

HP QuickTest Professional Delphi Add-in Extensibility

Software Version: 10.00

Developer Guide

Manufacturing Part Number: T6513-90047

Document Release Date: January 2009

Software Release Date: January 2009



Legal Notices

Warranty

The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

The information contained herein is subject to change without notice.

Restricted Rights Legend

Confidential computer software. Valid license from HP required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Third-Party Web Sites

HP provides links to external third-party Web sites to help you find supplemental information. Site content and availability may change without notice. HP makes no representations or warranties whatsoever as to site content or availability.

Copyright Notices

© 1992 - 2009 Mercury Interactive (Israel) Ltd.

Trademark Notices

Adobe® and Acrobat® are trademarks of Adobe Systems Incorporated.

Intel®, Pentium®, and Intel® Xeon™ are trademarks of Intel Corporation in the U.S. and other countries.

Java™ is a US trademark of Sun Microsystems, Inc.

Microsoft®, Windows®, Windows NT®, and Windows® XP are U.S registered trademarks of Microsoft Corporation.

Oracle® is a registered US trademark of Oracle Corporation, Redwood City, California.

Unix® is a registered trademark of The Open Group.

SlickEdit® is a registered trademark of SlickEdit Inc.

Documentation Updates

This guide's title page contains the following identifying information:

- Software Version number, which indicates the software version.
- Document Release Date, which changes each time the document is updated.
- Software Release Date, which indicates the release date of this version of the software.

To check for recent updates, or to verify that you are using the most recent edition of a document, go to:

<http://h20230.www2.hp.com/selfsolve/manuals>

This site requires that you register for an HP Passport and sign-in. To register for an HP Passport ID, go to:

<http://h20229.www2.hp.com/passport-registration.html>

Or click the **New users - please register** link on the HP Passport login page.

You will also receive updated or new editions if you subscribe to the appropriate product support service. Contact your HP sales representative for details.

Support

You can visit the HP Software Support web site at:

<http://www.hp.com/go/hpsoftwaresupport>

This web site provides contact information and details about the products, services, and support that HP Software offers.

HP Software Support Online provides customer self-solve capabilities. It provides a fast and efficient way to access interactive technical support tools needed to manage your business. As a valued support customer, you can benefit by using the HP Software Support web site to:

- Search for knowledge documents of interest
- Submit and track support cases and enhancement requests
- Download software patches
- Manage support contracts
- Look up HP support contacts
- Review information about available services
- Enter into discussions with other software customers
- Research and register for software training

Most of the support areas require that you register as an HP Passport user and sign in. Many also require a support contract.

To find more information about access levels, go to:

http://h20230.www2.hp.com/new_access_levels.jsp

To register for an HP Passport ID, go to:

<http://h20229.www2.hp.com/passport-registration.html>

Table of Contents

Welcome to This Guide	7
How This Guide Is Organized	8
Who Should Read This Guide	10
QuickTest Professional Online Documentation	10
Additional Online Resources.....	12
Chapter 1: Developing Support for Custom Delphi Controls	15
Glossary	16
Understanding How to Create Support for a Custom Delphi Control	16
Designing Your Test Object Configuration XML File.....	19
Designing Your Toolkit Configuration XML File	23
Designing Your Delphi Extensibility Code	24
Creating Support for Custom Grid Controls	30
Step-by-Step Instructions for Supporting Custom Delphi Controls.....	35
Chapter 2: Deploying the Toolkit Support Set	39
About Deploying the Custom Toolkit Support.....	40
Placing Your XML Files in the Correct Locations.....	41
Compiling Your Extensibility Code	42
Modifying Deployed Support	43
Removing Deployed Support	43
Chapter 3: Understanding the Toolkit Configuration XML File Structure	45
Understanding The Toolkit Configuration XML File	45
Mapping Application Controls to Test Object Classes	47
Mapping Test Object Classes to Inner Objects	49
An Example of a Complete Toolkit Configuration File	54
Index.....	55

Welcome to This Guide

Welcome to QuickTest Professional Delphi Add-in Extensibility.

QuickTest Professional Delphi Add-in Extensibility is an SDK (Software Development Kit) package that enables you to support testing applications that use third-party and custom Delphi controls that are not supported out-of-the-box by the QuickTest Professional Delphi Add-in.

The QuickTest Professional Delphi Add-in Extensibility SDK is included in the Delphi Add-in installation and provides the following:

- An API that enables you to extend the QuickTest Professional Delphi Add-in to support custom Delphi controls.
- A template that you can use when you create your extensibility code.
- The Delphi Add-in Extensibility Help (available from **Start > Programs > QuickTest Professional > Extensibility > Documentation**), which includes the following:
 - A Developer Guide
 - The QuickTest Test Object Schema Help
- A printer-friendly (PDF) version of the Developers Guide (in the **<QuickTest Professional installation folder>\help\Extensibility** folder).
- Sample Delphi Add-in Extensibility toolkit support sets that extend QuickTest support for the following Delphi custom controls:
 - TrackBar
 - StringDrawGrid (a table control)

This chapter includes:

- How This Guide Is Organized on page 8
- Who Should Read This Guide on page 10
- QuickTest Professional Online Documentation on page 10
- Additional Online Resources on page 12

How This Guide Is Organized

This guide explains how to use QuickTest Professional Delphi Add-in Extensibility to extend QuickTest support for third-party and custom Delphi controls.

This guide assumes you are familiar with QuickTest functionality and should be used together with the API file (**QuickTest Professional installation folder>\dat\Extensibility\Delphi\AgentExtensibilitySDK.pas**), and the extensibility code template (**<QuickTest Professional installation folder>\dat\Extensibility\Delphi\ExtensibilityImplementationTemplate.pas**), both of which contain comments and explanations, and together with the *HP QuickTest Professional Test Object Schema Help* provided in the Delphi Add-in Extensibility Help (**Start > Programs > QuickTest Professional > Extensibility > Documentation > Delphi Add-in Extensibility Help**). These documents should also be used in conjunction with the *HP QuickTest Professional User Guide*, the Delphi section of the *HP QuickTest Professional Add-ins Guide*, and the *HP QuickTest Professional Object Model Reference* (available from **Help > QuickTest Professional Help** in the QuickTest main window).

This guide contains:

Chapter 1 Developing Support for Custom Delphi Controls

Explains how to create a toolkit support set to extend the QuickTest Professional Delphi Add-in to support Delphi user interface controls that the add-in does not recognize out-of-the-box.

Chapter 2 Deploying the Toolkit Support Set

Explains how to enable QuickTest to use the toolkit support set that you developed to recognize the controls in the custom toolkit and run tests on them.

Chapter 3 Understanding the Toolkit Configuration XML File Structure

Explains the structure and syntax of the toolkit configuration XML file that you need to create as part of your toolkit support set.

Note: The information, examples, and screen captures in this guide focus specifically on working with QuickTest tests. However, much of the information applies equally to components.

Business components and scripted components are part of HP Business Process Testing, which utilizes a keyword-driven methodology for testing applications. For more information, see the *HP QuickTest Professional User Guide* and the *HP QuickTest Professional for Business Process Testing User Guide*.

Who Should Read This Guide

This guide is intended for programmers, QA engineers, systems analysts, system designers, and technical managers who want to extend QuickTest support for Delphi custom controls.

