same function library). By default, the scope is set to **Public**. (Only public functions can be registered to a test object.)

Note: If you create a user-defined function manually and do not define the scope as **Public** or **Private**, it will be treated as a public function, by default.

After you define a public function, you can register the function. Alternatively, if you defined a private function, or if you do not want to register the function, you can continue by specifying arguments for the function. For more information, see “Specifying Arguments for the Function” on page 213.

Registering a Function Using the Function Generator

You can register a public function to a test object to enable the function (operation) to be performed on a test object. When you register a function to a test object, you can choose to override the functionality of an existing operation, or you can register the function as a new operation for the test object.

After you register a function to a test object, it is displayed as an operation in the Step Generator when that test object is selected, and in the Keyword View **Operation** list when that test object is selected from the **Item** list, as well as in IntelliSense and in the general **Operation** list in the Step Generator. When you register a function to a test object, it can only be called by that test object.

If you choose to register the function to a test object, the Function Definition Generator automatically adds the argument, **test_object**, as the first argument in the Arguments area in the top-right corner of the Function Definition Generator. The Function Definition Generator also automatically adds a **RegisterUserFunc** statement with the correct argument values immediately after your function definition.

When you register a function to a test object, you can optionally define it as the default operation for that test object. This instructs QuickTest to display the function in the **Operation** column, by default, when you or the Subject Matter Expert choose the associated test object in the **Item** list. It also enables you to select the function from IntelliSense. When you define a function as the default function for a test object, the value **True** is specified as the fourth argument of the **RegisterUserFunc** statement.

If you do not register the function to a specific test object, the function is automatically defined as a global function. Global functions can be called by selecting the **Functions** category in the Step Generator, or the **Operation** item in the Keyword View. A list of global functions can be viewed alphabetically in the **Operation** box when the **Functions** category is selected in the Step Generator, in the **Operation** list when the **Operation** item is selected from the **Item** list in the Keyword View, and when using IntelliSense.

During run-time, QuickTest first searches the test for the specified function and then searches the function libraries in the order in which they are listed in the Resources tab. If QuickTest finds more than one function that matches the function name in a specific test or function library, it uses the last function it finds in that test or function library. If QuickTest finds two functions with the same name in two different function libraries, it uses the function from the function library that has the higher priority. To avoid confusion, it is recommended that you verify that within the resources associated with a test, each function has a unique name.

Tip: If you choose not to register your function at this time, you can manually register it later by adding a **RegisterUserFunc** statement after your function as shown in the following example:

```
RegisterUserFunc "WebEdit", "MySet", "MySetFunc"
```

In this example, the **MySet** method (operation) is added to the WebEdit test object using the **MySetFunc** user-defined function. If you choose the WebEdit test object from the **Item** list in the Keyword View, the **MySet** operation will then be displayed in the **Operation** list (together with other registered and out of the box operations for the WebEdit test object).

You can also register your function to other test objects by duplicating (copying and pasting) the **RegisterUserFunc** statement and modifying the argument values as needed when you save the function code in a function library.

To define this function as the default function, you define the value of the fourth argument of the **RegisterUserFunc** statement as **True**. For example:

```
RegisterUserFunc "WebEdit", "MySet", "MySetFunc", True
```

Note: A registered or global function can only be called from a test after it is added to the test script or a function library that is associated with the test.

To register the function to a test object:

- 1 Select the **Register to a test object** check box. The options in this area are enabled, and a new argument, **test_object**, is automatically added to the list of arguments in the **Arguments** area in the top-right corner of the Function Definition Generator. (The **test_object** argument receives the test object to which you want to register the function.)

Function definition

Name:

Type:

Scope:

Register to a test object

Test object: Operation:

Register as default operation

Arguments:	
Name	Pass Mode
test_object	By value

Note: If you clear the **Register to a test object** check box, the default **test_object** argument is automatically removed from the **Arguments** area (unless you renamed it).

- 2 Choose a **Test object** from the list of available objects. For example, for the sample **VerifyProperty** function, you might want to register it to the **Link** test object.
- 3 Specify the **Operation** that you want to add or override for the test object.
 - To define a new operation, enter a new operation name in the **Operation** box. For example, for the sample **VerifyProperty** function, you may want to define a new **VerifyProperty** operation.
 - To override the standard functionality of an existing operation, choose an operation from the list of available operations in the **Operation** box.

- 4 If you want the function to be displayed as the default operation in the **Operation** column when you or the Subject Matter Expert choose the associated item, select the **Register as default operation** check box.

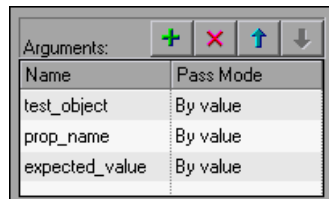
For example, if you were to define the **VerifyProperty** operation as the default operation for the Link test object, the value **True** would be defined as the fourth argument of the **RegisterUserFunc** statement, and the syntax would appear as follows:

```
RegisterUserFunc "Link", "VerifyProperty", "VerifyProperty", True
```

After you specify the test object registration information, you specify additional arguments for the function.

Specifying Arguments for the Function

After you define the basic function definition and specify the test object registration information, if any, you can specify the function's arguments.




For example, if you choose to register the function to a test object, as we did the example described in “Registering a Function Using the Function Generator” on page 209, you may want to assign the arguments **prop_name** (the name of the property to check) and **expected_value** (the expected value of the property), in addition to the first argument, **test_object**. You must define the required argument(s) for your function to run correctly.

You can list the arguments in any order. However, if you are registering the function to a test object, the first argument must always receive the test object.




To define the arguments for the function:

In the **Arguments** area, specify the argument(s) for the function. You can add as many arguments as needed. To ensure clarity, the name for each argument should indicate the value that needs to be entered.

- ▶ To add an argument, click  and enter a name for the argument. The argument name should clearly indicate the value that needs to be entered for the argument. Argument names may not contain non-English letters or characters. In addition, argument names must begin with a letter and cannot contain spaces or any of the following characters:

! @ # \$ % ^ & * () + = [] \ { } | ; ' : "" , / < > ?

By default, the **Pass Mode** is set as **By value**. This instructs QuickTest to pass the argument to the function by value. If you want to pass the argument by reference, choose **By reference** in the **Pass Mode** box.

- ▶ To remove an argument, select it and click . The argument is removed from the Function Definition Generator.
- ▶ To set the order of the arguments, use the  and  arrows. The order of the arguments only affects the readability of the function code (except if you want to register the public function—in this case, the first argument must receive the test object).

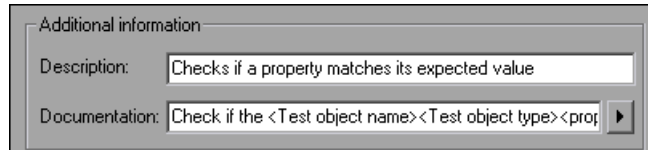
Documenting the Function

The Function Definition Generator enables you to add header information to your user-defined function. You can add a description, which is displayed as a tooltip when the cursor is positioned over the operation. You can then use this tooltip to determine which operation to choose from the list of available operations. (It is advisable to keep the description text as brief and clear as possible.)

In addition, you can add documentation that specifies exactly what a step using your function does. You can include the test object name, test object type, and any argument values in the text. You can also add text manually, as needed. This text that you add here is displayed in the Keyword View in the **Step documentation** box of the Step Generator and in the **Documentation** column. Therefore, the sentence must be a clear and understandable.

For example, if you were checking a link to “Mercury” from a search engine, you might define the following documentation using the Function Definition Generator:

```
'@Documentation Check if the <Test object name> <Test object type>
<prop_name> value matches the expected value: <expected_value>.
```





After choosing values for the arguments in the Keyword View, the above documentation might appear as follows: Check if the “Mercury Business Technology” link “text” value matches the expected value: “Mercury Business Technology Optimization (BTO) Software”.

Tip: You can right-click on any column header in the Keyword View and select the **Documentation only** option to view or print a list of steps. This instructs QuickTest to display only the **Documentation** column. You can also choose **Edit > Copy Documentation to Clipboard** and then paste the documentation in any application. Therefore, the sentence displayed for the step in this column must also be clear enough to use for manual testing instructions.

To document the function:

- 1 In the **Description** box, enter the text to be displayed as a tooltip when the cursor is positioned over the function name in the **Operation** list in the Step Generator, in the **Operation** column in the Keyword View, and in IntelliSense.

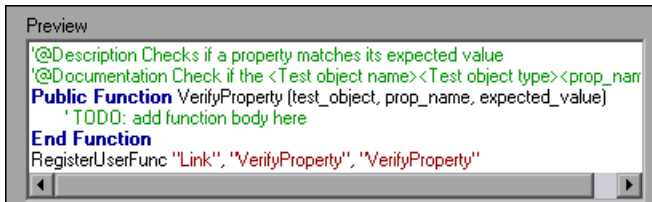
For example, for the sample **VerifyProperty** function, you may want to enter: Checks whether a property value matches the actual value.

- 2** In the **Documentation** box, enter the text to be displayed in the **Step documentation** box in the Step Generator in the Keyword View and in the **Documentation** column of the Keyword View. You can use arguments in the **Documentation** text by clicking  and selecting the relevant argument. If you selected the **Register to a test object** check box, clicking  also enables you to add the **Test object name** and/or **Test object type** items to the **Documentation** column from the displayed list. If you use these test object and argument items in the **Documentation** text, they are replaced dynamically by the relevant test object names and types or argument values.

Previewing the Function

The **Preview** area displays the function code as you define it, in read-only format. You can review your function and make any changes, as needed, in the various areas of the Function Definition Generator.

For example, for the sample **VerifyProperty** function, the **Preview** area displays the following code.



```

Preview
'@Description Checks if a property matches its expected value
'@Documentation Check if the <Test object name><Test object type><prop_name>
Public Function VerifyProperty (test_object, prop_name, expected_value)
    ' TODO: add function body here
End Function
RegisterUserFunc "Link", "VerifyProperty", "VerifyProperty"
  
```

After you review the code (before you insert it in the active document), you can choose either to generate another function definition or to finalize the code for the function you defined.

Generating Another User-Defined Function

After you preview the code—before you insert the function in the active document—you can decide whether you want to generate an additional function definition.

Note: If you do not want to define an additional function, continue to the next section.

To generate an additional user-defined function:

- 1** Select the **Insert another function definition** check box and click **Insert**. QuickTest inserts the function definition in the active document and clears the data from the Function Definition Generator. The Function Definition Generator remains open.
- 2** Define the new function beginning from “Defining the Function Definition” on page 208.

Finalizing the User-Defined Function

After you preview the code, you insert it in the active document. If you insert it in a function library, any test associated with the function library can access the function. If you insert the function directly in a test (in the Expert View), the test can contain a call to the function from anywhere within the specific action.

After you insert the code in the required location, you can finalize the function. For example, for the **VerifyProperty** function, the following code would be inserted in your function library or test:

```
'@Description Checks whether a property matches its expected value
'@Documentation Check whether the <Test object name> <Test object type>
<prop_name> value matches the expected value: <expected_value>.
Public Function VerifyProperty (test_object, prop_name, expected_value)
    'TODO: add function body here
End Function
RegisterUserFunc "Link", "VerifyProperty", "VerifyProperty"
```

Tip: The **RegisterUserFunc** statement (in the last line) registers the **VerifyProperty** function to the Link test object. If you want to register the function to more than one test object, you could copy this line and duplicate it for each test object, changing the argument values, as required.

To finalize the function, you add its content (replacing the TODO comment). For example, if you want the function to verify whether the expected value of a property matches the actual property value of a specific test object, you might add the following to the body of the function:

```
Dim actual_value
' Get the actual property value
actual_value = obj.GetROProperty(prop_name)
' Compare the actual value to the expected value
If actual_value = expected_value Then
    Reporter.ReportEvent micPass, "VerifyProperty Succeeded", "The " &
prop_name & " expected value: " & expected_value & " matches the actual
value"
    VerifyProperty = True
Else
    Reporter.ReportEvent micFail, "VerifyProperty Failed", "The " &
prop_name & " expected value: " & expected_value & " does not match the
actual value: " & actual_value
    VerifyProperty = False
End If
```

To finalize the user-defined function:

- 1 Click **OK**. QuickTest inserts the function definition in the active document and closes the Function Definition Generator.

Note: If you define a function directly in an action, the function can be called only in that action.

- 2 In your function library or test, add content to the function code, as required, by replacing the TODO line.

Tip: To display the function in the test results tree (Test Results window) after a run session, add a **Reporter.ReportEvent** statement to the function code (as shown in the example above).

Note that if your user-defined function uses a default test object method, this step will appear in the Test Results window after the run session. However, you can still add a **Reporter.ReportEvent** statement to the function code to provide additional information and to modify the test status, if required.

- 3 If you inserted the code in a function library, you must associate the function library with a test to enable access to the user-defined function(s). You also need to check its syntax to ensure that tests will have access to the functions, and that you will be able to see and use the functions. For more information, see “Working with Associated Function Libraries” on page 200.

Registering User-Defined Functions as Test Object Methods

In addition to using the QuickTest Function Definition Generator to register a function, as described in “Registering a Function Using the Function Generator” on page 209, you can also use the **RegisterUserFunc** statement to add new methods to test objects or to change the behavior of an existing test object method during a run session.

When you register a function to a test object, you can define it as the default operation for that test object, if required. The default operation is displayed by default in the Step Generator or the **Operation** column in the Keyword View when the test object to which it is registered is selected.

If you choose not to register a function to a test object, it becomes a global function. Global functions can be called by selecting the **Functions** category in the Step Generator, the **Operation** item in the Keyword View, or when using IntelliSense. You use the **UnregisterUserFunc** statement to disable new methods or to return existing methods to their original QuickTest behavior.

To register a method, you first define a function in your test or in an associated function library. You then enter a **RegisterUserFunc** statement at the end of the function that specifies the test object class, the function to use, and the method name that calls your function. You can register a new method for a test object class, or you can use an existing method name to (temporarily) override the existing functionality of the specified method.

Your registered method applies only to the test or function library in which you register it. In addition, QuickTest clears all function registrations at the beginning of each run session.

Preparing the User-Defined Function

You can write your user-defined function directly into your test if you want to limit its use only to the local action, or you can store the function in an associated function library to make it available to many actions and tests (recommended). If the same function name exists locally within your action and within an associated function library, QuickTest uses the function defined in the action.

When you run a statement containing a registered method, it sends the test object as the first argument. For this reason, your user-defined function must have at least one argument. Your user-defined function can have any number of arguments, or it can have only the test object argument. Make sure that if the function overrides an existing method, it has the exact syntax of the method it is replacing. This means that its first argument is the test object and the rest of the arguments match all the original method arguments.

Tip: You can use the **parent** test object property to retrieve the parent of the object represented by the first argument in your function. For example:
 ParentObj = obj.GetROProperty("parent")

When writing your function, you can use standard VBScript statements as well as any QuickTest reserved objects, methods, functions, and any method associated with the test object specified in the first argument of the function.

For example, suppose you want to report the current value of an edit box to the Test Results before you set a new value for it. You can override the standard QuickTest **Set** method with a function that retrieves the current value of an edit box, reports that value to the Test Results, and then sets the new value of the edit box.

The function would look something like this:

```
Function MyFuncWithParam (obj, x)
    dim y
    y = obj.GetROProperty("value")
    Reporter.ReportEvent micDone, "previous value", y
    MyFuncWithParam=obj.Set (x)
End Function
```

Note: This function defines a return value, so that each time it is called from a test, the function returns the **Set** method argument value.

Registering User-Defined Test Object Methods

You can use the `RegisterUserFunc` statement to instruct QuickTest to use your user-defined function as a method of a specified test object class for the duration of a test run, or until you unregister the method.

Note: If you call an external action that registers a method (and does not unregister it at the end of the action), the method registration also takes effect for the remainder of the test that called the action.

To register a user-defined function as a test object method, use the following syntax:

`RegisterUserFunc TOClass, MethodName, FunctionName, SetAsDefault`

Item	Description
<i>TOClass</i>	Any test object class. Note: You cannot register a method for a QuickTest reserved object (such as DataTable , Environment , Reporter , and so forth).
<i>MethodName</i>	The name of the method you want to register (and display in QuickTest, for example, in the Keyword View and IntelliSense). If you enter the name of a method already associated with the specified test object class, your user-defined function overrides the existing method. If you enter a new name, it is added to the list of methods that the object supports.
<i>FunctionName</i>	The name of the user-defined function that you want to call from your test. The function can be located in your test or in any associated function library.
<i>SetAsDefault</i>	Indicates whether the registered function is used as the default method for the test object. When you select a test object in the Keyword View or Step Generator, the default method is automatically displayed in the Operation column (Keyword View) or Operation box (Step Generator).

Tip: If the function you are registering is defined in a function library, it is recommended to include the **RegisterUserFunc** statement in the function library as well so that the method will be immediately available for use in any test using that function library.

For example, suppose that the Find Flights Web page contains a **Country** edit box, and by default, the box contains the value **USA**. The following example registers the **Set** method to use the **MySet** function in order to retrieve the default value of the edit box before the new value is entered.

```
Function MySet (obj, x)
```

```
    dim y
```

```
    y = obj.GetROProperty("value")
```

```
    Reporter.ReportEvent micDone, "previous value", y
```

```
    MySet=obj.Set(x)
```

```
End Function
```

```
RegisterUserFunc "WebEdit", "Set", "MySet"
```

```
Browser("MercuryTours").Page("FindFlights").WebEdit("Country").Set "Canada"
```

For more information and examples, refer to the *QuickTest Professional Object Model Reference*.

Unregistering User-Defined Test Object Methods

When you register a method using a **RegisterUserFunc** statement, your method becomes a recognized method of the specified test object for the remainder of the test, or until you unregister the method. If your method overrides a QuickTest method, unregistering the method resets the method to its normal behavior. Unregistering other methods removes them from the list of methods supported by the test object.

Unregistering methods is especially important when a reusable action contains registered methods that override QuickTest methods. For example, if you do not unregister a method that uses a function defined directly within a called action, then the calling test will fail if the registered method is called again in a later action, because it will not be able to find the function definition.

If the registered function was defined in a function library, then the calling test may succeed (assuming the function library is associated with the calling test). However, unexpected results may be produced as the author of the calling test may not realize that the called action contained a registered function, and therefore, may use the registered method in later actions, expecting normal QuickTest behavior.

To unregister a user-defined method, use the following syntax:

UnRegisterUserFunc *TOClass*, *MethodName*

Item	Description
<i>TOClass</i>	The test object class for which your method is registered.
<i>MethodName</i>	The method you want to unregister.

For example, suppose that the Find Flights Web page contains a **Country** edit box, and by default, the box contains the value USA. The following example registers the **Set** method to use the **MySet** function in order to retrieve the default value of the edit box before the new value is entered. After using the registered method in a **WebEdit.Set** statement for the **Country** edit box, the **UnRegisterUserFunc** statement is used to return the **Set** method to its standard functionality.


```
Function MySet (obj, x)
    dim y
    y = obj.GetROProperty("value")
    Reporter.ReportEvent micDone, "previous value", y
    MySet=obj.Set(x)
End Function

RegisterUserFunc "WebEdit", "Set", "MySet"
Browser("MercuryTours").Page("FindFlights").WebEdit("Country").Set "Canada"
UnRegisterUserFunc "WebEdit", "Set"
```

Additional Tips for Working with User-Defined Functions

When working with user-defined functions, consider the following tips and guidelines:

- ▶ For an in-depth view of the required syntax, you can define a function using the Function Definition Generator and experiment with the various options.
- ▶ When you register a function, it applies to an entire test object class. You cannot register a method for a specific test object.
- ▶ If you want to call a function from additional test objects, you can copy the RegisterUserFunc line, paste it immediately after another function and replace any relevant argument values.
- ▶ If the function you are registering is defined in a function library, it is recommended to include the RegisterUserFunc statement in the function library as well so that the method will be immediately available for use in any test using that function library.

- ▶ QuickTest clears all method registrations at the beginning of each run session.
- ▶ If you use a partial run or debug option, such as **Run from step** or **Start from step**, to begin running a test from a point after method registration was performed in a test step (and not in a function library), QuickTest does not recognize the method registration because it occurred prior to the beginning of the current run session.
- ▶ To use an **Option Explicit** statement in a function library associated with your test, you must include it in all the function libraries associated with the test. If you include an **Option Explicit** statement in only some of the associated function libraries, QuickTest ignores all the **Option Explicit** statements in all function libraries. You can use **Option Explicit** statements directly in your action scripts without any restrictions.
- ▶ Each function library must have unique variables in its global scope. If you have two associated function libraries that define the same variable in the global scope using a Dim statement or define two constants with the same name, the second definition causes a syntax error. If you need to use more than one variable with the same name in the global scope, include a Dim statement only in the last function library (since function libraries are loaded in the reverse order).
- ▶ By default, steps that use user-defined functions are not displayed in the test results tree of the Test Results window after a run session. If you want the function to appear in the test results tree, you must add a **Reporter.ReportEvent** statement to the function code. For example, you may want to provide additional information or to modify the test status, if required.
- ▶ If you delete a function in use from an associated function library, the test step using the function will display the  icon. In subsequent run sessions for the test, an error will occur when the step using the non-existent function is reached.
- ▶ If another user modifies a function library that is referenced by a test, or if you modify the function library using an external editor (not QuickTest), the changes will take effect only after the test is reopened.

- ▶ When more than one function with the same name exists in the test script or function library, the last function will always be called. (QuickTest searches the test script for the function prior to searching the function libraries.) To avoid confusion, make sure that you verify that within the resources associated with a test, each function has a unique name.
- ▶ If you register a method within a reusable action, it is strongly recommended to unregister the method at the end of the action (and then re-register it at the beginning of the next action if necessary), so that tests calling your action will not be affected by the method registration.
- ▶ You can re-register the same method to use different user-defined functions without first unregistering the method. However, when you do unregister the method, it resets to its original QuickTest functionality (or is cleared completely if it was a new method), and not to the previous registration.

For example, suppose you enter the following statements:

```
RegisterUserFunc "Link", "Click", "MyClick"  
RegisterUserFunc "Link", "Click", "MyClick2"  
UnRegisterUserFunc "Link", "Click"
```

After running the **UnRegisterUserFunc** statement, the **Click** method stops using the functionality defined in the **MyClick2** function, and returns to the original QuickTest **Click** functionality, and not to the functionality defined in the **MyClick** function.

- ▶ For more information about creating functions and subroutines using VBScript, you can view the VBScript documentation from the QuickTest Help menu (**Help > QuickTest Professional Help > VBScript Reference**).

Executing Externally-Defined Functions from Your Test

If you decide not to associate a function library (any VBScript file) with a test, but do want to be able to call its functions, subroutines, and so forth from an action in your test or from another function library, you can do so by inserting an **ExecuteFile** statement in your action.

When you run your test, the **ExecuteFile** statement executes all global code in the function library making all definitions in the file available from the global scope of the action's script.

Note: You cannot debug a file that is called using an **ExecuteFile** statement, or any of the functions contained in the file. In addition, when debugging a test that contains an **ExecuteFile** statement, the execution marker may not be correctly displayed.

Tip: If you want to include the same **ExecuteFile** statement in every action you create, you can add the statement to an action template. For more information, refer to “Creating an Action Template” on page 487 in the *QuickTest Professional Basic Features User's Guide*.

To execute an externally-defined function:

- 1** Create a VBScript file using standard VBScript syntax. For more information, refer to the Microsoft VBScript Language Reference (**Help > QuickTest Professional Help > VBScript Reference > VBScript**).
- 2** Store the file in any folder that you can access from the computer running your test.
- 3** Add an **ExecuteFile** statement to an action in your test using the following syntax:

ExecuteFile *FileName*

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

- 4 Use the functions, subroutines, and so forth, from the specified VBScript file as necessary in your action.

Notes:

The **ExecuteFile** statement utilizes the VBScript **ExecuteGlobal** statement. For more information, refer to the Microsoft VBScript Language Reference (choose **Help > QuickTest Professional Help > VBScript Reference > VBScript**).

When you run an **ExecuteFile** statement within an action, you can call the functions in the file only from the current action. To make the functions in a VBScript file available to your entire test, add the file name to the associated function libraries list in the Resources tab of the Test Settings dialog box. For more information, see “Working with Associated Function Libraries” on page 200.

7

Automating QuickTest Operations

Just as you use QuickTest to automate the testing of your applications, you can use the QuickTest Professional automation object model to automate your QuickTest operations. Using the objects, methods, and properties exposed by the QuickTest automation object model, you can write programs that configure QuickTest options and run tests instead of performing these operations manually using the QuickTest interface.

Automation programs are especially useful for performing the same tasks multiple times or on multiple tests, or quickly configuring QuickTest according to your needs for a particular environment or application.

This chapter describes:

- About Automating QuickTest Operations
- Deciding When to Use QuickTest Automation Programs
- Choosing a Language and Development Environment for Designing and Running Automation Programs
- Learning the Basic Elements of a QuickTest Automation Program
- Generating Automation Scripts
- Using the QuickTest Automation Object Model Reference

About Automating QuickTest Operations

You can use the QuickTest Professional automation object model to write programs that automate your QuickTest operations. The QuickTest automation object model provides objects, methods, and properties that enable you to control QuickTest from another application.

What is Automation?

Automation is a Microsoft technology that makes it possible to access software objects inside one application from other applications. These objects can be easily created and manipulated using a scripting or programming language such as VBScript or VC++. Automation enables you to control the functionality of an application programmatically.

An **object model** is a structural representation of software objects (classes) that comprise the implementation of a system or application. An object model defines a set of classes and interfaces, together with their properties, methods and events, and their relationships.

What is the QuickTest Automation Object Model?

Essentially all configuration and run functionality provided via the QuickTest interface is in some way represented in the QuickTest automation object model via objects, methods, and properties. Although a one-on-one comparison cannot always be made, most dialog boxes in QuickTest have a corresponding automation object, most options in dialog boxes can be set and/or retrieved using the corresponding object property, and most menu commands and other operations have corresponding automation methods.

You can use the objects, methods, and properties exposed by the QuickTest automation object model, along with standard programming elements such as loops and conditional statements to design your program.

Automation programs are especially useful for performing the same tasks multiple times or on multiple tests, or quickly configuring QuickTest according to your needs for a particular environment or application.

For example, you can create and run an automation program from Microsoft Visual Basic that loads the required add-ins for a test, starts QuickTest in visible mode, opens the test, configures settings that correspond to those in the Options, Test Settings, and Record and Run Settings dialog boxes, runs the test, and saves the test.

You can then add a simple loop to your program so that your single program can perform the operations described above for multiple tests.

You can also create an initialization program that opens QuickTest with specific configuration settings. You can then instruct all of your testers to open QuickTest using this automation program to ensure that all of your testers are always working with the same configuration.

Deciding When to Use QuickTest Automation Programs

Like the tests you design using QuickTest, creating a useful QuickTest automation program requires planning, design time, and testing. You must always weigh the initial investment with the time and human-resource savings you gain from automating potentially long or tedious tasks.

Any QuickTest operation that you must perform many times in a row or must perform on a regular basis is a good candidate for a QuickTest automation program.

The following are just a few examples of useful QuickTest automation programs:

- ▶ **Initialization programs**—You can write a program that automatically starts QuickTest and configures the options and the settings required for recording on a specific environment.
- ▶ **Maintaining your tests**—You can write a program that iterates over your collection of tests to accomplish a certain goal. For example:
 - ▶ **Updating values**—opening each test with the proper add-ins, running it in update run mode against an updated application, and saving it in order to update the values in all of your tests to match the updated values in your application.
 - ▶ **Applying new options to existing tests**—When you upgrade to a new version of QuickTest, you may find that the new version offers certain options that you want to apply to your existing tests. You can write a program that opens each existing test, sets values for the new options, then saves and closes it.
- ▶ **Calling QuickTest from other applications**—You can design your own applications with options or controls that run QuickTest automation programs. For example, you could create a Web form or simple Windows interface from which a product manager could schedule QuickTest runs, even if the manager is not familiar with QuickTest.

Choosing a Language and Development Environment for Designing and Running Automation Programs

You can choose from a number of object-oriented programming languages for your automation programs. For each language, there are a number of development environments available for designing and running your automation programs.

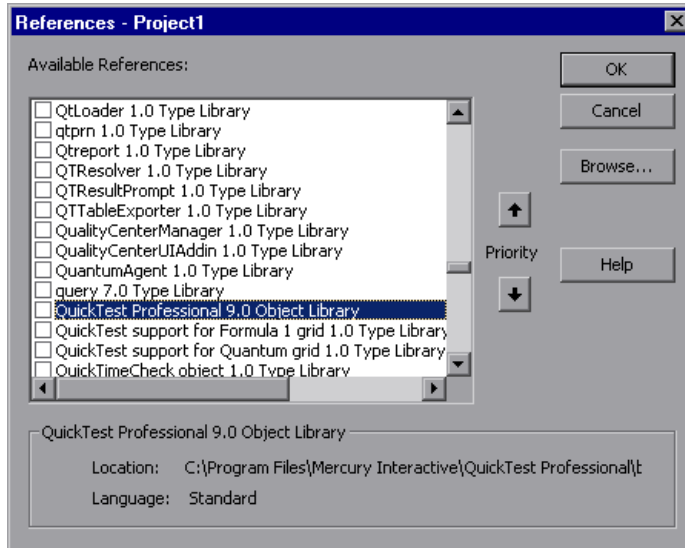
Writing Your Automation Program

You can write your QuickTest automation programs in any language and development environment that supports automation. For example, you can use: VBScript, JavaScript, Visual Basic, Visual C++, or Visual Studio.NET.

Some development environments support referencing a type library. A *type library* is a binary file containing the description of the objects, interfaces, and other definitions of an object model.

If you choose a development environment that supports referencing a type library, you can take advantage of features like Microsoft IntelliSense, automatic statement completion, and status bar help tips while writing your program. The QuickTest automation object model supplies a type library file named **QTOBJECTMODEL.dll**. This file is stored in **<QuickTest installation folder>\bin**.

If you choose an environment that supports it, be sure to reference the QuickTest type library before you begin writing or running your automation program. For example, if you are working in Microsoft Visual Basic, choose **Project > References** to open the References dialog box for your project. Then select **QuickTest Professional <Version> Object Library** (where <Version> is the current installed version of the QuickTest automation type library).



Running Your Automation Program

There are several applications available for running automation programs. You can also run automation programs from command line using Microsoft's Windows Script Host.

For example, you could use the following command line to run your automation program:

```
WScript.exe /E:VBSCRIPT myScript.vbs
```

Learning the Basic Elements of a QuickTest Automation Program

Like most automation object models, the root object of the QuickTest automation object model is the **Application** object. The **Application** object represents the application level of QuickTest. You can use this object to return other elements of QuickTest such as the **Test** object (which represents a test document), **Options** object (which represents the Options dialog box), or **Addins** collection (which represents a set of add-ins from the Add-in Manager dialog box), and to perform operations like loading add-ins, starting QuickTest, opening and saving tests, and closing QuickTest.

Each object returned by the **Application** object can return other objects, perform operations related to the object and retrieve and/or set properties associated with that object.

Every automation program begins with the creation of the QuickTest **Application** object. Creating this object does not start QuickTest. It simply provides an object from which you can access all other objects, methods and properties of the QuickTest automation object model.

Note: You can also optionally specify a remote QuickTest computer on which to create the object (the computer on which to run the program). For more information, refer to the “Running Automation Programs on a Remote Computer” section of the online *QuickTest Automation Object Model Reference*.

The structure for the rest of your program depends on the goals of the program. You may perform a few operations before you start QuickTest such as retrieving the associated add-ins for a test, loading add-ins, and instructing QuickTest to open in visible mode. After you perform these preparatory steps, if QuickTest is not already open on the computer, you can open QuickTest using the **Application.Launch** method. Most operations in your automation program are performed after the **Launch** method.

For information on the operations you can perform in an automation program, refer to the online *QuickTest Automation Object Model Reference*. For more information on this Help file, see “Using the QuickTest Automation Object Model Reference” on page 239.

When you finish performing the necessary operations, or you want to perform operations that require closing and restarting QuickTest, such as changing the set of loaded add-ins, use the **Application.Quit** method.

Generating Automation Scripts

The Properties tab of the Test Settings dialog box, the General tab of the Options dialog box, and the Object Identification dialog box each contain a **Generate Script** button. Clicking this button generates an automation script file (.vbs) containing the current settings from the corresponding dialog box.

You can run the generated script as is to open QuickTest with the exact configuration of the QuickTest application that generated the script, or you can copy and paste selected lines from the generated files into your own automation script.

For example, the generated script for the Options dialog box may look something like this:

```
Dim App 'As Application
Set App = CreateObject("QuickTest.Application")
App.Launch
App.Visible = True
App.Options.DisableVORRecognition = False
App.Options.AutoGenerateWith = False
App.Options.WithGenerationLevel = 2
App.Options.TimeToActivateWinAfterPoint = 500
...
...
App.Options.WindowsApps.NonUniqueListItemRecordMode = "ByName"
App.Options.WindowsApps.RecordOwnerDrawnButtonAs = "PushButtons"
App.Folders.RemoveAll
```

For more information on the **Generate Script** button and for information on the options available in the Options, Object Identification, and Test Settings dialog boxes, see Chapter 4, “Configuring Object Identification,” and refer to Chapter 24, “Setting Global Testing Options,” and Chapter 25, “Setting Options for Individual Tests” in the *QuickTest Professional Basic Features User’s Guide*.

Using the QuickTest Automation Object Model Reference

The QuickTest Automation Object Model Reference is a Help file that provides detailed descriptions, syntax information, and examples for the objects, methods, and properties in the QuickTest automation object model.

You can open the *QuickTest Automation Object Model Reference* from the:

- ▶ QuickTest program folder (**Start > Programs > QuickTest Professional > Documentation > QuickTest Automation Reference**)
- ▶ QuickTest Help menu (**Help > QuickTest Automation Object Model Reference**)

Part II

Managing and Merging Object Repositories

8

Managing Object Repositories

The Object Repository Manager enables you to manage all of the shared object repositories used in your organization from a single, central location, including adding and defining objects, modifying objects and their descriptions, parameterizing repositories to make them more generic, maintaining and organizing repositories, merging repositories, and importing and exporting repositories in XML format.

This chapter describes:

- About Managing Object Repositories
- Understanding the Object Repository Manager
- Working with Object Repositories
- Modifying Object Repositories
- Working with Repository Parameters
- Modifying Test Object Details
- Locating Objects
- Performing Merge Operations
- Performing Import and Export Operations

About Managing Object Repositories

The Object Repository Manager enables you to create and maintain shared object repositories. You can work with object repositories saved both in the file system and in a Quality Center project.

Each object repository contains the information that enables QuickTest to identify the objects in your application. QuickTest enables you to maintain the reusability of your tests by storing all the information regarding your test objects in a shared object repository. When objects in your application change, the Object Repository Manager provides a single, central location in which you can update test object information for multiple tests.

Note: Instead of, or in addition to, shared object repositories, you can choose to store all or some of the objects in a local object repository for each action. For more information on local object repositories, refer to Chapter 6, “Working with Test Objects” in the *QuickTest Professional Basic Features User’s Guide*.

If an object with the same name and description is located in both the local object repository and in a shared object repository that is associated with the same action, the action uses the local object definition. If an object with the same name and description is located in more than one shared object repository, and these shared object repositories are all associated with the same action, QuickTest uses the object definition from the first occurrence of the object, according to the order in which the shared object repositories are associated with the action. For more information on associating shared object repositories, refer to “Associating Object Repositories with Actions” on page 472 in the *QuickTest Professional Basic Features User’s Guide*.

