HPE UFT Delphi Add-in Extensibility

Software Version: 14.00

Developer Guide

Go to **HELP CENTER ONLINE**

http://uft-help.saas.hpe.com/

Hewlett Packard
Enterprise

Document Release Date: January 2017 | Software Release Date: January 2017

Legal Notices

Documentation Updates

The title page of this document 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: https://softwaresupport.hpe.com.

This site requires that you register for an HPE Passport and sign in. To register for an HPE Passport ID, go to https://softwaresupport.hpe.com and click **Register**.

Support

Visit the HPE Software Support Online web site at: https://softwaresupport.hpe.com

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

HPE Software online support 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 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 HPE 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 HPE Passport user and sign in. Many also require a support contract. To register for an HPE Passport ID, go to: https://softwaresupport.hpe.com and click **Register**.

To find more information about access levels, go to: https://softwaresupport.hpe.com/web/softwaresupport/access-levels.

HPE Software Solutions & Integrations and Best Practices

Visit **HPE Software Solutions Now** at https://softwaresupport.hpe.com/km/KM01702710 to explore how the products in the HPE Software catalog work together, exchange information, and solve business needs.

Visit **Hewlett Packard Enterprise Self-Solve Knowledge Search** at https://softwaresupport.hpe.com/group/softwaresupport to access a wide variety of best practice documents and materials.

Contents

Welcome to UFT Delphi Add-in Extensibility

HPE UFT 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 UFT Delphi Add-in.

This chapter includes:

About the UFT Delphi Add-in Extensibility SDK

The UFT 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 UFT 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, which includes the following:
  - A developer guide.
  - The UFT Test Object Schema Help.

  The Help is available online at: UFT Add-in Extensibility.

Accessing UFT Delphi Add-in Extensibility in Windows 8 Operating Systems

UFT files that were accessible from the **Start** menu in previous versions of Windows are accessible in Windows 8 from the **Start** screen or the **Apps** screen.

- **Applications (.exe files).** You can access UFT applications in Windows 8 directly from the **Start** screen. For example, to start UFT, double-click the **HP Unified Functional Testing** shortcut.
- **Non-program files.** You can access documentation from the **Apps** screen.

> **Note:** As in previous versions of Windows, you can access context sensitive help in UFT by pressing **F1**, and access complete documentation and external links from the **Help** menu.

About the UFT Delphi Add-in Extensibility Developer Guide

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

This guide assumes you are familiar with UFT functionality, and should be used together with the following documents:

- The API file, which contains comments and explanations. (**UFT installation folder>\dat\Extensibility\Delphi\AgentExtensibilitySDK.pas**)
- The extensibility code template, which contains comments and explanations. (**<UFT installation folder>\dat\Extensibility\Delphi\ExtensibilityImplementationTemplate.pas**)
- *HPE UFT Test Object Schema Help*, provided in the Delphi Add-in Extensibility Help. (**Start > All Programs > HPE Software > HP Unified Functional Testing > Extensibility > Documentation > Delphi Add-in Extensibility Help**)

These documents should also be used in conjunction with the following UFT documentation, available with the UFT installation (**Help > HP Unified Functional Testing Help** from the UFT main window):

- *HPE Unified Functional Testing User Guide*
- The Delphi section of the *HPE Unified Functional Testing Add-ins Guide*
- *HPE UFT Object Model Reference for GUI Testing*

Note:

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

Business components are part of HP Business Process Testing. For more information, see the *HPE Unified Functional Testing User Guide* and the *HPE Business Process Testing User Guide*.

When working in Windows 8, access UFT documentation and other files from the **Apps** screen.

To enable you to search this guide more effectively for specific topics or keywords, use the following options:

- **AND**, **OR**, **NEAR**, and **NOT** logical operators. Available from the arrow next to the search box.
- **Search previous results.** Available from the bottom of the **Search** tab.
- **Match similar words.** Available from the bottom of the **Search** tab.
- **Search titles only.** Available from the bottom of the **Search** tab.

Tip: When you open a Help page from the search results, the string for which you searched may be included in a collapsed section. If you cannot find the string on the page, expand all the drop-down sections and then use Ctrl-F to search for the string.