To use this guide, you should be familiar with:

- Major QuickTest features and functionality
- QuickTest Professional Object Model
- QuickTest Professional Delphi Add-in
- Delphi programming
- XML (basic knowledge)

QuickTest Professional Online Documentation

QuickTest Professional includes the following online documentation:

Readme provides the latest news and information about QuickTest. Select **Start > Programs > QuickTest Professional > Readme**.

HP QuickTest Professional Installation Guide explains how to install and set up QuickTest. Select **Help > Printer-Friendly Documentation > HP QuickTest Professional Installation Guide**.

HP QuickTest Professional Tutorial teaches you basic QuickTest skills and shows you how to design tests for your applications. Select **Help > QuickTest Professional Tutorial**.

Product Feature Movies provide an overview and step-by-step instructions describing how to use selected QuickTest features. Select **Help > Product Feature Movies**.

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

QuickTest Professional Help includes:

- **What's New in QuickTest Professional** describes the newest features, enhancements, and supported environments in the latest version of QuickTest.
- **HP QuickTest Professional User Guide** describes how to use QuickTest to test your application.
- **HP QuickTest Professional for Business Process Testing User Guide** provides step-by-step instructions for using QuickTest to create and manage assets for use with Business Process Testing.
- **HP QuickTest Professional Add-ins Guide** describes how to work with supported environments using QuickTest add-ins, and provides environment-specific information for each add-in.
- **HP QuickTest Professional Object Model Reference** describes QuickTest test objects, lists the methods and properties associated with each object, and provides syntax information and examples for each method and property.
- **HP QuickTest Professional Advanced References** contains documentation for the following QuickTest COM and XML references:
 - **HP QuickTest Professional Automation** 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.
 - **HP QuickTest Professional Test Results Schema** documents the test results XML schema, which provides the information you need to customize your test results.
 - **HP QuickTest Professional Test Object Schema** documents the test object XML schema, which provides the information you need to extend test object support in different environments.
 - **HP QuickTest Professional Object Repository Schema** documents the object repository XML schema, which provides the information you need to edit an object repository file that was exported to XML.

- **HP QuickTest Professional Object Repository Automation** documents the Object Repository automation object model, which provides the information you need to manipulate QuickTest object repositories and their contents from outside of QuickTest.
- **VBScript Reference** contains Microsoft VBScript documentation, including VBScript, Script Runtime, and Windows Script Host.

To access the QuickTest Professional Help, select **Help > QuickTest Professional Help**. You can also access the QuickTest Professional Help by clicking in selected QuickTest windows and dialog boxes and pressing F1. Additionally, you can view a description, syntax, and examples for a QuickTest test object, method, or property by placing the cursor on it and pressing F1.

Additional Online Resources

The **HP Software Web site** provides you with the most up-to-date information on HP Software products. This includes new software releases, seminars and trade shows, customer support, and more. The URL for this Web site is www.hp.com/go/software.

The following additional online resources are available from the QuickTest Professional **Help** menu:

Troubleshooting & Knowledge Base accesses the Troubleshooting page on the HP Software Support Web site where you can search the Self-solve knowledge base. Choose **Help > Troubleshooting & Knowledge Base**. The URL for this Web site is <http://h20230.www2.hp.com/troubleshooting.jsp>.

HP Software Support accesses the HP Software Support Web site. This site enables you to browse the Self-solve knowledge base. You can also post to and search user discussion forums, submit support requests, download patches and updated documentation, and more. Choose **Help > HP Software Support**. The URL for this Web site is www.hp.com/go/hpsoftwaresupport.

Most of the support areas require that you register as an HP Passport user and sign in. Many also require a support contract.

To find more information about access levels, go to:
http://h20230.www2.hp.com/new_access_levels.jsp

To register for an HP Passport user ID, go to:
<http://h20229.www2.hp.com/passport-registration.html>

1

Developing Support for Custom Delphi Controls

You can create a custom toolkit support set to extend the QuickTest Professional Delphi Add-in, and teach QuickTest to recognize Delphi user interface controls that the Delphi Add-in does not recognize out-of-the-box.

This chapter explains how to create support for custom Delphi user interface controls using QuickTest Professional Delphi Add-in Extensibility. It explains what files and units you have to create for the toolkit support set, the structure and content of these files, and how to use them to enable QuickTest to test applications that contain custom Delphi controls.

For information on where the toolkit support set files should be stored, and what you have to do to activate the support you design, see Chapter 2, “Deploying the Toolkit Support Set.”

This chapter includes:

- Glossary on page 16
- Understanding How to Create Support for a Custom Delphi Control on page 16
- Designing Your Test Object Configuration XML File on page 19
- Designing Your Toolkit Configuration XML File on page 23
- Designing Your Delphi Extensibility Code on page 24
- Creating Support for Custom Grid Controls on page 30
- Step-by-Step Instructions for Supporting Custom Delphi Controls on page 35

Glossary

- **Agent Object (AO).** A class written in Delphi and integrated into the QuickTest Professional Delphi Add-in's precompiled agent. The Agent Object provides the methods required to support retrieving properties and recording and running test object methods on the custom control. The Agent Object acts as an interface between QuickTest and the Delphi application being tested.
- **Custom toolkit.** A set of custom controls for which you implement QuickTest support.
- **Custom toolkit support set.** The set of files, units and objects that you create to extend QuickTest support for the controls in the custom toolkit.
- **Inner test object.** The Standard Windows test object class whose implementation QuickTest uses for any properties or methods for which specific support is not provided by the Agent Object that you develop.

Understanding How to Create Support for a Custom Delphi Control

You can create a custom toolkit support set to extend the QuickTest Professional Delphi Add-in, and to teach QuickTest to recognize Delphi user interface controls that the Delphi Add-in does not recognize out-of-the-box.

A custom toolkit support set consists of the following items:

- **Required:** A toolkit configuration XML file, in which you map the custom controls to test object classes (either existing Delphi test object classes or new ones that you define).

- **Optional:** A test object configuration XML file, in which you define the test object classes that will represent your custom controls in QuickTest tests and components.

You do not need to define a test object class to represent a custom control if an existing Delphi test object class has all of the necessary test object methods and identification properties.

- **Optional:** A Delphi unit of extensibility code, in which you develop an Agent Object for every type of control that you want to support, as well as a factory function that creates these Agent Objects.

You do not need to create an Agent Object for a control that can be adequately supported by mapping to an existing Delphi test object class. You need to create an Agent Object in the following situations:

- You map the custom control to a new (custom) test object class.
- You map the custom control to an existing test object class, but you want to override the implementation of a test object method or provide the value of an identification property.
- You are creating support for a custom grid control.

For more information, see:

- “Designing Your Test Object Configuration XML File” on page 19
- “Designing Your Toolkit Configuration XML File” on page 23
- “Designing Your Delphi Extensibility Code” on page 24
- “Step-by-Step Instructions for Supporting Custom Delphi Controls” on page 35

Using the Delphi Add-in Extensibility Samples

The Delphi Add-in Extensibility SDK includes the following samples to help you learn about Delphi Add-in Extensibility. You can also use the samples as a basis for your extensibility files.