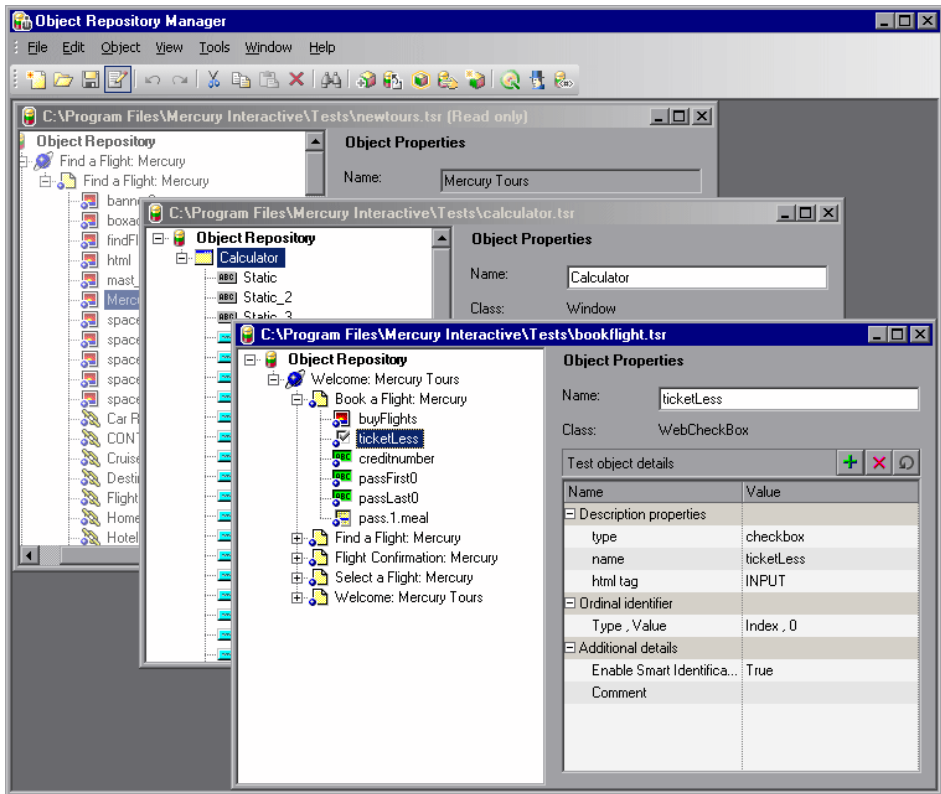
You can use the same shared object repository with multiple actions. You can also use multiple object repositories with each action. In addition, you can save objects directly with an action in a local object repository. This enables them to be accessed only from that action.

If one or more of the property values of an object in your application differ from the property values QuickTest uses to identify the object, your test may fail. Therefore, when the property values of objects in your application change, you should modify the corresponding test object property values in the corresponding object repository so that you can continue to use your existing tests.

You can modify objects in a shared object repository using the Object Repository Manager, as described in this chapter. You can modify objects stored in a local object repository using the Object Repository window. For information on the Object Repository window, refer to Chapter 6, “Working with Test Objects” in the *QuickTest Professional Basic Features User’s Guide*.

Understanding the Object Repository Manager

You open the Object Repository Manager by choosing **Resources > Object Repository Manager**. The Object Repository Manager enables you to open multiple shared object repositories and modify them as needed. You can open shared object repositories both from the file system and from a Quality Center project.



Tip: While the Object Repository Manager is open, you can continue working with other QuickTest windows.

You can open as many shared object repositories as you want. Each shared object repository opens in a separate document window. You can then resize, maximize, or minimize the windows to arrange them as you require to copy, drag, and move objects between different shared object repositories, as well as perform operations on a single object repository. For more information on the information shown in the shared object repository windows, see “Understanding the Shared Object Repository Windows” on page 251.











You open shared object repositories from the Open Shared Object Repository dialog box. In this dialog box, the **Open in read-only mode** check box is selected, by default. If you clear this check box, the shared object repository opens in editable mode. Otherwise, the shared object repository opens in read-only mode and you must click the **Enable Editing** button to modify it. For more information, see “Editing Object Repositories” on page 260.









When you choose a menu item or click a toolbar button in the Object Repository Manager, the operation you select is performed on the shared object repository whose window is currently active (in focus). The name and file path of the shared object repository is shown in the title bar of the window. For more information on the Object Repository Manager toolbar buttons, see “Using the Object Repository Manager Toolbar” on page 248.




Many of the shared object repository operations you can perform in the Object Repository Manager are done in a similar way to how you modify objects stored in a local object repository (using the Object Repository window). For this reason, many of the procedures are actually described in Chapter 6, “Working with Test Objects” in the *QuickTest Professional Basic Features User’s Guide*. Most of the procedures apply equally to the Object Repository Manager and the Object Repository window, but the windows and options may differ slightly.

Using the Object Repository Manager Toolbar

You can access frequently performed operations using the Object Repository Manager toolbar. The Object Repository Manager toolbar contains the following buttons:

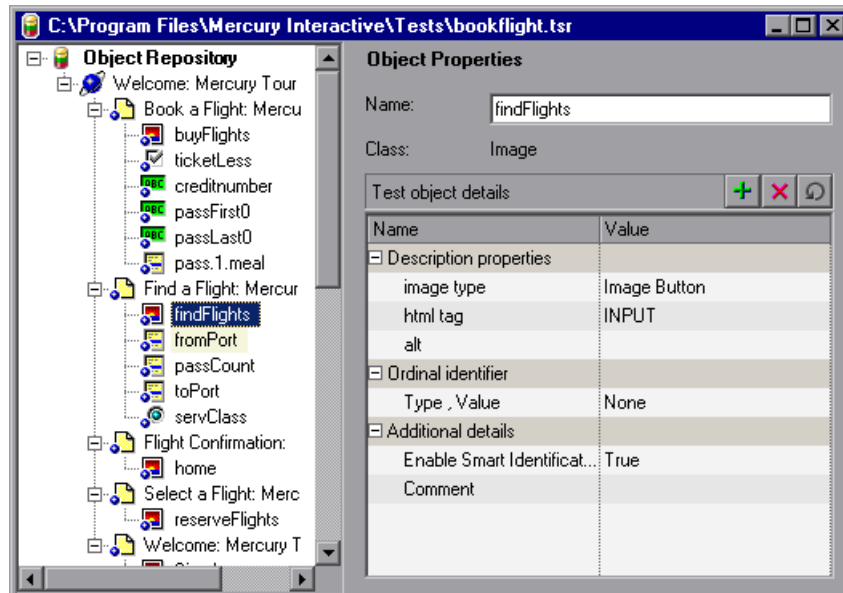
Button	Description
	Enables you to create a new shared object repository. For more information, see “Creating New Object Repositories” on page 253.
	Enables you to open a shared object repository from the file system or from Quality Center. For more information, see “Opening Object Repositories” on page 254.
	Enables you to save the active shared object repository to the file system or to Quality Center. For more information, see “Saving Object Repositories” on page 255.
	Enables you to edit the active shared object repository, by making the shared object repository editable. For more information, see “Editing Object Repositories” on page 260.
	Enables you to undo the previous operation performed in the active shared object repository. You do this in the same way as in a local object repository. For more information, see “Copying, Pasting, and Moving Objects in the Object Repository” on page 191.
	Enables you to redo the operation that was previously undone in the active shared object repository. You do this in the same way as in a local object repository. For more information, see “Copying, Pasting, and Moving Objects in the Object Repository” on page 191.
	Enables you to cut the selected item or object in the active shared object repository. You do this in the same way as in a local object repository. For more information, see “Copying, Pasting, and Moving Objects in the Object Repository” on page 191.
	Enables you to copy the selected item or object to the Clipboard in the active shared object repository. You do this in the same way as in a local object repository. For more information, see “Copying, Pasting, and Moving Objects in the Object Repository” on page 191.

Button	Description
	Enables you to paste the data from the Clipboard to the active shared object repository. You do this in the same way as in a local object repository. For more information, see “Copying, Pasting, and Moving Objects in the Object Repository” on page 191.
	Enables you to delete the selected item or object in the active shared object repository. You do this in the same way as in a local object repository. For more information, see “Deleting Objects from the Object Repository” on page 194.
	Enables you to find an object, property, or property value in the active shared object repository. You can also find and replace specified property values. You do this in the same way as in a local object repository. For more information, see “Finding Objects in an Object Repository” on page 195.
	Enables you to add objects to the active shared object repository. You do this in the same way as in a local object repository. For more information, see “Adding Objects to the Object Repository” on page 180.
	Enables you to update test object properties in the active shared object repository according to the actual properties of the object in your application. You do this in the same way as in a local object repository. For more information, see “Updating Test Object Properties from an Object in Your Application” on page 162.
	Enables you to select an object in the active shared object repository and highlight it in your application. You do this in the same way as in a local object repository. For more information, see “Highlighting an Object in Your Application” on page 198.
	Enables you to select an object in your application and highlight it in the active shared object repository. You do this in the same way as in a local object repository. For more information, see “Locating an Object in the Object Repository” on page 199.
	Enables you to define a test object that does not yet exist in your application and add it to the active shared object repository. You do this in the same way as in a local object repository. For more information, see “Defining New Test Objects” on page 188.

Button	Description
	<p>Enables you to connect to Quality Center to work with object repository files stored in a Quality Center project. You can connect to Quality Center from the main QuickTest window or from the Object Repository Manager. For more information, see “Connecting QuickTest to Quality Center” on page 364.</p>
	<p>Enables you to open the Object Spy to view run-time or test object properties and values of objects in your application. For more information, refer to “Viewing Object Properties Using the Object Spy” on page 66 in the <i>QuickTest Professional Basic Features User’s Guide</i>.</p>
	<p>Enables you to add, edit, and delete repository parameters in the active shared object repository. For more information, see “Managing Repository Parameters” on page 262.</p>

Understanding the Shared Object Repository Windows

Each shared object repository that you open in the Object Repository Manager is displayed in a standalone document window. Each shared object repository window displays a tree of all objects in the object repository, together with test object information for the selected object.



For each test object you select in the tree, the Object Repository window displays information about the selected test object. You can view the test object description of any test object in the shared object repository, modify test objects and their properties, and add objects to the shared object repository. For more information, see “Modifying Object Repositories” on page 258 and “Modifying Test Object Details” on page 268.

Each object repository window contains the following information:

Information	Description
Object Repository tree	Contains all test objects in the shared object repository.
Name	The name that QuickTest assigns to the selected test object. You can change the test object name. For more information, refer to “Renaming Test Objects” on page 164 in the <i>QuickTest Professional Basic Features User’s Guide</i> .
Class	The class of the selected object.
Test object details	Enables you to view and modify the properties and property values used to identify the selected object during a run session. For more information, see “Modifying Test Object Details” on page 268.

Note: Even when steps containing a test object are deleted from your action, the objects remain in the object repository. You can delete objects from a shared object repository using the Object Repository Manager, in much the same way as you delete objects from a local object repository. For more information, refer to “Deleting Objects from the Object Repository” on page 194 in the *QuickTest Professional Basic Features User’s Guide*.

Working with Object Repositories

You can use the Object Repository Manager to create new object repositories, open and modify existing object repositories, and save and close them when you are finished.

Creating New Object Repositories

You can create a new object repository, add objects to it, and then save it. You can then associate one or more actions with the object repository from within QuickTest. For more information on associating shared object repositories, refer to “Associating Object Repositories with Actions” on page 472 in the *QuickTest Professional Basic Features User’s Guide*.

To create a new object repository:



In the Object Repository Manager, choose **File > New** or click the **New** button. A new object repository opens. You can now add objects to it, modify it, and save it. For more information, see “Modifying Object Repositories” on page 258 and “Saving Object Repositories” on page 255.

Opening Object Repositories

You can open existing object repositories to view or modify them. You can open object repositories from the file system or from a Quality Center project.



You connect to a Quality Center project either from QuickTest or from the Object Repository Manager by choosing **File > Quality Center Connection** or clicking the **Quality Center Connection** button. For more information on connecting to Quality Center, see “Connecting QuickTest to Quality Center” on page 364.

Note for users of previous QuickTest versions:

When you open an object repository that was created using an earlier version of QuickTest, QuickTest must convert it to the current format before you can edit it.

If the object repository contains test objects from external add-ins, the relevant add-in must be installed to convert the object repository to the current format. Otherwise, you can open it only in read-only format.

If you do not want to convert the object repository, you can view it in read-only format. After the file is converted and you save it, you cannot use it with earlier versions of QuickTest.

To open an object repository:



- 1 In the Object Repository Manager, choose **File > Open** or click the **Open** button. The Open Shared Object Repository dialog box opens.

Note: If you are connected to Quality Center, the dialog box that opens is different from the standard file system dialog box. You can switch between the two dialog box versions by clicking the **File System** and **Quality Center** buttons in the Open Shared Object Repository dialog box.

- 2 Select the object repository you want to open, and click **Open** or **OK** (depending on whether you are opening it from the file system or a Quality Center project). The object repository opens.

By default, the object repository opens in read-only mode. You can open it in editable format by clearing the **Open in read-only mode** check box in the Open Shared Object Repository dialog box. You can also enable editing for an object repository as described in “Editing Object Repositories” on page 260.

If the object repository is editable, you can add objects to it, modify it, and save it. For more information, see “Modifying Object Repositories” on page 258 and “Saving Object Repositories” on page 255.

Tip: You can also open an object repository from the **Recent Files** list in the **File** menu.

Saving Object Repositories

After you finish creating or modifying an object repository, you should save it. When you modify an object repository, an asterisk (*) is displayed in the title bar until the object repository is saved.



You can save an object repository to the file system or to a Quality Center project (if you are connected to a Quality Center project). You connect to a Quality Center project either from QuickTest or from the Object Repository Manager by choosing **File > Quality Center Connection** or clicking the **Quality Center Connection** button. For more information on connecting to Quality Center, see “Connecting QuickTest to Quality Center” on page 364.

Note: All changes you make to an object repository are automatically updated in all tests open on the same computer that use the object repository as soon as you make the change—even if you have not yet saved the object repository with your changes. If you close the object repository without saving your changes, the changes are rolled back in any open tests that were open at the time. When you open a test on the same computer on which you modified the object repository, the test is automatically updated with all saved changes made in the associated object repository. To see saved changes in a test or repository open on a different computer, you must open the test or object repository file or lock it for editing on your computer to load the changes.

To save an object repository:

1 Make sure that the object repository you want to save is the active window.



2 Choose **File > Save** or click the **Save** button. If the file has already been saved, the changes you made are saved. If the file has not yet been saved, the Save Shared Object Repository dialog box opens.

Note: If you are connected to Quality Center, the dialog box that opens is different from the standard file system dialog box. You can switch between the two dialog box versions by clicking the **File System** and **Quality Center** buttons in the Open Shared Object Repository dialog box.

3 Select the folder in which you want to save the object repository.

- 4 Enter a name for the object repository in the **File name** or **Attachment Name** box (depending on whether you are saving it to the file system or a Quality Center project). Use a descriptive name that will help you easily identify the file.

Note: You cannot use any of the following characters in the object repository name:

\ / : * " ? < > |

- 5 Click **Save** or **OK** (depending on whether you are saving it to the file system or a Quality Center project). QuickTest saves the object repository with a **.tsr** extension in the specified location and displays the object repository name and path in the title bar of the repository window.

Closing Object Repositories

After you finish modifying or using an object repository, you should close it. When you close the file it is automatically unlocked so that it can be used or modified by others. You can also choose to close all open object repositories.

Note: If you close QuickTest, the Object Repository Manager also closes. If you have made changes that are not yet saved, you are prompted to do so before the Object Repository Manager closes.

To close an object repository:

- 1 Make sure that the object repository you want to close is the active window.
- 2 Choose **File > Close** or click the **Close** button in the object repository window's title bar. The object repository is closed and is automatically unlocked. If you have made changes that are not yet saved, you are prompted to do so before the file closes.

To close all open object repositories:

Choose **File > Close All Windows**, or **Window > Close All Windows**. All open object repositories are closed and are automatically unlocked. If you have made changes that are not yet saved, you are prompted to do so before the files close.

Modifying Object Repositories

You can modify your object repositories in a variety of ways to either prepare them for initial use or update them throughout the testing process. You can add and modify objects and object properties in a shared object repository, copy or move objects from one object repository to another, drag objects to a different location in the hierarchy, delete objects, and rename objects. When you modify an object repository, an asterisk (*) is displayed in the title bar until the object repository is saved.



Tip: You can use the **Edit > Undo** and **Edit > Redo** options or **Undo** and **Redo** buttons to cancel or repeat your changes as necessary. The **Undo** and **Redo** options are related to the active document. When you save an object repository, you cannot undo and redo operations that were performed on that file before the save operation.

If you opened the object repository in read-only mode, you must enable editing for the object repository before you can modify it. This locks the object repository and prevents it from being modified simultaneously by multiple users.

Note: All changes you make to an object repository are automatically updated in all tests open on the same computer that use the object repository as soon as you make the change—even if you have not yet saved the object repository with your changes. If you close the object repository without saving your changes, the changes are rolled back in any open tests that were open at the time. When you open a test on the same computer on which you modified the object repository, the test is automatically updated with all saved changes made in the associated object repository. To see saved changes in a test or repository open on a different computer, you must open the test or object repository file or lock it for editing on your computer to load the changes.

Tip: You can also modify a shared object repository by merging it with another shared object repository. If you merge two shared object repositories, a new shared object repository is created, containing the content of both object repositories. If you merge a shared object repository with a local object repository, the shared object repository is updated with the content of the local object repository. For more information, see Chapter 9, “Merging Shared Object Repositories.”

After making sure that your shared object repository is editable, and that it is the active window, you modify it in the same way as you modify a local object repository. For more information, see:

- ▶ “Editing Object Repositories,” below.
- ▶ “Adding Objects to the Object Repository” on page 180 in the *QuickTest Professional Basic Features User’s Guide*.
- ▶ “Copying, Pasting, and Moving Objects in the Object Repository” on page 191 in the *QuickTest Professional Basic Features User’s Guide*.
- ▶ “Deleting Objects from the Object Repository” on page 194 in the *QuickTest Professional Basic Features User’s Guide*.

Editing Object Repositories

When you open an object repository, it is opened in read-only mode by default. You can open it in editable format by clearing the **Open in read-only mode** check box in the Open Shared Object Repository dialog box when you open it.

If you opened the object repository in read-only mode, you must enable editing for the object repository before you can modify it. You do not need to enable editing for an object repository if you only want to view it or copy objects from it to another object repository.

When you enable editing for an object repository, it locks the object repository so that it cannot be modified by other users. To enable other users to modify the object repository, you must first unlock it (by disabling edit mode, or by closing it). If an object repository is already locked by another user, it is saved in read-only format, or if you do not have the permissions required to open it, you cannot enable editing for it.

Note for users of previous QuickTest versions: If you want to edit an object repository that was created using an earlier version of QuickTest, QuickTest must convert it to the current format before you can edit it. If you do not want to convert it, you can view it in read-only format. After the file is converted and saved, you cannot use it with earlier versions of QuickTest.

To enable editing for an object repository:

- 1 Make sure that the object repository you want to edit is the active window.
- 2 Choose **File > Enable Editing** or click the **Enable Editing** button. The object repository becomes editable.

Working with Repository Parameters

Repository parameters enable you to specify that certain property values should be parameterized, but leave the actual parameterization to be defined in each test that is associated with the object repository that contains the parameterized test object property values.

Repository parameters are useful when you want to create and run tests on an object that changes dynamically. An object may change dynamically if it is frequently updated in the application, or if its property values are set using dynamic content, for example, from a database.

For example, you may have a button whose text property value changes in a localized application depending on the language of the user interface. You can parameterize the name property value using a repository parameter, and then in each test that uses the object repository you can specify the location from which the property value should be taken. For example, in one test that uses this object repository you can specify that the property value comes from an environment variable, in another test it can come from the Data Table, and in a third test you can specify it as a constant value.


You define all the repository parameters for a specific object repository using the Manage Repository Parameters dialog box. You define each repository parameter together with an optional default value and meaningful description. For more information, see “Managing Repository Parameters” on page 262.

When you open a test that uses an object repository with a repository parameter that has no default value, an indication that there is a repository parameter that needs mapping is displayed in the Missing Resources pane. You can then map the repository parameter as needed in the test. You can also map repository parameters that have default values, and change mappings for repository parameters that are already mapped. For more information on mapping repository parameters, refer to “Handling Unmapped Shared Object Repository Parameter Values” on page 499 in the *QuickTest Professional Basic Features User’s Guide*.

Managing Repository Parameters

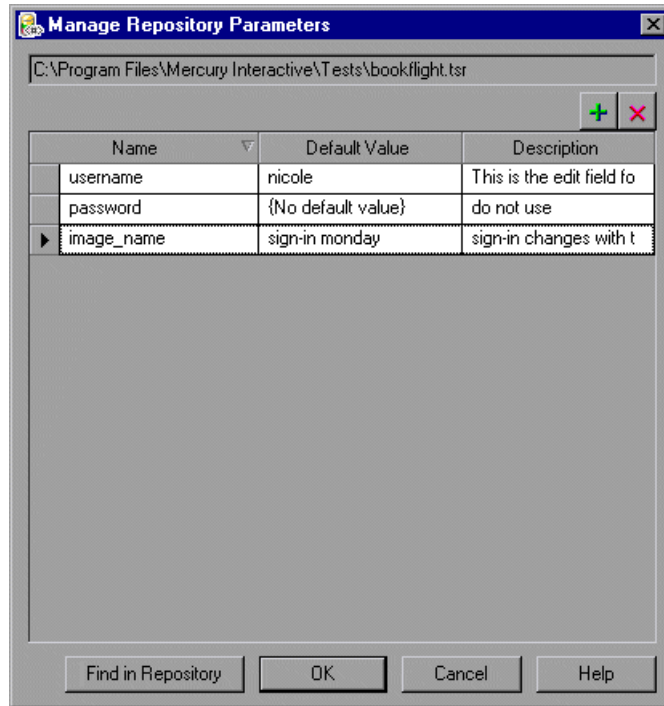
The Manage Repository Parameters dialog box enables you to add, edit, and delete repository parameters for a single shared object repository.

To manage repository parameters:



- 1 Make sure that the object repository whose parameters you want to manage is the active window.
- 2  If the object repository is in read-only format, choose **File > Enable Editing** or click the **Enable Editing** button. The object repository becomes editable.



- 3 Choose **Tools > Manage Repository Parameters** or click the **Manage Repository Parameters** button. The Manage Repository Parameters dialog box opens.



The Manage Repository Parameters dialog box contains the following information and options:

Option	Description
Repository name	Displays the name and path of the object repository whose repository parameters you are managing.
	Enables you to add a new repository parameter. For more information, see “Adding Repository Parameters” on page 264.
	Enables you to delete the currently selected repository parameter(s). For more information, see “Deleting Repository Parameters” on page 267.
Parameter list	Displays the list of repository parameters currently defined in this object repository. You can modify a parameter’s default value and description directly in the parameter list. For more information, see “Modifying Repository Parameters” on page 266.
Find in Repository	Searches for and highlights the first test object in the object repository tree that uses the selected repository parameter. You can click this button again to find the next occurrence of the selected parameter, and so forth.

Adding Repository Parameters

The Add Repository Parameter dialog box enables you to define a new repository parameter. You can also specify a default value for the parameter, and a meaningful description to help identify it when it is used in a test step.

To add a repository parameter:

- 1 In the Manage Repository Parameters dialog box, click the **Add Repository Parameter** button. The Add Repository Parameter dialog box opens.

- 2 In the **Name** box, specify a meaningful name for the parameter. Parameter names must start with an English letter and can contain only alphanumeric characters and underscores.
- 3 In the **Default value** box, you can specify a default value to be used for the repository parameter. This value is used if you do not map the repository parameter to a value or parameter type in a test that uses this object repository. If you do not specify a default value, the repository parameter will appear as unmapped in any tests that use this shared object repository.





Tip: If you specify a default value, you can later remove it by clicking in the **Default Value** cell of the relevant parameter in the Manage Repository Parameters dialog box and then clicking the **Clear Default Value** button. The text **{No Default Value}** is displayed in the cell.

- 4 In the **Description** box, you can enter a description of the repository parameter. The description will help you identify the parameter when mapping repository parameters within a test.
- 5 Click **OK** to add the parameter to the list of parameters in the Manage Repository Parameters dialog box.

Modifying Repository Parameters

You can modify the default value of a repository parameter or modify a repository parameter description directly in the Manage Repository Parameters dialog box. However, you cannot modify a repository parameter name.

To modify a repository parameter:


- 1 In the Manage Repository Parameters dialog box, select the required parameter.
 -  2 To modify the default value, click in the **Default Value** cell of the required parameter. You can either modify the default value by entering a new value, or you can remove the default value by clicking the **Clear Default Value** button. If you remove the default value, the text **{No Default Value}** is displayed in the cell. If you do not specify a default value, the repository parameter will appear as unmapped in any tests that use this shared object repository.
-
-  **Note:** If you delete the text manually, it does not remove the default value. It creates a default value of an empty string. You must click the **Clear Default Value** button if you want to remove the default value.
-
- 3 To modify the parameter description, click in the **Description** cell of the required parameter and enter the required description.

Deleting Repository Parameters

You can delete a repository parameter definition if it is no longer needed. When you delete a repository parameter that is used in a test object definition, the test object property value remains mapped to the parameter, even though the parameter no longer exists. Therefore, before deleting a repository parameter, you should make sure that it is not used in any test object descriptions, otherwise tests that have steps using these test objects will fail when you run them.

Tip: You can use the **Find in Repository** button in the Manage Repository Parameters dialog box to see where a repository parameter is being used.

To delete a repository parameter:

- 1 In the Manage Repository Parameters dialog box, select the repository parameter(s) that you want to delete by clicking in the selection area to the left of the parameter name.
- 2  Click the **Delete Repository Parameter** button. The selected repository parameter is deleted.

Modifying Test Object Details

The **Test object details** area for shared object repositories open in the Object Repository Manager enables you to view and modify the properties and property values used to identify an object during a run session.

After making sure that your shared object repository is editable, and that it is the active window, you modify test object details for objects in a shared object repository in the same way as you modify them for local objects. For more information, see the *QuickTest Professional Basic Features User's Guide*:

- ▶ “Adding Properties to a Test Object Description” on page 167
- ▶ “Defining New Test Object Properties” on page 171
- ▶ “Updating Test Object Properties from an Object in Your Application” on page 162
- ▶ “Restoring Default Properties for a Test Object” on page 164
- ▶ “Removing Properties from a Test Object Description” on page 173
- ▶ “Specifying Ordinal Identifiers” on page 174
- ▶ “Renaming Test Objects” on page 164



Note: You can use the **Edit > Undo** and **Edit > Redo** options or **Undo** and **Redo** buttons to cancel or repeat your changes as necessary. The **Undo** and **Redo** options are related to the active document. When you save a repository, you cannot undo and redo operations that were performed on that file before the save operation.

You use the Object Repository Manager to specify property values for test object descriptions in a shared object repository. The options available when specifying property values for objects in shared object repositories are different from those available when specifying properties for objects in local repositories. For more information on specifying property values for objects in shared object repositories, see “Specifying a Property Value,” on page 269.

Specifying a Property Value

You can specify or modify values for properties in the test object description. You can specify a value using a constant value (either a simple value or a constant value that includes regular expressions) or you can parameterize it using a repository parameter. For more information on repository parameters, see “Working with Repository Parameters” on page 261.

To specify a property value:

- 1 Select the test object whose property value you want to specify.
- 2 In the **Test object details** area, click in the value cell for the required property.
- 3 Specify the property value in one of the following ways:
 - ▶ If you want to specify a simple constant value, enter it in the value cell. The remaining steps in this procedure are not necessary if you specify a constant value in the value cell. You can also specify a constant value using a regular expression in the Repository Parameter dialog box, as described below.
 - ▶ If you want to parameterize the value using a repository parameter, click the parameterization button in the value cell. The Repository Parameter dialog box opens.



Repository Parameter

Specify a constant value or select a repository parameter name for the property you want to parameterize.

Constant: Regular expression

Parameter:

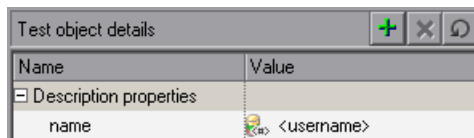
Default value:

OK Cancel Help

- 4 Choose one of the following options to specify a value for the property:
 - ▶ Select the **Constant** radio button and specify a constant value. You can also enter a constant value directly in the value cell of the **Test object details** area. If you used a regular expression in the constant value, select the **Regular expression** check box.
 - ▶ Select the **Parameter** radio button and select a repository parameter from the list of defined parameters. If a default value is defined for the parameter, it is also shown.

Note: You define repository parameters using the Manage Repository Parameters dialog box. For more information, see “Managing Repository Parameters” on page 262.

- 5 Click **OK** to close the Repository Parameter dialog box. If you parameterized the value, the parameter name is shown with an icon in the **Value** column of the **Test object details** area, as shown below. Otherwise, the constant value you specified is shown in the **Value** column.



Locating Objects

You can search for a specific object in your object repository in several ways. You can search for an object according to its type. For example, you can search for a specific edit box, or you can point to an object in your application to automatically highlight that same object in your repository. You can replace specific property values with other property values. For example, you can replace a property value `userName` with the value `user name`. You can also select an object in your object repository and highlight it in your application to check which object it is.

After making sure that your shared object repository is the active window, you locate an object in a shared object repository in the same way as you locate it in a local object repository. If you want to replace property values, you must also make sure that the object repository is editable.

For more information, see the *QuickTest Professional Basic Features User's Guide*:

- ▶ “Finding Objects in an Object Repository” on page 195
- ▶ “Highlighting an Object in Your Application” on page 198
- ▶ “Locating an Object in the Object Repository” on page 199

Performing Merge Operations

The Object Repository Merge Tool enables you to merge objects from the local object repository of one or more actions to a shared object repository using the **Update from Local Repository** option in the Object Repository Manager (**Tools > Update from Local Repository**). For example, you may have learned objects locally in a specific action in your test and want to add them to the shared object repository so they are available to all actions in different tests that use that object repository. You can also use the Object Repository Merge Tool to merge two shared object repositories into a single shared object repository.

You open the Object Repository Merge Tool by choosing **Tools > Object Repository Merge Tool** in the Object Repository Manager. For more information on performing merge operations and updating object repositories with local objects, see Chapter 9, “Merging Shared Object Repositories.”

Note: While the Object Repository Merge Tool is open, you cannot work with the Object Repository Manager.

Performing Import and Export Operations

You can import and export object repositories from and to XML files. XML provides a structured, accessible format that enables you to make changes to object repositories using the XML editor of your choice and then import them back into QuickTest. You can view the required format for the object repository by exporting a saved object repository.

You can import and export files either from and to the file system or a Quality Center project (if QuickTest is connected to Quality Center).



You connect to a Quality Center project either from QuickTest or from the Object Repository Manager by choosing **File > Quality Center Connection** or clicking the **Quality Center Connection** button. For more information on connecting to Quality Center, see “Connecting QuickTest to Quality Center” on page 364.

Importing from XML

You can import an XML file (created using the required format) as an object repository. The XML file can either be an object repository that you exported to XML format using the Object Repository Manager, or an XML file created using a tool such as QuickTest Siebel Test Express or a custom built utility. You must adhere to the XML structure and format.

Tip: To view the required XML structure and format, you can export an existing shared object repository to XML and then use the XML file as a guide. For more information, see “Exporting to XML” on page 273.

To import from XML:

- 1 Choose **File > Import from XML**. The Import from XML dialog box opens.

Note: If you are connected to Quality Center, the dialog box that opens is different from the standard file system dialog box. You can switch between the two dialog box versions by clicking the **File System** and **Quality Center** buttons in the Import from XML dialog box.

- 2 Select the XML file you want to import, and click **Open** or **OK** (depending on whether you are opening it from the file system or a Quality Center project).
- 3 The XML file is imported and a summary message box opens showing information regarding the number of objects, parameters, and metadata that were successfully imported from the specified file.
- 4 Click **OK** to close the message box. The imported XML file is opened as a new object repository. You can now modify it as required and save it as an object repository.

Exporting to XML

You can export the contents of an object repository to an XML file. This enables you to easily edit it using any XML editor, and also enables you to save it in an accessible, versatile format.

To export to XML:

- 1** Make sure that the object repository you want to export is the active window.
- 2** Choose **File > Export to XML**. The Export to XML dialog box opens.

Note: If you are connected to Quality Center, the dialog box that opens is different from the standard file system dialog box. You can switch between the two dialog box versions by clicking the **File System** and **Quality Center** buttons in the Export to XML dialog box.

- 3** Select the location in which to save the file, specify the file or attachment name, and click **Save** or **OK** (depending on whether you are saving it to the file system or a Quality Center project).
- 4** The object repository is exported to the specified XML file and a summary message box opens showing information regarding the number of objects, parameters, and metadata that were successfully exported to the specified file.
- 5** Click **OK** to close the message box. You can now open the XML file and view or modify it with any XML editor.

9

Merging Shared Object Repositories

QuickTest Professional enables you to merge two shared object repositories into a single shared object repository using the Object Repository Merge Tool. You can also use this tool to merge objects from the local object repository of one or more actions into a shared object repository.

This chapter describes:

- ▶ About Merging Shared Object Repositories
- ▶ Understanding the Object Repository Merge Tool
- ▶ Using Object Repository Merge Tool Commands
- ▶ Defining Default Settings
- ▶ Merging Two Object Repositories
- ▶ Updating a Shared Object Repository from Local Object Repositories
- ▶ Viewing Merge Statistics
- ▶ Understanding Object Conflicts
- ▶ Resolving Object Conflicts
- ▶ Filtering the Target Repository Pane
- ▶ Synchronizing Object Repository Views
- ▶ Finding Specific Objects
- ▶ Saving the Target Object Repository

About Merging Shared Object Repositories

QuickTest Professional provides the ability to merge existing assets from two repositories into a single shared object repository using the Object Repository Merge Tool. This tool enables you to merge two shared object repositories (called the **primary** object repository and the **secondary** object repository), into a new third object repository, called the **target** object repository. Objects in the primary and secondary object repositories are automatically compared and then added to the target object repository according to preconfigurable rules that define how conflicts between objects are resolved.

After the merge process, the Object Repository Merge Tool provides a graphic presentation of the original objects in the primary and secondary repositories, which remain unchanged, as well as the objects in the merged target object repository. Objects that had conflicts are highlighted. The conflict of each object that you select in the target object repository is described in detail. The Object Repository Merge Tool provides specific options that enable you to keep the suggested resolution for each conflict, or modify each conflict resolution individually, according to your requirements.

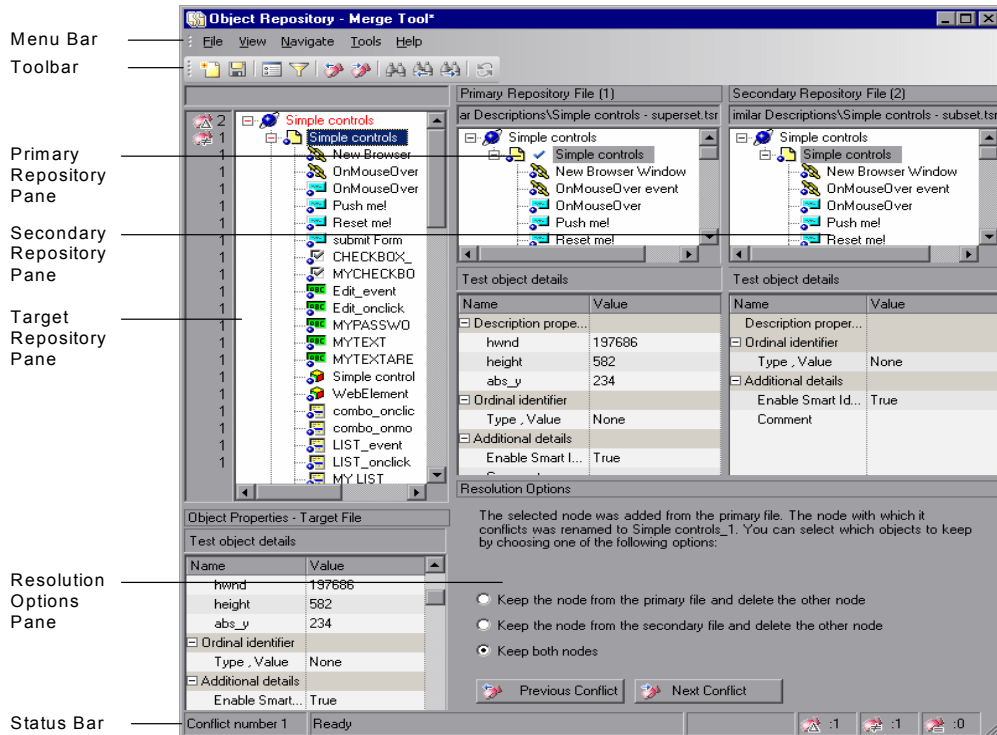
The Object Repository Merge Tool also enables you to merge objects from the local object repository of one or more actions into a shared object repository. For example, you may have learned objects locally in a specific action in your test and want to add them to the shared object repository, so that they are available to all actions in different tests that use that object repository.