To check for recent updates, or to verify that you are using the most recent edition of a document, go to the HP Software Product Manuals Web site (http://h20230.www2.hp.com/selfsolve/manuals).

Who Should Read This Guide

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

To use this guide, you should be familiar with:

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

Additional Online Resources

The following additional online resources are available:

Resource	**Description**
Troubleshooting & Knowledge Base	The Troubleshooting page on the HPE Software Support Web site where you can search the Self-solve knowledge base. The URL for this Web site is https://softwaresupport.hpe.com/group/softwaresupport/search-result?keyword=.
HPE Software Web site	The HPE Software Web site. This 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 https://www.hpe.com/us/en/home.html

Chapter 1: Developing Support for Custom Delphi Controls

You can create a custom toolkit support set to extend the UFT Delphi Add-in, and teach Unified Functional Testing (UFT) 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 UFT 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 UFT 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 "Deploying the Toolkit Support Set" on page 30.

This chapter includes:

Glossary

- **Agent Object (AO).** A class written in Delphi and integrated into the UFT 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 UFT and the Delphi application being tested.
- **Custom toolkit.** A set of custom controls for which you implement UFT support.
- **Custom toolkit support set.** The set of files, units and objects that you create to extend UFT support for the controls in the custom toolkit.
- **Inner test object.** The Standard Windows test object class whose implementation UFT 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 UFT Delphi Add-in, and to teach UFT 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 UFT 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 16
- "Designing Your Toolkit Configuration XML File" on page 19
- " Designing Your Delphi Extensibility Code" on page 20
- "Step-by-Step Instructions for Supporting Custom Delphi Controls" on page 27

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 UFT 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 UFT to treat a custom grid control as a table. For more information, see "Creating Support for Custom Grid Controls" on page 24.

The samples are located under **<UFT 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 UFT's interaction with custom controls. To do this, create and run a UFT test on the sample custom control before and after deploying the sample toolkit support set to UFT. 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 UFT's interaction with the TTrackBar custom control:

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

   You can see that UFT 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 "Deploying the Toolkit Support Set" on page 30.
3. Use the UFT Object Spy to see how UFT 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, UFT uses a **DelphiTrackBar** test object (and a customized icon) to represent the control. The **DelphiTrackBar** test object supports the **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, UFT records test steps with the **Set** operation.

To open the Delphi Add-in grid extensibility sample project in Delphi Studio:

The Delphi Add-in grid extensibility sample project, **DelphiGridExtSample**, references the **TStringDrawGrid** custom control. Therefore, to successfully open **DelphiGridExtSample** in Delphi Studio, you must first register **TStringDrawGrid** in Delphi Studio.

To do this, perform the following steps:

1. In Delphi Studio, select **Component > Install Component**. The Install Component dialog box opens.
2. In the **Unit file name** box, specify the full path for the **StringDrawGrid.pas** file. The file is located in: **<UFT installation folder>\samples\DelphiGridExtSample\Application**.
3. Click **OK**. If confirmation or information dialog boxes open, click **Yes** and/or **OK**, as necessary.

   The Package dialog box opens.

   > **Note:** If Delphi Studio displays the **StringDrawGrid.pas** file for editing at this point instead of opening the Package dialog box, repeat these steps to start again.

4. Click **Compile**.
5. Close the Package dialog box, and click **Yes** in the save confirmation box that opens.
6. Open **DelphiGridExtSample**.

Designing Your Test Object Configuration XML File

In this file, you define any custom test object classes that you want UFT 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, the test object methods they support, their identification properties, and so on).

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 a test object method, it appears in the list of test object methods in UFT, 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.
- If you add an identification property, it appears in UFT 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.

In the test object configuration XML file, you create a **ClassInfo** element for each test object class that you want to define. 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 UFT 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 UFT opens, UFT 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.

To ensure the structural correctness of your test object configuration file, you can validate it against the **ClassesDefintions.xsd** file. This file is installed with UFT, in the **<UFT installation folder>\dat** folder. (For backward compatibility reasons, UFT still supports certain XML structures that do not pass validation against this XSD.)