- The basic Delphi Add-in extensibility sample provides a toolkit support set for the **TTrackBar** control, which is not supported out-of-the-box by the Delphi Add-in. This toolkit support set provides a comprehensive example of how to extend QuickTest support for a custom control.
- The Delphi Add-in grid extensibility sample provides a toolkit support set for the **TStringDrawGrid** control, which is a custom grid control that inherits from the **TCustomGrid** class. This toolkit support set demonstrates only how to teach QuickTest to treat a custom grid control as a table. For more information, see “Creating Support for Custom Grid Controls” on page 30.

The samples are located under **<QuickTest Professional installation folder>\samples**, in the **DelphiExtSample** and **DelphiGridExtSample** folders. Within each of these folders, the custom control and its source files are located in the **Application** sub-folder, and the toolkit support set files (configuration files and extensibility unit) are located in the **ToolkitSupportSet** sub-folder.

You can use these samples in the following ways:

- Study the content of the toolkit support set files to gain a better understanding of how to develop your own toolkit support sets.
- Copy the toolkit support set files (or parts of them) and use them as a basis for designing your toolkit support sets.
- Learn how extensibility can affect QuickTest’s interaction with custom controls. To do this, create and run a QuickTest test on the sample custom control before and after deploying the sample toolkit support set to QuickTest. The procedure described below guides you through this process for the **TTrackBar** sample. You can perform a similar procedure using the **TStringDrawGrid** sample.

To analyze how the TTrackBar extensibility sample affects QuickTest’s interaction with the TTrackBar custom control:

- 1 Use the QuickTest Object Spy to see how QuickTest recognizes the **TTrackBar** control. Create and run a simple test on the control.

You can see that QuickTest uses a generic DelphiObject test object to represent the track bar control. To set the location of the thumb on the track bar, you must use **Click**, **Drag**, and **Drop** operations.

- 2 Deploy the toolkit support set for the **TTrackBar** control according to the instructions in Chapter 2, “Deploying the Toolkit Support Set.”
- 3 Use the QuickTest Object Spy to see how QuickTest recognizes the **TTrackBar** control now that the extensibility support is enabled. Create and run a simple test on the control.

You can see that with extensibility support enabled, QuickTest uses a **DelphiTrackBar** test object (and a customized icon) to represent the control. The **DelphiTrackBar** test object supports **Set**, **Next**, and **Prev** operations for modifying the location of the thumb on the track bar. In addition, when you drag the track bar thumb during a recording session, QuickTest records test steps with the **Set** operation.

Designing Your Test Object Configuration XML File

In this file, you define any custom test object classes that you want QuickTest to use to represent your custom controls in tests and components. Define a test object class for each custom control that cannot be adequately represented by an existing Delphi test object class.

In a test object configuration XML, you define the test object classes (for example, their identification properties, the test object methods they support, and so on). To do this you define a **ClassInfo** element for each test object class. In addition, you define the name of the environment or custom toolkit for which the test object classes are intended (in the **PackageName** attribute of the **TypeInformation** element), and the QuickTest add-in which these test object classes extend (in the **AddinName** attribute of the **TypeInformation** element).

If the relevant add-in is not loaded when QuickTest opens, QuickTest does not load the information in this XML. Similarly, if the name of the environment or custom toolkit is displayed in the Add-in Manager dialog box and its check box is not selected, the information in this XML is not loaded.

A test object class definition can include the following:

- The name of the new test object class and its attributes, including the base class—the test object class that the new test object class extends.

The test object class name must be unique among all of the environments whose support a QuickTest user might load simultaneously. For example, do not use names of test object classes from existing QuickTest add-ins, such as DelphiButton, DelphiEdit, etc.

Note: A test object class inherits the base class' test object operations (methods and properties), generic type, and default operation. Identification properties are not inherited.

- The generic type for the new test object class, if you want the new test object class to belong to a different generic type than the one to which its base class belongs. (For example, if your new test object class extends DelphiObject (whose generic type is **object**), but you would like QuickTest to group this test object class with the **edit** test object classes.)

Note: Generic types are used when filtering objects (for example, in the Select Object for Step dialog box and when adding multiple test objects to the object repository. Generic types are also used when creating documentation strings for the Documentation column of the Keyword View (if they are not specifically defined in the test object configuration file).

- The path of the icon file to use for this test object class (Optional. If not defined, a default icon is used.) The file can be a **.dll** or **.ico** file.

- A context-sensitive Help topic to open when F1 is pressed for the test object in the Keyword View or Expert View. The definition includes the Help file path and the relevant Help ID within the file.
- A list of methods for the test object class, including the following information for each method:
 - The arguments, including the argument type (for example, **String** or **Integer**), direction (In or Out), whether the argument is mandatory, and, if not, its default value.
 - The method description (shown as a tooltip in the Keyword View, Expert View, and Step Generator).
 - The Documentation string (shown in the **Documentation** column of the Keyword View and in the Step Generator).
 - The return value type.
 - A context-sensitive Help topic to open when F1 is pressed for the test object method in the Keyword View or Expert View, or when the **Operation Help** button is clicked for the method in the Step Generator. The definition includes the Help file path and the relevant Help ID within the file.
- The test object operation that is selected by default in the Keyword View and Step Generator when a step is generated for an object of this class.
- A list of identification properties for the test object class.
 - The identification properties that are used for the object description.
 - The identification properties that are used for **smart identification**. (This information is relevant only if smart identification is enabled for the test object class. To enable smart identification, use the Object Identification dialog box in QuickTest.)
 - The identification properties that are available for use in checkpoints.
 - The identification properties that are selected by default for checkpoints (in the QuickTest Checkpoint Properties dialog box).

Note: You can also create a definition for an existing test object class in the test object configuration XML. This definition is added to the existing definition of this test object class, affecting all test objects of this class. It is therefore not recommended to modify existing test object classes in this way.

For example, if you add an identification property, it appears in QuickTest in the list of properties for all test objects of this class, but has no value unless it is implemented for the specific supported object.

If you add a test object method, it appears in the list of test object methods in QuickTest, but if you use the test object method in a test, and it is not implemented for the specific object, a run-time error occurs.

If you add test object methods to existing test object classes, you might add a prefix to the method name that indicates the toolkit support for which you added the method (for example, **CustomButtonClick**, **CustomEditSet**). This enables test designers to easily identify the custom methods and use them in test steps only if they know that the custom method is supported for the specific object.

For information on the structure and syntax of this XML, see the QuickTest Test Object Schema Help, available in the QuickTest Professional Delphi Add-in Extensibility Help (**DelphiExtensibility.chm** Help file in the <QuickTest Professional installation folder>\help\Extensibility folder).

Designing Your Toolkit Configuration XML File

In this file you define two types of mapping:

- **Mapping application controls to test objects.** You map the custom controls in your Delphi application to the Delphi test object classes that should represent them in QuickTest tests and components. You can map custom controls to new Delphi test object classes that you define in the test object configuration file or to existing Delphi test object classes.

Use new Delphi test object classes, if you want to customize the test object name, description (the set of properties QuickTest uses to identify the object in the application), or any properties and methods that are specific to your control.

Note: If you want to customize only how a certain test object method is performed on the control (and not the method syntax), you can map the control to the existing test object class. You can override the test object method implementation in the Agent Object that you develop.