Note: When the Object Repository Merge Tool is open, you cannot work with the Object Repository Manager. For more information on the Object Repository Manager, see Chapter 8, “Managing Object Repositories.”

Understanding the Object Repository Merge Tool

You open the Object Repository Merge Tool by choosing **Tools > Object Repository Merge Tool** in the Object Repository Manager.

An example of the Object Repository - Merge Tool window is shown below:



The Object Repository - Merge Tool window contains the following key elements:

- ▶ **Menu bar**—Displays menus of Object Repository Merge Tool commands. These commands are described in various places throughout this chapter. Shortcut keys for menu commands are described in “Performing Commands Using Shortcut Keys” on page 283.
- ▶ **Toolbar**—Contains buttons of commonly used menu commands to assist you in merging, managing, and saving object repositories. Toolbar buttons are described in “Using Toolbar Commands” on page 282.

- ▶ **Target Repository Pane**—Displays the test objects that were merged from the primary and secondary repositories. You can also choose to show or hide the Target Repository Object Properties pane, which displays the properties of any test object that is selected in the Target Repository pane. For more information, see “Target Repository Pane” on page 279.
- ▶ **Primary Repository Pane**—Displays the test objects in the primary object repository. For more information, see “Primary and Secondary Repository Panes” on page 280.
- ▶ **Secondary Repository Pane**—Displays the test objects in the secondary object repository. For more information, see “Primary and Secondary Repository Panes” on page 280.
- ▶ **Resolution Options Pane**—Provides source, conflict, and resolution details about the objects in the target repository pane, and enables you to modify the resolution method that was applied to any conflict. For more information, see “Resolution Options Pane” on page 280.
- ▶ **Status Bar**—Provides source, conflict, and resolution details about the object selected in the target repository pane, and an icon legend. For more information, see “Status Bar” on page 281.

Changing the View

You can change the view presented by the Object Repository Merge Tool according to your working preferences.

- ▶ Drag the edges of the panes to resize them in the Object Repository Merge Tool window.
- ▶ Choose **Primary Repository**, **Secondary Repository**, **Target Repository Object Properties**, or **Resolution Options** from the **View** menu to hide or show these panes in the Object Repository Merge Tool.
- ▶ Choose **View > Set as Default Layout** to set your current view as the default view, which displays each time you open the Object Repository Merge Tool. You can choose **View > Restore Default Layout** to restore the view to the default settings after you have made any changes.

Target Repository Pane

The target repository pane displays a hierarchy of the test objects, as well as their respective properties and values, that were merged from the primary and secondary repositories. In the column to the left of the object hierarchy, the pane displays the source file of each object (**1** is displayed for the primary file and **2** for the secondary file), and an icon representing the type of conflict, if any.

When you save the target object repository, the file path is displayed above the object hierarchy.

Note: To make it easier to see the status of an object at a glance, the text colors of the object names in the target object repository can be set according to their source and whether they caused a conflict. For more information, see “Specifying Color Settings” on page 284.

The target repository pane provides the following functionality:

- ▶ When you select an object in the target object repository, the corresponding object in the primary and/or secondary source file hierarchy is located and indicated by a check mark.
- ▶ When you select an object in the target object repository, its properties and values are displayed in the **Object Properties - Target File** area at the bottom of the target repository pane (**View > Target Repository Object Properties**).
- ▶ If the merge results in a conflict, an icon is displayed to the left of the conflicting object in the target object repository. You can see a tooltip description of the conflict type by positioning your pointer over the icon.
- ▶ When you right-click an object, a context-sensitive menu opens. You can choose an option to expand or collapse the entire hierarchy in the target object repository, or, when applicable, to change the conflict resolution method and result.
- ▶ You can expand or collapse the hierarchy of the node by double-clicking a node. You can also expand or collapse the entire hierarchy in the target object repository by choosing **Collapse All** or **Expand All** from the **View** menu.



- ▶ You can jump directly to the next or previous conflict in the target object repository hierarchy by choosing **Next Conflict** or **Previous Conflict** from the **Navigate** menu, or by clicking the **Next Conflict** or **Previous Conflict** buttons in the toolbar or Resolution Options pane.
- ▶ You can locate one or more objects in the target object repository by using the Find dialog box. For more information, see “Finding Specific Objects” on page 304.
- ▶ You can show or hide the target repository object properties by choosing **View > Target Repository Object Properties**.

Primary and Secondary Repository Panes

The primary and secondary repository panes display the hierarchies of the test objects, and their properties and values, in the original source repositories that you chose to merge. The file path is shown above each object hierarchy.

The panes provide the following functionality:

- ▶ You can expand or collapse the hierarchy of a selected item by double-clicking the item.
- ▶ You can view the properties and values of an object in the **Test object details** area by selecting it in the relevant pane.
- ▶ You can show or hide the panes by selecting or clearing **Primary Repository** or **Secondary Repository** in the **View** menu.

Resolution Options Pane

The Resolution Options pane provides information about any conflict encountered during the merge for the object selected in the target object repository. The pane also provides options that enable you to keep or change the conflict resolution method that was applied using the default resolution options.

The Resolution Options pane provides the following functionality:

- ▶ When you select a conflicting object in the target object repository, the pane displays a textual description of the conflict and the resolution method used by the Object Repository Merge Tool. A choice of alternative resolution methods is offered.
- ▶ You can select a radio button to choose an alternative resolution method for the conflict. Every time you make a change, the target object repository is automatically updated and is redisplayed.
- ▶ You can jump directly to the next or previous conflict in the target repository hierarchy by clicking the **Previous Conflict** or **Next Conflict** buttons.
- ▶ You can show or hide the pane by selecting or clearing **Resolution Options** in the **View** menu.

Status Bar

The status bar shows the conflict number (if any) of the object selected in the target repository pane, and a legend of the icons used in the target repository pane.

The following icons may be displayed in the status bar and (and in the target repository pane):



- ▶ Similar Description Conflict



- ▶ Same Name Different Description Conflict



- ▶ Same Description Different Name Conflict

Position your pointer over the icon to see a tooltip description of the conflict type.

For more information on conflict types, see “Understanding Object Conflicts” on page 297.

Tips:

Click any of the conflict icons in the Status bar to view the Statistics dialog box. For more information, see “Viewing Merge Statistics” on page 296.



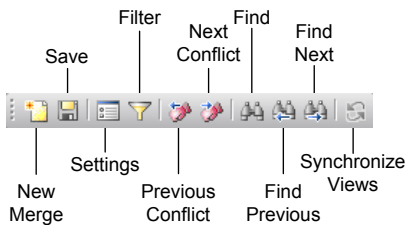
Click in the box to the left of the icons to view the Filter dialog box. This area shows a Filter icon when a filter is currently in use. For more information, see “Filtering the Target Repository Pane” on page 302.

Using Object Repository Merge Tool Commands

You can select Object Repository Merge Tool commands from the menu bar or from the toolbar. Certain commands can be executed by pressing shortcut keys, as described in “Performing Commands Using Shortcut Keys” on page 283. You can also select an object in the target repository pane and choose commands from the context-sensitive (right-click) menu.

Using Toolbar Commands

You can perform frequently used commands by clicking buttons in the toolbar.



Performing Commands Using Shortcut Keys

You can perform some Object Repository Merge Tool commands by pressing shortcut keys. The shortcut keys listed below are shown next to the respective menu commands.

You can perform the following **File** menu commands by pressing the corresponding shortcut keys:

Command	Shortcut Key	Function
New Merge	CTRL+N	Enables you to specify two object repositories with which to perform a new merge operation.
Save	CTRL+S	Saves the merged shared object repository.

You can perform the following **Navigate** menu commands by pressing the corresponding shortcut keys:

Command	Shortcut Key	Function
Next Conflict	F4	Finds the next conflicting object in the merged object repository.
Previous Conflict	SHIFT+F4	Finds the previous conflicting object in the merged object repository.
Find	CTRL+F	Opens the Find dialog box.
Find Next	F3	Finds the next object in the merged object repository according to the search specifications in the Find dialog box.
Find Previous	SHIFT+F3	Finds the previous object in the merged object repository according to the search specifications in the Find dialog box.

Defining Default Settings

The Object Repository Merge Tool is supplied with predefined settings that are used when merging object repositories. These are the default settings:

- ▶ Specify the text color of the object names that are displayed in the target object repository.
- ▶ Configure how the Object Repository Merge Tool deals with conflicting objects in the primary and secondary repositories (or local and shared repositories when updating a shared object repository from local object repositories).

You can change these settings at any time to create new default settings. After you change the settings, all new merges are performed according to the new default settings.

Tip: If you want to change the settings before merging two repositories, you must click **Cancel** to close the New Merge dialog box, change the settings as described in the next sections, and then perform the merge.

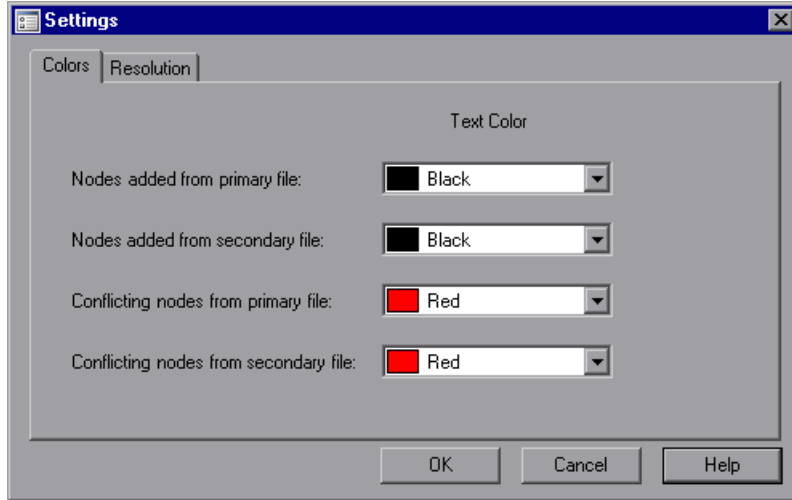
Specifying Color Settings


You can specify the color in which object names are displayed in the target object repository according to their source, and whether they caused a conflict. This enables you to see more easily the status of each object.

Note: The options in the Colors tab of the Settings dialog box apply equally to objects added from the local (primary) and shared (secondary) object repositories, when performing an **Update from Local Repository** operation.

To specify color settings:

- 1 Choose **Tools > Settings** or click the **Settings** button. The Settings dialog box opens.



- 2 For each item in the Colors tab, click the down arrow  next to the text box and select an identifying color.
- 3 Click **OK**. Object names in the target object repository are displayed in the selected color according to your selections.

Specifying Default Resolution Settings

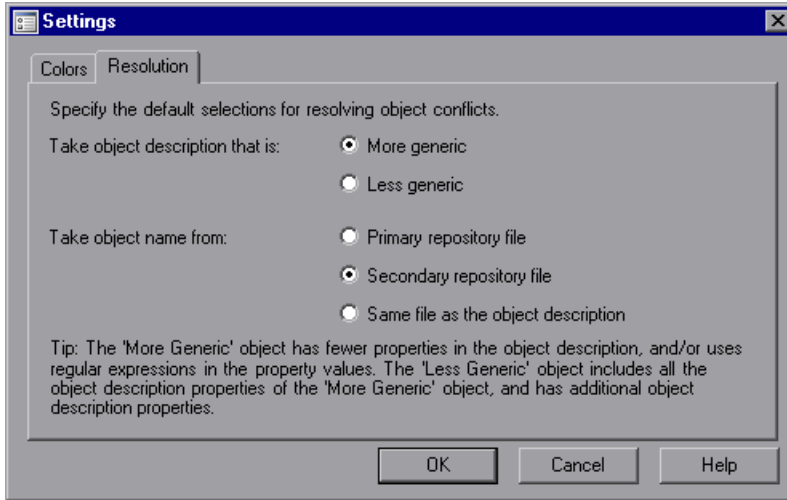
You can configure how the Object Repository Merge Tool automatically deals with conflicting objects during the merge process.

Note: The options in the Resolution tab of the Settings dialog box also apply to objects added from the local and shared object repositories, when performing an **Update from Local Repository** operation.

To specify default resolution settings:



- 1 Choose **Tools > Settings** or click the **Settings** button. The Settings dialog box opens.
- 2 Click the **Resolution** tab.



- 3 Select the appropriate radio buttons to specify the default resolution settings that the Object Repository Merge Tool applies when dealing with conflicting objects.
 - ▶ **Take object description that is**—Specifies how to resolve conflicts in which two test objects have the same name, but their descriptions differ. You can specify that the target object repository takes the object description that is more generic or less generic.
 - **More generic**—Instructs the Object Repository Merge Tool to take the object that has fewer identifying properties than the object with which it conflicts, or uses regular expressions in its property values. This is the default setting.
 - **Less generic**—Instructs the Object Repository Merge Tool to take the object that has all the identifying properties of the object with which it conflicts, plus additional identifying properties.

- ▶ **Take object name from**—Specifies how to resolve conflicts where two test objects have the same or similar descriptions, but their names differ. You can select the source from which the target object repository takes the object name:
 - **Primary repository file**—The target object repository takes the object name from the object in the primary object repository. This is the default setting.
 - **Secondary repository file**—The target object repository takes the object name from the object in the secondary object repository.
 - **Same file as the object description**—The target object repository takes the object name from the object in the same object repository from which it took the object description.
- 4 Click **OK**. The Object Repository Merge Tool will apply your selections when resolving conflicts between objects in all future repository merges.

Note: If you make any change to the resolution settings while you have a merged object repository open, you are asked whether you want to merge the open files again with the new settings. Click **Yes** to merge the files again with the new settings, or click **No** to keep the existing merge created with the previous settings. If you click **No**, the new settings will apply only to future merges.

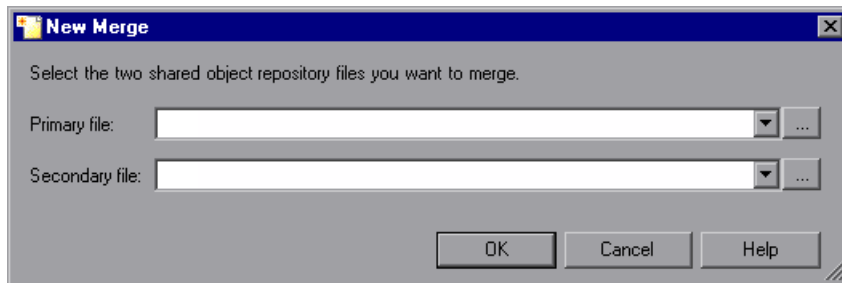
Merging Two Object Repositories

Using the Object Repository Merge Tool, you can merge two source object repositories to create a new shared object repository. Objects in the repositories are automatically compared and added to the new repository according to configurable rules that define how conflicts between objects are resolved. The original source files are not changed.

Note: An object repository that is currently open by another user is locked. If you try to merge the locked file, a warning message displays, but you can still perform the merge because the merge process does not modify the source files. Note that changes made to the locked file by the other user may not be included in the merged object repository.

To merge two object repositories:


- 1 In the Object Repository Manager, choose **Tools > Object Repository Merge Tool**. The New Merge dialog box opens on top of the Object Repository - Merge Tool window.



Tips:

If the Object Repository - Merge Tool window is already open, you can choose **File > New Merge** or click the **New Merge** button to open the New Merge dialog box.

If you want to change the configured settings before merging the repositories, click **Cancel** to close the New Merge dialog box, change the settings as described in “Defining Default Settings” on page 284, and then perform the merge.

- 2** In the **Primary file** and **Secondary file** boxes, enter or browse to and select the **.tsr** object repositories that you want to merge into a single repository. You can click the down arrow  next to each box to view and select recently used files.
-

Notes:

It is recommended that you select as your primary repository the object repository in which you have invested the most effort, meaning the repository with more objects, object properties, and values.



A warning icon is displayed next to the relevant text box if you enter the name of a file without a **.tsr** suffix, a file with an incorrect path, or a file that does not exist. You can position your pointer over the icon to see a tooltip explanation of the error. Enter or select an existing **.tsr** file with the correct path.

If you want to merge an object repository that was created using an earlier version of QuickTest, you must first open and save it in the Object Repository Manager to update it to the new format.

- 3** Click **OK**. The Object Repository Merge Tool automatically merges the selected object repositories into a new target object repository according to the configured resolution settings, and displays the results in the Statistics dialog box on top of the Object Repository - Merge Tool window.

- 4 Review the merge statistics, as described in “Viewing Merge Statistics” on page 296, and click **Close**.

In the Object Repository - Merge Tool window, you can:

- ▶ Modify any conflict resolutions between objects from the source repositories, if necessary, as described in “Resolving Object Conflicts” on page 300.
- ▶ Filter the objects in the target object repository, as described in “Filtering the Target Repository Pane” on page 302.
- ▶ Find specific objects in the target object repository, as described in “Synchronizing Object Repository Views” on page 303.
- ▶ Save the target object repository to the file system or to a Quality Center project, as described in “Saving the Target Object Repository” on page 305.

Updating a Shared Object Repository from Local Object Repositories

You can update a shared object repository by merging local object repositories associated with actions in one or more tests into the shared object repository. The objects that are merged from the local object repositories are then available to any actions that use that shared object repository in any tests.

In the merge process, the objects in the local object repository for the selected action are moved to the target shared object repository. The action then uses the objects from the updated shared object repository.

If you choose to add local object repositories for more than one action, QuickTest performs multiple merges, merging each action's local object repository with the target object repository one at a time, for all the actions in the list. You can view and modify the results of each merge if necessary.

Note: You can only merge local object repositories from actions that are associated with the shared object repository you are updating.

To update a shared object repository from a local object repository:

- 1** Choose **Resources > Object Repository Manager**. The Object Repository Manager opens.

Note: For more information on the Object Repository Manager, see Chapter 8, "Managing Object Repositories."



- 2** In the Object Repository Manager, choose **File > Open** or click the **Open** button. The Open Shared Object Repository dialog box opens.

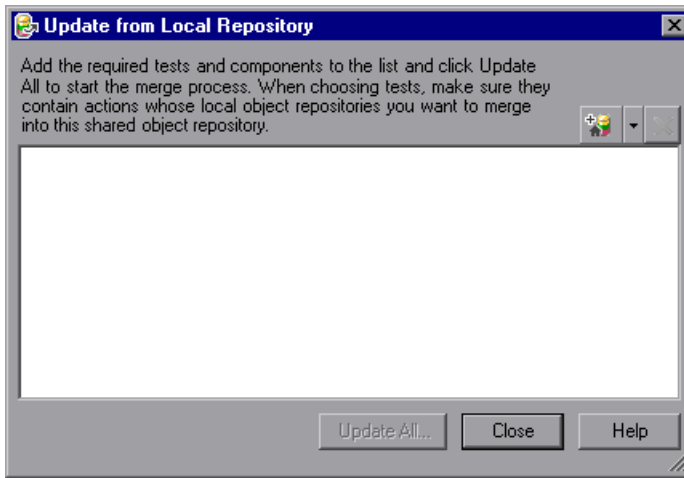
If you are currently connected to a Quality Center project, the Open Shared Object Repository dialog box displays the test plan tree for the project. Select a test to view the shared object repositories attached to the test.


- 3** Browse to the **.tsr** file that contains the shared object repository you want to update, clear the **Open in read-only mode** check box, and click **Open**, or click **OK** in the case of Quality Center attached files. The file opens with the objects and properties displayed in editable format.



Tip: If you opened the object repository in read-only mode, choose **File > Enable Editing** or click the **Enable Editing** button in the Object Repository Manager toolbar. The object repository file is made editable.

- 4 Choose **Tools > Update from Local Repository**. The Update from Local Repository dialog box opens.



- 5 Click the down arrow  next to the **Add Tests** button, and choose **Browse for Test**. The Open Test dialog box opens. If you are currently connected to a Quality Center project, the Open Test from Quality Center Project dialog box opens.

Browse to the test containing actions whose local object repositories you want to merge into the shared object repository.

Note: You can only add a test containing actions that are associated with the shared object repository you are updating and whose local object repositories contain objects.

- 6 Repeat step 5 to add additional tests if required.



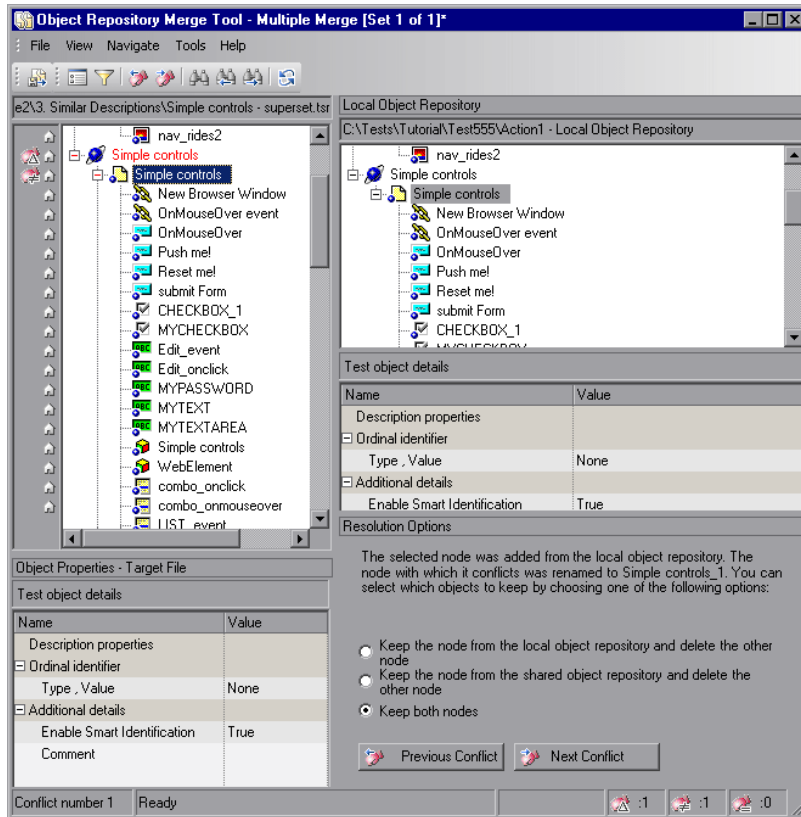
Note: The local object repositories associated with all the actions contained in the listed tests are included in the merge. If you want to remove an action from the merge, select it in the list and click **Delete**.

- 7 Click **Update All**. QuickTest automatically merges the first action local object repository into the shared object repository according to the configured settings, and displays the results in the Statistics dialog box on top of the Object Repository Merge Tool window.


Note: Before each merge, QuickTest checks whether the local object repository is in use by another user. If so, the local object repository is locked and the objects for the selected action cannot be moved to the target shared object repository. A warning message is displayed. The merge can be performed when the local object repository is no longer in use by the other user.


- 8 Review the merge statistics, as described in “Viewing Merge Statistics” on page 296, and click **Close**.

The Object Repository - Merge Tool window for a local object repository merge displays the local object repository as the primary object repository, and the shared object repository as the target object repository.



At the left of each object in the target object hierarchy is an icon that indicates the source of the objects:

 indicates that the node was added from the local object repository

 indicates that the node already existed in the shared object repository

Note: If you specified more than one action in the Update from Local Repository dialog box, QuickTest performs multiple merges, merging each action's local object repository with the target object repository one at a time. The Statistics dialog box and the Object Repository Merge Tool - Multiple Merge window displayed after this step show the merge results of the first merge (the local object repository of the first action being merged into the shared object repository). QuickTest enables you to view, and modify if necessary, the results of each merge in sequence. The number of each merge set in a multiple merge is displayed in the title bar, for example, [Set 2 of 3].

- 9 For each object merged into the shared object repository, you can accept the automatic merge or use the Resolution Options pane to:
 - ▶ Add a specific object to the shared object repository and remove it from the local object repository.
 - ▶ Keep a specific object in the local object repository and not add it to the shared object repository.

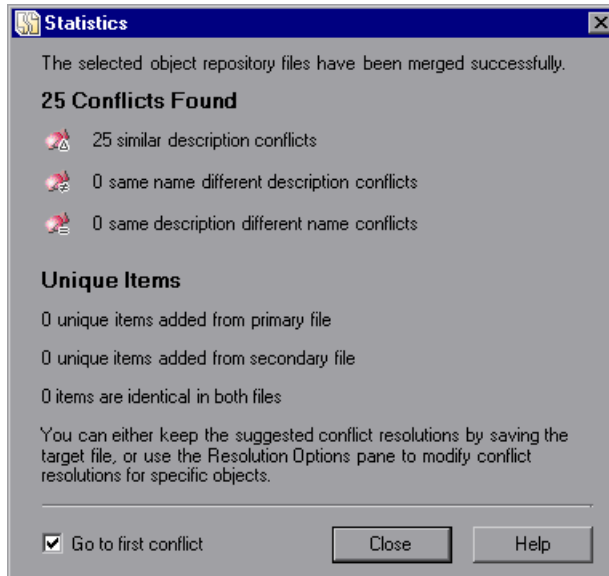
For more information, see “Resolving Object Conflicts” on page 300.



- 10 If you are performing multiple merges, click the **Save and Merge Next** button in the Object Repository Merge Tool toolbar to perform the next merge (the local object repository of the next action being merged into the shared object repository).
- 11 Click **Yes** to save your changes between merges. If you click **No**, the current merge (objects merged from the last action) will not be saved.
- 12 Repeat steps 8 through 11 to complete the multiple merges.
- 13 Choose **File > Exit**, then click **Yes** to save the updated object repository.

Viewing Merge Statistics

After you merge two object repositories, the Object Repository Merge Tool displays the Statistics dialog box, which describes how the files were merged, and the number and type of any conflicts that were resolved during the merge.



Note: The statistics shown after performing an **Update from Local Repository** operation differ slightly from the options shown above.

Tip: You can view the merge statistics in the Statistics dialog box at any time by choosing **View > Statistics** in the Object Repository - Merge Tool window or by clicking a conflict icon in the status bar.

The Statistics dialog box displays the following information:

- ▶ The number and type of any conflicts between the objects added to the target object repository. Conflict types are described in “Resolving Object Conflicts” on page 300.
- ▶ The number of items added to the target object repository that are unique in each of the primary or secondary (or local) files, or are identical in both files.

Tip: Select the **Go to first conflict** check box to jump to the first conflict in the target object repository immediately after you close the Statistics dialog box.

Understanding Object Conflicts

Merging two object repositories can result in conflicts arising from similarities between the test objects they contain. The Object Repository Merge Tool identifies three possible conflict types:



- ▶ **Similar Description Conflict**—Two test objects which have the same name and the same object hierarchy, but which have slightly different descriptions. In this conflict type, one of the objects always has a subset of the properties set of the other object. These conflicts are described on page 298.

By default, the conflict resolution settings for conflicts of this type are configured so that the target object repository takes the object that has fewer identifying properties than the object with which it conflicts. For information on changing the default settings, see “Defining Default Settings” on page 284.



- ▶ **Same Name Different Description Conflict**—Two test objects which have the same name and the same object hierarchy, but differ somehow in their description (for example, they have different properties, or the same property with different values). These conflicts are described on page 299.

By default, the conflict resolution settings for conflicts of this type are configured so that the target object repository takes the object from both files. The object that is added from the secondary file is renamed by adding an incremental numeric suffix to the name, for example, `Edit_1`. For information on changing the default settings, see “Defining Default Settings” on page 284.



- ▶ **Same Description Different Name Conflict**—Two test objects which have identical descriptions, have the same object hierarchy, but differ in their object names. These conflicts are described on page 299.

By default, the conflict resolution settings for conflicts of this type are configured so that the target object repository takes the object name from the primary source file. For information on changing the default settings, see “Defining Default Settings” on page 284.

Note: Objects that do not have a description, such as Page or Browser objects, are compared by name only. If the same object is contained in both the source repositories but with different names, they will be merged into the target object repository as two separate objects.

Similar Description Conflict

An object in the primary object repository and an object in the secondary object repository have the same name, and they have similar, but not identical, description properties and values. One of the objects always has a subset of the properties set of the other object. For example, an object named `Button_1` in the secondary object repository has the same description properties and values as an object named `Button_1` in the primary object repository, but also has additional properties and values.

You can resolve this conflict type by:

- ▶ Taking the object description from the object that is added from the primary repository.
- ▶ Taking the object description from the object that is added from the secondary repository.
- ▶ Taking both objects into the target object repository. In this case, the Object Repository Merge Tool automatically renames the object that is added from the secondary file by adding an incremental numeric suffix to the name, for example, Edit_1.

Same Name Different Description Conflict

An object in the primary object repository and an object in the secondary object repository have the same name, but completely different description properties and values.

You can resolve this conflict type by:

- ▶ Keeping the object added from the primary repository only
- ▶ Keeping the object added from the secondary repository only
- ▶ Keeping the object from both repositories. In this case, the Object Repository Merge Tool automatically renames the object that is added from the secondary file by adding an incremental numeric suffix to the name, for example, Edit_1.

Same Description Different Name Conflict

An object in the primary object repository and an object in the secondary object repository have different names, but the same description properties and values.

You can resolve this conflict type by:

- ▶ Taking the object name from the object in the primary repository
- ▶ Taking the object name from the object in the secondary repository

Resolving Object Conflicts

Conflicts between objects in the primary and secondary object repositories are resolved automatically by the Object Repository Merge Tool according to the default resolution settings that you can configure before performing the merge. For more information, see “Defining Default Settings” on page 284.

However, the Object Repository Merge Tool also allows you to change the way the merge was performed for each individual object that causes a conflict.

For example, an object in the primary repository could have the same name as an object in the secondary repository, but have a different description. You may have defined in the default settings that in this case, the object with the more generic object description, meaning the object with fewer properties, should be added to the target object repository. However, when you review the conflicts after the automatic merge, you could decide to handle the specific conflict differently, for example, by keeping both objects.

Note: Changes that you make to the default conflict resolution can themselves affect the target object repository by causing new conflicts. In the above example, keeping both objects would cause a name conflict. Therefore, the target object repository is updated after each conflict resolution change and redisplayed.

You can identify objects that caused conflicts, and the conflict type, by the icon displayed to the left of the object name in the target object repository pane of the Object Repository Merge Tool and the text color. When you select a conflicting object, a full description of the conflict, including how it was automatically resolved by the Object Repository Merge Tool, is displayed in the Resolutions Options pane.

The Resolutions Options pane offers alternative resolution options. You can choose to keep the default resolution if it suits your needs, or use the alternative options to resolve the conflict in a different way.

Tip: You can also change the default resolution settings and merge the files again. For more information, see “Defining Default Settings” on page 284.

To resolve object conflicts:

- 1** In the target object repository, select an object that has a conflict, as indicated by the icon to the left of the object name. The conflicting objects are highlighted in the source repositories.

A description of the conflict and the resolution method used by the Object Repository Merge Tool is described in the Resolution Options pane. A radio button for each possible alternative resolution method is displayed. For information on each of the conflict types, see “Understanding Object Conflicts” on page 297.

- 2** In the Resolution Options pane, select a radio button to choose an alternative resolution method. The target object repository is updated according to your selection and redisplayed.
- 3** In the Resolution Options pane, click the **Previous Conflict** or **Next Conflict** buttons to jump directly to the next or previous conflict in the target object repository hierarchy.
- 4** Repeat steps 1 through 3 to modify additional conflict resolutions, as necessary.
- 5** Save the target object repository, as described in “Saving the Target Object Repository” on page 305.

Filtering the Target Repository Pane

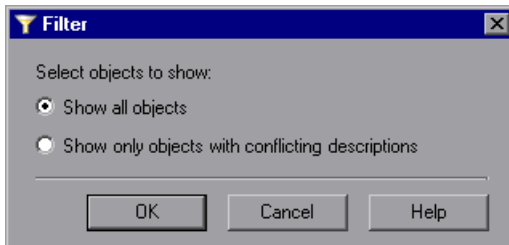
Merging two object repositories can result in a target object repository containing a large number of objects. To make navigation and the location of specific objects easier in the target repository pane, the Object Repository Merge Tool enables you to filter the objects in the pane and show only the objects that have conflicts that were resolved during the merge.

Note: The filter only affects which objects are displayed in the target repository pane. It does not affect which objects are included in the target object repository.

To filter the objects in the target repository pane:



- 1 Choose **Tools > Filter** or click the **Filter** button. The Filter dialog box opens.



Tip: You can also click in the box to the left of the icons in the status bar to view the Filter dialog box. A Filter icon is shown in this area when a filter is currently in use.

- 2 Select a radio button according to the objects you want to view in the target object repository.
 - ▶ **Show all objects**—Shows all objects in the target object repository
 - ▶ **Show only objects with conflicting descriptions**—Shows only objects in the target object repository that had description conflicts
- 3 Click **OK**. The objects in the pane are filtered and the target object repository displays only the requested object types.

Synchronizing Object Repository Views

The Object Repository Merge Tool enables you to navigate the target, primary, and secondary object repositories independently. You can also resize the various panes to display only some of the objects contained in the repositories. When using large object repositories, this can result in the various panes displaying different areas of the repository hierarchies, making it difficult to locate and track specific objects affected by the merge process.



To synchronize the repositories to display the same object in both views, select the object in the primary or secondary object repository in which it is currently visible and click **Synchronize Views**.

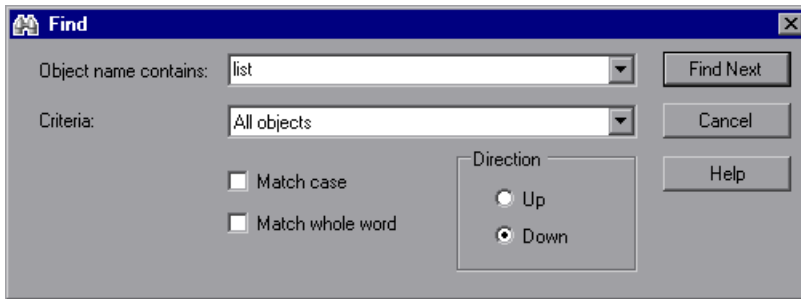
Finding Specific Objects

You can use the Find feature in the Object Repository Merge Tool to locate one or more objects in the target object repository whose name contains a specified string. The located object is also highlighted in the relevant primary and/or secondary repositories.

To find an object:



- 1 Choose **Navigate > Find** or click the **Find** button. The Find dialog box opens.



- 2 In the **Object name contains** box, enter the full or partial name of the object you want to find.
- 3 In the **Criteria** box, refine your search by selecting which objects to search. The following criteria are available:
 - **All objects**
 - **Objects from one source**
 - **Objects with conflicts**
 - **Objects with conflicts or from one source**
- 4 Select one or both of the following options to help fine-tune your search:
 - **Match case**—Distinguishes between upper-case and lower-case characters in the search. When **Match case** is selected, QuickTest finds only those occurrences in which the capitalization exactly matches the text you entered in the **Object name contains** box.
 - **Match whole word**—Searches for occurrences that are whole words only and not part of larger words.

- 5 Specify the direction from the current cursor location in which you want to search: **Up** or **Down**

Tip: Selecting **Up** or **Down** searches to the beginning or the end of the target object repository from the current cursor location. To search the entire repository, select the first (or last) object in the hierarchy and select **Down** (or **Up**).

- 6 Click **Find Next** to highlight the next object that matches the specified criteria in the target object repository.

You can also close the Find dialog box and use the following commands:



- ▶ Click the **Find Next** button or choose **Navigate > Find Next** to highlight the next object that matches the specified criteria.



- ▶ Click the **Find Previous** button or choose **Navigate > Find Previous** to highlight the previous object that matches the specified criteria.

Saving the Target Object Repository

When you are sure that the object conflicts are resolved satisfactorily, you can save the target object repository to the file system or to a Quality Center project (if QuickTest is currently connected to the Quality Center project).