For information on the structure and syntax of this XML, see the UFT Test Object Schema Help.

The sections below describe the information that you can include in a test object class definition.

Class Name and Base Class

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. A new test object class extends an existing DelphiUFT test object class, directly or indirectly. The base class may be a class delivered with UFT or a class defined using Delphi Add-in Extensibility.

By default, the base class is DelphiObject.

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

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

Generic Type

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 UFT to group this test object class with the **edit** test object classes.)

Generic types are used when filtering objects (for example, in the Step Generator's 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).

Test Object Operations

A list of operations for the test object class, including the following information for each operation:

- 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 operation description (shown in the Object Spy and as a tooltip in the Keyword 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 operation in the Keyword View or Editor, or when the **Operation Help** button is clicked for the operation in the Step Generator. The definition includes the Help file path and the relevant Help ID within the file.

Default Operation

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.

Identification Properties

A list of identification properties for the test object class. You can also define:

- 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 UFT.)
- The identification properties that are available for use in checkpoints and output values.
- The identification properties that are selected by default for checkpoints (in the UFT Checkpoint Properties dialog box).

When defining identification properties, it is recommended to start by copying existing identification properties from the base object in your schema or from similar objects in related toolkits.

Icon File

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

Help File

A context-sensitive Help topic to open when **F1** is pressed for the test object in the Keyword View or Editor. The definition includes the **.chm** Help file path and the relevant Help ID within the file.

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 UFT 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 UFT 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 UFT 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 21.

- **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 UFT Standard Windows test object classes that serve as inner objects. UFT 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 UFT GUI testing 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 *HPE UFT Object Model Reference for GUI Testing***.**

In some dialog boxes, UFT 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 UFT 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 "Understanding the Toolkit Configuration XML File Structure" on page 34.

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 UFT and the Delphi controls in the application being tested.
- One factory function that creates the appropriate Agent Object for each control. When UFT first interacts with a control, it calls the factory function to create the corresponding Agent Object.

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: **<UFT 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 UFT Delphi Add-in: **<UFT 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 the next page.
- 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 **<UFT 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 "Deploying the Toolkit Support Set" on page 30.

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

UFT interacts with the application's controls by setting and retrieving the published properties provided by the Agent Object and the control itself. UFT 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:

- **__QTPReplayMtd_**

  Use this prefix for all Agent Object properties designed to implement running UFT 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 24.
- **__QTPRecording**

  This property is used to implement support for the UFT recording capability. For more information, see "Supporting the UFT Recording Capability" on the next page.

The implementation for recording and running tests and components is described in the following sections.

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:** UFT uses only lowercase letters in identification property names. Therefore, the names of published properties that support identification properties must contain only lowercase letters (even if the identification property name in the test object configuration file contains uppercase letters).

A UFT 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 UFT 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 UFT.

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 UFT Recording Capability

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

During a recording session, when an event occurs on a custom Delphi control, UFT 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 UFT.
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 UFT.

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

   When UFT 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 **<UFT 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	lParam
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 37.

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 UFT 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 **<UFT 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 14.

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 UFT to Treat Your Custom Grid Control as a Table " on the next page.

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 UFT 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 **<UFT installation folder>\dat\Extensibility\Delphi\AgentExtensibilitySDK.pas**.)

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

> **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 UFT. 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 UFT to Treat Your Custom Grid Control as a Table

> **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.

In the toolkit configuration XML file, define the following:

- Instruct UFT 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 UFT to treat this type of test object as a table test object when creating checkpoints and output values.

To instruct UFT 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="CustReplayProgID"
                        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.

Alternatively, you could create a single **Key** element to instruct UFT 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 UFT 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 UFT 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 "Understanding the Toolkit Configuration XML File Structure" on page 34.

Step-by-Step Instructions for Supporting Custom Delphi Controls

This section guides you through the process of creating a UFT 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, create and deploy the toolkit configuration file as described in the procedure below, but compile the application you are testing with the **MicDelphiAgent.pas** module, as described in the Delphi section of the *HPE Unified Functional Testing 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 UFT tests and components. You can map your custom controls to existing Delphi test objects classes, or to custom test object classes that you define later in this task.