The same is true for identification properties—if a test object class includes an identification property that is relevant for your control but QuickTest does not retrieve its value, you can design your Agent Object to provide the necessary value. For more information, see “Creating Your Extensibility Code” on page 25.

- **Mapping custom test object classes to inner objects.** You map the test object classes that you defined in the test object configuration file (if any) to existing QuickTest Standard Windows test object classes that serve as inner objects. QuickTest uses the inner object’s implementation for any properties or methods for which specific support is not provided by the Agent Object that you develop.

Mapping your new Delphi test object classes to inner objects enables you to take advantage of existing QuickTest support for common methods and properties. This can be helpful, for example, when creating support for subclassed Standard Windows controls such as `TListView`. For a list of existing Standard Windows test object classes, see the *HP QuickTest Professional Object Model Reference*.

Important: In some dialog boxes, QuickTest displays a list of available test object classes per environment (for example, in the Define New Test Object dialog box, the Object Identification dialog box, and the Step Generator). If you want QuickTest to display your custom Delphi test object classes in these dialog boxes, you must define this in the toolkit configuration file as well.

For information on the structure and syntax of this XML, see Chapter 3, “Understanding the Toolkit Configuration XML File Structure.”

Designing Your Delphi Extensibility Code

The Delphi unit that you develop for extensibility must include the following items:

- One or more Agent Objects; one Agent Object for each type of control that you want to support. The Agent Objects interface between QuickTest and the Delphi controls in the application being tested.
- One factory function that creates the appropriate Agent Object for each control. When QuickTest first interacts with a control, it calls the factory function to create the corresponding Agent Object.

Note: For some custom controls, mapping the control to an existing Delphi test object class might provide sufficient support. In such cases, you do not have to design an Agent Object for the control.

Creating Your Extensibility Code

To create the extensibility Delphi unit that you develop to support your custom controls, use the template unit provided with the Delphi Add-in:

<QuickTest Professional installation folder>\dat\Extensibility\Delphi\ExtensibilityImplementationTemplate.pas

In your extensibility code you must do the following:

- Import and use the AgentExtensibilitySDK unit provided with the QuickTest Professional Delphi Add-in: **<QuickTest Professional installation folder>\dat\Extensibility\Delphi\AgentExtensibilitySDK.pas**
- Design an Agent Object for each type of control that you want to support. The Agent Object must inherit from **TMicAO** or one of the other Agent Object base classes defined in the Delphi Add-in Extensibility SDK. In the Agent Objects, develop published properties that support the test object methods and identification properties required for your controls. For more information, see “Working with Published Properties to Support Test Object Methods and Identification Properties” on page 26.
- Create a factory function that receives an object reference of a Delphi user interface control and returns a new Agent Object. The factory function must be able to recognize the custom controls for which you are creating support, and create the appropriate Agent Object.
- In the initialization section of your extensibility unit, call the **AddExtensibilityServer** API function to register your factory function to the Delphi Add-in.

For more information, see **<QuickTest Professional installation folder>\dat\Extensibility\Delphi\AgentExtensibilitySDK.pas**.

Before you can run the support that you develop, you must compile the Delphi application you are testing with the extensibility unit you designed and with the Delphi Add-in precompiled agent. For more information, see Chapter 2, “Deploying the Toolkit Support Set.”

Working with Published Properties to Support Test Object Methods and Identification Properties

QuickTest interacts with the application's controls by setting and retrieving the published properties provided by the Agent Object and the control itself. QuickTest first accesses the published properties of the Agent Object and then, if necessary, the published properties of the Delphi object.

When you develop your Agent Object, design published properties to support the identification properties and test object methods that you defined in the test object configuration file. For example, you can create published properties in your Agent Object to enable access to (public) unpublished member variables of the control.

The following reserved properties are used for the implementation of recording and running tests and components, as described in the sections below:

➤ **__QTPReplayMtd_**

Use this prefix for all Agent Object properties designed to implement running QuickTest test object methods.

➤ **__CellRect, __CellData, and __TableContent**

These properties are used to implement support for grid objects. For more information, see “Creating Support for Custom Grid Controls” on page 30.

➤ **__QTPRecording**

This property is used to implement support for the QuickTest recording capability.

Supporting Identification Properties

For each identification property that you want to support, make sure there is a published property with the same name in the Delphi control or the Agent Object.

Note: QuickTest uses only lower-case letters in identification property names. Therefore, the names of published properties that support identification properties must contain only lower-case letters (even if the identification property name in the test object configuration file contains upper-case letters).

A QuickTest user can access the published properties of the Agent Object and the Delphi control using the **GetROProperty** and **SetROProperty** methods. In addition, these published properties can be verified using checkpoints in a QuickTest test and viewed in the Object Spy.

If a property name begins with a double underscore ('__'), it is not displayed in the Object Spy and cannot be accessed by checkpoints or output values. Such hidden properties can be accessed by the **GetROProperty** and **SetROProperty** test object methods, and can be accessed directly in user defined functions created in QuickTest.

Supporting Test Object Methods

For each test object method that you want to support, create a published property named **__QTPReplayMtd_<Test Object Method Name>** in the Agent Object. For example, a published property named **__QTPReplayMtd_MyTOMethod** provides the implementation for running the **MyTOMethod** test object method.

Supporting the QuickTest Recording Capability

An Agent Object must implement the **__QTPRecording** published property to support the QuickTest recording capability.

During a recording session, when an event occurs on a custom Delphi control, QuickTest sends the Windows message to the Agent Object and queries the **__QTPRecording** property to retrieve the corresponding line to add to the test or component.

In most cases, you do not have to implement this property in your extensibility code. If you want to create an Agent Object that supports recording, have your Agent Object inherit from the **TMicRecordableAO** base class.

The **TMicRecordableAO** agent object base class implements **__QTPRecording**, to perform the following:

- 1** Process the parameters passed by QuickTest.
- 2** Call the **ProcessMessage** function to determine what step to record for the event that occurred (and the recording mode).
- 3** Convert the recording information to the format required by QuickTest.

In the Agent Object that you develop, you need only implement the **ProcessMessage** function.

When QuickTest accesses the **__QTPRecording** property, it passes the window message parameters and window handle to the Agent Object, and expects in return, an array (of type **safearray**) that contains the recorded step information (the operation and its arguments) and the recording mode. For more information on recording modes, see <QuickTest Professional installation folder>\dat\Extensibility\Delphi\AgentExtensibilitySDK.pas.

Input parameters:

Parameter Index	Value Type	Description
0	VT_I4	Window handle
1	VT_I4	Message
2	VT_I4	IParam
3	VT_I4	wParam

Output safearray format:

Parameter Index	Value Type	Description
0	VT_I4	Recording mode
1	VT_BSTR	Test object method to record
2..end		Test object method arguments

Note: If you create a custom test object class to support the custom control, you can use the **ExtObjRecFilter** attribute in the toolkit configuration file to specify the level of events that trigger recording. For more information, see “Mapping Test Object Classes to Inner Objects” on page 49.

Creating Support for Custom Grid Controls

To create support for a custom grid control you need to map the control to an appropriate test object class, develop an Agent Object that implements the support, and (optionally) instruct QuickTest to treat the control as a table.