The file you can save depends on the type(s) of object repositories that were merged. If you merged two shared object repositories, you can save the new target object repository file that was created. If you merged one or more local object repositories with a shared object repository, you can save the existing shared object repository file that now contains the objects and data from the local object repositories.

Saving the Object Repository to the File System

You can save the new merged shared object repository to the file system at any time.

To save an object repository to the file system:



- 1 Choose **File > Save** or click the **Save** button. If the file was saved previously, the current changes you made are saved. If the file has not yet been saved, the Save Shared Object Repository dialog box opens.

Note: If you are connected to Quality Center, the Save Shared Object Repository dialog box is different from the standard file selection dialog box. You can switch to save the file to the file system by clicking the **File System** button in that dialog box.

- 2 Navigate to and select the folder in which you want to save the object repository. Enter a name for the object repository in the **File name** box.

Use a descriptive name that will help you easily identify the file. You cannot use the following characters in an object repository name:

\ / : " ? < > | *

- 3 Click **Save**. QuickTest saves the object repository with a **.tsr** extension in the specified location and displays the file name and path above the target object repository in the Object Repository - Merge Tool window.

Saving the Object Repository to a Quality Center Project

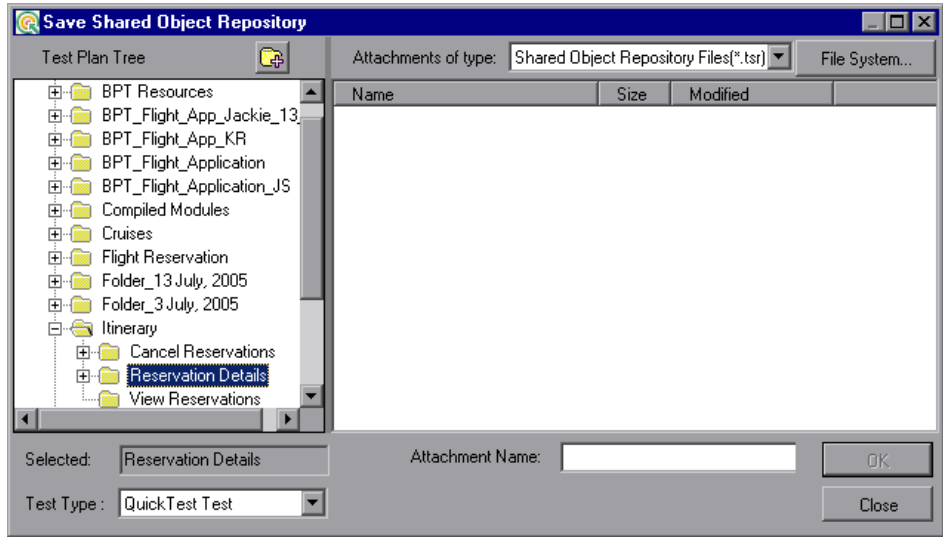
If you are connected to Quality Center, you can save your merged shared object repository as an attachment to a test in the test plan tree of your project.

Note: You cannot overwrite an existing object repository in Quality Center.

To save an object repository in a Quality Center project:



- 1 Choose **File > Save** or click the **Save** button. If the file was saved to Quality Center previously, the current changes you made are saved to the object repository. If the file has not yet been saved, the Save Shared Object Repository dialog box opens.



- 2 In the test plan tree, select the test or folder in which you want to save the object repository.



You can also click the **New Folder** button to create a new test folder in the test plan tree in Quality Center.

Note: You can switch to save the file to the file system by clicking the **File System** button in the Save Shared Object Repository dialog box. You can switch back to the Save Shared Object Repository dialog box for Quality Center by clicking the **Quality Center** button.

- 3 Enter a name for the object repository in the **Attachment Name** box.

Use a descriptive name that will help you easily identify the object repository. You cannot use the following characters in an object repository name:

\ / : " ? < > | *

Note: You cannot overwrite an existing object repository.

- 4 Click **OK**. QuickTest saves the object repository to Quality Center and displays the file name and path above the target object repository in the Object Repository - Merge Tool window. In Quality Center, the file is shown in the Attachments tab of the relevant test or folder.

Part III

Configuring Advanced Settings

10

Configuring Web Event Recording

If QuickTest does not record Web events in a way that matches your needs, you can configure the events you want to record for each type of Web object.

This chapter describes:

- ▶ About Configuring Web Event Recording
- ▶ Selecting a Standard Event Recording Configuration
- ▶ Customizing the Event Recording Configuration
- ▶ Recording Right Mouse Button Clicks
- ▶ Saving and Loading Custom Event Configuration Files
- ▶ Resetting Event Recording Configuration Settings

About Configuring Web Event Recording

QuickTest creates your test by recording the events you perform on your Web-based application. An event is a notification that occurs in response to an operation, such as a change in state, or as a result of the user clicking the mouse or pressing a key while viewing the document. You may find that you need to record more or fewer events than QuickTest automatically records by default. You can modify the default event recording settings by using the Web Event Recording Configuration dialog box to select one of three standard configurations, or you can customize the individual event recording configuration settings to meet your specific needs.

For example, QuickTest does not generally record mouseover events on link objects. If, however, you have mouseover behavior connected to a link, it may be important for you to record the mouseover event. In this case, you could customize the configuration to record mouseover events on link objects whenever they are connected to a behavior.

Notes:

Event configuration is a global setting and therefore affects all tests that are recorded after you change the settings.

Changing the event configuration settings does not affect tests that have already been recorded. If you find that QuickTest recorded more or less than you need, change the event recording configuration and then re-record the part of your test that is affected by the change.

Changes to the custom Web event recording configuration settings do not take effect on open browsers. To apply your changes for an existing test, make the changes you need in the Web Event Recording Configuration dialog box, refresh any open browsers, and then start a new recording session.

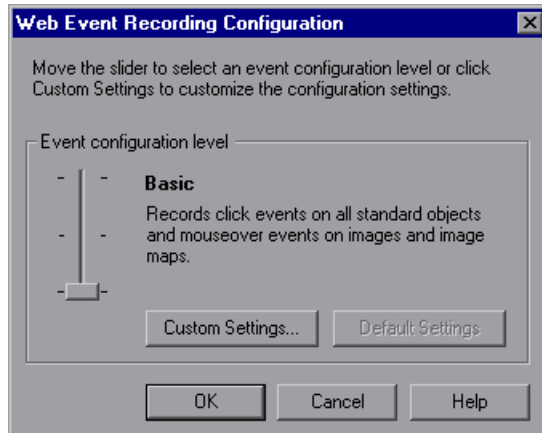
Selecting a Standard Event Recording Configuration

The Web Event Recording Configuration dialog box offers three standard event-configuration levels. By default, QuickTest uses the **Basic** recording-configuration level. If QuickTest does not record all the events you need, you may require a higher event-configuration level.

Level	Description
Basic	<p>Default</p> <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 <i>handler</i> or <i>behavior</i> connected. For more information on handlers and behaviors, see “Listening Criteria” on page 321. • Records the mouseover event on images and image maps only if the event following the mouseover is performed on the same object.
Medium	Records click events on the <DIV>, , and <TD> HTML tag objects, in addition to the objects recorded in the basic level.
High	<p>Records mouseover, mousedown, and double-click events on objects with <i>handlers</i> or <i>behaviors</i> attached, in addition to the objects recorded in the basic level.</p> <p>For more information on handlers and behaviors, see “Listening Criteria” on page 321.</p>

To set a standard event-recording configuration:

- 1** Choose **Tools > Web Event Recording Configuration**. The Web Event Recording Configuration dialog box opens.



- 2** Use the slider to select your preferred standard event recording configuration.

Tip: You can click the **Custom Settings** button to open the Custom Web Event Recording dialog box where you can customize the event recording configuration. For more information, see “Customizing the Event Recording Configuration,” below.

You can click the **Default Settings** button to return the scale to the **Basic** level.

- 3** Click **OK**.

Customizing the Event Recording Configuration

If the standard event configuration levels do not exactly match your recording needs, you can customize the event recording configuration using the Custom Web Event Recording Configuration dialog box.

The Custom Web Event Recording Configuration dialog box enables you to customize event recording in several ways. You can:

- ▶ Add or delete objects to which QuickTest should apply special listening or recording settings.
- ▶ Add or delete events for which QuickTest should listen.
- ▶ Modify the listening or recording settings for an event.

To customize the event recording configuration:

- 1** Choose **Tools > Web Event Recording Configuration**. The Web Event Recording Configuration dialog box opens.
- 2** Click the **Custom Settings** button. The Custom Web Event Recording Configuration dialog box opens.

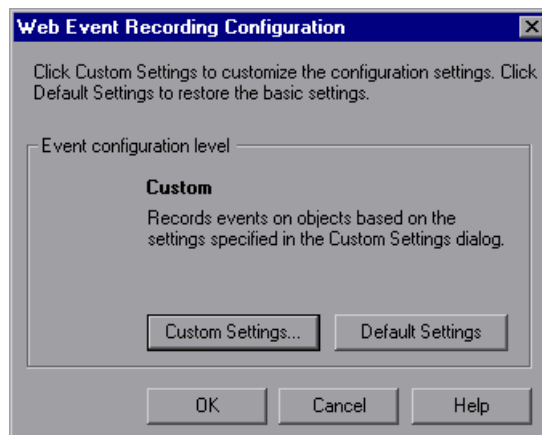


3 Customize the event recording configuration using the following options:

Option	Description
Objects pane	<p>Displays a list of Web test object classes and HTML tag objects.</p> <ul style="list-style-type: none"> • To add an object, choose Object > Add. • Only HTML Tag objects can be deleted. To delete an HTML object from the list, choose Object > Delete. <p>For more information, see “Adding and Deleting Objects in the Custom Configuration Object List” on page 317.</p>
Events pane	<p>Displays a list of events associated with the object.</p> <ul style="list-style-type: none"> • To add an event to the Events pane, choose Event > Add. • To delete an event, choose Event > Delete. <p>For more information, see “Adding and Deleting Listening Events for an Object” on page 319.</p>
Event Name	The name of the event.
Listen	<p>The criteria for when QuickTest listens to the event.</p> <ul style="list-style-type: none"> • Always—Always listens to the event. • If Handler—Listens to the event if a handler is attached to it. A handler is code in a Web page, typically a function or routine written in a scripting language, that receives control when the corresponding event occurs. • If Behavior—Listens to the event if a DHTML behavior is attached to it. A DHTML behavior encapsulates specific functionality or behavior on a page. When applied to a standard HTML element on a page, a behavior enhances that element's default behavior. • If Handler or Behavior—Listens to the event if a handler or behavior is attached to it. • Never—Never listens to the event. <p>For more information, see “Modifying the Listening and Recording Settings for an Event” on page 321.</p>

Option	Description
Record	Enables or disables recording of the event for the selected object, or enables recording of the event only if the subsequent event occurs on the same object.
Reset	Enables you to reset your settings to a preconfigured level.

- 4 Click **OK**. The Custom Web Event Recording Configuration dialog box closes. The slider scale on the Web Event Recording Configuration dialog box is hidden and the configuration description displays **Custom**.



- 5 Click **OK** to close the Web Event Recording Configuration dialog box.

Adding and Deleting Objects in the Custom Configuration Object List

The Custom Web Event Recording Configuration dialog box lists objects in an object hierarchy. The top of the hierarchy is **Any Web Object**. The settings for **Any Web Object** apply to any object on the Web page being tested, for which there is no specific event recording configuration set. Below this are the **Web Objects** and **HTML Tag Objects** categories, each of which contains a list of objects.

When working with the objects in the Custom Web Event Recording Configuration dialog box, keep the following principles in mind:

- ▶ If an object is listed in the Custom Web Event Recording Configuration dialog box, then the settings for that object override the settings for **Any Web Object**.
- ▶ You cannot delete or add to the list of objects in the **Web Objects** category, but you can modify the settings for any of these objects.
- ▶ You can add any HTML Tag object in your Web page to the **HTML Tag Objects** category.

To add objects to the event configuration object list:

- 1** In the Custom Web Event Recording Configuration dialog box, choose **Object > Add**. A **New Object** object is displayed in the HTML Tag Objects list.



- 2** Click **New Object** to rename it. Enter the exact HTML Tag name.

By default the new object is set to listen and record **onclick** events with handlers attached.

For more information on adding or deleting events, see “Adding and Deleting Listening Events for an Object,” below. For more information on listening and recording settings, see “Modifying the Listening and Recording Settings for an Event” on page 321.

To delete objects from the HTML Tag Objects list:

- 1** From the Custom Web Event Recording Configuration dialog box, select the object in the HTML Tag Objects category that you want to delete.
- 2** Choose **Object > Delete**. The object is deleted from the list.

Note: You cannot delete objects from the **Web Objects** category.

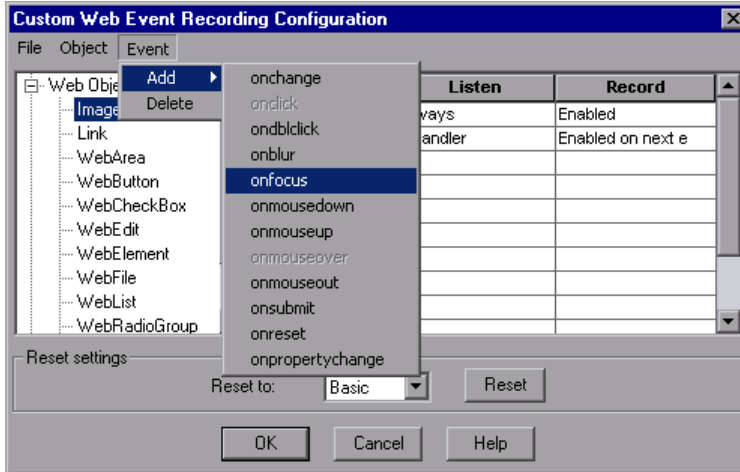
Adding and Deleting Listening Events for an Object

You can add or delete events from the list of events that trigger QuickTest to listen to an object.

To add listening events for an object:

- 1** In the Custom Web Event Recording Configuration dialog box, select the object to which you want to add an event, or select **Any Web Object**.

- 2 Choose **Event > Add**. A list of available events opens.



- 3 Select the event you want to add. The event is displayed in the Event Name column in alphabetical order. By default, QuickTest listens to the event when a handler is attached and always records the event (as long as it is listened to at some level).

For more information on listening and recording settings, see “Modifying the Listening and Recording Settings for an Event,” below.

To delete listening events for an object:

- 1 In the Custom Web Event Recording Configuration dialog box, select the object from which you want delete an event, or select **Any Web Object**.
- 2 Select the event you want to delete from the **Event Name** column.
- 3 Choose **Event > Delete**. The event is deleted from the **Event Name** column.

Modifying the Listening and Recording Settings for an Event

You can select the listening criteria and set the recording status for each event listed for each object.

Note: The listen and record settings are mutually independent. This means that you can choose to listen to an event for particular object, but not record it, or you can choose not to listen to an event for an object, but still record the event. For more information, see “Tips for Working with Event Listening and Recording” on page 323.

Listening Criteria

For each event, you can instruct QuickTest to listen every time the event occurs on the object if an event handler is attached to the event, if a DHTML behavior is attached to the event, if an event handler or DHTML behavior are attached to the event, or to never listen to the event.

An event handler is code in a Web page, typically a function or routine written in a scripting language, that receives control when the corresponding event occurs.

A DHTML behavior encapsulates specific functionality or behavior on a page. When applied to a standard HTML element on a page, a behavior enhances that element's default behavior.

To specify the listening criterion for an event:

- 1 From the Custom Web Event Recording Configuration dialog box, select the object for which you want to modify the listening criterion or select **Any Web Object**.

- 2 In the row of the event you want to modify, select the listening criterion you want from the **Listen** column.

Event Name	Listen	Record
onclick	If Handler	Enabled
onkeydown	Always	Enabled
onmouseover	If Handler	Disabled
4	Always	
5	If Handler	
6	If Behavior	
7	If Handler or Behavior	
8	Never	
9		
10		

You can select **Always**, **If Handler**, **If Behavior**, **If Handler or Behavior**, or **Never**.

Recording Status

For each event, you can enable recording, disable recording, or enable recording only if the next event is dependent on the selected event.

- **Enabled**—Records the event each time it occurs on an object as long as QuickTest listens to the event on the selected object, or on another object to which the event bubbles.

Bubbling is the process whereby, when an event occurs on a child object, the event can travel up the chain of hierarchy within the HTML code until it encounters an event handler to process the event.

- **Disabled**—Does not record the specified event and ignores event bubbling where applicable.
- **Enabled on next event**—Same as **Enabled**, except that it records the event only if a subsequent event occurs on the same object. For example, suppose a mouseover behavior modifies an image link. You may not want to record the mouseover event each time you happen to move the mouse over this image. It is essential, though, that the mouseover event be recorded before a click event on the same object because only the image that is displayed after the mouseover event enables the link event. This option applies only to the Image and WebArea objects.

To set the recording status for an event:

- 1 From the Custom Web Event Recording Configuration dialog box, select the object for which you want to modify the recording status or select **Any Web Object**.
- 2 In the row of the event you want to modify, select a recording status from the Record column.

Event Name	Listen	Record
onclick	Always	Enabled
onmouseover	If Handler	Enabled on next ev
3		Disabled
4		Enabled
5		Enabled on next ever
6		
7		
8		
9		

Tips for Working with Event Listening and Recording

It can sometimes be difficult to find the ideal listen and recording settings. When defining these settings, keep in mind the following guidelines:

- If settings for different objects in the Objects Pane conflict, QuickTest gives first priority to settings for specific **HTML Tag Objects** and second priority to **Web Objects** settings. QuickTest only applies the settings for **Any Web Object** to Web objects that were not defined in the **HTML Tag Object** or **Web Objects** areas.
- To record an event on an object, you must instruct QuickTest to listen for the event, and to record the event when it occurs. You can listen for an event on a child object, even if a parent object contains the handler or behavior, or you can listen for an event on a parent object, even if the child object contains the handler or behavior.

However, you must enable recording for the event on the source object (the object on which the event actually occurs, regardless of which parent object contains the handler or behavior).

For example, suppose a table cell with an onmouseover event handler contains two images. When the mouse moves over either of the images, the event also bubbles up to the cell, and the bubbling includes information on the image that the mouse moved over. You can record this mouseover event by:

- ▶ Setting **Listen** on the <TD> tag mouseover event to **If Handler** (so that QuickTest “hears” the event when it occurs), while disabling recording on it, and then setting **Listen** on the tag mouseover event to **Never**, while setting **Record** on the tag to **Enable** (to record the mouseover event on the image after it is listened to at the <TD> level).
- ▶ Setting **Listen** on the tag mouseover event to **Always** (to listen for the mouseover event even though the image tag does not contain a behavior or handler), and setting **Record** on the tag to **Enabled** (to record the mouseover event on the image).
- ▶ Instructing QuickTest to listen for many events on many objects may lower performance, so it is recommended to limit **Listen** settings to the required objects.
- ▶ In rare situations, listening to the object on which the event occurs (the source object) can interfere with the event.

If you find that your application works properly until you begin recording on the application using QuickTest, your **Listen** settings may be interfering.

If this problem occurs with a mouse event, try selecting the appropriate **Use standard Windows mouse events** option(s) in the Advanced Web Options dialog box. For more information, refer to “Advanced Web Options” on page 733 in the *QuickTest Professional Basic Features User’s Guide*.

If this problem occurs with a keyboard or internal event, or the **Use standard Windows mouse events** option does not solve your problem, set the **Listen** settings for the event to **Never** on the source object (but keep the record setting enabled on the source object), and set the **Listen** settings to **Always** for a parent object.

Recording Right Mouse Button Clicks

QuickTest enables you to record clicks made using left, center, and right mouse buttons. By default, only left clicks are recorded, but you can modify the configuration to record clicks from the right and center buttons, as well.

QuickTest records the **Click** statement when the **OnClick** event is triggered. QuickTest differentiates between the mouse buttons by listening for events configured for each of the mouse buttons. By default, it listens for the **OnMouseUp** event, but you can also configure it to listen for the **OnMouseDown** event using the Web Event Recording Configuration dialog box.

Notes:

Recording of simultaneous clicking of more than one mouse button is not supported.

QuickTest does not record the right click that opens the browser context menu, or the selection of an item from the context menu. For more information on modifying the script manually to enable these options, refer to the following Knowledge Base articles:

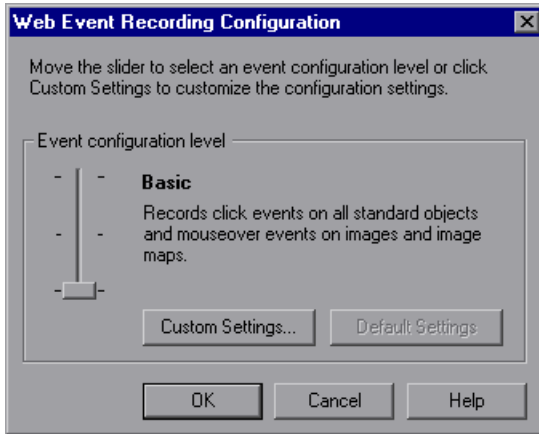
- ▶ **Problem ID 31270:** How to replay right-clicking on an object to open a pop-up menu
 - ▶ **Problem ID 27184:** How to select an item from a right-click menu
-

Configuring QuickTest to Record Right Mouse Clicks

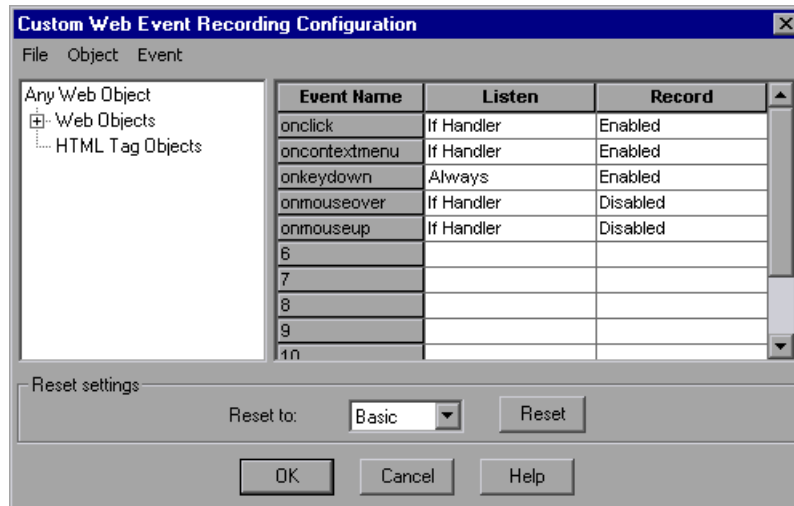
You instruct QuickTest to record right mouse clicks by modifying the configuration file manually and then loading it.

To configure QuickTest to record right mouse clicks:

- 1 Choose **Tools > Web Event Recording Configuration**. The Web Event Recording Configuration dialog box opens.



- Click the **Custom Settings** button. The Custom Web Event Recording Configuration dialog box opens.



- In the Custom Web Event Recording Configuration dialog box, choose **File > Save Configuration As**. The Save As dialog box opens.
- Navigate to the folder in which you want to save the web event recording configuration file, and enter a configuration file name. The extension for configuration files is **.xml**.
- Click **Save** to save the file and close the dialog box.
- Open the saved configuration file for editing in any text editor. The configuration file uses a defined structure. For more information on the XML file structure, see “Understanding the Web Event Recording Configuration XML Structure” on page 330.

The beginning of the file, which is relevant for Web objects, is shown below.

```
- <XML>
- <Object Name="Any Web Object">
  <Event Name="onclick" Listen="2" Record="2" />
  <Event Name="oncontextmenu" Listen="2" Record="2" />
  <Event Name="onkeydown" Listen="1" Record="2" />
  <Event Name="onmouseover" Listen="2" Record="1" />
- <Event Name="onmouseup" Listen="2" Record="1">
  <Property Name="button" Value="2" Listen="2" Record="2" />
```

The **Property Name** argument controls the recording of the mouse buttons. The value of the mouse buttons are defined as follows:

- 1—Left
- 2—Right
- 4—Middle

7 Edit the file as follows:

- To record a left mouse click for the **onmouseup** event, add the following line:

```
<Property Name="button" Value="1" Listen="2" Record="2"/>
```

- To record right and left mouse clicks for the **onmousedown** event, add the following lines:

```
<Event Name="onmousedown" Listen="2" Record="1">
```

```
  <Property Name="button" Value="2" Listen="2" Record="2"/>
```

```
  <Property Name="button" Value="1" Listen="2" Record="2"/>
```

```
</Event>
```

Note: Only one event, either **onmouseup** or **onmousedown**, should be used to handle mouse clicks. If both events are used, QuickTest will record two clicks instead of one. By default, QuickTest listens for the **onmouseup** event.

8 Save the file.

9 In the Custom Web Event Recording Configuration dialog box, choose **File > Load Configuration**. The Open dialog box opens.

10 Navigate to the folder in which you saved the edited configuration file, select the file, and click **Open**. The Custom Web Recording Configuration dialog box reopens.

11 Click **OK**. The new configuration is loaded, with all preferences corresponding to those you defined in the XML configuration file. Any Web objects you now record will be recorded according to these new settings.

Saving and Loading Custom Event Configuration Files

You can save the changes you make in the Custom Web Event Recording Configuration dialog box, and load them at any time.

You can also modify the XML file before loading it. For more information on the XML file structure, see “Understanding the Web Event Recording Configuration XML Structure” on page 330.

To save a custom configuration:

- 1** Customize the event recording configuration as desired. For more information on how to customize the configuration, see “Customizing the Event Recording Configuration” on page 315.
- 2** In the Custom Web Event Recording Configuration dialog box, Choose **File > Save Configuration As**. The Save As dialog box opens.
- 3** Navigate to the folder in which you want to save your event configuration file and enter a configuration file name. The extension for configuration files is **.xml**.
- 4** Click **Save** to save the file and close the dialog box.

To load a custom configuration:

- 1** Choose **Tools > Web Event Recording Configuration** and then click **Custom Settings** to open the Custom Web Event Recording Configuration dialog box.
- 2** Choose **File > Load Configuration**. The Open dialog box opens.
- 3** Locate the event configuration file (**.xml**) that you want to load and click **Open**. The dialog box closes and the selected configuration is loaded.

Understanding the Web Event Recording Configuration XML Structure

The Web event recording configuration XML file is structured in a certain format. If you are modifying the file, or creating your own file, you must ensure that you adhere to this format, in order for your settings to take effect.

Following is a sample XML file:

```
<XML>
  <Object Name="Any Web Object">
    <Event Name="onclick" Listen="2" Record="2"/>
    <Event Name="onmouseup" Listen="2" Record="1">
      <Property Name="button" Value="2" Listen="2" Record="2"/>
    </Event>
  </Object>
  ...
  ...
  ...
  <Object Name="WebList">
    <Event Name="onblur" Listen="1" Record="2"/>
    <Event Name="onchange" Listen="1" Record="2"/>
    <Event Name="onfocus" Listen="1" Record="2"/>
  </Object>
</XML>
```


You define the listening criteria and recording status options in the XML using the following possible values:

Settings	Possible Values
Listen	1—Always 2—If Handler 4—If Behavior 6—If Handler or Behavior 0—Never
Record	1—Disabled 2—Enabled 6—Enabled on Next Event

Resetting Event Recording Configuration Settings

You can restore standard settings after you set custom settings by resetting the event recording configuration settings to the basic level from the Web Event Recording Configuration dialog box. You can also restore the default custom level settings from the Custom Web Event Recording Configuration dialog box.

Note: When you choose to reset standard settings, your custom settings are cleared completely. If you do not want to lose your changes, be sure to save your settings in an event configuration file. For more information, see “Saving and Loading Custom Event Configuration Files” on page 329.

To reset basic level configuration settings from the Web Event Recording Configuration dialog box:

- 1** Choose **Tools > Web Event Recording Configuration**. The Web Event Recording Configuration dialog box opens.
- 2** Click **Default**. The standard configuration slider is displayed again and all event settings are restored to the **Basic** event recording configuration level.
- 3** If you want to select a different standard configuration level, see “Selecting a Standard Event Recording Configuration” on page 313.

You can also restore the settings to a specific (base) custom configuration from within the Custom Web Event Recording Configuration dialog box so that you can begin customizing from that point.

To reset the settings to a custom level from the Custom Web Event Recording Configuration dialog box:

- 1** Choose **Tools > Web Event Recording Configuration**. The Web Event Recording Configuration dialog box opens.
- 2** Click the **Custom Settings** button. The Custom Web Event Recording Configuration dialog box opens.
- 3** In the **Reset to** box, select the standard event recording level you want.
- 4** Click **Reset**. All event settings are restored to the defaults for the level you selected.

11

Customizing the Expert View and Function Library Windows

You can customize the way your test is displayed when you work in the Expert View and the way functions are displayed in the function library windows. Any changes you make are applied globally to the Expert View and to all function library windows.

This chapter describes:

- ▶ About Customizing the Expert View and Function Library Windows
- ▶ Customizing Editor Behavior
- ▶ Customizing Element Appearance
- ▶ Personalizing Editing Commands

About Customizing the Expert View and Function Library Windows

QuickTest includes a powerful and customizable editor that enables you to modify many aspects of the Expert View and function library windows.

The Editor Options dialog box enables you to change the way scripts and function libraries are displayed in the Expert View and function library windows. You can also change the font style and size of text in your scripts and function libraries, and change the color of different elements, including comments, strings, QuickTest reserved words, operators, and numbers. For example, you can display all text strings in red.

QuickTest includes a list of default keyboard shortcuts that enable you to move the cursor, delete characters, and cut, copy, and paste information to and from the Clipboard. You can replace these shortcuts with shortcuts you prefer. For example, you could change the **Line start** command from the default HOME to ALT + HOME.

You can also modify the way your script or function library is printed using options in the Print dialog box. For more information, refer to “Printing a Test” on page 103 in the *QuickTest Professional Basic Features User’s Guide* and see “Printing a Function Library” on page 198.

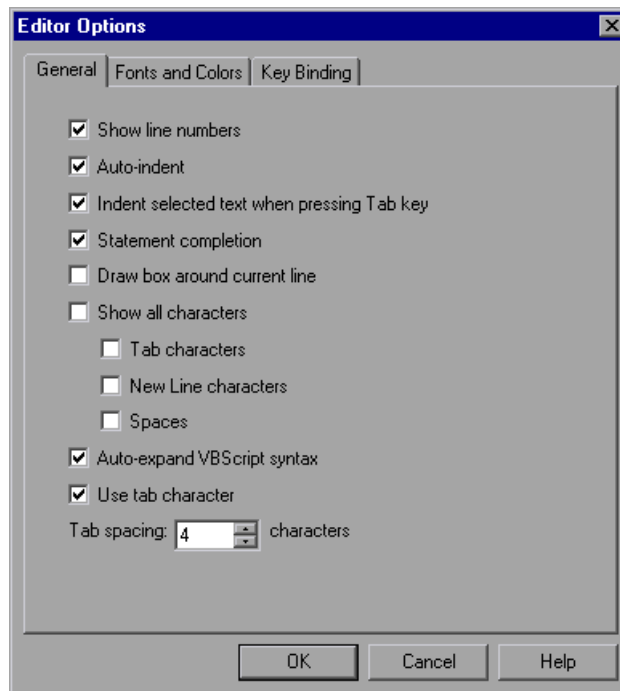
For more information on using the Expert View, see Chapter 5, “Working with the Expert View and Function Library Windows.” For more information on working with function libraries, see Chapter 6, “Working with User-Defined Functions and Function Libraries.”

Customizing Editor Behavior

You can customize how scripts and function libraries are displayed in the Expert View and function library windows. For example, you can show or hide character symbols, and choose to display line numbers. For more information on using the Expert View, see Chapter 5, “Working with the Expert View and Function Library Windows.” For more information on working with function libraries, see Chapter 6, “Working with User-Defined Functions and Function Libraries.”

To customize editor behavior:

- 1 When the Expert View or a function library window is active, choose **Tools > View Options**. The Editor Options dialog box opens.
- 2 Click the **General** tab.



3 Choose from the following options:

Options	Description
Show line numbers	Displays a line number to the left of each line in the script or function.
Auto-indent	Causes lines following an indented line to automatically begin at the same point as the previous line. You can click the HOME key on your keyboard to move the cursor back to the left margin.
Indent selected text when pressing Tab key	Pressing the TAB key indents the selected text. When this option is not enabled, pressing the Tab key replaces the selected text with a single Tab character.
Statement completion	<p>When this option is selected, if you type:</p> <ul style="list-style-type: none"> • a dot after a test object—QuickTest displays a list of available test objects and methods that you can add after the object you typed. • an open parenthesis after an object—QuickTest displays a list of all test objects of this type in the object repository. If there is only one object of this type in the object repository, QuickTest automatically enters its name in quotes after the open parenthesis. • a method—QuickTest displays the syntax for the method, including its specific mandatory and optional arguments. • the Object property—if the object data is currently available in the Active screen or the open application, QuickTest displays native methods and properties of any run-time object in your application. <p>For more information on using the statement completion (IntelliSense) feature, see “Generating Statements in the Expert View or a Function Library” on page 130.</p>
Draw box around current line	Displays a box around the line of the test in which the cursor is currently located.

Options	Description
Show all characters	Displays all Tab, New Line and Space character symbols. You can also select to display only some of these characters by selecting or clearing the relevant check boxes.
Auto-expand VBScript syntax	<p>Automatically recognizes the first two characters of keywords and adds the relevant VBScript syntax or blocks to the script, when you type the relevant keyword.</p> <p>For example, if you enter the letters if and then enter a space at the beginning of a line in the Expert View, QuickTest automatically enters:</p> <pre>If Then End If</pre>
Use tab character	Inserts a TAB character when the TAB key on the keyboard is used. When this option is not enabled, the specified number of space characters is inserted when you press the TAB key.

- 4 Click **OK** to save the changes and close the dialog box.

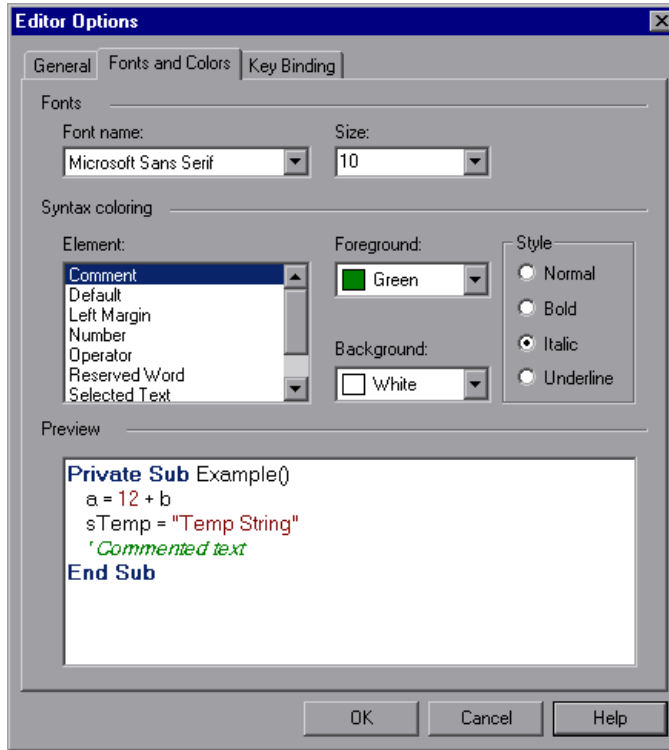
Customizing Element Appearance

QuickTest scripts and function libraries contain many different elements, such as comments, strings, QuickTest and VBScript reserved words, operators, and numbers. Each element of a QuickTest script can be displayed in a different color. You can also specify the font style and size to use for all elements in the Expert View. You can create your own personalized color scheme for each script element. For example, all comments in your scripts could be displayed as blue letters on a yellow background.

To set font and color preferences for elements:

- 1 When the Expert View or a function library window is active, choose **Tools > View Options**. The Editor Options dialog box opens.

2 Click the **Fonts and Colors** tab.



3 In the **Fonts** area, select the **Font name** and **Size** that you want to use to display all elements. By default, the editor uses the Microsoft Sans Serif font, which is a Unicode font.