A sample toolkit support set, which provides support for a custom grid control (**TStringDrawGrid**), is located in the **<QuickTest Professional installation folder>\samples\DelphiGridExtSample** folder. After reading this section, you can use the sample to gain a better understanding of how to create support for custom grid controls. For more information, see “Using the Delphi Add-in Extensibility Samples” on page 18.

Mapping a Custom Grid Control to a Test Object Class

You can map the custom grid control to the **DelphiTable** test object class, or to a custom grid test object class that you define in the test object configuration file.

If you map the custom grid control to the **DelphiTable** test object class, you do not have to create any of the definitions described in “Instructing QuickTest to Treat Your Custom Grid Control as a Table” on page 32.

Developing an Agent Object to Support a Custom Grid Control

The Agent Object must provide support for all of the test object methods and identification properties defined in the test object class mapped to the grid control. These include grid operations such as **SetCellData** and **GetCellData**, and any other methods and properties that you define in the test object class.

The QuickTest Professional Delphi Add-in provides a test object extension (**Mercury.DelphiTableSrv**) that implements much of the design required to support grid controls.

This extension implements basic grid operations like **SetCellData**, **GetCellData**, **SelectCell**, and so on. The extension delegates these test object methods to the Agent Object using the **__CellRect**, **__CellData**, **__TableContent**, **RowCount**, and **ColCount** published properties.

To create support for a custom grid control, you must design an Agent Object that inherits from **TMicGridAOBase** or **TCustomGridAOBase** and implements these published properties. (For more information about the **TMicGridAOBase** or **TCustomGridAOBase** base classes, see `<QuickTest Professional installation folder>\dat\Extensibility\Delphi\AgentExtensibilitySDK.pas`.)

In addition, you must instruct QuickTest to use the grid test object extension to support your custom grid. For more information on how to do this, see “Instructing QuickTest to Treat Your Custom Grid Control as a Table” on page 32.

Note: If you map the custom control to a custom test object class, design the Agent Object to support any additional test object methods and identification properties defined in the test object class.

Implementing the Published Properties for Supporting a Grid

The support that you develop for a custom grid control is based on the Delphi Add-in grid test object extension. Therefore, you must implement the following published properties in your Agent Object:

- **__CellRect** must return the rectangle at which the cell is located, in the format: x;y;width;height;; where x and y are the coordinates of the top left corner of the rectangle.
- **__CellData** is used to set and retrieve the value contained in a cell (in String format). The **TMicGridAOBase** agent object base class implements this property to call the abstract functions **GetCellDataEx** and **SetCellDataEx**. Implement these functions in the derived class that you design for your Agent Object.
- **__TableContent** is used to write the content (data) of the whole table to the specified file and return **true** or **false** indicating success or failure. The file is specified in the parameter passed to the Agent Object from QuickTest. Write the table content to the file in string format, with tabs separating cell data and new-line characters separating rows.

The grid test object extension uses this property to support table checkpoints. The **TMicGridAOBase** agent object base class implements this property to call the abstract **CaptureTableEx** function. Implement this function in the derived class that you design for your Agent Object.

Instructing QuickTest to Treat Your Custom Grid Control as a Table

In the toolkit configuration XML file, define the following:

- Instruct QuickTest to use the Delphi Add-in grid test object extension to support your custom grid control (or all controls mapped to a specific custom grid test object class).
- If you defined a custom grid test object class (in the test object configuration file), instruct QuickTest to treat this type of test object as a table test object when creating checkpoints and output values.

Note: For custom grid controls mapped to the DelphiTable test object class, you do not have to create any of the definitions described in this section.

To instruct QuickTest to use the grid test object extension for this type of control:

Add the following definitions to your toolkit configuration XML file (**bold** text represents the lines you need to add):

```
<MicTest>
  <Key Name="Packages">
    <Key Name="DelphiPackage">
      <Key Name="CustomServers">
        <Key Name="TCustomGridNativeClass">
          <Value Name="CustReplayProglD"
            Type="BSTR"> Mercury.DelphiTableSrv
          </Value>
        </Key>
      </Key>
    </Key>
  </Key>
  ...
</MicTest>
```

- Replace **TCustomGridNativeClass** with the window class name of the grid control for which you are developing support.
- Within the **Key** element where **Name="CustomServers"**, create a separate **Key** value for each custom grid class that you want to support.

Note: Alternatively, you could create a single **Key** element to instruct QuickTest to use the grid test object extension for all custom controls mapped to a certain custom test object class. To do this, replace **TCustomGridNativeClass** in the section above with the name of the custom grid test object class, prefixed with the string **MC2CSMapping_** (for example, **MC2CSMapping_DelphiCustomTable**).

To instruct QuickTest to treat this type of test object as a table test object when creating checkpoints and output values:

Add the following definitions to your toolkit configuration file in the section that you create to map your custom test object class to an inner objects (**bold** text represents the lines you need to add):

```
<MicTest>
...
<Key Name="Test Objects">
  <Key Name="TheDelphiCustomTestObject you are mapping">
...
    <!-- enables table checkpoint and output value -->
    <Key Name="CustomStepCfg">
      <Value Name="Checkpoint" Type="BSTR">Mercury.MultiVerUI</Value>
      <Value Name="Output Value" Type="BSTR">Mercury.MultiVerUI</Value>
    </Key>
    <!-- Enables use of the Define/Modify Row Range dialog box -->
    <Key Name="CustomStepCfgDlg">
      <Value Name="Checkpoint" Type="BSTR">
        Mercury.TableTOConcigUI
      </Value>
      <Value Name="Output Value" Type="BSTR">
        Mercury.TableTOConcigUI
      </Value>
    </Key>
  </Key>
</Key>
</MicTest>
```

If you define more than one custom Delphi grid test object class, add these definitions within the **Key** element that you define for each of the relevant test object classes.

This instructs QuickTest to use the Table Checkpoint Properties and Table Output Value Properties dialog boxes for this type of test object.

For information on the structure and syntax of the toolkit configuration XML, see Chapter 3, “Understanding the Toolkit Configuration XML File Structure.”

Step-by-Step Instructions for Supporting Custom Delphi Controls

This section guides you through the process of creating a QuickTest Professional Delphi Add-in Extensibility toolkit support set.

Note: In some cases, it is sufficient to map your custom control to an existing Delphi test object class. In this case, perform only steps 1 and 2 in the procedure below, and compile the application you are testing with the **MicDelphiAgent.pas** module, as described in the Delphi section of the *HP QuickTest Professional Add-ins Guide*.

In other cases, mapping the custom control to an existing test object class does not provide adequate support, even though the test object class includes all of the necessary test object methods and identification properties. In these cases, you do not need to create a new test object class to support the custom control (steps 3 and 4), but you do need to create a Delphi extensibility unit that supports the test object methods and properties (steps 6 to 9).

To create the support set for your custom control:

1 Create your toolkit configuration file.

- a** Decide which test object classes will represent your custom controls in QuickTest tests and components. You can map your custom controls to existing Delphi test objects classes, or to custom test object classes that you define in step 3.
- b** Decide which custom test object classes you will create to represent your custom controls (if any), and which existing Standard Windows test object classes will serve as their inner objects.
- c** Copy the sample toolkit configuration file, `<QuickTest Professional installation folder>\samples\DelphiExtSample\ToolkitSupportSet\TrackerSampleToolkitCfg.xml`, to create your toolkit configuration file.

- d** For each custom control that you want to support, make a copy of the **Value** element that maps the **TTrackBar** custom control to the custom test object class **DelphiTrackBar**. Replace **TTrackBar** and **DelphiTrackBar** with the appropriate names.
- e** For each custom test object class that you create (if any), make a copy of the **Key** element that contains the settings for the **DelphiTrackBar** custom test object class. Replace the test object name, the **InnerProgId**, and the **InnerMicClass** values, with the appropriate test object names.

For information on the structure and syntax of this XML, see Chapter 3, “Understanding the Toolkit Configuration XML File Structure.”

Note: You must select an inner object that matches the functionality of the control you are supporting. In most cases, the inner object should be a generic test object class like **WinObject**. You can use a more specific test object class when you are sure that it is appropriate for the type of control you are supporting (for example, you can use the **WinListView** test object as an inner object when creating support a control that subclasses **TListView**).

In order to verify that the test object class you are using for an inner object is appropriate, use the QuickTest Object Mapping dialog box to map this test object class to your control and ensure that you can successfully use the Object Spy on your control.

For more information on the Object Mapping dialog box, see the *HP QuickTest Professional User Guide*.

2 Deploy your toolkit configuration file.

Copy the toolkit configuration XML file to the <QuickTest Professional installation folder>\dat\Settings folder.

3 Create your test object configuration file.

If all of your custom controls are mapped to existing Delphi test object classes, skip to step 5.

Otherwise, create a test object configuration XML file in which you define your custom test object classes. Define description properties, identification properties, and test object methods for each custom test object class. For more information, see “Designing Your Test Object Configuration XML File” on page 19.

4 Deploy your test object configuration file.

If you created a test object configuration XML file, copy it to the **<QuickTest Professional installation folder>\dat\Extensibility\Delphi** folder.

5 Verify the test object class mapping.

Open QuickTest and make sure that:

- When you use the Object Spy, QuickTest recognizes your custom controls correctly.
- When QuickTest learns your custom controls, the corresponding test objects are added to the Object Repository.

Note: If mapping your custom control to an existing Delphi test object provides sufficient support for creating and running tests and components on this control, you do not need to perform any of the remaining steps in this procedure.

6 Create a basic extensibility Delphi unit.

Create your extensibility code using the Agent Object Implementation Template **<QuickTest Professional installation folder>\dat\Extensibility\Delphi\ExtensibilityImplementationTemplate.pas**. Define some visible published properties in the Agent Object.

7 Compile your extensibility code.

Compile the application you are testing with the QuickTest Professional Delphi Add-in precompiled agent and with your extensibility unit. To do this, perform the procedure described in “Compiling Your Extensibility Code” on page 42.

8 Verify the functionality of your extensibility code.

Run your Delphi application and verify that your Agent Object is functioning correctly by testing that you can use the QuickTest Object Spy to view the properties that you defined in step 6.

9 Complete the development of your extensibility Delphi unit.

Implement the rest of the Agent Object’s published properties to support the identification properties and test object methods that you defined in the test object configuration file.

2

Deploying the Toolkit Support Set

The final stage of extending QuickTest support for a custom toolkit is deploying the toolkit support set. This means enabling QuickTest to use the toolkit support set that you developed to recognize the controls in the toolkit and run tests on them.

While you are developing the toolkit support set, deploying it to QuickTest enables you to test and debug the support that you create. After the toolkit support set is complete, you can deploy it on any computer with QuickTest installed, to extend the QuickTest Professional Delphi Add-in.

This chapter includes:

- About Deploying the Custom Toolkit Support on page 40
- Placing Your XML Files in the Correct Locations on page 41
- Compiling Your Extensibility Code on page 42
- Modifying Deployed Support on page 43
- Removing Deployed Support on page 43

About Deploying the Custom Toolkit Support

To deploy a QuickTest Professional Delphi Add-in Extensibility toolkit support set, you must place the XML files that you created in the correct locations on a computer with QuickTest installed, and compile the application you are testing with the QuickTest Professional Delphi Add-in precompiled agent and with the extensibility unit that you developed.

For more information, see:

- “Placing Your XML Files in the Correct Locations” on page 41
- “Compiling Your Extensibility Code” on page 42

From the QuickTest user’s perspective, after you deploy the toolkit support set on a computer on which QuickTest is installed, the Delphi Add-in recognizes your custom controls just as it recognizes any other Delphi object.

Placing Your XML Files in the Correct Locations

To deploy the toolkit support set that you create, you must place the XML files in specific locations within the QuickTest installation folder. The following table describes the appropriate location for each of the toolkit support files:

File	Location
Test Object Configuration file	<ul style="list-style-type: none"> ➤ <QuickTest installation folder>\dat\Extensibility\Delphi ➤ <QuickTest Add-in for Quality Center installation folder>\dat\Extensibility\Delphi (Optional. Required only if QuickTest Add-in for Quality Center is installed)
Toolkit Configuration file	<QuickTest installation folder>\dat\Settings
Icon files for custom test object classes (optional)	<p>The file can be a .dll or .ico file, located on the computer on which QuickTest is installed, or in an accessible network location.</p> <p>Specify the location in the test object configuration file.</p>
Help files for the test object classes (optional)	<p>Must be a .chm file, located on the computer on which QuickTest is installed.</p> <p>Specify the location in the test object configuration file.</p>

Note: The recommended location for the toolkit support set icon files is **<QuickTest installation folder>\dat\Extensibility\Delphi\Toolkits\<custom toolkit name>\res**.

The recommended location for the toolkit support set Help files is **<QuickTest installation folder>\dat\Extensibility\Delphi\Toolkits\<custom toolkit name>\help**.

In the test object configuration file, you can specify these locations using relative paths. For more information, see the *QuickTest Test Object Schema Help* (available with the Delphi Add-in Extensibility Help).

Compiling Your Extensibility Code

If you developed an extensibility Delphi unit, you must compile the application you are testing with the QuickTest Professional Delphi Add-in precompiled agent and with your extensibility unit. To do this, perform the following steps:

- 1** Add the **<QuickTest Professional Installation folder>\dat\Extensibility\Delphi** folder to the search path of the application's project or copy the contents of the **<QuickTest Professional Installation folder>\dat\Extensibility\Delphi** folder to the project folder.
- 2** Add **MicDelphiAgent** to the **Uses** section of your application's project file.
- 3** If your application includes the **TwwDBGrid** from InfoPower, add **MicWWSupport** to the **Uses** section of your application's project file after **MicDelphiAgent**.
- 4** Add the location of your extensibility code to the search path of the application's project or place your file in the project folder.
- 5** Add the name of your extensibility unit to the **Uses** section of your application's project file.
- 6** Compile the Delphi application project.

Modifying Deployed Support

If you modify the extensibility Delphi unit you developed, you must recompile the application you are testing (with the QuickTest Professional Delphi Add-in precompiled agent and with the extensibility unit that you developed) and re-run the Delphi application for the changes to take effect.

If you modify the XML files of a deployed toolkit support set, you must close and reopen QuickTest for the changes to take effect.