Note: When testing in a Unicode environment, you must select a Unicode-compatible font. Otherwise, elements in your test or function library may not be correctly displayed in the Expert View or function library windows. However, the test or function library will still run in the same way, regardless of the font you choose. If you are working in an environment that is not Unicode-compatible, you may prefer to choose a fixed-width font, such as Courier, to ensure better character alignment.

- 4 Select an element from the **Element** list.
- 5 Choose a foreground color and a background color.
- 6 Choose a font style for the element (**Normal**, **Bold**, **Italic**, or **Underline**).
An example of your change is displayed in the **Preview** pane at the bottom of the dialog box.
- 7 Repeat steps 4 to 6 for each element you want to modify.
- 8 Click **OK** to apply the changes and close the dialog box.

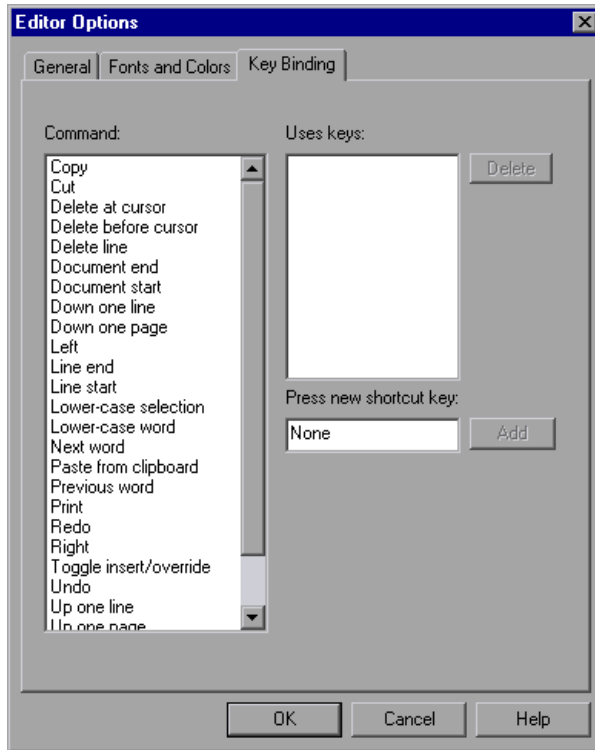
Personalizing Editing Commands

You can personalize the default keyboard shortcuts you use for editing. QuickTest includes keyboard shortcuts that let you move the cursor, delete characters, and cut, copy, or paste information to and from the Clipboard. You can replace these shortcuts with your preferred shortcuts. For example, you could change the **Line end** command from the default END to ALT + END.

Note: The default QuickTest menu shortcut keys override any key bindings that you may define. For example, if you define the Paste command key binding to be CTRL+P, it will be overridden by the default QuickTest shortcut key for opening the Print dialog box (corresponding to the **File > Print** option). For a complete list of QuickTest menu shortcut keys, refer to “Executing Commands Using Shortcut Keys” on page 43 in the *QuickTest Professional Basic Features User’s Guide*.

To personalize editing commands:

- 1 When the Expert View or a function library window is active, choose **Tools > View Options**. The Editor Options dialog box opens.
- 2 Click the **Key Binding** tab.



- 3 Select a command from the **Command** list.
- 4 Click in the **Press new shortcut key** box and then press the key(s) you want to use for the selected command. For example, press and hold the CTRL key while you press the number 4 key to enter CTRL+4.

5 Click **Add**.

Note: If the key combination you specify is not supported, or is already defined for another command, a message to this effect is displayed below the shortcut key box.

- 6** Repeat steps 3 - 5 for any additional commands.
- 7** If you want to delete a key sequence from the list, select the command in the **Command** list, then highlight the key(s) in the **Uses keys** list, and click **Delete**.
- 8** Click **OK** to apply the changes and close the dialog box.

12

Setting Testing Options During the Run Session

You can control how QuickTest records and test runs by setting and retrieving testing options during a run session.

This chapter describes:

- ▶ About Setting Testing Options During the Run Session
- ▶ Setting Testing Options
- ▶ Retrieving Testing Options
- ▶ Controlling the Test Run
- ▶ Adding and Removing Run-Time Settings

About Setting Testing Options During the Run Session

QuickTest testing options affect how you record and run tests. For example, you can set the maximum time that QuickTest allows for finding an object in a page.

You can set and retrieve the values of testing options during a run session using the **Setting** object in the Expert View. For more information on working in the Expert View, see Chapter 5, “Working with the Expert View and Function Library Windows.”

By retrieving and setting testing options using the **Setting** object, you can control how QuickTest runs a test.

You can also set many testing options using the Options dialog box (global testing options) and the Test Settings dialog box (test-specific settings). For more information, refer to Chapter 24, “Setting Global Testing Options” and Chapter 25, “Setting Options for Individual Tests” in the *QuickTest Professional Basic Features User’s Guide*.

This chapter describes some of the QuickTest testing options that can be used with the **Setting** object from within a test script. For detailed information on all the available methods and properties for the **Setting** object, refer to the **Utility** section of the *QuickTest Professional Object Model Reference*.

Note: You can also control QuickTest options as well as most other QuickTest operations from an external application using automation programs. For more information, see “Automating QuickTest Operations” on page 231, or refer to the *QuickTest Automation Object Model Reference* (**Help > QuickTest Automation Object Model Reference**).

Setting Testing Options

You can use the **Setting** object to set the value of a testing option from within the test script. To set the option, use the following syntax:

Setting (*testing_option*) = *new_value*

Some options are global and others are per-test settings.

Using the **Setting** object with a global testing option changes a testing option globally, and this change is reflected in the Options dialog box.

For example, if you execute the following statement:

```
Setting("AutomaticLinkRun")=1
```

QuickTest disables automatically created checkpoints in the test. The setting remains in effect during your current QuickTest session until it is changed again, either with another **Setting** statement, or by clearing the **Ignore automatic checkpoints while running tests** check box in the Advanced Web Options dialog box (Choose **Tools** > **Options** > **Web** tab, and click **Advanced**).

Using the **Setting** object to set per-test options is also reflected in the Test Settings dialog box. You can also use the **Setting** object to change a setting for a specific part of a specific test. For more information see “Controlling the Test Run” on page 346.

For example, if you execute the following statement:

```
Setting("WebTimeOut")=50000
```

QuickTest automatically changes the amount of time it waits for a Web page to load before running a test step to 50000 milliseconds. The setting remains in effect during your current QuickTest session until it is changed again, either with another **Setting** statement, or by setting the **Browser Navigation Timeout** option in the Web tab of the Test Settings dialog box.

Note: Although the changes you make using the **Setting** object are reflected in the Options and Test Settings dialog boxes, these changes are not saved when you close QuickTest, unless you make other changes in the same dialog box manually and click **Apply** or **OK** (which saves all current settings in that dialog box).

Retrieving Testing Options

You can also use the **Setting** object to retrieve the current value of a testing option.

To store the value in a variable, use the syntax:

```
new_var = Setting ( testing_option )
```

To display the value in a message box, use the syntax:

```
MsgBox (Setting (testing_option) )
```

For example:

```
LinkCheckSet = Setting("AutomaticLinkRun")
```

assigns the current value of the AutomaticLinkRun setting to the user-defined variable LinkCheckSet.

Other examples of testing options that you can use to retrieve a setting are shown in “Setting Testing Options” on page 344.

Controlling the Test Run

You can use the retrieve and set capabilities of the **Setting** object together to control a run session without changing global settings. For example, if you want to change the **DefaultTimeOut** testing option to 5 seconds for objects on one Web page only, insert the following statement after the Web page opens in your test script:

```
'Keep the original value of the DefaultTimeOut testing option  
old_delay = Setting ("DefaultTimeOut")
```

```
'Set temporary value for the DefaultTimeOut testing option  
Setting("DefaultTimeOut")= 5000
```


To return the **DefaultTimeOut** testing option to its original value at the end of the Web page, insert the following statement immediately before linking to the next page in the script:

```
'Change the DefaultTimeOut testing option back to its original value.  
Setting("DefaultTimeOut")=old_delay
```

Adding and Removing Run-Time Settings

In addition to the global and specific settings, you can also add, modify, and remove custom run-time settings. These settings are applicable during the run session only.

To add a new run-time setting, use the syntax:

```
Setting.Add "testing_option", "value"
```

For example, you could create a setting that indicates the name of the current tester and displays the name in a message box.

```
Setting.Add "Tester Name", "Mark Train"  
MsgBox Setting("Tester Name")
```

Note: When using a **Setting.Add** statement, an error occurs if you try to add an existing key value. To avoid this error you should use a **Setting.Exists** statement first. For more details about all the **Setting** methods, refer to the *QuickTest Professional Object Model Reference*.

To modify a run-time setting that has already been initialized, use the same syntax you use for setting any standard setting option:

Setting (*testing_option*) = *new_value*

For example:

```
Setting("Tester Name")="Alice Wonderlin"
```

To delete a custom run-time setting, use the following syntax:

Setting.Remove (*testing_option*)

For example:

```
Setting.Remove ("Tester Name")
```

Part IV

Working with Other Mercury Products

13

Working with WinRunner

When you work with QuickTest, you can also run WinRunner tests and call TSL or user-defined functions in compiled modules.

This chapter describes:

- ▶ About Working with WinRunner
- ▶ Calling WinRunner Tests
- ▶ Calling WinRunner Functions

About Working with WinRunner

If you have WinRunner 7.5 or later installed on your computer, you can include calls to WinRunner tests and functions in your QuickTest test.

Note: For WinRunner versions earlier than 7.6, you cannot run WinRunner tests on Web pages (using WinRunner's WebTest Add-in) from QuickTest if the QuickTest Web Add-in is loaded. For WinRunner 7.6, you can enable this functionality by installing patch **WR76P10 - Support WR/QTP integration** from the patch database on the Mercury Customer Support site (<http://support.mercury.com>). For future versions of WinRunner, this functionality will be provided built-in.

Once you create a call to a WinRunner test or function, you can modify the argument values in call statements by editing them in the Expert View or Keyword View.

When QuickTest is connected to a Quality Center project that contains WinRunner tests or compiled modules, you can call a WinRunner test or function that is stored in that Quality Center project.

Calling WinRunner Tests

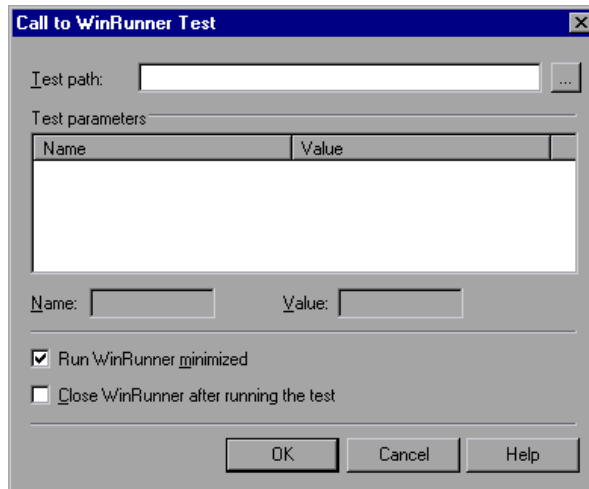
When QuickTest links to WinRunner to run a test, it starts WinRunner, opens the test, and runs it. Information about the WinRunner test run is displayed in the QuickTest Test Results window.

You can insert a call to a WinRunner test using the Call to WinRunner Test dialog box or by entering a **TSLTest.RunTestEx** statement in the Expert View.

Note: You cannot call a WinRunner test that includes calls to QuickTest tests.

To insert a call to a WinRunner test using the Call to WinRunner Test dialog box:

- 1 Choose **Insert > Call to WinRunner > Test**. The Call to WinRunner Test dialog box opens.



- 2 In the **Test path** box, enter the path of the WinRunner test or browse to it.

If you are connected to Quality Center when you click the browse button, the Open WinRunner Test from Quality Center project dialog box opens so that you can select the module from the Quality Center project. For more information on this dialog box, see “Opening Tests from a Quality Center Project” on page 375.

- 3 The Parameters box lists any test parameters required for the WinRunner test. To enter values for the parameters:

- ▶ Highlight the parameter in the **Test Parameters** list. The selected parameter is displayed in the **Name** box below the list
- ▶ Enter the new value in the **Value** box.

Note: You can also use the parameter values from a QuickTest random environment parameter or from the QuickTest Data Table as the parameters for your WinRunner test. You do this by entering the parameter information manually in the **TSLTest.RunTestEx** statement. For more information, see “Passing QuickTest Parameterized Values to a WinRunner Test” on page 354.

- 4 Select **Run WinRunner minimized** if you do not want to view the WinRunner window while the test runs. (This option is supported only for WinRunner 7.6 and later.)
- 5 Select **Close WinRunner after running the test** if you want the WinRunner application to close when the step calling the WinRunner test is complete. (This option is supported only for WinRunner 7.6 and later.)
- 6 Click **OK** to close the dialog box.

For information on WinRunner test parameters, refer to the *WinRunner User's Guide*.

In QuickTest, the call to the WinRunner test is displayed as:

- ▶ a WinRunner **RunTestEx** step in the Keyword View. For example:

	Operation	Value
TSLTest	RunTestEx	"C:\WinRunner\Tests\basic flight",True,0,MyValue

- ▶ a **TSLTest.RunTestEx** statement in VBScript in the Expert View. For example:

```
TSLTest.RunTestEx "C:\WinRunner\Tests\basic_flight",TRUE, 0, "MyValue"
```

The **RunTestEx** method has the following syntax:

```
TSLTest.RunTestEx TestPath , RunMinimized , CloseApp [ , Parameters ]
```

Note: Tests created in QuickTest 6.0 may contain calls to WinRunner tests using the **RunTest** method, which has slightly different syntax. Your tests will continue to run successfully with this method. However, if you are working with WinRunner 7.6 or later, it is recommended to update your tests to the **RunTestEx** method (and corresponding argument syntax). For more information on these methods, refer to the *QuickTest Professional Object Model Reference*.

For additional information on the **RunTestEx** method and an example of usage, refer to the *QuickTest Professional Object Model Reference*.

Passing QuickTest Parameterized Values to a WinRunner Test

Rather than setting fixed values for the parameters required for a WinRunner test, you can pass WinRunner parameter values defined in a QuickTest Data Table, random or environment parameter. You specify these parameterized values by entering the appropriate statement as the *Parameters* argument in the **TSLTest.RunTestEx** statement.

For example, suppose you want to run a WinRunner test on a Windows-based Flight Reservation application, and that the test includes parameterized statements for the number of passengers on the flight and the seat class. You can pass the WinRunner test the value for its first parameter from a QuickTest random parameter (that generates a random number between 1 and 100), and pass it the value for the seat class from a QuickTest Data Table column labeled **Class**. Your **TSLTest.RunTestEx** statement in QuickTest might look something like this:

```
TSLTest.RunTestEx "D:\test1", TRUE, FALSE, RandomNumber(1, 100) ,
DataTable("Class", dtGlobalSheet)
```

For more information on the syntax and usage of the **RandomNumber**, **Environment**, and **DataTable** methods, refer to the Utility section of the *QuickTest Professional Object Model Reference*.

Viewing the Results

When you run a call to a WinRunner test, and WinRunner 7.6 or later is installed on your computer, your QuickTest results include a node for each event that would normally be included in the WinRunner results. When you select a node corresponding to a WinRunner step, the right pane displays a summary of the WinRunner test and details about the selected step.

Note: You can also view the results of the called WinRunner test from the results folder of the WinRunner test. For WinRunner tests stored in Quality Center, you can also view the WinRunner test results from Quality Center.

For more information, refer to “Viewing WinRunner Test Steps in the Test Results” on page 684 in the *QuickTest Professional Basic Features User’s Guide*.

For more information on designing and running WinRunner tests, refer to your WinRunner documentation.

Calling WinRunner Functions

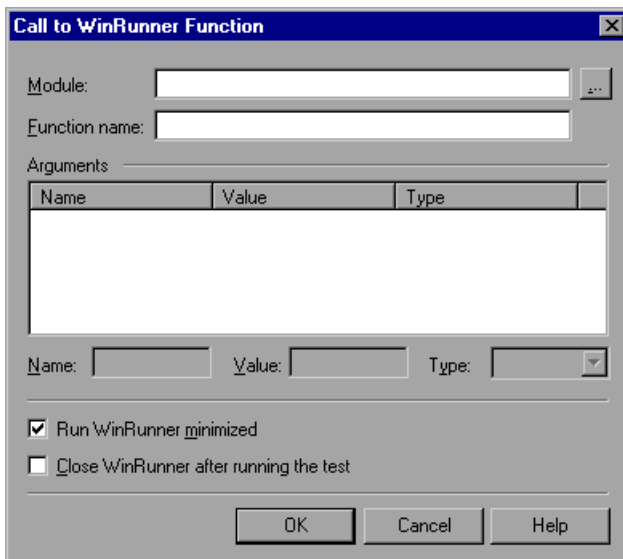
When QuickTest links to WinRunner to call a function, it starts WinRunner, loads the compiled module, and calls the function. This is useful when you want to use a user-defined function from WinRunner in QuickTest.

You call a WinRunner function from QuickTest by specifying the function and the compiled module containing the function.

Note: You cannot retrieve the values returned by the WinRunner function in your QuickTest test. However, you can view the returned value in the results.

To call a user-defined function from a WinRunner compiled module:

- 1 Choose **Insert > Call to WinRunner > Function**. The Call to WinRunner Function dialog box opens.



- 2 In the **Module** box, enter the path of the compiled module containing the function or browse to it.

If you are connected to Quality Center when you click the browse button, the Open WinRunner Test from Quality Center project dialog box opens so that you can select the compiled module from the Quality Center project.

To call a WinRunner TSL function, enter the path of any compiled module.

- 3 In the **Function name** box, enter the name of a function defined in the specified compiled module, or enter any WinRunner TSL function.
- 4 Click inside the **Arguments** box. If WinRunner is currently open on your computer, the **Arguments** box displays the argument names as defined for the selected function. If WinRunner is not open, the **Arguments** box lists **p1-p15**, representing a maximum of fifteen (15) possible arguments for the function.
- 5 Enter values for **in** or **inout** arguments as follows:
 - Highlight the argument in the **Arguments** box. The argument name is displayed in the **Name** box.
 - In the **Type** box, select the correct argument type (**in/out/inout**).
 - If the argument type is “in” or “inout,” enter the value in the **Value** box.

Note: You can also use the parameter values from a QuickTest random or environment parameter or from the QuickTest Data Table as the **in** or **inout** arguments for your function. You do this by entering the argument information manually in the **TSLTest.CallFuncEx** statement. For more information, see “Passing QuickTest Parameters to a WinRunner Function,” below.

For more information on function parameters, refer to the *WinRunner User's Guide*.

- 6 Select **Run WinRunner minimized** if you do not want to view the WinRunner window while the function runs. (This option is supported only for WinRunner 7.6 and later.)

- 7 Select **Close WinRunner after running the test** if you want the WinRunner application to close when the step calling the WinRunner function is complete. (This option is supported only for WinRunner 7.6 and later.)
- 8 Click **OK** to close the dialog box.

In QuickTest, the call to the TSL function is displayed as:

- ▶ a WinRunner **CallFuncEx** step in the Keyword View. For example:

	Operation	Value
TSLTest	CallFuncEx	"C:\WinRunner\Tests\TISStep","TISStep",True,0,"MyArg1"

- ▶ a **TSLTest.CallFuncEx** statement in VBScript in the Expert View. For example:

```
CallFuncEx "C:\WinRunner\Tests\TISStep","TISStep1",TRUE, 0, "MyArg1"
```

The **CallFuncEx** function has the following syntax:

TSLTest.CallFuncEx *ModulePath, Function, RunMinimized, CloseApp [, Arguments]*

Note: Tests created in QuickTest 6.0 may contain calls to WinRunner tests using the **CallFunc** method, which has slightly different syntax. Your tests will continue to run successfully with this method. However, if you are working with WinRunner 7.6 or later, it is recommended to update your tests to the **CallFuncEx** method (and corresponding argument syntax). For more information on these methods, refer to the *QuickTest Professional Object Model Reference*.

For additional information on the **CallFuncEx** method and an example of usage, refer to the *QuickTest Professional Object Model Reference*.

For information on WinRunner functions, function arguments, and WinRunner compiled modules, refer to the *WinRunner User's Guide* and the *WinRunner TSL Reference Guide*.

Passing QuickTest Parameters to a WinRunner Function

Rather than setting fixed values for the in and inout arguments in a WinRunner function, you can instruct QuickTest to have WinRunner use the parameter values defined in a QuickTest random or environment parameter, or in a QuickTest Data Table. You specify these parameters by entering the appropriate statement as the *Parameters* argument in the **TSLTest.CallFuncEx** statement.

For example, suppose you created a user-defined function in WinRunner that runs an application and enters the user name and password for the application.

You can instruct QuickTest to have WinRunner take the value for the user name and password from QuickTest Data Table columns labeled **FlightUserName** and **FlightPwd**. Your **TSLTest.CallFuncEx** statement in QuickTest might look something like this:

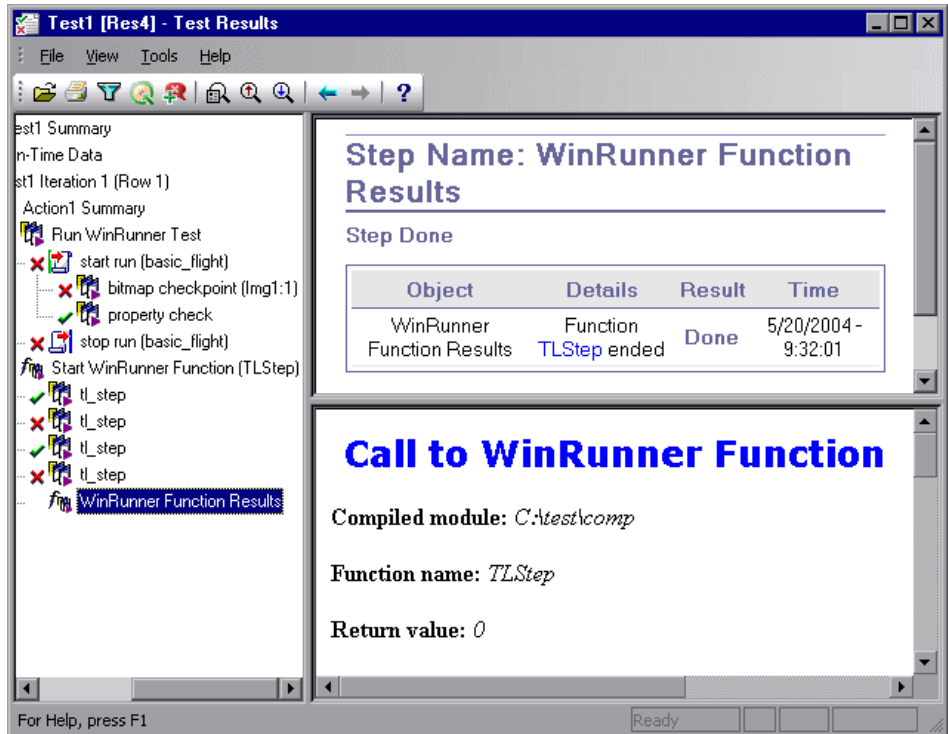
```
TSLTest.CallFuncEx "D:\flightfuncs", "run_flight", TRUE, FALSE,
DataTable("FlightUserName", dtGlobalSheet), DataTable("FlightPwd",
dtGlobalSheet)
```

For more information on the syntax and usage of the **RandomNumber**, **Environment** and **DataTable** methods, refer to the **Utility** section of the *QuickTest Professional Object Model Reference*.

Viewing the Results

After you run a WinRunner function in WinRunner 7.6 or later from QuickTest, you can view the results of your function call. The QuickTest Test Results window shows the start of the WinRunner function and the WinRunner function results. If the called function included events such as **report_msg** or **tl_step**, information about the results of these events are also included.

Highlight the **WinRunner Function Results** item in the results tree to display the function return value and additional information about the call to the function.



For more information on working with WinRunner functions and compiled modules, refer to your WinRunner documentation.

14

Working with Quality Center

To ensure comprehensive testing of your application or applications, you typically must create and run many tests. Mercury Quality Center, the centralized quality solution (formerly TestDirector), can help you organize and control the testing process.

Note: References to Quality Center features and options in this chapter apply to all currently supported versions of both Quality Center and TestDirector. Refer to the *QuickTest Professional Readme* for a list of the supported versions of Quality Center and TestDirector.

This chapter describes:

- About Working with Quality Center
- Connecting to and Disconnecting from Quality Center
- Saving Tests to a Quality Center Project
- Opening Tests from a Quality Center Project
- Working with Template Tests
- Running a Test Stored in a Quality Center Project from QuickTest
- Managing Test Versions in QuickTest
- Setting Preferences for Quality Center Test Runs

About Working with Quality Center

QuickTest integrates with Quality Center, the Mercury centralized quality solution. Quality Center helps you maintain a project of all kinds of tests (such as QuickTest tests, business process tests, manual tests, tests created using other Mercury products, and so forth) that cover all aspects of your application's functionality. Each test in your project is designed to fulfill a specified testing requirement of your application. To meet the goals of a project, you organize the tests in your project into unique groups.

Quality Center provides an intuitive and efficient method for scheduling and running tests, collecting results, analyzing the results, and managing test versions. It also features a system for tracking defects, enabling you to monitor defects closely from initial detection until resolution.

A Quality Center project is a database for collecting and storing data relevant to a testing process. For QuickTest to access a Quality Center project, you must connect to the local or remote Web server where Quality Center is installed. When QuickTest is connected to Quality Center, you can create tests and save them in your Quality Center project. After you run your tests, you can view the results in Quality Center.

You must have the following access permissions to use QuickTest with Quality Center:

- ▶ Full read and write permissions to the Quality Center cache folder
- ▶ Full read and write permissions to the QuickTest Add-in for Quality Center installation folder

When working with Quality Center, you can associate tests with external files attached to a Quality Center project. You can associate external files for all tests or for a single test. For example, suppose you set the shared object repository mode as the default mode for new tests. You can instruct QuickTest to use a specific object repository stored in Quality Center.

For more information on specifying external files for all tests, refer to Chapter 24, "Setting Global Testing Options" in the *QuickTest Professional Basic Features User's Guide*. For more information on specifying external files for a single test, refer to Chapter 25, "Setting Options for Individual Tests" in the *QuickTest Professional Basic Features User's Guide*.

You can report defects to a Quality Center project either automatically as they occur, or manually directly from QuickTest's Test Results window. For information on manually or automatically reporting defects to a Quality Center project, refer to "Submitting Defects Detected During a Run Session" on page 682 in the *QuickTest Professional Basic Features User's Guide*.

For more information on working with Quality Center, refer to the *Mercury Quality Center User's Guide*. For the latest information and tips regarding QuickTest and Quality Center integration, refer to the *QuickTest Professional Readme* (available from **Start > Programs > QuickTest Professional > Readme**).

Connecting to and Disconnecting from Quality Center

If you are working with both QuickTest and Quality Center, QuickTest can communicate with your Quality Center project.

You can connect or disconnect QuickTest to or from a Quality Center project at any time during the testing process. However, do not disconnect QuickTest from Quality Center while a QuickTest test is opened from Quality Center or while QuickTest is using a shared resource from Quality Center (such as a shared object repository or Data Table file).

Note: You can connect to any currently supported version of Quality Center or TestDirector. Refer to the *QuickTest Professional Readme* for a list of the supported versions of Quality Center and TestDirector. For more information, see "Working with the Quality Center Connectivity Add-in" on page 373.

Connecting QuickTest to Quality Center

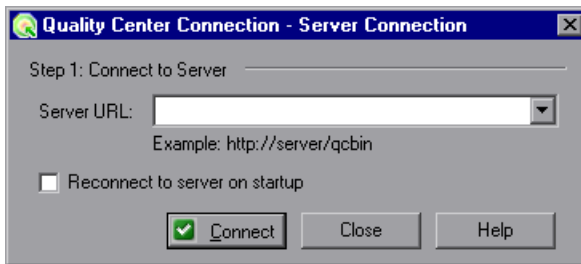
The connection process has two stages. First, you connect QuickTest to a local or remote Quality Center server. This server handles the connections between QuickTest and the Quality Center project.

Next, you log in and choose the project you want QuickTest to access. The project stores tests and run session information for the Web site or application you are testing. Note that Quality Center projects are password protected, so you must provide a user name and a password.

To connect QuickTest to a Quality Center server:



- 1** Choose **File > Quality Center Connection** or click the **Quality Center Connection** toolbar button. The Quality Center Connection - Server Connection dialog box opens.



- 2** In the **Server URL** box, type the URL address of the Web server where Quality Center is installed.

Note: You can choose a Quality Center server accessible via a Local Area Network (LAN) or a Wide Area Network (WAN).

- 3** To automatically reconnect to the Quality Center server the next time you open QuickTest, select the **Reconnect to server on startup** check box.

4 Click **Connect**.

The second stage of the connection process depends on the version of the server to which you connected. Refer to the relevant section:

- “Connecting to a Project Using an 8.x Server,” on page 365
- “Connecting to a Project Using a 9.0 Server,” on page 367

Connecting to a Project Using an 8.x Server

If you connected to a TestDirector 8.0 Service Pack 2 server or a Mercury Quality Center 8.2 Service Pack 1 server, you specify the domain and project to which you want to connect and then log in to the project.

To connect to a project using an 8.x server:

- 1 If you connected to a project in TestDirector 8.0 Service Pack 2 or Mercury Quality Center 8.2 Service Pack 1, the Quality Center Connection - Project Connection dialog box opens.

The server's name is displayed in read-only format in the QC Server box.

- 2 In the **Domain** box, select the domain that contains the Quality Center project.
- 3 In the **Project** box, select the project with which you want to work.

Note: If you select a project for which you do not have access permission, a notification is displayed when you click **Connect**.

- 4 In the **User name** box, type a user name for opening the selected project.
- 5 In the **Password** box, type the password for the selected project.
- 6 Click **Connect** to connect QuickTest to the selected project.
Once the connection to the selected project is established, the fields in the **Project Connection** area are displayed in read-only format.
- 7 To automatically reconnect to the Quality Center server the next time you open QuickTest, select the **Reconnect to server on startup** check box.
- 8 If the **Reconnect to server on startup** check box is selected, then the **Reconnect to project on startup** check box is enabled. To automatically connect to the selected project on startup, select the **Reconnect to project on startup** check box.
- 9 If the **Reconnect to project on startup** check box is selected, the **Save password for reconnection on startup** check box is enabled. To save your password for reconnection on startup, select the **Save password for reconnection on startup** check box.

If you do not save your password, you will be prompted to enter it when QuickTest connects to Quality Center on startup.

- 10 Click **Close** to close the Quality Center Connection - Project Connection dialog box. The Quality Center icon is displayed on the status bar to indicate that QuickTest is currently connected to a Quality Center project.



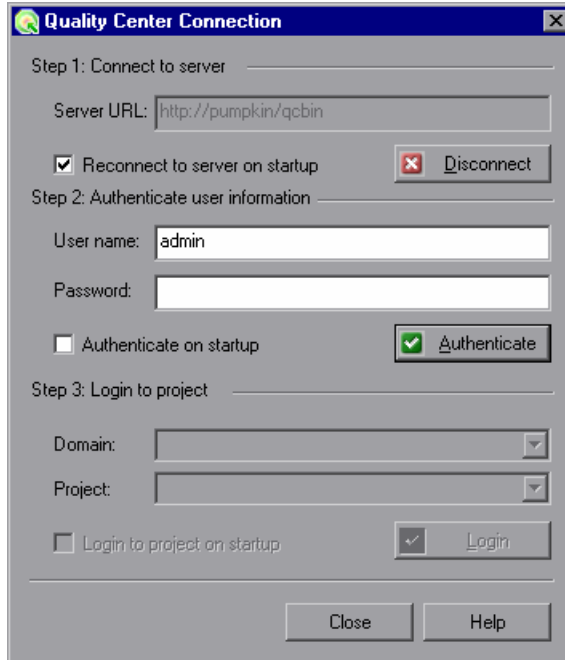
Tip: To view the current Quality Center connection, point to the **Quality Center** icon on the status bar. A tooltip displays the Quality Center server name and project to which QuickTest is connected. To open the Quality Center Connection dialog box, double-click the **Quality Center** icon.

Connecting to a Project Using a 9.0 Server

If you connected to a Mercury Quality Center 9.0 server, you specify the domain and project to which you want to connect and then log in to the project.

To connect to a project using a 9.0 server:

- 1 If you connected to a project in Quality Center 9.0, the Quality Center Connection dialog box opens.



The Quality Center server's name is displayed in read-only format in the Server URL box.

- 2 In the **User name** box, type your Quality Center user name.
- 3 In the **Password** box, type your Quality Center password.

- 4 Click **Authenticate** to authenticate your user information against the Quality Center server.

Once your user information has been authenticated, the fields in the **Authenticate user information** area are displayed in read-only format. The **Authenticate** button changes to a **Change User** button.

Tip: You can log in to the same Quality Center server using a different user name by clicking **Change User**, and then entering a new user name and password and clicking **Authenticate** again.

- 5 In the **Domain** box, select the domain that contains the Quality Center project. Only those domains that you have permission to connect to are displayed.
- 6 In the **Project** box, select the project with which you want to work. Only those projects that you have permission to connect to are displayed.
- 7 Click **Login**.
- 8 To automatically reconnect to the Quality Center server the next time you open QuickTest, select the **Reconnect to server on startup** check box.
- 9 If the **Reconnect to server on startup** check box is selected, then the **Authenticate on startup** check box is enabled. To automatically authenticate your user information the next time you open QuickTest, select the **Authenticate on startup** check box.
- 10 If the **Authenticate on startup** check box is selected, the **Login to project on startup** check box is enabled. To log in to the selected project on startup, select the **Login to project on startup** check box.

- 11 Click **Close** to close the Quality Center Connection dialog box. The Quality Center icon is displayed on the status bar to indicate that QuickTest is currently connected to a Quality Center project.



Tip: To view the current Quality Center connection, point to the **Quality Center** icon on the status bar. A tooltip displays the Quality Center server name and project to which QuickTest is connected. To open the Quality Center Connection dialog box, double-click the **Quality Center** icon.

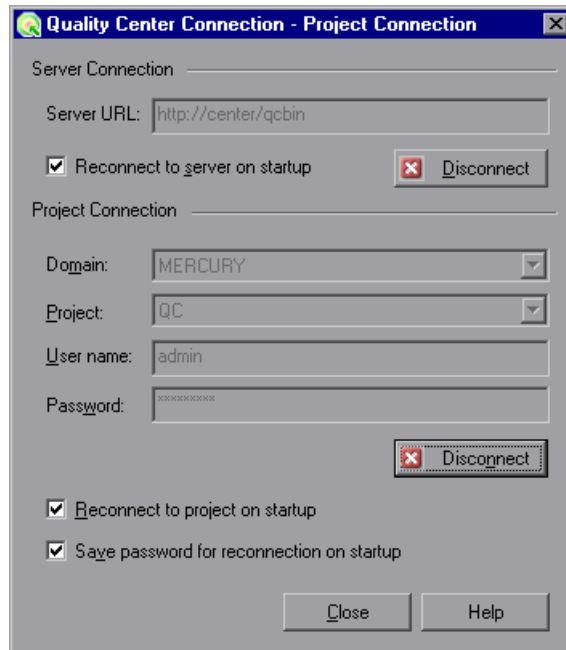
Disconnecting QuickTest from Quality Center

You can disconnect from a Quality Center project or server. Note that if you disconnect QuickTest from a Quality Center server without first disconnecting from a project, QuickTest's connection to that project database is automatically disconnected.

Note: If a Quality Center test, or shared file (such as a shared object repository or Data Table file) is open when you disconnect from Quality Center, then QuickTest closes it.

To disconnect QuickTest from an 8.x server:

- 1 Choose **File > Quality Center Connection** or click the **Quality Center Connection** toolbar button. The Quality Center Connection - Project Connection dialog box opens.