Removing Deployed Support

If you want to remove support for a custom toolkit from QuickTest after it is deployed, you must delete its toolkit configuration file from: **<QuickTest installation folder>\dat\Settings** and compile the application you are testing without the extensibility Delphi unit you developed.

If none of the test object class definitions in a test object configuration file are used to represent any custom controls (meaning they are no longer needed), you can delete the file from:

<QuickTest installation folder>\dat\Extensibility\Delphi (and **<QuickTest Add-in for Quality Center installation folder>\dat\Extensibility\Delphi** if relevant).

3

Understanding the Toolkit Configuration XML File Structure

A Delphi toolkit support set must include a toolkit configuration file that maps your custom Delphi controls to the test object classes that represent and support them in QuickTest.

This chapter includes:

- Understanding The Toolkit Configuration XML File on page 45
- Mapping Application Controls to Test Object Classes on page 47
- Mapping Test Object Classes to Inner Objects on page 49
- An Example of a Complete Toolkit Configuration File on page 54

Understanding The Toolkit Configuration XML File

In the toolkit configuration XML file you must define two types of mapping, as described in “Designing Your Toolkit Configuration XML File” on page 23:

- Mapping application controls to test object classes
- Mapping custom test object classes to inner objects

The root element of the toolkit configuration XML file is a **MicTest** element, which must contain two **Key** elements, each with different **Name** attributes (Packages and Test Objects). These two **Key** elements divide the XML file into two sections, each used for a different type of mapping.

```
<?xml version="1.0"?>
<MicTest>
  <Key Name="Packages">
    ...
  </Key>

  <Key Name="Test Objects">
    ...
  </Key>
</MicTest>
```

- Within the **Key** element where **Name="Packages"**, you map application controls to test objects.

This is also the section in which you can map a custom grid control (or a custom grid test object class) to **Mercury.DelphiTableSrv**, if you want to use the built-in grid test object extension. This built-in extension implements much of the design required to support grid controls. For more information, see “Creating Support for Custom Grid Controls” on page 30.

- Within the **Key** element where **Name="Test Objects"**, you map custom test object classes to inner objects.

This is also the section in which you set additional settings for custom test object classes, such as instructing QuickTest to display them in certain dialog boxes or to treat them as tables when creating checkpoints.

For information on the structure and syntax of each of these sections, see:

- “Mapping Application Controls to Test Object Classes” on page 47
- “Mapping Test Object Classes to Inner Objects” on page 49

Mapping Application Controls to Test Object Classes

Within the **MicTest** root element of the toolkit configuration XML file, under the **Key** element where **Name="Packages"**, create a structure similar to the following:

```
<?xml version="1.0"?>
<MicTest>
  <Key Name="Packages">
    <Key Name="DelphiPackage">
      <Key Name="ClassPatterns">
        <!-- One Value element for each custom control that you want to map -->
        <!-- This example shows the mapping for the TStringDrawGrid custom control. -->
        <Value Name="TStringDrawGrid" Type="BSTR">DelphiCustomTable</Value>
      </Key>
      <Key Name="CustomServers">
        <!-- One Key element for each custom grid control that you want to support -->
        <!-- This example shows the definition for the TStringDrawGrid custom control. -->
        <Key Name="TStringDrawGrid">
          <Value Name="CustReplayProgID"
                Type="BSTR"> Mercury.DelphiTableSrv
          </Value>
        </Key>
      </Key>
    </Key>
  </Key>
  ...
</MicTest>
```

- Within the **Key** element whose **Name** attribute is **ClassPatterns**, you define one **Value** element for each type of custom control that you want to support.

In each **Value** element:

- The **Name** attribute contains the window class name of your custom control.
- The **Type** attribute is set to **BSTR**.
- The element value contains the name of the custom Delphi test object class that QuickTest should use to represent the control in tests and components.

- If you are creating support for custom grid controls, create a **Key** element whose **Name** attribute is `CustomServers`. Within this **Key** attribute, you define one **Key** element for each custom grid control that you want to support.

The **Key** element for each custom grid control contains:

- A **Name** attribute that contains the window class name of your custom control.
- The following **Value** element:

```
<Value Name="CustReplayProgID" Type="BSTR"> Mercury.DelphiTableSrv </Value>
```

Alternatively, you could create a single **Key** element (within the `CustomServers` **Key** element) to instruct QuickTest to use the grid test object extension for all custom controls mapped to a certain custom test object class. To do this, set the **Name** attribute of the **Key** element to `MC2CSMapping_<custom grid test object class name>`.

For more information on supporting custom grid controls, see “Creating Support for Custom Grid Controls” on page 30.

Mapping Test Object Classes to Inner Objects

Within the **MicTest** root element of the toolkit configuration XML file, under the **Key** element where **Name**="Test Objects", you define one **Key** element for each custom Delphi test object class that you want to map.

The **Name** attribute of this **Key** element must contain the name of the custom Delphi test object class. For example, the excerpt below is part of the test object configuration file that maps the **DelphiTrackBar** test object class to the **WinObject** test object class that serves as its inner object:

```
<?xml version="1.0"?>
<MicTest>
...
<Key Name="Test Objects">
  <!-- mapping for DelphiTrackBar -->
  <Key Name="DelphiTrackBar">
...
    </Key>
  </Key>
</MicTest>
```

Within the **Key** element for each custom Delphi test object class define the following **Value** elements:

Name Attribute	Type Attribute	Element Use	Element Value
BottomLevelObject	I4	Optional	Specifies whether test objects of this type can contain other objects. Possible values: ► 0 : Test object is not a bottom-level object. It can contain child objects. (Default) ► 1 : Test object is a bottom-level object. It cannot contain child objects.
CLSID	BSTR	Required	Required value: {A990252E-48C1-4d6c-9B55-4701BC29919C}

Name Attribute	Type Attribute	Element Use	Element Value
(Default)	BSTR	Required	The name of the custom Delphi test object class that you want to map.
ExtObjRecFilter	I4	Optional	<p>Specifies which Windows messages QuickTest passes to the Agent Object for recording.</p> <p>Possible values:</p> <ul style="list-style-type: none"> ➤ 0: Only messages addressed to the control's window are passed to the Agent Object for recording. (All other messages are ignored.) ➤ 1: All Windows messages are passed to the Agent Object for recording. ➤ 2: Only messages addressed to the control or its children are passed to the Agent Object for recording. (All other messages are ignored.) <p>If this element is not defined, all recording is handled by the inner object.</p>
InnerMicClass	BSTR	Required	<p>The name of the existing Standard Windows test object class that you want to use as the inner object.</p> <p>For a list of available test object classes, see the Standard Windows section of the <i>HP QuickTest Professional Object Model Reference</i>.</p>

Name Attribute	Type Attribute	Element Use	Element Value
InnerProgId	BSTR	Required	Mercury.<InnerMicClass> Exceptions: For the following InnerMicClass values, set InnerProgId as specified below: Window —Mercury.StdWindow Dialog —Mercury.WinDialog Static —Mercury.WinStatic Desktop —Mercury.MicDesktop
tag query name	BSTR	Required	Required value: delphi_name

Additional Definitions Within the Test Objects Key Element

This section describes additional optional elements that you can define within the **Key** element defined for each custom Delphi test object class:

- In some dialog boxes, QuickTest displays a list of available test object classes per environment (for example, the Define New Test Object dialog box, the Object Identification dialog box, and the Step Generator dialog box). If you want QuickTest to display your custom Delphi test object classes in these dialog boxes, you must add the following lines in the **Key** element for each custom Delphi test object class:

```
<Key Name="Info">
  <Value Name="package" Type="BSTR">DelphiPackage</Value>
</Key>
```

- If you want QuickTest to treat your custom Delphi test object class as a grid (table), you must add additional elements within the **Key** element that you define for this test object class. For more information, see “To instruct QuickTest to treat this type of test object as a table test object when creating checkpoints and output values:” on page 34.

- The Checkpoint Properties dialog box in QuickTest includes a **Checkpoint timeout** value (in seconds). You can customize the default checkpoint timeout value that is used when creating new checkpoints on your custom test object class. (Otherwise the default is **0**). To set the default checkpoint timeout value, add an additional **Key** element, like the one below, within the **Key** element defined for your custom Delphi test object class. Replace the number 10 in these lines with the default you want QuickTest to use:

```
<Key Name="CheckpointTimeout">  
  <Value Name="DefaultTimeout" Type="I4">10<!--0xA--></Value>  
</Key>
```

An Example for the Test Objects Key Element

The example below shows the entire **Key** element defined to map the **DelphiTrackBar** test object class to the existing **WinObject** test object class:

```
<?xml version="1.0"?>
<MicTest>
...
<Key Name="Test Objects">
  <!-- The mapping definitions for the DelphiTrackBar test object -->
  <Key Name="DelphiTrackBar">
    <!-- The name of the custom Delphi test object class being mapped.-->
    <Value Name="(Default)" Type="BSTR">DelphiTrackBar</Value>
    <!-- The name (and ProgId) of the Standard Windows inner object.-->
    <Value Name="InnerProgId" Type="BSTR">Mercury.WinObject</Value>
    <Value Name="InnerMicClass" Type="BSTR">WinObject</Value>
    <!-- ExtObjRecFilter Value element is set to 0: only messages addressed to the
        control's window are passed to the Agent Object for recording.
        All other messages are ignored. -->
    <Value Name="ExtObjRecFilter" Type="I4">0</Value>
    <!-- BottomLevelObject Value element is set to 1: Test objects of this class
        do not have child objects. -->
    <Value Name="BottomLevelObject" Type="I4">1</Value>
    <!-- These elements are defined identically for every mapped test object class.-->
    <Value Name="tag query name" Type="BSTR">delphi_name</Value>
    <Value Name="CLSID" Type="BSTR">
      {A990252E-48C1-4d6c-9B55-4701BC29919C}
    </Value>
    <Key Name="Info">
      <Value Name="package" Type="BSTR">DelphiPackage</Value>
    </Key>
  </Key>
</Key>
</MicTest>
```

An Example of a Complete Toolkit Configuration File

An example of a toolkit configuration file is shown below. This file maps the **TTrackBar** Delphi object to the new **DelphiTrackBar** test object class, and the **DelphiTrackBar** test object class to the existing **WinObject** test object class:

```
<?xml version="1.0"?>
<MicTest>
  <!-- Mapping the window class of the application controls to the
    custom Delphi test object classes that should represent them in QuickTest.-->
  <Key Name="Packages">
    <Key Name="DelphiPackage">
      <Key Name="ClassPatterns">
        <!-- Mapping the TTrackBar control to the DelphiTrackBar test object -->
        <Value Name="TTrackBar" Type="BSTR">DelphiTrackBar</Value>
      </Key>
    </Key>
  </Key>
  <!-- Mapping the custom Delphi test object classes to inner objects. -->
  <Key Name="Test Objects">
    <!-- The mapping definitions for the DelphiTrackBar test object -->
    <Key Name="DelphiTrackBar">
      <!-- The name of the custom Delphi test object class being mapped.-->
      <Value Name="(Default)" Type="BSTR">DelphiTrackBar</Value>
      <!-- The name (and ProgId) of the Standard Windows inner object.-->
      <Value Name="InnerProgId" Type="BSTR">Mercury.WinObject</Value>
      <Value Name="InnerMicClass" Type="BSTR">WinObject</Value>
      <!-- ExtObjRecFilter Value element is set to 0: only messages addressed to the
        control's window are passed to the Agent Object for recording. -->
      <Value Name="ExtObjRecFilter" Type="I4">0</Value>
      <!-- BottomLevelObject Value element is set to 1: Test objects of this class
        do not have child objects. -->
      <Value Name="BottomLevelObject" Type="I4">1</Value>
      <!-- These elements are defined identically for every mapped test object class.-->
      <Value Name="tag query name" Type="BSTR">delphi_name</Value>
      <Value Name="CLSID" Type="BSTR">
        {A990252E-48C1-4d6c-9B55-4701BC29919C}
      </Value>
      <Key Name="Info">
        <Value Name="package" Type="BSTR">DelphiPackage</Value>
      </Key>
    </Key>
  </Key>
</MicTest>
```

Index

A

- Agent Object 16, 24
- API information location 8
- assistive properties
 - defining 19

C

- checkpoints
 - customizing default timeout 52
 - supporting 27
- custom test object class
 - defining 19
 - displaying in QuickTest dialog boxes 51
- custom toolkit support set. *See* toolkit support set
- custom toolkit support. *See* support

D

- Define New Test Object dialog box 51
- description properties
 - defining 19

E

- extensibility code 24
 - API information location 8
 - template location 8
- extensibility SDK
 - install 7

F

- factory function 24

G

- grid controls, supporting 30
- grid test object extension 30, 32

H

- HP Software Support Web site 12
- HP Software Web site 12

I

- identification properties
 - supporting 27
- inner object 16, 23
- inner test object. *See* inner object

K

- Knowledge Base 12

M

- mapping
 - controls to test object classes 23, 47
 - custom test object classes to inner objects 23, 49
 - to an existing test object class 16, 23, 24, 35

N

- new test object class. *See* custom test object class

O

- Object Identification dialog box 51
- online documentation 10
- online resources 12

P

- published properties 26
 - for grid support 31

R

- Readme 10
- recording
 - filtering 50
 - support for 28

S

- samples 18
- Smart Identification properties
 - defining 19
- Smart identification, for extensibility test objects 19
- Step Generator dialog box 51
- support
 - activating 40
 - compiling 42
 - creating 16
 - deploying 40
 - developing 24
 - file locations 41
 - for grid (table) controls 30
 - for identification properties 27
 - for recording 28, 50
 - for table checkpoints 32
 - for table output values 32
 - for test object methods 27
 - modifying 43
 - removing 43
 - samples 18
 - Smart identification 19
 - step-by-step 35
 - See also* toolkit support set

T

- table checkpoints 32
- table controls, supporting 30
- table output values 32
- template location 8
- templates 18
- test object classes
 - defining 19
 - mapping 23
 - new 19
- test object methods, supporting 27
- toolkit support set 16
 - Delphi unit 24
 - test object configuration 19
 - toolkit configuration 23, 45, 54
 - See also* support
- Troubleshooting and Knowledge Base 12
- TStringDrawGrid sample 18
- TTrackBar sample 18