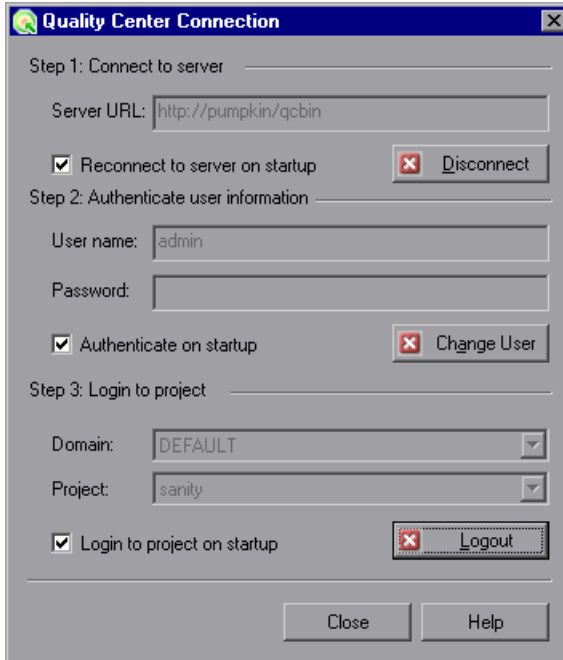


- 2 To disconnect QuickTest from the selected project, in the **Project Connection** area, click **Disconnect**.
- 3 To disconnect QuickTest from the selected Quality Center server, in the **Server Connection** area, click **Disconnect**.
- 4 Click **Close** to close the Quality Center Connection - Project Connection dialog box.

To disconnect QuickTest from a 9.0 server:



- 1 Choose **File > Quality Center Connection** or click the **Quality Center Connection** toolbar button. The Quality Center Connection dialog box opens.



- 2 To disconnect QuickTest from the selected project, in the **Step 3: Login to project** area, click **Logout**.
- 3 To disconnect QuickTest from the selected Quality Center server, in the **Step 1: Connect to server** area, click **Disconnect**.

Tip: You can log in to the same Quality Center server using a different user name by clicking **Change User** and then entering a new user name and password and clicking **Authenticate** again.

- 4 Click **Close** to close the Quality Center Connection dialog box.

Working with the Quality Center Connectivity Add-in

To connect to Quality Center, the Quality Center Connectivity Add-in must be installed. This add-in is installed automatically when you connect to Quality Center using the Quality Center Connection dialog box.



To view the version of the Quality Center Connectivity Add-in that is currently installed on your computer, choose **Help > About** and then click the **Product Information** button. For more information, refer to “Viewing Product Information” on page 54 in the *QuickTest Professional Basic Features User's Guide*.

To manually install the Quality Center Connectivity Add-in, choose **Quality Center Connectivity** from the Quality Center Add-ins page (available from the Quality Center main screen).

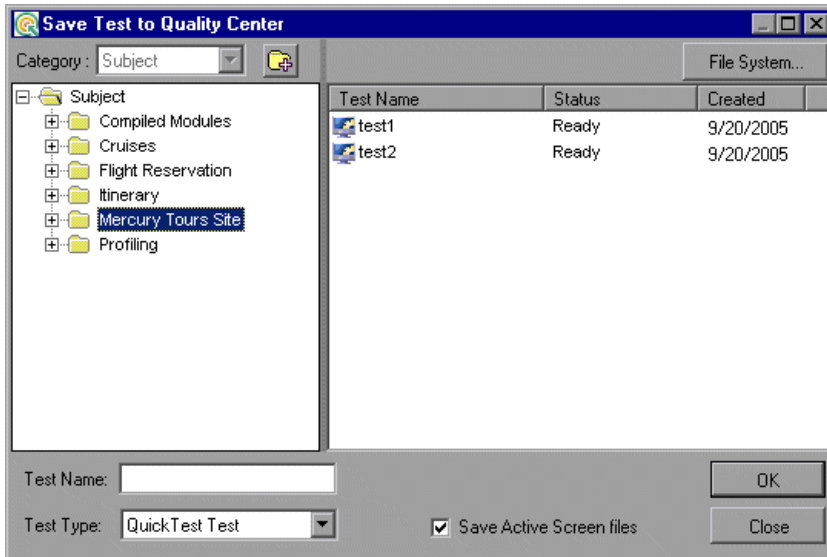
Note: The Quality Center Connectivity Add-in also enables access to TDOTA functionality (i.e. via an automation program), even if Quality Center is not installed on your computer. For more information on accessing TDOTA using automation programs, refer to the *QuickTest Automation Object Model Reference*.

Saving Tests to a Quality Center Project

When QuickTest is connected to a Quality Center project, you can create new tests in QuickTest and save them directly to your project. To save a test, you give it a descriptive name and associate it with the relevant subject in the test plan tree. This helps you to keep track of the tests created for each subject and to quickly view the progress of test planning and creation.

To save a test to a Quality Center project:

- 1 Connect to a Quality Center server and project. For more information, see “Connecting QuickTest to Quality Center” on page 364.
- 2 In QuickTest, click **Save** or choose **File > Save** to save the test. The Save Test to Quality Center dialog box opens and displays the test plan tree.



Note that the Save Test to Quality Center dialog box opens only when QuickTest is connected to a Quality Center project.

To save a test directly in the file system, click the **File System** button to open the Save QuickTest Test dialog box. (From the Save QuickTest Test dialog box, you can return to the Save Test to Quality Center project dialog box by clicking the **Quality Center** button.)

- 3 Select the relevant subject folder in the test plan tree. To expand the tree and view a sublevel, double-click a closed folder. To collapse a sublevel, double-click an open folder.
- 4 In the **Test Name** box, enter a name for the test. Use a descriptive name that will help you easily identify the test. Note that a test name cannot exceed 220 characters (including the path), cannot begin or end with spaces, and cannot include the following characters:
\\ : * ? " < > | % ' ;
- 5 Confirm that the **Save Active Screen files** is selected if you want to save the Active Screen files with your test. Note that if you clear this box, your Active Screen files will be deleted, and you will not be able to edit your test using Active Screen options. For more information, refer to “Saving a Test” on page 100 in the *QuickTest Professional Basic Features User’s Guide*.
- 6 Click **OK** to save the test and close the dialog box. Note that the text in the status bar changes while QuickTest saves the test.

The next time you start Quality Center, the new test will be included in Quality Center’s test plan tree. For more information, refer to the *Mercury Quality Center User’s Guide*.

Opening Tests from a Quality Center Project

When QuickTest is connected to a Quality Center project, you can open QuickTest tests that are a part of your Quality Center project. You locate tests according to their position in the test plan tree, rather than by their actual location in the file system. You can also open tests from the recent tests list in the **File** menu.

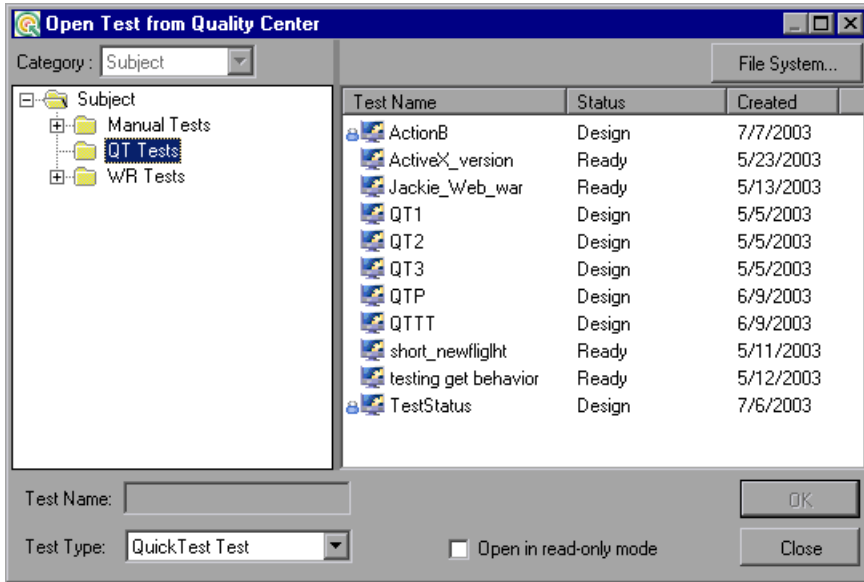
When you open a test in a Quality Center project with version control support, icons indicate the test’s version control status.

To open a test from a Quality Center project:

- 1 Connect to a Quality Center server and project. For more information, see “Connecting QuickTest to Quality Center” on page 364.



- 2 In QuickTest, click **Open** or choose **File > Open > Test** to open the test. The Open Test from Quality Center dialog box opens and displays the test plan tree.



Note that the Open Test from Quality Center Project dialog box opens only when QuickTest is connected to a Quality Center project.

Note: To open a test directly from the file system while you are connected to Quality Center, click the **File System** button to open the Open Test dialog box. (From the Open Test dialog box, you can click the **Quality Center** button to return to the Open Test from Quality Center Project dialog box.)

- 3 Click the relevant subject in the test plan tree. To expand the tree and view sublevels, double-click closed folders. To collapse the tree, double-click open folders.

Note that when you select a subject, the tests that belong to the subject are displayed in the right pane of the Open Test from Quality Center Project dialog box.

- ▶ If the test is stored in a Quality Center project with version control support, icons next to the **Test Name** indicate the test's version control status. For more information, see “Opening Tests from a Quality Center Project with Version Control Support” on page 378.
 - ▶ The **Test Name** column lists the names of the tests that belong to the selected subject.
 - ▶ The **Status** column indicates whether each test is in **Design** stage or is **Ready** for test runs. Note that by default, tests saved to a Quality Center project from QuickTest are labeled as **Design**. The status can be changed only from the Quality Center client.
 - ▶ The **Created** column indicates the date on which each test was created.
- 4** Select a test in the **Test Name** list. The test is displayed in the read-only **Test Name** box.
 - 5** If you want to open the test in read-only mode, select the **Open in read-only mode** check box.
 - 6** Click **OK** to open the test.

As QuickTest downloads and opens the test, the operations it performs are displayed in the status bar.

When the test opens, the QuickTest title bar displays [Quality Center], the full subject path and the test name. For example:

[Quality Center] Subject\System\qa_test1

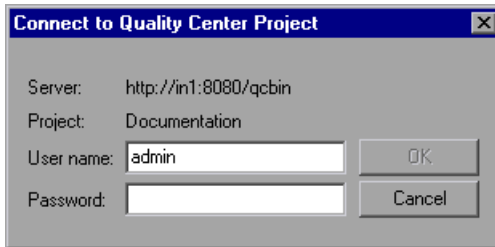
The test opens in read-only mode if:

- ▶ You selected **Open in read-only mode**
- ▶ You opened a test that is currently checked in to the Quality Center version control database (for projects that support version control)
- ▶ You opened a test that is currently checked out to another user (for projects that support version control)

For more information, see “Opening Tests from a Quality Center Project with Version Control Support” on page 378.

Opening Tests from the Recent Files List

You can open Quality Center tests from the recent files list in the File menu. If you select a test located in a Quality Center project, but QuickTest is currently not connected to Quality Center or to the correct project for the test, the Connect to Quality Center Project dialog box opens and displays the correct server, project, and the name of the user who most recently opened the test on this computer.



Log in to the project, and click **OK**.

The Connect to Quality Center Project dialog box also opens if you choose to open a test that was last edited on your computer using a different Quality Center user name. You can either log in using the displayed name or you can click **Cancel** to stay logged in with your current user name.




Opening Tests from a Quality Center Project with Version Control Support



When you click the **Open** toolbar button or choose **File > Open > Test** to open a test from a Quality Center project with version control support, the Open QuickTest Test from Quality Center Project dialog box displays icons that indicate the version control status of each test in the selected subject.

When you open a test from a Quality Center project with version control support, the test opens in read-write or read-only mode depending on the current version control status of the test.

The table below summarizes the version control status icons and the open mode for each status:

Icon	Description	Open Mode
<None>	The test is currently checked in to the version control database.	Read-only
	The test is currently checked out to you.	Read-write
	The test is currently checked out to another user.	Read-only
	An old version of the test is currently open on your computer.	As is

For more information about working with tests stored in a Quality Center project with version control, see “Managing Test Versions in QuickTest” on page 389.

Working with Template Tests

Template tests serve as the basis for all QuickTest tests created in Quality Center. A template test is a QuickTest test that contains default test settings. For example, a template test might specify the QuickTest add-ins, associated function libraries, and recovery scenarios that are associated with a test. You can modify these test settings in the Test Settings dialog box (**File > Settings**) in QuickTest.

In addition to default test settings, a template test can also contain any comments or steps you want to include with all new QuickTest tests created in Quality Center. For example, you may want to add a comment notifying users which add-ins are associated with the template test, or you may want to add a step that opens a specific Web page or application at the beginning of every test. Any steps or comments you add to a template test are included in all new tests created in Quality Center that are based on that template test.

A default template test is installed on each Quality Center client when the QuickTest Professional Add-in for Quality Center is installed. You can modify this default template test, or you can create other template tests with various test settings.

If you decide to modify the default template test, it is recommended to copy the modified template test to the relevant **Templates** folder on all computers from which Quality Center users might create tests. This overwrites the local template test and ensures that all Quality Center users will create QuickTest tests based on the same template test (and not their default local copy). For more information, see “Working with the Default Template Test” below.

All template tests are saved in your Quality Center project (except for the default template test, which is located on the Quality Center client). These template tests do not need to be copied to each user’s local computer. This enables users to customize their local template tests, if needed, and still have access to globally maintained template tests. For more information, see “Working with New Template Tests” on page 381.

When tests based on a specific template test are run from Quality Center, QuickTest automatically loads the associated add-ins and applies the required settings, as defined in the test.

Working with the Default Template Test

When you install the QuickTest Add-in for Quality Center, default template tests for all supported QuickTest versions are installed in the <**QuickTest Add-in for Quality Center folder**>\bin\Templates folder on your computer (for example: C:\Program Files\Mercury Interactive\QuickTest Add-in for Quality Center\bin\Templates\Template90).

When a Quality Center user creates a new QuickTest test in Quality Center, the default template test for the installed QuickTest version is automatically associated with the test unless the users selects another template test, as described in “Creating a QuickTest Test in Quality Center” on page 384.

You can modify the template test that is installed by default with the QuickTest Add-in for Quality Center. Because the default template test is installed locally, any changes you make to the template test are applied only to tests created on your computer (using the Quality Center client). Therefore, if you want to modify the template test for a group of users, you should copy your modified template test to all Quality Center client computers. This ensures that every new test created in Quality Center based on the default template test has the same basic test settings defined.

Alternatively, you can create a new template test, as described in the following sections.

For more information on applying the default template test to a new QuickTest in Quality Center, see “Creating a QuickTest Test in Quality Center” on page 384.

Working with New Template Tests

When you create new template tests, they are stored in your Quality Center project, making them available as the basis for new QuickTest tests created in that Quality Center project.

You can create multiple template tests, each for a specific testing purpose. For example, you may want to create one template test for QuickTest tests that test Web applications with ActiveX controls, and another for QuickTest tests that test standard Windows applications. You would associate the ActiveX and Web Add-ins with the first template test. For the second template test, you would not associate any QuickTest add-ins at all, but you might specify the Windows application on which you want to record and run. You could also make other modifications to the test settings for each of the template tests, as needed.

As you create each template test, you can save it with a descriptive name that clearly indicates its purpose, such as, `ActiveX_Web_Addins_Template` or `Std_Windows_Template_Test`. Users can then choose the appropriate template test when creating QuickTest tests in Quality Center.

Note: When you define a template test that associates specific QuickTest add-ins, make sure that the add-ins are actually installed on the QuickTest computer on which the test will eventually run. Otherwise, when the test is run, QuickTest will not be able to load the required add-ins and the test may fail. For more information on running QuickTest tests from Quality Center, refer to the *Mercury Quality Center* documentation.

Creating a New Template Test


You create a template test by first creating a blank test in QuickTest with the required test settings. Then, in the Test Plan module of your Quality Center project, you browse to your QuickTest test and save it as a **Template Test**.

Note: When you save the test in QuickTest, you should apply a descriptive name that clearly indicates its purpose. For example, if the template test is to be used to associate the ActiveX and Web Add-ins with a new test, you could call it `ActiveX_Web_Addins_Template`.


Tip: In the Quality Center test plan tree (Test Plan module), you may want to create a special folder for your template tests. This will enable other users to quickly locate the relevant template test when they create new QuickTest tests in Quality Center.

To create a template test:

In QuickTest:

- 1** Open QuickTest with the required add-ins loaded. For more information on loading QuickTest add-ins, refer to “Loading QuickTest Add-ins” on page 795 in the *QuickTest Professional Basic Features User’s Guide*.
- 2** Define the required settings in the Test Settings dialog box (**File > Settings**). For more information, refer to “Using the Test Settings Dialog Box” on page 741 in the *QuickTest Professional Basic Features User’s Guide*.
- 3** If you want to include comments or steps in all tests based on this template test, add them.
-  **4** Click the **Save** button or choose **File > Save** to save the test. The Save Test to Quality Center dialog box opens. Save the test to your Quality Center project using a descriptive name that clearly indicates its purpose. For more information, see “Saving Tests to a Quality Center Project” on page 374.

In Quality Center:

-  **5** Open the project in Quality Center, click the **Test Plan** button on the sidebar to open the Test Plan module, and browse to the test you saved in step 4.
- 6** Right-click the test and choose **Template Test**. The test is converted to a template test.
- 7** Repeat steps 1 to 6 to create additional template tests, as needed.

Creating a QuickTest Test in Quality Center

In Quality Center, you create QuickTest tests in the Test Plan module. When you create a QuickTest test, you apply a template test to it. You can choose either the default template test stored on your QuickTest client, or a template test that is saved in your Quality Center project.

If you do not have any template tests saved in your Quality Center project, or if you choose <None> in the Template box (in the Create New Test dialog box shown on page 375), Quality Center uses the settings defined in the template test that was installed with the **QuickTest Add-in for Quality Center** on your Quality Center client. For more information, see “Working with the Default Template Test” on page 380. Otherwise, if you have at least one template test saved in your Quality Center project, you can select it when creating a new QuickTest test. For more information, see “Working with New Template Tests” on page 381.

Note: When you create a QuickTest test in Quality Center, you must choose a template test that specifies the QuickTest add-ins to be associated with the test. Otherwise the required QuickTest add-ins will not be loaded during the run session.

Your new QuickTest test will use all of the settings defined in the template test you choose. When the test runs from Quality Center, QuickTest uses the settings specified in the Test Settings dialog box, and automatically loads the required QuickTest add-ins.

Note: The following procedure describes how to create a test in Quality Center using a template test. This procedure may be different depending on your version of Quality Center. For the most updated instructions on creating a new test in Quality Center, refer to the *Mercury Quality Center User's Guide*.

To create a test in Quality Center using a template test:

1 In Quality Center, click the **Test Plan** button on the sidebar to open the Test Plan module.

2 In the test plan tree, choose a folder.



3 Click the **New Test** button, or choose **Test > New Test**. The Create New Test dialog box opens.

Note: The **Template** box is displayed only if the **Quality Center Add-in** or **QuickTest Professional Add-in** is installed on your computer. If the **Template** box is not displayed, you must install the **Quality Center Add-in** from the QuickTest Professional CD-ROM or the **QuickTest Professional Add-in** from the More Mercury Quality Center Add-ins page (opened from the Mercury Quality Center options or login windows > **Add-ins Page** link).

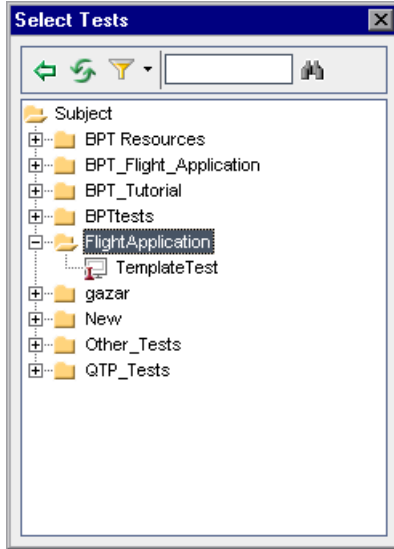
4 From the **Test Type** list, select **QUICKTEST_TEST**.

5 In the **Test Name** box, type a name for the test using alphanumeric characters (and underscores, if needed). Note that a test name cannot exceed 220 characters (including the path), cannot begin or end with spaces, and cannot include the following characters:

\ / : * ? " < > | % ' ;

6 Click the **Template** box browse button. The Select Tests dialog box opens.

- 7 Expand the folder containing your template test.



- 8 Select the template test on which to base your new test and click the **Add** button. The Select Tests dialog box closes and the template test you selected is displayed in the **Template** box (in the Create New Test dialog box).
- 9 In the Create New Test dialog box, click **OK**. The new test is created with the test settings defined in the template test.
- 10 Click **OK** to close the Create New Test dialog box. The new test is displayed in the test plan tree under the subject folder you selected.

Note: If the Required Fields dialog box opens, set the required values and click **OK**. For more information, refer to the *Mercury Quality Center Administrator's Guide*.

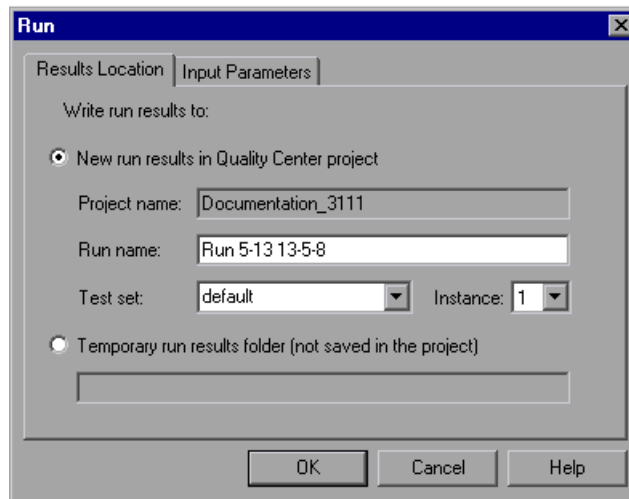
- 11 Continue creating the test. For more information on creating tests in Quality Center, refer to the *Mercury Quality Center User's Guide*.

Running a Test Stored in a Quality Center Project from QuickTest

QuickTest can run a test from a Quality Center project and save the run results in the project. To save the run results, you specify a name for the run session and a test set in which to store the results.

To save run results to a Quality Center project:

- 1 In QuickTest, click the **Run** button or choose **Automation > Run**. The Run dialog box opens.



- 2 The **Project name** box displays the Quality Center project to which you are currently connected.

To save the run results in the Quality Center project, accept the default **Run name**, or type a different one in the box.

- 3 Accept the default **Test set**, or browse to select another one.

- 4 If there is more than one instance of the test in the test set, specify the instance of the test for which you want to save the results in the **Instance** box.

Note: A *test set* is a group of tests selected to achieve specific testing goals. For example, you can create a test set that tests the user interface of the application or the application's performance under stress. You define test sets when working in Quality Center's test run mode. For more information, refer to your Quality Center documentation.

To run the test, overwriting the previous test run results, select the **Temporary run results folder (not saved in the project)** option.

Note: QuickTest stores temporary test run results for all tests in **<System Drive:\Temp\TempResults>**. The path in the text box of the **Temporary run results folder (not saved in the project)** option is read-only and cannot be changed.

- 5 Click **OK**. The Run dialog box closes and QuickTest begins running the test. As QuickTest runs the test, it highlights each step in the Keyword View.

When the test stops running, the Test Results window opens unless you have cleared the **View results when test run ends** check box in the Run tab of the Options dialog box. For more information about the Options dialog box, refer to Chapter 24, "Setting Global Testing Options" in the *QuickTest Professional Basic Features User's Guide*.

When the test stops running, **Uploading** is displayed in the status bar. The Test Results window opens when the uploading process is completed.

Note: You can report defects to a Quality Center project either automatically as they occur, or manually directly from QuickTest's Test Results window. For more information, refer to "Submitting Defects Detected During a Run Session" on page 682 in the *QuickTest Professional Basic Features User's Guide*.

Managing Test Versions in QuickTest

When QuickTest is connected to a Quality Center project with version control support, you can update and revise your automated test scripts while maintaining old versions of each test. This helps you keep track of the changes made to each test script, see what was modified from one version of a script to another, or return to a previous version of the test script.

You add a test to the version control database by saving it in a project with version control support. You manage test versions by checking tests in and out of the version control database.

The test with the latest version is the test that is located in the Quality Center test repository and is used by Quality Center for all test runs.

Notes:

A Quality Center project with version control support requires the installation of version control software as well as Quality Center's Version Control Add-in. For more information, refer to your Quality Center documentation.

The **Quality Center Version Control** options in the **File** menu are available only when you are connected to a Quality Center project database with version control support and you have a Quality Center test open.

Adding Tests to the Version Control Database

When you use **Save As** to save a new test in a Quality Center project with version control support, QuickTest automatically saves the test in the project, checks the test into the version control database with version number 1.1.1 and then checks it out so that you can continue working.

The QuickTest status bar indicates each of these operations as they occur. Note, however, that saving your changes to an existing test does not check them in. Even if you save and close the test, the test remains checked out until you choose to check it in. For more information, see “Checking Tests into the Version Control Database” on page 392.

Checking Tests Out of the Version Control Database

When you choose **File > Open > Test** to open a test that is currently checked in to the version control database, it is opened in read-only mode.

Note: The Open Test from Quality Center Project dialog box displays icons that indicate the version control status of each test in your project. For more information, see “Opening Tests from a Quality Center Project” on page 375.

You can review the checked-in test. You can also run the test and view the results.

To modify the test, you must check it out. When you check out a test, Quality Center copies the test to your unique check-out directory (automatically created the first time you check out a test), and locks the test in the project database. This prevents other users of the Quality Center project from overwriting any changes you make to the test. However, other users can still run the version that was last checked in to the database.

You can save and close the test, but it remains locked until you return the test to the Quality Center database. To release the test either check the test in, or undo the check out operation. For more information on checking tests in, see “Checking Tests into the Version Control Database” on page 392. For more information on undoing the check-out, see “Canceling a Check-Out Operation” on page 397.

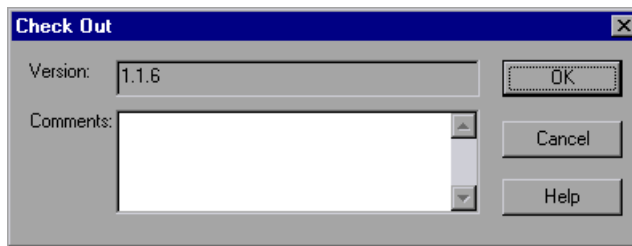
By default, the check out option accesses the latest version of the test. You can also check out older versions of the test. For more information, see “Using the Version History Dialog Box” on page 394.

To check out the latest version of a test:

- 1 Open the test you want to check out. For more information, see “Opening Tests from a Quality Center Project” on page 375.

Note: Make sure the test you open is currently checked in. If you open a test that is checked out to you, the **Check Out** option is disabled. If you open a test that is checked out to another user, all **Quality Center Version Control** options, except the **Version History** option, are disabled.

- 2 Choose **File > Quality Center Version Control > Check Out**. The Check Out dialog box opens and displays the test version to be checked out.



- 3 You can enter a description of the changes you plan to make in the **Comments** box.
- 4 Click **OK**. The read-only test closes and automatically reopens as a writable test.

5 View or edit your test as necessary.

Note: You can save changes and close the test without checking the test in, but your changes will not be available to other Quality Center users until you check it in. If you do not want to check your changes in, you can undo the check-out. For more information on checking tests in, see “Checking Tests into the Version Control Database” on page 392. For more information on undoing the check-out, see “Canceling a Check-Out Operation” on page 397.

Checking Tests into the Version Control Database

While a test is checked out, Quality Center users can run the previously checked-in version of your test. For example, suppose you check out version 1.2.3 of a test and make a number of changes to it and save the test. Until you check the test back in to the version control database as version 1.2.4 (or another number that you assign), Quality Center users can continue to run version 1.2.3.

When you have finished making changes to a test and you are ready for Quality Center users to use your new version, you check it in to the version control database.

Note: If you do not want to check your changes into the Quality Center database, you can undo the check-out operation. For more information, see “Canceling a Check-Out Operation” on page 397.

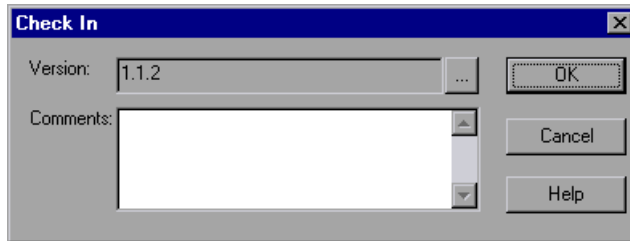
When you check a test back into the version control database, Quality Center deletes the test copy from your checkout directory and unlocks the test in the database so that the test version will be available to other users of the Quality Center project.

To check in the currently open test:

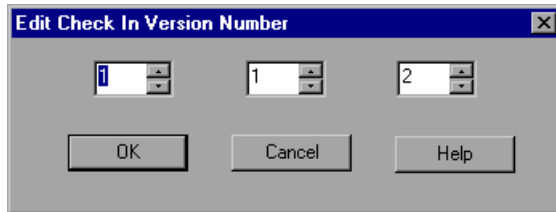
- 1 Confirm that the currently open test is checked out to you. For more information, see “Viewing Version Information For a Test” on page 394.

Note: If the open test is currently checked in, the **Check In** option is disabled. If you open a test that is checked out to another user, all **Quality Center Version Control** options, except the **Version History** option, are disabled.

- 2 Choose **File > Quality Center Version Control > Check In**. The Check In dialog box opens.



- 3 Accept the default new version number and proceed to step 7, or click the browse button to specify a custom version number. If you click the browse button, The Edit Check In Version Number dialog box opens.



- 4 Modify the version number manually or using the up and down arrows next to each element of the version number. You can enter numbers 1-900 in the first element. You can enter numbers 1-999 in the second and third elements. You cannot enter a version number lower than the most recent version of this test in the version control database.

- 5 Click **OK** to save the version number and close the Edit Check In Version Number dialog box.
- 6 If you entered a description of your change when you checked out the test, the description is displayed in the **Comments** box. You can enter or modify the comments in the box.
- 7 Click **OK** to check in the test. The test closes and automatically reopens as a read-only test.

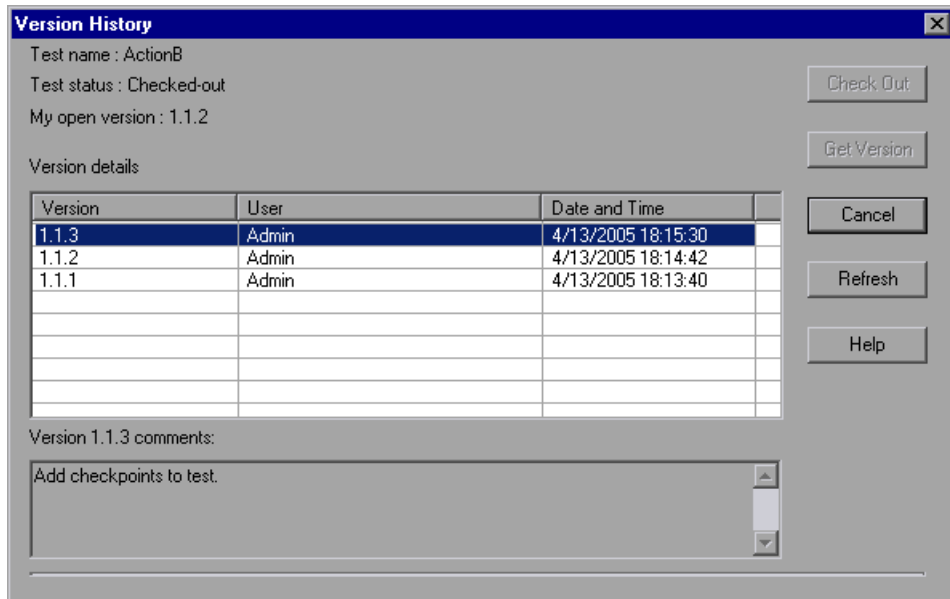
Using the Version History Dialog Box

You can use the Version History dialog box to view version information about the currently open test and to view or retrieve an older version of the test.

Viewing Version Information For a Test

You can view version information for any open test that has been stored in the Quality Center version control database, regardless of its current status.

To open the Version History dialog box for a test, open the test and choose **File > Quality Center Version Control > Version History**.



The Version History dialog box provides the following information:

Test name—The name of the currently open test.

Test status—The status of the test. The test can be:

- **Checked-in**—The test is currently checked in to the version control database. It is currently open in read-only format. You can check out the test to edit it.
- **Checked-out**—The test is checked out by you. It is currently open in read-write format.
- **Checked-out by <another user>**—The test is currently checked out by another user. It is currently open in read-only format. You cannot check out or edit the test until the specified user checks in the test.

My open version—The test version that is currently open on your QuickTest computer.

Version details—The version details for the test.

- **Version**—A list of all versions of the test.
- **User**—The user who checked in each listed version.
- **Date and Time**—The date and time that each version was checked in.

Version comments—The comments that were entered when the selected test version was checked in.

Working with Previous Test Versions

You can view an old version of a test in read-only mode or you can check out an old version and then check it in as the latest version of the test.

To view an old version of a test:

- 1** Open the Quality Center test. The latest version of the test opens. For more information, see “Opening Tests from a Quality Center Project” on page 375.
- 2** Choose **File > Quality Center Version Control > Version History**. The Version History dialog box opens.
- 3** Select the test version you want to view in the **Version details** list.

- 4 Click the **Get Version** button. QuickTest reminds you that the test will open in read-only mode because it is not checked out.
- 5 Click **OK** to close the QuickTest message. The selected version opens in read-only mode.

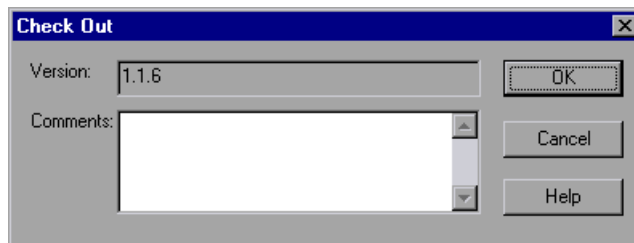
Tips:

To confirm the version number that you now have open in QuickTest, look at the **My open version** value in the Version History dialog box.

After using the **Get Version** option to open an old version in read-only mode, you can check-out the open test by choosing **File > Quality Center Version Control > Check Out**. This is equivalent to using the **Check Out** button in the Version History dialog box.

To check out an old version of a test:

- 1 Open the Quality Center test. The latest version of the test opens. For more information, see “Opening Tests from a Quality Center Project” on page 375.
- 2 Choose **File > Quality Center Version Control > Version History**. The Version History dialog box opens.
- 3 Select the test version you want to view in the **Version details** list.
- 4 Click the **Check Out** button. A confirmation message opens.
- 5 Confirm that you want to check out an older version of the test. The Check Out dialog box opens and displays the test version to be checked out.



- 6** You can enter a description of the changes you plan to make in the **Comments** box.
- 7** Click **OK**. The open test closes and the selected version opens as a writable test.
- 8** View or edit the test as necessary.
- 9** If you want to check in your test as the new, latest version in the Quality Center database, choose **File > Quality Center Version Control > Check In**. If you do not want to upload the modified test to Quality Center, choose **File > Quality Center Version Control > Undo Check out**.

For more information on checking tests in, see “Checking Tests into the Version Control Database” on page 392. For more information on undoing the check-out, see “Canceling a Check-Out Operation” on page 397.

Canceling a Check-Out Operation

If you check out a test and then decide that you do not want to upload the modified test to Quality Center you should cancel the check-out operation so that the test will be available for check out by other Quality Center users.

To cancel a check-out operation:

- 1** If it is not already open, open the checked-out test.
- 2** Choose **File > Quality Center Version Control > Undo Check out**.
- 3** Click **Yes** to confirm the cancellation of your check-out operation. The check-out operation is cancelled. The checked-out test closes and the previously checked-in version reopens in read-only mode.

Setting Preferences for Quality Center Test Runs

You can run QuickTest tests that are stored in a Quality Center database via QuickTest, via a Quality Center client that is installed on your computer, or via a remote Quality Center client or server. When Quality Center runs your QuickTest test, it uses the associated add-ins list to load the proper add-ins for your test on the QuickTest computer. For more information, refer to “Modifying Associated Add-Ins” on page 745 and “Working with Template Tests” on page 379 in the *QuickTest Professional Basic Features User’s Guide*.

Note: You cannot run QuickTest tests from Quality Center if the QuickTest computer is logged off or locked.

You can instruct QuickTest to report a defect for each failed step when Quality Center test runs on your QuickTest computer. You can also submit defects to Quality Center manually from the QuickTest Test Results window. For more information, refer to “Submitting Defects Detected During a Run Session” on page 682 in the *QuickTest Professional Basic Features User’s Guide*.

Before you instruct a remote Quality Center client to run QuickTest tests on your computer, you must give Quality Center permission to use your QuickTest application. You can also view or modify the QuickTest Remote Agent Settings.

Enabling Quality Center to Run Tests on a QuickTest Computer

For security reasons, remote access to your QuickTest application is not enabled. If you want to allow Quality Center (or other remote access clients) to open and run QuickTest tests, you must select the **Allow other Mercury products to run tests and components** option.

In addition, if you want to run QuickTest tests remotely from Quality Center, and QuickTest is installed on Windows XP Service Pack 2 or Windows 2003 Server, you must first change DCOM permissions and open firewall ports. For more information, refer to the *QuickTest Professional Installation Guide*, or access the knowledge base (<http://support.mercury.com/cgi-bin/portal/CSO/kbBrowse.jsp>) and search for article number 43245.

To enable remote Quality Center clients to run tests on your QuickTest computer:

- 1 Open QuickTest.
- 2 Choose **Tools > Options** or click the **Options** toolbar button. The Options dialog box opens.
- 3 Click the **Run** tab.
- 4 Select the **Allow other Mercury products to run tests and components** check box.

For more information on this option, refer to “Setting Run Testing Options” on page 709 in the *QuickTest Professional Basic Features User’s Guide*.

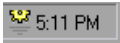
Tip: To access QuickTest tests from Quality Center, you must also have the QuickTest Add-in for Quality Center installed on the Quality Center computer. For more information on this add-in, refer to the QuickTest Professional Add-in screen (accessible from the main Quality Center screen).

Setting QuickTest Remote Agent Preferences

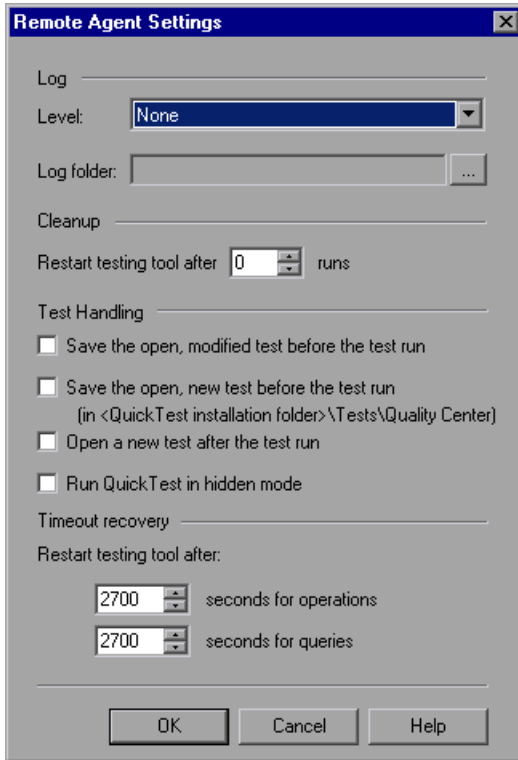
When you run a QuickTest test from Quality Center, the QuickTest Remote Agent opens on the QuickTest computer. The QuickTest Remote Agent determines how QuickTest behaves when a test is run by a remote application such as Quality Center.

You can open the Remote Agent Settings dialog box at any time to view or modify the settings that your QuickTest application uses when Quality Center runs a QuickTest test on your computer.

To open the Remote Agent Settings dialog box:



- 1 Choose **Start > Programs > QuickTest Professional > Tools > Remote Agent**. The Remote Agent opens and the Remote Agent icon is displayed in the task bar tray.
- 2 Right-click the Remote Agent icon and choose **Settings**. The Remote Agent Settings dialog box opens.



- 3 View or modify the settings in the dialog box. For more information, see “Understanding the Remote Agent Settings Dialog Box,” below.
- 4 Click **OK** to save your settings and close the dialog box.
- 5 Right-click the Remote Agent icon and choose **Exit** to end the Remote Agent session.

Understanding the Remote Agent Settings Dialog Box

The Remote Agent Settings dialog box enables you to view or modify the settings that your QuickTest application uses when Quality Center runs a QuickTest test on your computer.

The Remote Agent Settings dialog box contains the following options:

Option	Description
Level	<p>The level of detail to include in the log that is created when Quality Center runs a QuickTest test.</p> <p>None (default)—No log is created.</p> <p>Low—The log lists any Quality Center-QuickTest communication errors.</p> <p>Medium—The log includes Quality Center-QuickTest communication errors and information on other major operations that result in Quality Center-QuickTest communication.</p> <p>High—The log includes all available information related to Quality Center-QuickTest communications.</p>
Log folder	<p>The folder path for storing the log file. Required if a log type is specified in the Level option.</p>
Restart testing tool after ___ runs	<p>Restarts the QuickTest application after the Quality Center completes the specified number of test runs. When QuickTest restarts, it continues with the next test in the test set.</p> <p>You may want to use this option to maximize available memory.</p> <p>If you do not want QuickTest to restart during a test set, enter 0 (default).</p>
Save the open, modified test before the test run	<p>If an existing (named) test is open in QuickTest when the Remote Agent begins running a test, this option ensures that any modifications to the test are saved.</p>

Option	Description
<p>Save the open, new test before the test run</p>	<p>If a new (untitled) test is open in QuickTest when the Remote Agent begins running a test, this option saves the test in: <QuickTest installation folder>\Tests\Quality Center with a sequential test name.</p>
<p>Open a new test after the test run</p>	<p>By default, the last test run by the remote agent stays open in QuickTest when it finishes running all tests. However, if any shared resources (such as a shared object repository or Data Table file) are associated with the open test, those resources are locked to other users until the test is closed. You can select this option to ensure that the last test that Quality Center runs is closed, and a blank test is open instead.</p>
<p>Run QuickTest in hidden mode</p>	<p>Specifies whether to run QuickTest in hidden (silent) mode.</p>
<p>Restart testing tool after</p>	<p>Restarts QuickTest if there is no response after the specified number of seconds for:</p> <p>Operations—QuickTest operations such as Open or Run.</p> <p>Queries—Standard status queries that remote applications perform to confirm that the application is responding (such as Quality Center's get_status query).</p> <p>The default value for both options is 2700 seconds (45 minutes). However, while QuickTest operations may take a long time between responses, queries usually take only several seconds. Therefore, you may want to set different values for each of these options.</p>

15

Working with Business Process Testing

When you are connected to a Quality Center project with Business Process Testing support, QuickTest enables you to create and/or implement the steps for the components that are used in Quality Center business process tests.

This chapter describes:

- ▶ About Working with Business Process Testing
- ▶ Understanding Business Process Testing Roles
- ▶ Understanding Business Process Testing Methodology

About Working with Business Process Testing

Business Process Testing enables Subject Matter Experts to create tests using a keyword-driven methodology for testing as well as an improved automated testing environment.

Business Process Testing integrates QuickTest with Quality Center and can be enabled by purchasing a specific Business Process Testing license. To work with Business Process Testing from within QuickTest, you must connect to a Quality Center project with Business Process Testing support.

This section provides an overview of the Business Process Testing model. For more information, refer to the *Business Process Testing User's Guide* (included in the Quality Center documentation package) and the *QuickTest Professional for Business Process Testing User's Guide*.

Understanding Business Process Testing Roles

The Business Process Testing model is role-based, allowing non-technical Subject Matter Experts (working in Quality Center) to collaborate effectively with Automation Engineers (working in QuickTest Professional). Subject Matter Experts define and document business processes, business components, and business process tests, while Automation Engineers define the required resources and settings, such as shared object repositories, function libraries, and recovery scenarios. Together, they can build, data-drive, document, and run business process tests, without requiring programming knowledge on the part of the Subject Matter Expert.

Note: The role structure and the tasks performed by various roles in your organization may differ from those described here according to the methodology adopted by your organization. These roles are flexible and depend on the abilities and time resources of the personnel using Business Process Testing. For example, the tasks of the Subject Matter Expert and the Automation Engineer may be performed by the same person. There are no product-specific rules or limitations controlling which roles must be defined in a particular organization, or which types of users can do which Business Process Testing tasks (provided that the users have the correct permissions).

The following user roles are identified in the Business Process Testing model:

Subject Matter Expert—The Subject Matter Expert has specific knowledge of the application logic, a high-level understanding of the entire system, and a detailed understanding of the individual elements and tasks that are fundamental to the application being tested. This enables the Subject Matter Expert to determine the operating scenarios or business processes that must be tested and identify the key business activities that are common to multiple business processes.

Using the Business Components module in Quality Center, the Subject Matter Expert creates business components that describe the specific tasks that can be performed in the application, and the condition or state of the application before and after those tasks. The Subject Matter Expert then defines the individual steps for each business component comprising the business process in the form of manual, or non-automated steps.

During the design phase, the Subject Matter Expert works with the Automation Engineer to identify the resources and settings needed to automate the components, enabling the Automation Engineer to prepare them. When the resources and settings are ready, the Subject Matter Expert automates the manual steps by converting them to keyword-driven components. Part of this process entails choosing an application area for each component. The application area contains all of the required resource files and settings that are specific to a particular area of the application being tested. Associating each component with an application area enables the component to access these resources and settings.

Using the Quality Center Test Plan module, the Subject Matter Expert combines the business components into business process tests, composed of a serial flow of the components. For example, most applications require users to log in before they can access any of the application functionality. The Subject Matter Expert could create one business component that represents this login procedure. This component procedure can be used in many business process tests, resulting in easier and more cost-efficient maintenance, updating, and test management.

The Subject Matter Expert configures the values used for business process tests, runs them in test sets, and reviews the results. The Subject Matter Expert is also responsible for maintaining the testing steps for each of the individual business components.

While defining components, Subject Matter Experts continue collaborating with the Automation Engineer. For example, they may request new operations (functions) for a component or discuss future changes planned for the component.

Automation Engineer—The Automation Engineer is an expert in using an automated testing tool, such as QuickTest Professional. The Automation Engineer works with the Subject Matter Expert to identify the resources that are needed for the various business process tests.

The Automation Engineer then prepares the resources and settings required for testing the features associated with each specific component, and stores them in an application area within the same Quality Center project used by the Subject Matter Experts who create and run the business process tests for the specific application.

Each application area serves as a single entity in which to store all of the resources and settings required for a component, providing a single point of maintenance for all elements associated with the testing of a specific part of an application. Application areas generally include one or more shared object repositories, a list of keywords that are available for use with a component, function libraries containing automated functions (operations), recovery scenarios for failed steps, and other resources and settings that are needed for a component to run correctly. Components are linked to the resources and settings in the application area. Therefore, when changes are made in the application area, all associated components are automatically updated.

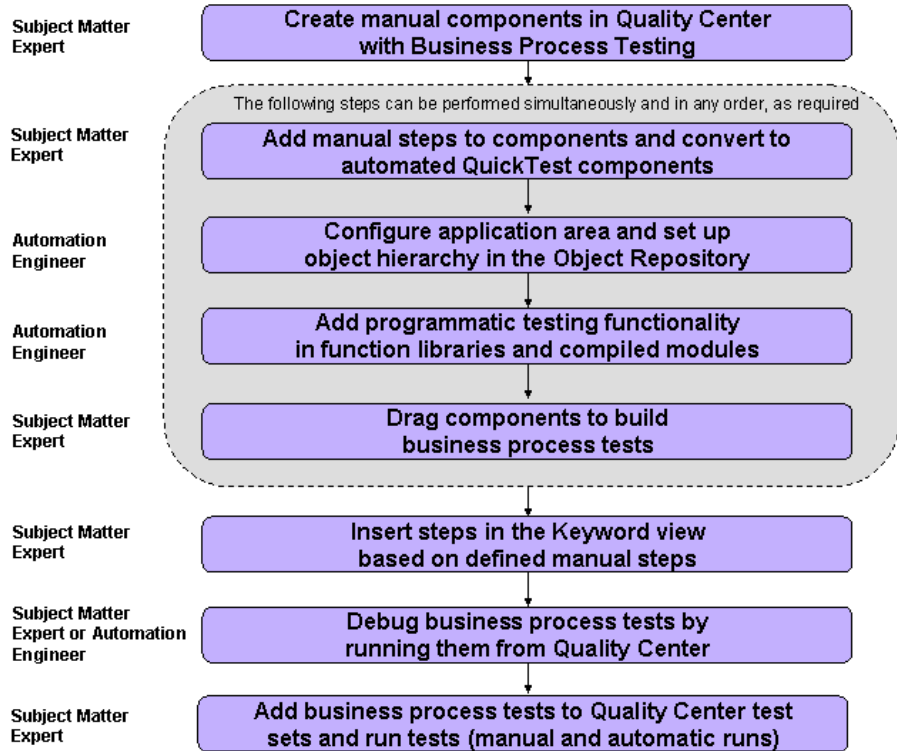
The Automation Engineer uses QuickTest's features and functionality to create these resources from within QuickTest. For example, in QuickTest, the Automation Engineer can create and populate various object repositories with test objects that represent the different objects in the application being tested, even before the application is fully developed. The Automation Engineer can then add repository parameters, and so forth, as needed. The Automation Engineer can manage the various object repositories using the Object Repository Manager, and merge repositories using the Object Repository Merge Tool. Automation Engineers can also use QuickTest's function library editor to create and debug function libraries containing functions that use programming logic to encapsulate the steps needed to perform a particular task.

Using the resources created by the Automation Engineer, the Subject Matter Experts can automate component steps, and create and maintain components and business process tests.

Automation Engineers can also create, debug, and modify components in QuickTest, if required.

Understanding the Business Process Testing Workflow

The Business Process Testing workflow may differ according to your testing needs. Following is an example of a common workflow:



Understanding Business Process Testing Methodology

Each scenario that the Subject Matter Expert creates is a **business process test**. A business process test is composed of a serial flow of **components**. Each component performs a specific task. A component can pass data to a subsequent component.

Understanding Components

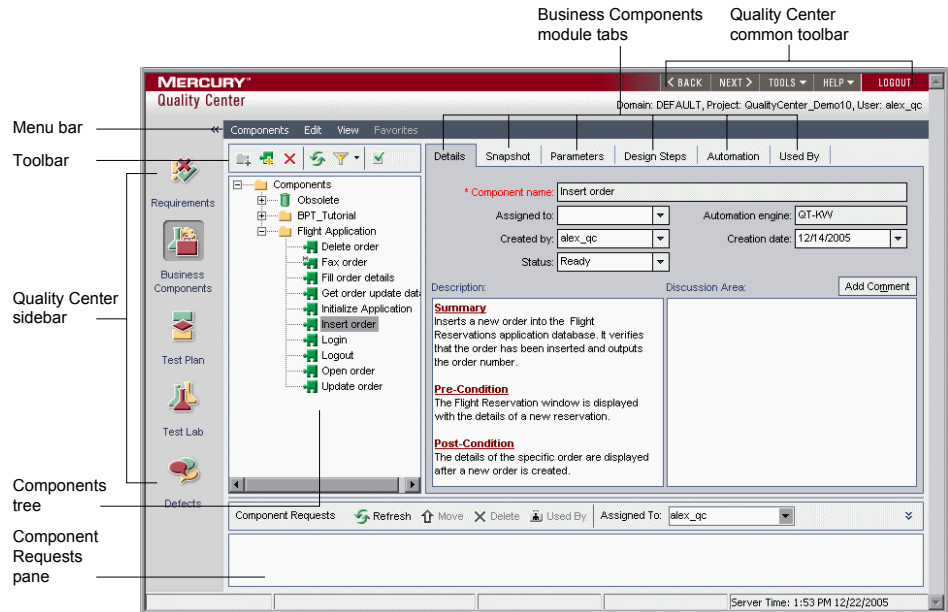
Components are easily-maintained reusable scripts that perform a specific task, and are the building blocks from which an effective business process testing structure can be produced. Components are parts of a business process that has been broken down into smaller parts. For example, in most applications users need to log in before they can do anything else. A Subject Matter Expert can create one component that represents the login procedure for an application. Each component can then be reused in different business process tests, resulting in easier maintenance, updating, and test management.

Components are comprised of steps. For example, the login component's first step may be to open the application. Its second step could be entering a user name. Its third step could be entering a password, and its fourth step could be clicking the **Enter** button.

You can create and edit components in QuickTest by adding steps on any supported environment, parameterizing selected items, and enhancing the component by incorporating functions (operations) that encapsulate the steps needed to perform a particular task. In Quality Center, a Subject Matter Expert creates components and combines them into business process tests, which are used to check that the application behaves as expected.

Creating Components in the Quality Center Business Components Module

The Subject Matter Expert can create a new component and define it in the Quality Center Business Components module.



The Business Component module includes the following elements:

- ▶ **Details**—Provides a general summary of the component’s purpose or goals, and the condition of the application before and after a component is run (its pre-conditions and post-conditions).
- ▶ **Snapshot**—Displays an image that provides a visual cue or description of the component’s purpose or operations.
- ▶ **Parameters**—Specifies the input and output component parameter values for the business component. Implementing and using parameters enables a component to receive data from an external source and to pass data to other components in the business process test flow.
- ▶ **Design Steps**—Enables you to create or view the manual steps of your business component, and to automate it if required.

- ▶ **Automation**—Displays or provides access to automated components. For keyword-driven components, enables you to create and modify the steps of your automated business component in a keyword-driven, table format, and provides a plain-language textual description of each step of the implemented component.
- ▶ **Used by**—Provides details about the business process tests that include the currently selected business component. The tab also includes a link to the relevant business process test in the Test Plan module.
- ▶ **Component Requests pane**—Enables you to handle new component requests that were generated in the Test Plan module. Component requests are requests to add a new business component to the project.

Implementing Components in QuickTest Professional

Generally, components are created by Subject Matter Experts in Quality Center, although they can also be created and debugged in QuickTest.

In QuickTest, you create components by recording steps on any supported environment or by adding steps manually (if the object repository is populated and the required operations are available). You can parameterize selected items. You can also view and set options specific to components.

QuickTest enables you to create and modify two types of components: **business components** and **scripted components**. A business component is an easily-maintained, reusable unit comprising one or more steps that perform a specific task. A scripted component is an automated component that can contain programming logic. Scripted components share functionality with both test actions and business components. For example, you can use the Keyword View, the Expert View, and other QuickTest tools and options to create, view, modify, and debug scripted components in QuickTest. Due to their complexity, scripted components can be edited only in QuickTest.

In Quality Center, the Subject Matter Expert can open components created in QuickTest. The Subject Matter Expert can then view and edit business components, but can only view the details for scripted components.

Creating Business Process Tests in the Quality Center Test Plan Module

To create a business process test, the Subject Matter Expert selects (drags and drops) the components that apply to the business process test and configures their run settings.

Each component can be used differently by different business process tests. For example, in each test the component can be configured to use different input parameter values or run a different number of iterations.

If, while creating a business process test, the Subject Matter Expert realizes that a component has not been defined for an element that is necessary for the business process test, the Subject Matter Expert can submit a component request from the Test Plan module.

Running Business Process Tests and Analyzing the Results

You can use the run and debug options in QuickTest to run and debug an individual component.

You can debug a business process test by running the test from the Test Plan module in Quality Center. When you choose to run from this module, you can choose which components to run in debug mode. (This pauses the run at the beginning of a component.)

When the business process test has been debugged and is ready for regular test runs, the Subject Matter Expert runs it from the Test Lab module similar to the way any other test is run in Quality Center. Before running the test, the Subject Matter Expert can define run-time parameter values and iterations using the **Iterations** column in the Test Lab module grid.

From the Test Lab module, you can view the results of the entire business process test run. The results include the value of each parameter, and the results of individual steps reported by QuickTest.

You can click the **Open Report** link to open the complete QuickTest report. The hierarchical report contains all the different iterations and components within the business process test run.

Understanding the Differences Between Components and Tests

If you are already familiar with using QuickTest to create action-based tests, you will find that the procedures for creating and editing components are quite similar. However, due to the design and purpose of the component model, there are certain differences in the way you create, edit, and run components. The guidelines below provide an overview of these differences.

- ▶ A component is a single entity. It cannot contain multiple actions or have calls to other actions or to other components.
- ▶ When working with components, all external files are stored in the Quality Center project to which you are currently connected.
- ▶ The name of the component node in the Keyword View is the same as the saved component. You cannot rename the node.
- ▶ Business components are created in the Keyword View, not the Expert View.
- ▶ You add resources via the component's application area, and not directly to the component.
- ▶ Components use custom keywords created in function libraries to perform operations, such as verifying property values and opening the application you are testing.

16

Working with Mercury Performance Testing and Business Availability Center Products

After you use QuickTest to create and run a suite of tests that test the functional capabilities of your application, you may want to test how much load your application can handle or to monitor your application as it runs.

Mercury LoadRunner tests the performance of applications under controlled and peak load conditions. To generate load, LoadRunner runs hundreds or thousands of virtual users. These virtual users provide consistent, repeatable, and measurable load to exercise your application just as real users would.

Mercury Business Availability Center enables real-time monitoring of the end user experience. Business Process Monitor runs virtual users to perform typical activities on the monitored application.

If you have already created and perfected a test in QuickTest that is a good representation of your users' actions, you may be able to use your QuickTest test as the basis for performance testing and application management activities. You can use Silent Test Runner to check in advance that a QuickTest test will run correctly from LoadRunner and Business Process Monitor.

This chapter describes:

- ▶ About Working with Mercury Performance Testing and Business Availability Center Products
- ▶ Using QuickTest Performance Testing and Business Availability Center Features

- ▶ Designing QuickTest Tests for Use with LoadRunner or Business Process Monitor
- ▶ Inserting and Running Tests in LoadRunner or Business Process Monitor
- ▶ Measuring Transactions
- ▶ Using Silent Test Runner

About Working with Mercury Performance Testing and Business Availability Center Products

QuickTest enables you to create complex tests that examine the full spectrum of your application's functionality to confirm that every element of your application works as expected in all situations.

The run mechanisms used in all Mercury Performance Testing and Mercury Business Availability Center products are the same. This means that you can create tests that are compatible with LoadRunner and Business Process Monitor, enabling you to take advantage of tests or test segments that have already been designed and debugged in QuickTest.

For example, you can add QuickTest tests to specific points in a LoadRunner scenario to confirm that the application's functionality is not affected by the extra load at those sensitive points. You can also run QuickTest tests on Business Process Monitor to simulate end user experience and ensure that your application is running correctly and in a timely manner.

QuickTest also offers several features that are designed specifically for integration with LoadRunner and Business Process Monitor. However, since LoadRunner and Business Process Monitor are designed to run tests using virtual users representing many users simultaneously performing standard user operations, some QuickTest features may not be available when integrating these products with QuickTest.

If you do plan to use a single test in both QuickTest and LoadRunner and/or Business Process Monitor, you should take into account the different options supported in each product as you design your test. For more information, see “Designing QuickTest Tests for Use with LoadRunner or Business Process Monitor” on page 416 and “Inserting and Running Tests in LoadRunner or Business Process Monitor” on page 417.

Using QuickTest Performance Testing and Business Availability Center Features

You can use the **Services** object and its associated methods to insert statements that are specifically relevant to Performance Testing and Business Availability Center. These include **AddWastedTime**, **EndDistributedTransaction**, **EndTransaction**, **GetEnvironmentAttribute**, **LogMessage**, **Rendezvous**, **SetTransaction**, **SetTransactionStatus**, **StartDistributedTransaction**, **StartTransaction**, **ThinkTime**, and **UserDataPoint**. For more information on these methods, refer to the **Services** section of the *QuickTest Professional Object Model Reference* and your LoadRunner or Business Availability Center documentation.



You can also insert **StartTransaction** and **EndTransaction** statements using the **Insert > Start Transaction** and **Insert > End Transaction** menu options or toolbar buttons to insert the statement. For more information on these options, see “Measuring Transactions” on page 419.

Note: LoadRunner and Business Process Monitor use only the data that is included within a transaction, and ignore any data in a test outside of a transaction.

Designing QuickTest Tests for Use with LoadRunner or Business Process Monitor

The QuickTest tests you use with LoadRunner or Business Process Monitor should be simple, designed to pinpoint specific operations, and should avoid using external actions and references to other external files (including resources stored in Quality Center). Also, when working with action iterations, corresponding **StartTransaction** and **EndTransaction** statements must be contained within the same action.

Designing Tests for LoadRunner

Consider the following guidelines when designing tests for use with LoadRunner:

- ▶ Do not include references to external actions or other external resources (including resources stored in Quality Center), such as an external Data Table file, environment variable file, shared object repositories, function libraries, and so forth. This is because LoadRunner may not have access to the external action or resource. (However, if the resource can be found on the network, QuickTest will use it.)
- ▶ Every QuickTest test must contain at least one transaction to provide useful information in LoadRunner.
- ▶ Make sure that the last step(s) in the test closes the application being tested, as well as any child processes that are running. This enables the next iteration of the test to open the application again.

Designing Tests for Business Process Monitor

Consider the following guidelines when designing tests for use with Business Process Monitor:

- ▶ Every QuickTest test must contain at least one transaction to provide useful information in Business Process Monitor.
- ▶ When measuring a distributed transaction over two different Business Process Monitor profiles, the profile with the **StartDistributedTransaction** statement must be run before the profile with the associated **EndDistributedTransaction**.

- ▶ When measuring distributed transactions, make sure that you relate the tests to a single Business Process Monitor instance. Business Process Monitor searches for the end transaction name in all instances, and may close the wrong distributed transaction if it is included in more than one instance.
- ▶ When measuring a distributed transaction over two Business Process Monitor profiles, make sure that the timeout value you specify is large enough so that the profile that contains the **StartDistributedTransaction** step and all the profiles that run before the profile that contains the **EndDistributedTransaction** step, will finish running in a time that is less than the value of the specified timeout.
- ▶ Business Process Monitor does not support running QuickTest Professional tests that require access to external resources, including resources stored in Quality Center (such as a shared object repository, function library, external Data Table, external actions, and so forth). Tests that require external resources may fail to run on Business Process Monitor. (However, if the resource can be found on the network, QuickTest will use it.)
- ▶ Make sure that the last step(s) in the test closes the application being tested, as well as any child processes that are running. This cleanup step enables the next test run to open the application again.

Inserting and Running Tests in LoadRunner or Business Process Monitor

Before you insert and run your QuickTest test in LoadRunner or Business Process Monitor, you should consider the guidelines below.

Note: You can simulate how the test will run from LoadRunner or Business Process Monitor by using Silent Test Runner. For more information, see “Using Silent Test Runner” on page 423.

Inserting and Running Tests in a LoadRunner Scenario

- ▶ You can run only one GUI Vuser concurrently per computer. (A GUI Vuser is a Vuser that runs a QuickTest test.)
- ▶ To insert a QuickTest test in a LoadRunner scenario, in the Controller Open Test dialog box, browse to the test folder and select **Astra Tests** in the **Files of type** box. This enables you to view QuickTest tests in the folder.
- ▶ Ensure that QuickTest is closed on the QuickTest computer before running a QuickTest test in LoadRunner.
- ▶ Transaction breakdown is not supported for tests (scripts) recorded with QuickTest.
- ▶ QuickTest cannot run on a computer that is:
 - ▶ logged off or locked. In these cases, consider running QuickTest on a terminal server.
 - ▶ already running a QuickTest test. Make sure that the test is finished before starting to run another QuickTest test.
- ▶ The settings in the LoadRunner Run-time Settings dialog box are not relevant for QuickTest tests.
- ▶ You cannot use the **ResultDir** QuickTest environment variable when running a test in LoadRunner.

For more information on working with LoadRunner, refer to your LoadRunner documentation.

Inserting and Running Tests from Business Process Monitor

- ▶ Before you try to run a QuickTest test in Business Process Monitor, check which versions of QuickTest are supported by your version of Business Process Monitor. For more information, refer to the Business Process Monitor documentation.
- ▶ Business Process Monitor can run only one QuickTest test at a time. Make sure that the previous QuickTest test is finished before starting to run another QuickTest test.
- ▶ Ensure that QuickTest is closed on the QuickTest computer before running a QuickTest test in Business Process Monitor.

- ▶ Transaction breakdown is not supported for tests recorded with QuickTest.
- ▶ If you make changes to your local copy of a QuickTest test after uploading it to Business Availability Center, you will need to upload the zipped test again to enable Business Process Monitor to run the test with your changes.
- ▶ QuickTest cannot run tests on a computer that is logged off, locked, or running QuickTest as a non-interactive service.
- ▶ You cannot use the **ResultDir** QuickTest environment variable when running a test in Business Process Monitor.

For more information on working with Business Availability Center, refer to your Mercury Business Availability Center documentation.

Measuring Transactions

You can measure how long it takes to run a section of your test by defining *transactions*. A transaction represents the process in your application that you are interested in measuring. Your test must include transactions to be used by LoadRunner or the Business Process Monitor. LoadRunner and the Business Process Monitor use only the data that is included within a transaction, and ignore any data in a test outside of a transaction.

You define transactions within your test by enclosing the appropriate sections of the test with *start* and *end* transaction statements. For example, you can define a transaction that measures how long it takes to reserve a seat on a flight and for the confirmation to be displayed on the client's terminal.

During the test run, the **StartTransaction** step signals the beginning of the time measurement. The time measurement continues until the **EndTransaction** step is reached. The test report displays the time it took to perform the transaction.

For information on the statements you can use in transactions, refer to the *QuickTest Professional Object Model Reference*.

There is no limit to the number of transactions that can be added to a test. You can also insert a transaction within a transaction.

Part of a sample test with a transaction is shown below, as it is displayed in the Keyword View:

Start transaction	Services	StartTransaction	"ReserveSeat"	Start the "ReserveSeat" transaction.
	Find a Flight: Mercury			
	fromPort	Select	"London"	Select the "London" item in the "fromPort" list.
	toPort	Select	"Frankfurt"	Select the "Frankfurt" item in the "toPort" list.
	toDay	Select	"12"	Select the "12" item in the "toDay" list.
	servClass	Select	"Business"	Select radio button "Business" in the "servClass" radio button group.
	airline	Select	"Blue Skies Airlines"	Select the "Blue Skies Airlines" item in the "airline" list.
	findFlights	Click	65,12	Click the "findFlights" image.
	Select a Flight: Mercury...			
	outFlight	Select	"Blue Skies Airlines"	Select radio button "Blue Skies Airlines" in the "outFlight" radio button group.
inFlight	Select	"Blue Skies Airlines"	Select radio button "Blue Skies Airlines" in the "inFlight" radio button group.	
reserveFlights	Click	46,8	Click the "reserveFlights" image.	
End transaction	Services	EndTransaction	"ReserveSeat"	End the "ReserveSeat" transaction.
	Book a Flight: Mercury_2			

The same part of the test is displayed in the Expert View as follows:

```

Services.StartTransaction "ReserveSeat"
Browser("Welcome: Mercury Tours").Page("Find a Flight: Mercury").
  WebList("fromPort").Select "London"
Browser("Welcome: Mercury Tours").Page("Find a Flight: Mercury").
  WebList("toPort").Select "Frankfurt"
Browser("Welcome: Mercury Tours").Page("Find a Flight: Mercury").
  WebList("toDay").Select "12"
Browser("Welcome: Mercury Tours").Page("Find a Flight: Mercury").
  WebRadioGroup("servClass").Select "Business"
Browser("Welcome: Mercury Tours").Page("Find a Flight: Mercury").
  WebList("airline").Select "Blue Skies Airlines"
Browser("Welcome: Mercury Tours").Page("Find a Flight: Mercury").
  Image("findFlights").Click 65,12
Browser("Welcome: Mercury Tours").Page("Select a Flight: Mercury").
  WebRadioGroup("outFlight").Select "Blue Skies Airlines"
Browser("Welcome: Mercury Tours").Page("Select a Flight: Mercury").
  WebRadioGroup("inFlight").Select "Blue Skies Airlines"
Browser("Welcome: Mercury Tours").Page("Select a Flight: Mercury").
  Image("reserveFlights").Click 46,8
Services.EndTransaction "ReserveSeat"
  
```

You can insert a variety of transaction-related statements using the Step Generator or Expert View. For more information, refer to the **Services** section of the *QuickTest Professional Object Model Reference*. You can also enter Start Transaction and End Transaction steps using options in the QuickTest window.

For more information, see:

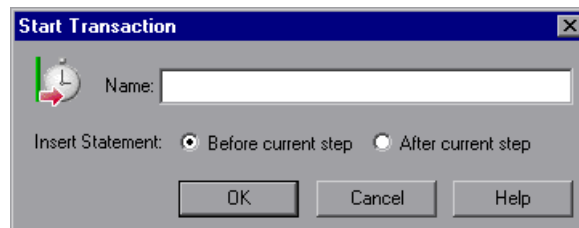
- ▶ “Inserting Transactions,” below
- ▶ “Ending Transactions” on page 422

Inserting Transactions

During the test run, the Start Transaction signals the beginning of the time measurement. You define the beginning of a transaction in the Start Transaction dialog box.

To insert a transaction:

- 1 Click the step where you want the transaction timing to begin. The page is displayed in the Active Screen tab.
- 2 Click the **Start Transaction** button or choose **Insert > Start Transaction**. The **Start Transaction** dialog box opens.



- 3 Enter a meaningful name in the **Name** box.

Note: You cannot include spaces in a transaction name.

- 4 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**.
- 5 Click **OK**. A **Start Transaction** step is added to the Keyword View.

Ending Transactions

During the test run, the End Transaction signals the end of the time measurement. You define the end of a transaction in the End Transaction dialog box.

Note: There may be cases in which you want to instruct QuickTest to perform all the steps in a transaction, even though an error occurs during the run session. In the Run tab of the Test Settings dialog box (**File > Settings**), select **proceed to next step** from the **When error occurs during run session** list.

To end a transaction:

- 1 Click the step where you want the transaction timing to end. The page opens in the Active Screen.
- 2 Click the **End Transaction** button or choose **Insert > End Transaction**. The **End Transaction** dialog box opens.



- 3 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.

- 4 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**.
- 5 Click **OK**. An **End Transaction** step is added to the Keyword View.

Using Silent Test Runner

Silent Test Runner enables you to simulate the way a QuickTest test runs from LoadRunner and Business Availability Center. When you run a test using Silent Test Runner, it runs without opening the QuickTest user interface, and the test runs at the same speed as when it is run from LoadRunner or Business Availability Center. At the end of the test run, you can view information about the test run and transaction times.

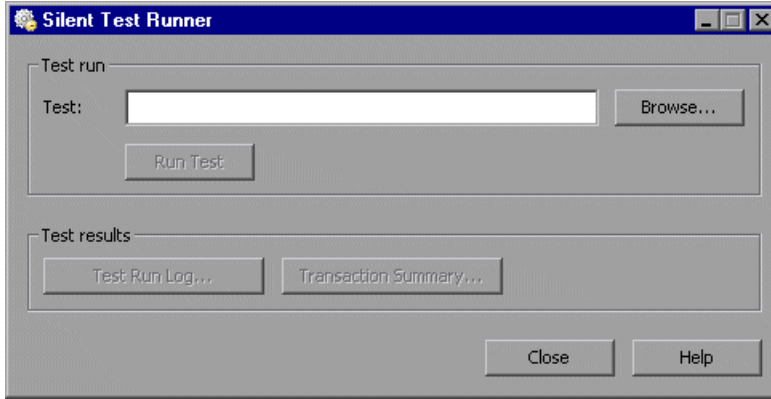
You can also use Silent Test Runner to verify that your QuickTest test is compatible with LoadRunner and Business Availability Center. A test will fail when run using Silent Test Runner if it uses a feature that is not supported by LoadRunner or Business Availability Center. For more information on features that are not supported, see “Designing QuickTest Tests for Use with LoadRunner or Business Process Monitor” on page 416, and “Inserting and Running Tests in LoadRunner or Business Process Monitor” on page 417.

Note: You cannot run Silent Test Runner if QuickTest is already open or another test is currently running. You must close QuickTest and wait for its process to end before running your test using Silent Test Runner.

You cannot use the **ResultDir** QuickTest environment variable when running a test from Silent Test Runner.

To run a QuickTest test using Silent Test Runner:

- 1 To open Silent Test Runner, choose **Start > Programs > QuickTest Professional > Tools > Silent Test Runner**. The Silent Test Runner dialog box opens.



- 2 Click the **Browse** button to navigate to your test. The Open Test dialog box opens and displays the tests located in your <QuickTest Professional>\Tests folder.
- 3 Select the test you want to run and click **Open**. The Open Test dialog box closes, the test name appears in the **Test** box of the Silent Test Runner dialog box, and the **Run Test** button is enabled.

Note: If you select a test that you ran previously, the **Test Run Log** and **Transaction Summary** buttons are enabled and you can display information about the last run of the selected test. The first time you run a test, the **Test Run Log** and **Transaction Summary** buttons are disabled.

- 4 Click **Run Test** to run your test. The test runs without opening the QuickTest user interface. The text **Running test...** appears next to the **Run Test** button while the test is running.

Note: After you start a test run, you cannot stop the test run from Silent Test Runner. If you close Silent Test Runner, the test continues to run. You can end a test by ending the **mdrv.exe** process.

- 5 When the test run finishes, the text **Running test...** is replaced with the text **Test run completed**. If Silent Test Runner was unable to run your test, the text **Test could not be run** appears. If previously disabled, the **Test Run Log** button is enabled. The **Transaction Summary** button is also enabled if you ran a test with transactions and the button was previously disabled. For more information about viewing the log files, see ““Viewing Test Run Information,” below.

Viewing Test Run Information

Silent Test Runner provides test run information in log files. Each test generates a test run log, and any test with transactions generates an additional transaction summary.

Viewing the Test Run Log

The test run log is saved as **output.txt** in the **<QuickTest Professional>\Tests\<test name>** folder. A log file is saved for each test run with Silent Test Runner and is overwritten when you rerun the test. To open the log file, click **Test Run Log**.

The log file displays information about the test run. For example, information is shown about each iteration, action call, step transaction, failed step, and so forth. Each line displays a message or error ID. For more information on message and error codes in the log file, refer to your Performance Center or Business Availability Center documentation.

Viewing the Transaction Summary

The transaction summary is saved as **transactions.txt** in the <QuickTest Professional>\Tests**<test name>** folder. A transaction summary is saved for each test that includes transactions and is overwritten when you rerun the test. To open the log file, click **Transaction Summary**. The transaction summary displays a line for each transaction in the test. For each transaction, the status is displayed together with the total duration time and any wasted time (in seconds). The transaction measurements in Silent Test Runner are exactly the same as if the test was run from LoadRunner or Business Availability Center.

Notes:

A transaction summary is available only for a test that contains transactions ending with an **EndTransaction** statement. If a transaction started but did not end because of test failure, it is not included in the transaction summary.

Distributed transactions (transactions that start in one test and end in another) are not reported in the transaction summary but are included in the test run log.

Any transaction information included in the transaction summary is also included in the test run log.

Part V

Appendix

A

Working with QuickTest—Frequently Asked Questions

This chapter answers some of the questions that are asked most frequently by **advanced users** of QuickTest. The questions and answers are divided into the following sections:

- ▶ Recording and Running Tests
- ▶ Programming in the Expert View
- ▶ Working with Dynamic Content
- ▶ Advanced Web Issues
- ▶ Test Maintenance
- ▶ Testing Localized Applications
- ▶ Improving QuickTest Performance

Recording and Running Tests

► **How does QuickTest capture user processes in Web pages?**

QuickTest hooks the Microsoft Internet Explorer browser. As the user navigates the Web-based application, QuickTest records the user actions. (For information about modifying which user actions are recorded, see Chapter 10, “Configuring Web Event Recording.”) QuickTest can then run the test by running the steps as they originally occurred.

► **How can I record on objects or environments not supported by QuickTest?**

You can do this in a number of ways:

- By default, QuickTest supports several developmental environments. You can also enable support for additional environments, such as Java, Oracle, .NET, SAP Solutions, Siebel, PeopleSoft, terminal emulators, and Web services, by installing and loading any of the external add-ins that are available for QuickTest Professional.
- You can map objects of an unidentified or custom class to standard Windows classes. For more information on object mapping, see “Mapping User-Defined Test Object Classes” on page 119.
- You can define **virtual objects** for objects that behave like test objects, and then record in the normal recording mode. For more information on defining virtual objects, see Chapter 2, “Learning Virtual Objects.”
- You can record your clicks and keyboard input based on coordinates in the *low-level recording* or *analog* modes. For more information on low-level and analog recording, refer to “Choosing the Recording Mode” on page 90 in the *QuickTest Professional Basic Features User’s Guide*.

Programming in the Expert View

► Can I store functions and subroutines in a function library?

You can define functions within an individual test, or you can create one or more VBScript function libraries containing your functions, and then call them from any test.

You can also register your functions as methods for QuickTest test objects. Your registered methods can override the functionality of an existing test object method for the duration of a run session, or you can register a new method for a test object class.

For more information, see Chapter 6, “Working with User-Defined Functions and Function Libraries.”

Working with Dynamic Content

► How can I record and run tests on objects that change dynamically from viewing to viewing?

Sometimes the content of objects in a Web page or application changes due to dynamic content. You can create dynamic descriptions of these objects so that QuickTest will recognize them when it runs the test. For more information, refer to Chapter 6, “Working with Test Objects” in the *QuickTest Professional Basic Features User’s Guide*.

► How can I check that a child window exists (or does not exist)?

Sometimes a link in one window creates another window.

You can use the **Exist** property to check whether or not a window exists. For example:

```
Browser("Window_name").Exist
```

You can also use the **ChildObjects** method to retrieve all child objects (or the subset of child objects that match a certain description) on the Desktop or within any other parent object.

For more information on the **Exist** property and **ChildObjects** method, refer to the *QuickTest Professional Object Model Reference*.

► **How does QuickTest record on dynamically generated URLs and Web pages?**

QuickTest actually clicks links as they are displayed on the page. Therefore, QuickTest records how to find a particular object, such as a link on the page, rather than the object itself. For example, if the link to a dynamically generated URL is an image, then QuickTest records the “IMG” HTML tag, and the name of the image. This enables QuickTest to find this image in the future and click on it.

Advanced Web Issues

► **How does QuickTest handle cookies?**

Server side connections, such as CGI scripts, can use cookies both to store and retrieve information on the client side of the connection.

QuickTest stores cookies in the memory for each user, and the browser handles them as it normally would.

► **How does QuickTest handle session IDs?**

The server, not the browser, handles session IDs, usually by a cookie or by embedding the session ID in all links. This does not affect QuickTest.

► **How does QuickTest handle server redirections?**

When the server redirects the client, the client generally does not notice the redirection, and misdirections generally do not occur. In most cases, the client is redirected to another script on the server. This additional script produces the HTML code for the subsequent page to be viewed. This has no effect on QuickTest or the browser.

► **How does QuickTest handle meta tags?**

Meta tags do not affect how the page is displayed. Generally, they contain information only about who created the page, how often it is updated, what the page is about, and which keywords represent the page's content. Therefore, QuickTest has no problem handling meta tags.

► **Does QuickTest work with .asp?**

Dynamically created Web pages utilizing Active Server Page technology have an .asp extension. This technology is completely server-side and has no bearing on QuickTest.

► **Does QuickTest work with COM?**

QuickTest complies with the COM standard.

QuickTest supports COM objects embedded in Web pages (which are currently accessible only using Microsoft Internet Explorer) and you can drive COM objects in VBScript.

► **Does QuickTest work with XML?**

XML is eXtensible Markup Language, a pared-down version of SGML for Web documents, that enables Web designers to create their own customized tags. QuickTest supports XML and recognizes XML tags as objects.

You can also create XML checkpoints to check the content of XML documents in Web pages, frames or files. QuickTest also supports XML output and schema validation.

For more information, see Chapter 13, “Checking XML” in the *QuickTest Professional Basic Features User's Guide*.

Test Maintenance

► **How do I maintain my test when my application changes?**

The way to maintain a test when your application changes depends on how much your application changes. This is one of the main reasons you should create a small group of tests rather than one large test for your entire application. When your application changes, you can re-record part of a test. If the change is not significant, you can manually edit a test to update it.

You can also use QuickTest's action feature to design more modular and efficient tests. While recording, you divide your test into several actions, based on functionality. When your application changes, you can rerecord a specific action, without changing the rest of the test. Whenever possible, insert calls to reusable actions rather than creating identical pieces of script in several tests. This way, changes to your original reusable action are automatically applied to all tests calling that action. For more information, see Chapter 1, "Working with Advanced Action Features."

If you have many tests and actions that contain the same test objects, it is recommended to work with shared object repositories so that you can update object information in a centralized location.

To update the information in your checkpoints, the Active Screen, or about your test object properties when object properties change, or to add new objects or steps on an Active Screen image without rerecording steps, use the **Update Run Mode** option. For more information, refer to "Updating a Test" on page 604 in the *QuickTest Professional Basic Features User's Guide*.

► **Can I increase or decrease Active Screen information after I finish recording a test?**

If you find that the information saved in the Active Screen after recording is not sufficient for your test editing needs, or if you no longer need Active Screen information, and you want to decrease the size of your test, there are several methods of changing the amount of Active Screen information saved with your test.

- ▶ To decrease the disk space used by your test, you can delete Active Screen information by selecting **Save As**, and clearing the **Save Active Screen files** check box. For more information, refer to “Saving a Test” on page 100 in the *QuickTest Professional Basic Features User’s Guide*.
- ▶ If you chose not to save all information in the Active Screen when testing a Windows application, you can use one of several methods to increase the information stored in the Active Screen.

Confirm that the Active Screen capture preference in the Active Screen tab of the Options dialog box is set to capture the amount of information you need and then:

- Perform an **Update Run Mode** operation to save the required amount of information in the Active Screen for all existing steps. For more information on the **Update Run Mode** options, refer to “Updating a Test” on page 604 in the *QuickTest Professional Basic Features User’s Guide*.
- Re-record the step(s) containing the object(s) you want to add to the Active Screen.

To re-record the step, select the step *after* which you want to record your step, position your application to match the selected location in your test, and then begin recording. Alternatively, place a breakpoint in your test at the step *before* which you want to add a step and run your test to the breakpoint. This will bring your application to the correct place in order to record the step. For more information on setting breakpoints, refer to “Setting Breakpoints” on page 585 in the *QuickTest Professional Basic Features User’s Guide*.

For more information on changing the amount of information saved in the Active Screen for Windows applications, refer to “Setting Active Screen Options,” on page 701 in the *QuickTest Professional Basic Features User’s Guide*.

Testing Localized Applications

- ▶ **I am testing localized versions of a single application, each with localized user interface strings. How do I create efficient tests in QuickTest?**

You can parameterize these user interface strings using parameters from the global Environment variable list. This is a list of variables and corresponding values that can be accessed from any test. For more information, refer to Chapter 15, “Parameterizing Values” in the *QuickTest Professional Basic Features User’s Guide*.

- ▶ **I am testing localized versions of a single application. How can I efficiently input different data in my tests, depending on the language of the application?**

If you are running a single iteration of your test, or if you want values to remain constant for all iterations of an action or test, use environment variables, and then change the active environment variable file for each test run.

If you are running multiple iterations of your test or action, and you want the input data to change in each iteration, you can create an external Data Table for each localized version of your application. When you change the localized version of the application you are testing, you simply switch the Data Table file for your test in the Resources tab of the Test Settings dialog box.

For more information on working with Data Tables, refer to Chapter 19, “Working with Data Tables” in the *QuickTest Professional Basic Features User’s Guide*. For more information on selecting the Data Table file for your test, refer to “Defining Resource Settings for Your Test” on page 751 in the *QuickTest Professional Basic Features User’s Guide*.

Improving QuickTest Performance

► How can I improve the working speed of QuickTest?

You can improve the working speed of QuickTest by doing any of the following:

- Do not load unnecessary add-ins in the Add-in Manager when QuickTest starts. This will improve both recording time and run session performance. For more information about loading add-ins, refer to “Loading QuickTest Add-ins” on page 795 in the *QuickTest Professional Basic Features User's Guide*.
- Run your tests in "fast mode." From the Run tab in the Options dialog box, select the **Fast** option. This instructs QuickTest to run your test without displaying the execution arrow for each step, enabling the test to run faster. For more information on the Run tab of the Options dialog box, refer to “Setting Run Testing Options” on page 709 in the *QuickTest Professional Basic Features User's Guide*.
- If you are not using the Active Screen while editing your test, hide the Active Screen while editing your test to improve editing response time. Choose **View > Active Screen**, or toggle the Active Screen toolbar button to hide the Active Screen. For more information, refer to Chapter 2, “QuickTest at a Glance” in the *QuickTest Professional Basic Features User's Guide*.
- Decide if and how much information you want to capture and save in the Active Screen. The more information you capture, the easier it is to add steps to your test using the many Active Screen options, but more captured information also leads to slower recording and editing times. You can choose from the following Active Screen options to improve performance:
 - If you are testing Windows applications, you can choose to save all Active Screen information in every step, save information only in certain steps, or to disable Active Screen captures entirely. You set this preference in the Active Screen tab of the Options dialog box. For more information, refer to “Setting Active Screen Options” on page 701 in the *QuickTest Professional Basic Features User's Guide*.

- If you are testing Web applications, you can disable screen capture of all steps in the Active Screen. From the Active Screen tab of the Options dialog box, click **Custom Level** to open the Custom Active Screen Capture Settings dialog box.

Select the **Disable Active Screen Capture** option. This will improve recording time. For more information on the Active Screen tab of the Options dialog box, refer to “Setting Active Screen Options” on page 701 in the *QuickTest Professional Basic Features User’s Guide*.

- When you save a new test, or when you save a test with a new name using **Save As**, you can choose not to save the captured Active Screen files with the test by clearing the **Save Active Screen files** option in the Save or Save As dialog box. This is especially useful when you have finished designing your test and you plan to use your test only for test runs. Tests without Active Screen files open more quickly and use significantly less disk space.
- Decide when you want to capture and save images of the application for the test results. From the Run tab in the Options dialog box, select an option from the **Save step screen capture to test results** box. You can improve test run time and reduce disk space by saving screen captures only in certain situations or by not saving the images at all. For more information on the Active Screen tab of the Options dialog box, refer to “Setting Active Screen Options” on page 701 in the *QuickTest Professional Basic Features User’s Guide*.

Tip: If you need to recover Active Screen files after you save a test without Active Screen files, re-record the necessary steps or use the **Update Run Mode** option to recapture screens for all steps in your test. For more information, refer to “Updating a Test” on page 604 in the *QuickTest Professional Basic Features User’s Guide*.

► **How can I decrease the disk space used by QuickTest?**

You can decrease the disk space used by QuickTest by doing any of the following:

- Decide when you want to capture and save images of the application for the test results. From the Run tab in the Options dialog box, select an option from the **Save step screen capture to test results** box. You can reduce disk space and improve test run time by saving screen captures only in certain situations or not saving images at all. For more information on the Active Screen tab of the Options dialog box, refer to “Setting Active Screen Options” on page 701 in the *QuickTest Professional Basic Features User’s Guide*.
- Decide if and how much information you want to capture and save in the Active Screen. The more information you capture, the easier it is to add steps to your test using the many Active Screen options, but more captured information also leads to slower recording and editing times. You can choose from the following Active Screen options to improve performance:
 - If you are testing Windows applications, you can choose to Save all Active Screen information in every step, save information only in certain steps, or to disable Active Screen captures entirely. You set this preference in the Active Screen tab of the Options dialog box. For more information, refer to “Setting Active Screen Options” on page 701 in the *QuickTest Professional Basic Features User’s Guide*.
 - If you are testing Web applications, you can disable screen capture of all steps in the Active Screen. From the Active Screen tab, click **Custom Level** to open the Custom Active Screen Capture Settings dialog box. Select the **Disable Active Screen Capture** option. This will improve recording time. For more information on the Active Screen tab of the Options dialog box, refer to “Setting Active Screen Options” on page 701 in the *QuickTest Professional Basic Features User’s Guide*.

- When you save a new test, or when you save a test with a new name using Save As, you can choose not to save the captured Active Screen files with the test by clearing the **Save Active Screen files** option in the Save or Save As dialog box. This is especially useful when you have finished designing your test and you plan to use your test only for test runs. Tests without Active Screen files use significantly less disk space.

Tip: If you need to recover Active Screen files after you save a test without Active Screen files, re-record the necessary steps or use the **Update Run Mode** option to recapture screens for all steps in your test. For more information, refer to “Updating a Test” on page 604 in the *QuickTest Professional Basic Features User’s Guide*.

► **Is there a recommended length for tests?**

Although there is no formal limit regarding test length, it is recommended that you divide your tests into actions and that you use reusable actions in tests, whenever possible. An action should contain no more than a few hundreds steps and, ideally, no more than a few dozen. For more information, refer to Chapter 17, “Working with Actions” in the *QuickTest Professional Basic Features User’s Guide*.

Index

A

- action call
 - iterations 22
 - parameter values 23
 - properties 21
 - run properties 22
 - setting run properties 22
- action parameters 16
 - guidelines 19
- actions 3
 - basic syntax 29
 - diagram 4, 5
 - inserting
 - call to 9
 - copy of 6
 - existing 4
 - nesting 16
 - overview 4
 - sharing values 26
 - using Dictionary objects 28
 - syntax 29
 - syntax for parameters 29
 - syntax for storing return values 31
- Active Screen
 - increasing/decreasing saved information 434
- Active Server Page technology 433
- Add Repository Parameter dialog box 264
- Add/Remove dialog box, object
 - identification 98, 114
- adding tests to version control 390
- add-ins, associating with QuickTest test in Quality Center 379
- Agent, Remote 399
- Allow other Mercury tools to run tests 398
- API
 - using Windows 180

- Application crash trigger 54
- Application Management, integrating with QuickTest 413
- applications
 - closing 165
 - running 165
 - sample xvii
 - testing localized versions 436
- asp files 433
- assistive properties, configuring 96
- associated function libraries 200
 - modifying 203
- associating
 - current function library 202
 - add-ins with test created in Quality Center 382
- attribute property 178
- auto-expand VBScript syntax 337
- automation
 - Application object 237
 - definition 232
 - development environment 235
 - language 235
 - object model 231
 - type library 235
- Automation Engineer 405

B

- Basic event recording configuration level 313
- Basic Features User's Guide, QuickTest Professional xv
- behavior, DHTML 321
- bitmaps
 - checking. *See* QuickTest Professional Basic Features User's Guide
- bookmarks 137

Index

- bubbling 322
- business analyst
 - role in Business Process Testing 404
- Business Process Monitor, integrating with QuickTest 413
- Business Process Testing 403
 - roles 404
 - workflow 407
- Business Process Testing User's Guide, QuickTest Professional for xv
- business process tests 408
 - running 411

C

- calculations, in the Expert View and function libraries 169
- Call to WinRunner Function dialog box 356
- Call to WinRunner Test dialog box 352
- calling TSL functions
 - from QuickTest 356
- CGI scripts 432
- Check In command 390, 392
- Check Out command 390
- checking tests out of version control 390
- checkpoints
 - in Expert View 128
- Close application process operation 66
- Close method 165
- closing
 - object repositories 257
- collection, properties. *See* programmatic descriptions
- collections, of virtual objects 33
- color settings
 - Object Repository Merge Tool 284
- COM 433
- comments
 - in the Expert View and function libraries 167
- Completing the Recovery Scenario Wizard screen 77
- configuration levels
 - customizing 315
 - standard 313

- conflict resolution settings
 - Object Repository Merge Tool 285
- conflicts, resolving in merged object repository 300
- connecting QuickTest to Quality Center 363
- conventions. *See* typographical conventions
- converting
 - object repositories 254
- cookies 432
- copying function code 217
- creation time identifier. *See* ordinal identifier
- CreationTime property, understanding 105
- custom event-recording configuration 315
 - adding listening events 319
 - adding objects to the list 318
 - deleting objects from the list 319
 - procedure 315
 - specifying listening criteria 321
- custom objects, mapping 119
- custom web event configuration files
 - loading 329
 - saving 329
- Custom Web Event Recording Configuration dialog box 315, 326
- customer support, Web site xvii
- customizing function libraries 333
 - general options 335
 - highlighting elements 337
- customizing test scripts 333
 - general options 335
 - highlighting script elements 337

D

- data tables. *See* QuickTest Professional Basic Features User's Guide
- databases
 - checking. *See* QuickTest Professional Basic Features User's Guide
- debugging
 - function libraries 197
 - See also* QuickTest Professional Basic Features User's Guide
- decreasing Active Screen information 434
- default object identification settings 108
- defining arguments 208

- deleting
 - objects from list 319
 - repository parameters 267
- descriptive programming. *See* programmatic descriptions
- Dictionary object 28
- Dim statement, in the Expert View and function libraries 149
- disconnecting from Quality Center 370
- disk space, saving 437
- Do...Loop statement, in the Expert View and function libraries 171
- documentation
 - Basic Features User's Guide xv
 - Installation Guide xv
 - online xvii
 - Tutorial xv
 - updates xviii
- documenting a function 214
- DOS commands, run within tests 179
- dynamic Web content 431
- dynamically generated URLs and Web pages 432

E

- Editor Options dialog box 335
- End Transaction dialog box 422
- ending transactions 422
- errors in VBScript syntax 153
- event-recording configuration 311
 - customizing levels 315
 - resetting 331
 - standard levels 313
- ExecuteFile function 228
- ExecuteFile statement
 - using 201
- Exist property 431
- existing actions, inserting 4

- Expert View 123, 431
 - action syntax 29
 - basic action syntax 29
 - checkpoints 128
 - closing applications 165
 - customizing appearance of 333
 - finding text 140
 - replacing text 142
 - running applications 165
 - syntax for action parameters 29
 - syntax for action return values 31
 - understanding 125
 - understanding parameters 129
- exporting
 - object repository to XML file 273
- expressions, using in the Expert View and function libraries 145
- eXtensible Markup Language (XML) 433
- external functions, executing from script 228

F

- FAQs 429
- Filter dialog box
 - Object Repository Merge Tool 302
- filter properties (Smart Identification) 109
- filtering
 - target repository 302
- finalizing function code 217
- Find dialog box
 - Object Repository Merge Tool 304
- finding text in Expert View 140
- For...Each statement, in the Expert View and function libraries 171
- For...Next statement, in the Expert View and function libraries 170
- frequently asked questions 429
- function arguments, passing parameters
 - from QuickTest to WinRunner 359
- Function call operation 66

Index

Function Definition Generator 208
 defining a function 208
 documenting a function 214
 opening 206
 previewing function code 216
 registering a function 209
 using 204
function libraries 185
 associated 200
 associating current 202
 creating 189
 customizing appearance of 333
 debugging 197
 editing 195
 general options 335
 highlighting elements 337
 managing 188
 modifying associated 203
 navigating 194
 opening 191, 199
 read-only, editing 197
 saving 189
functions, user-defined 185

G

general options 335
Generate Script option 238
GetROProperty method 176
Go To dialog box 136
guidelines
 user-defined functions 225

H

handler 321
Help, online, from within QuickTest
 Professional xvii
High event recording configuration level 313
home page, Mercury xix

I

If...Then...Else statement, in Expert View and
 function libraries 173
importing
 object repository from XML file 272

increasing Active Screen information 434
index identifier. *See* ordinal identifier
Index property, programmatic
 descriptions 164
Index property, understanding 103
initialization scripts 233
inserting transactions 421
Installation Guide, QuickTest Professional xv
IntelliSense 130, 336
iterations 22

K

key assignments, in Expert View 339
key assignments, in function libraries 339
Keyboard or mouse operation 66
keyboard shortcuts
 in Expert View 339
 in function libraries 339
Keyword view. *See* QuickTest Professional
 Basic Features User's Guide
knowledge base xix

L

LoadRunner, integrating with QuickTest 413
local object repositories
 merging 290
localized applications, testing 436
location identifier. *See* ordinal identifier
Location property, understanding 104
low-level recording 430

M

Manage Repository Parameters dialog
 box 262
managing tests
 testing process 361
mandatory properties, configuring 96
mapping
 custom objects 119
measuring transactions 419
Medium event recording configuration
 level 313
Mercury Application Management,
 integrating with QuickTest 413

Mercury Best Practices xix
 Mercury Quality Center. *See* Quality Center
 Mercury Tours, sample application xix
 merging

- local object repositories 290
- shared object repositories 275

 meta tags 433
 methods

- adding new or changing behavior of 219
- run-time objects 177
- user-defined 219

 modifying

- object repositories 260
- repository parameters 266

N

Name and Description screen 76
 nesting actions 16
 New Merge dialog box 288

O

object identification

- generating automation scripts 108
- restoring defaults 108

 Object Identification dialog box 95
 Object Mapping dialog box 119
 object model

- automation 231
- definition 232

 Object property, run-time methods 178
 object repositories

- closing 257
- converting 254
- creating 253
- exporting to XML 273
- importing from XML 272
- managing 244
- modifying 260
- opening 254
- saving 255

Object Repository Manager 246
 Object Repository Merge Tool 275

- changing the view 278
- color settings 284
- conflict resolution settings 285
- conflicts 297
- filtering the target repository 302
- primary repository pane 280
- resolution options pane 280
- resolving conflicts 300
- secondary repository pane 280
- target repository pane 279
- window 277

 Object state trigger 54
 objects

- identification 93
- methods, run-time 177
- properties, run-time 177

 Open Test from Quality Center Project dialog box 376, 378
 opening tests

- in a Quality Center project 375

 Option Explicit statement 226
 Options dialog box

- Generate Script option 238

 ordinal identifier 102
 output values. *See* QuickTest Professional Basic Features User's Guide
 output.txt log file 425

P

parameter values for action calls 23
 parameterizing

- property values using repository parameters 269

 parameterizing values. *See* QuickTest Professional Basic Features User's Guide

Index

- parameters
 - action 16
 - action guidelines 19
 - adding repository 264
 - deleting repository 267
 - in the Expert View 129
 - managing repository 262
 - modifying repository 266
 - repository 261
 - syntax for calling action 29
 - passing parameters
 - to a WinRunner function 359
 - to a WinRunner test 354
 - performance testing products, integrating with QuickTest 413
 - performance, improving 437
 - Pop-up window trigger 54
 - post-recovery test run options 45
 - Post-Recovery Test Run Options screen 74
 - power users, advanced features for 429
 - previewing function code 216
 - primary repository 276
 - primary repository pane 280
 - printing
 - function library 198
 - priority
 - setting for recovery scenarios 88
 - programmatically descriptions 155
 - for description objects 160
 - for WebElement objects 163
 - in statement 157
 - using the With statement 159
 - using the Index property 164
 - using variables 157
 - programming 431
 - in Expert View and function libraries 123
 - in VBScript 146
 - programming logic. *See* QuickTest Professional Basic Features User's Guide
 - project (Quality Center)
 - connecting to 363
 - disconnecting from 370
 - opening tests in 375
 - saving tests to 374
 - properties
 - CreationTime 105
 - Index 103
 - Location 104
 - run-time objects 177
 - setting for action calls 21
 - viewing for recovery scenarios 81, 88
 - property collection. *See* programmatic descriptions
 - property values
 - specifying in the test object description 269
- ## Q
- QA engineer
 - role in Business Process Testing 404
 - QA Tester
 - role in Business Process Testing 404
 - Quality Center 361
 - associated function libraries 200
 - Connectivity Add-in 373
 - disconnecting from 370
 - opening tests in 375
 - project
 - connecting QuickTest to 363
 - running QuickTest tests remotely 398
 - version control for 389
 - Quality Center Connection dialog box 367
 - Quality Center Connection - Project Connection dialog box 365
 - Quality Center Connection - Server Connection dialog box 364
 - Quality Center project
 - saving tests to 374

- QuickTest
 - automation object model 231
 - integrating with Business Process Monitor 413
 - integrating with LoadRunner 413
 - integrating with Mercury Application Management 413
 - integrating with performance testing products 413
 - introduction. *See* QuickTest Professional Basic Features User's Guide
 - overview. *See* QuickTest Professional Basic Features User's Guide
- QuickTest Automation Object Model Reference 239

- R**
- Readme, QuickTest Professional xvii
- recording
 - low-level 430
 - right mouse button clicks 325
 - status, options 322
 - time, improving 437
- recovery
 - associating scenarios with tests 85
 - copying scenarios 83
 - deleting scenarios 82
 - disabling scenarios 89
 - files 48
 - modifying scenarios 82
 - operations 45
 - removing scenarios from tests 89
 - saving scenarios 78
 - scenarios 45
 - setting scenario priority 88
 - viewing scenario properties 81, 88
- recovery operation
 - Close application process 66
 - Function call 66
 - Keyboard or mouse operation 66
 - Restart Microsoft Windows 66
 - Recovery Operation - Click Button or Press Key screen 67
 - Recovery Operation - Close Processes screen 69
 - Recovery Operation - Function Call screen 71
 - Recovery Operation screen 66
 - Recovery Operations screen 64
 - Recovery Scenario Manager Dialog Box 48
 - Recovery Scenario Wizard 52
 - Click Button or Press Key screen 67
 - Close Processes screen 69
 - Completing the Recovery Scenario Wizard screen 77
 - Function screen 71
 - Name and Description screen 76
 - Post-Recovery Test Run Options screen 74
 - Recovery Operation screen 66
 - Recovery Operations screen 64
 - Select Object screen 58
 - Select Processes screen 62
 - Select Test Run Error screen 61
 - Select Trigger Event screen 54
 - Set Object Properties and Values screen 60
 - Specify Pop-up Window Conditions screen 56
 - redirection of server 432
 - registering functions 209
 - registering methods 219
 - using the RegisterUserFunc statement 222
 - RegisterUserFunc statement 209, 219
 - regular expressions
 - using in the Expert View and function libraries 145
 - See also* QuickTest Professional Basic Features User's Guide
 - remote access to QuickTest 398
 - Remote Agent 399
 - replacing text in Expert View 142
 - reports, filter 182
 - Repository Parameter dialog box 269

Index

- repository parameters 261
 - adding 264
 - deleting 267
 - managing 262
 - modifying 266
 - parameterizing values 269
- repository. *See also* object repository
- reserved objects 200
- resolution options pane 280
- resolving conflicts
 - Object Repository Merge Tool 300
- Restart Microsoft Windows operation 66
- right mouse button
 - configuring QuickTest to record 325
 - recording clicks 325
- roles in Business Process Testing 404
- run properties, setting for action calls 22
- running tests
 - advanced issues 430
 - from a Quality Center project 387
 - running WinRunner tests 352
 - using Silent Test Runner 424
- run-time
 - objects 177
 - settings, adding and removing 347
- S**
- sample application, Mercury Tours xix
- Save Shared Object Repository dialog box 306, 307
- Save Test to Quality Center Project dialog box 374
- saving
 - object repositories 255
 - recovery scenarios 78
 - target repository 305
- saving tests
 - to a Quality Center project 374
- scenarios
 - associating with tests 85
 - copying recovery 83
 - deleting recovery 82
 - disabling recovery 89
 - modifying recovery 82
 - recovery 45
 - removing recovery from tests 89
 - saving recovery 78
 - setting recovery priority 88
 - viewing recovery properties 81, 88
- scripts
 - general options 335
 - highlighting script elements 337
- scripts, test. *See* test scripts
- secondary repository 276
- secondary repository pane 280
- Select Action dialog box 7, 10
- Select Object screen 58
- Select Processes screen 62
- Select Test Run Error screen 61
- Select Trigger Event screen 54
- server
 - Quality Center, disconnecting from 370
 - redirections 432
 - server-side connections 432
- session IDs 432
- Set Object Properties and Values screen 60
- Set statement, in the Expert View and function libraries 149
- Setting object 344
- SGML 433
- shared object repositories
 - merging 275
- shared object repository window 251
- sharing action values
 - using Dictionary objects 28
 - using environment variables 27
 - via the global Data Table 26

- shortcuts
 - in Expert View 339
 - in function libraries 339
 - in Object Repository Merge Tool 283
- Silent Test Runner 423
 - opening 424
 - running tests from 424
- Silent Test Runner dialog box 424
- Smart Identification
 - configuring 109
 - enabling from the Object Identification dialog box 107, 108
- Smart Identification Properties dialog box 114
- Specify Pop-up Window Conditions screen 56
- standard event-recording configuration 313
- Start Transaction dialog box 421
- statement completion 130, 336
- Statistics dialog box 296
- status bar
 - Object Repository Merge Tool 281
- steps
 - adding with programming logic. *See* QuickTest Professional Basic Features User's Guide
- Subject Matter Expert (SME) 404
- support
 - knowledge base xix
 - Web site xix
- syntax
 - for action parameters 29
 - for action return values 31
 - for actions 29
- syntax errors, VBScript 153
- SystemUtil.Run method 165

T

- target repository 276
- target repository pane 279
- technical support. *See* customer support
- template tests 379
 - creating 382
- test database, maintaining 233

- Test Object Model. *See* QuickTest Professional Basic Features User's Guide
- test objects
 - property values, retrieving and setting 176
- test results
 - enabling and filtering 182
- Test run error trigger 54
- Test Run Log 425
- test run time, improving 437
- test scripts
 - customizing 333
- test set 388
- Test Settings dialog box
 - Generate Script option 238
- test versions in QuickTest 389
- TestDirector. *See* Quality Center
- testing options
 - during a test run 343
 - restoring 346
 - retrieving 346
 - run-time 347
 - setting 344
- testing process
 - introduction. *See* QuickTest Professional Basic Features User's Guide
- tests
 - and components, a comparison 412
 - associating recovery scenarios with 85
 - creating in Quality Center using a template test 384
 - diagram 4, 5
 - disabling recovery scenarios 89
 - opening in a Quality Center project 375
 - removing recovery scenarios from 89
 - running using Silent Test Runner 424
 - running. *See* QuickTest Professional Basic Features User's Guide
 - saving to a Quality Center project 374

Index

text

- checking. *See* QuickTest Professional Basic Features User's Guide
- finding in Expert View 140
- replacing in Expert View 142

timing transactions 419

toolbar

- Object Repository Merge Tool 282

transactions 419

- defining 419
- ending 422
- inserting 421
- measuring 419

trigger

- Application crash 54
- events 45
- Object state 54
- Pop-up window 54
- test run error 54

TSL functions

- calling from QuickTest 356

Tutorial, QuickTest Professional xvii

typographical conventions xxi

U

unregistering methods, using the

- UnregisterUserFunc statement 224

UnregisterUserFunc statement 219

update from local

- merging repositories 290

updates, documentation xviii

user-defined

- functions 185
- methods 219
- properties, accessing 178
- test objects, mapping 119

user-defined function

- adding a tooltip to 214
- documenting 214
- finalizing 217
- Function Definition Generator 204
- generating additional 216
- guidelines for 225
- previewing code in Function Definition Generator 216
- registering 209

V

values

- configuring. *See* QuickTest Professional Basic Features User's Guide

variables

- unique in global scope 226

VB Script

- formatting text 151

VBScript

- associated function libraries with Quality Center 200
- auto-expand syntax 337
- documentation 167
- syntax 146
- syntax errors 153

version control 389

- adding tests to 390
- checking tests in to 390, 392
- checking tests out of 390

version manager 389

Virtual Object Manager 42

Virtual Object wizard 38

virtual objects 33

- defining 37
- removing 42

W

- Web content, dynamic 431
- Web Event Recording Configuration dialog box 314, 326
- Web site, Mercury xvii
- WebElement objects, programmatic descriptions 163
- Web-event-recording configuration 311
 - customizing 315
 - standard 313
- What's New xv
- While statement, in the Expert View and function libraries 172
- Windows API 180
- WinRunner
 - calling tests
 - from QuickTest 352
 - calling TSL functions
 - from QuickTest 356
 - function arguments, passing
 - parameters from QuickTest 359
 - tests, passing parameters from QuickTest 354
 - working with 351
- With statements
 - entering manually 174
- workflow in Business Process Testing 407
- working test 390

X

- XML
 - exporting from object repository 273
 - importing as object repository 272