   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, **<UFT installation folder>\samples\DelphiExtSample\ToolkitSupportSet\TrackerSampleToolkitCfg.xml**, to create your toolkit configuration file. For information on the structure and syntax of this XML, see "Understanding the Toolkit Configuration XML File Structure" on page 34.

   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.

   > **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 UFT 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 *HPE Unified Functional Testing User Guide*.

2. **Deploy your toolkit configuration file**

   Copy the toolkit configuration XML file to the **<UFT 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 verifying the test object class mapping.

   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 16.

4. **Deploy your test object configuration file**

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

5. **Verify the test object class mapping**

   Open UFT, open a GUI test, and make sure that:

   - When you use the Object Spy, UFT recognizes your custom controls correctly.
   - When UFT 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 **<UFT 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 UFT Delphi Add-in precompiled agent and with your extensibility unit. To do this, perform the procedure described in "Compiling Your Extensibility Code" on page 32.

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 UFT Object Spy to view the properties that you defined when you created a basic extensibility Delphi unit.

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.

Chapter 2: Deploying the Toolkit Support Set

The final stage of extending UFT GUI testing support for a custom toolkit is deploying the toolkit support set. This means enabling UFT 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 UFT 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 UFT installed, to extend the UFT Delphi Add-in.

This chapter includes:

About Deploying the Custom Toolkit Support

To deploy a UFT Delphi Add-in Extensibility toolkit support set, you must place the XML files that you created in the correct locations on a computer with UFT installed, and compile the application you are testing with the UFT 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 the next page
- "Compiling Your Extensibility Code" on the next page

From the UFT user's perspective, after you deploy the toolkit support set on a computer on which UFT 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 UFT installation folder. The following table describes the appropriate location for each of the toolkit support files:

File	Location
Test Object Configuration file	• **<UFT installation folder>\dat\Extensibility\ Delphi** • **<UFT Add-in for ALM installation folder>\dat\Extensibility\ Delphi** (Optional. Required only if the folder exists, which means the UFT Add-in for ALM was installed independently from the ALM Add-ins page and not as part of the UFT installation.)
Toolkit Configuration file	**<UFT installation folder>\dat\Settings**
Icon files for custom test object classes (optional)	The file can be a **.dll** or **.ico** file, located on the computer on which UFT is installed, or in an accessible network location. Recommended location: **<UFT installation folder>\dat\Extensibility\Delphi\Toolkits\ <custom toolkit name>\res** Specify the location in the test object configuration file.
Help files for the test object classes (optional)	Must be a **.chm** file, located on the computer on which UFT is installed. Recommended location: **<UFT installation folder>\dat\Extensibility\Delphi\Toolkits\ <custom toolkit name>\help** Specify the location in the test object configuration file.

Note: In the test object configuration file, you can specify these locations using relative paths. For more information, see the *UFT Test Object Schema Help*

Compiling Your Extensibility Code

If you developed an extensibility Delphi unit, you must compile the application you are testing with the UFT Delphi Add-in precompiled agent and with your extensibility unit.

To do this, perform the following steps:

1. Add the <**UFT Installation folder>\dat\Extensibility\Delphi** folder to the search path of the application's project or copy the contents of the <**UFT 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 UFT 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 UFT for the changes to take effect.

Removing Deployed Support

If you want to remove support for a custom toolkit from UFT after it is deployed, you must delete its toolkit configuration file from: **<UFT 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: **<UFT installation folder>\dat\Extensibility\Delphi** (and **<UFT Add-in for ALM installation folder>\dat\Extensibility\Delphi** if relevant).

Chapter 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 UFT.

This chapter includes:

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 19:

- 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`):

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

The two **Key** elements within the **MicTest** element divide the XML file into two sections, each used for a different type of mapping:

- 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 24.
- 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 UFT 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" below
- "Mapping Test Object Classes to Inner Objects" on page 37

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 DelphiPackage, create a `ClassPatterns` **Key** element and, optionally, a `CustomServers` **Key** element.

The ClassPatterns Key Element

Within the **Key** element whose **Name** attribute is `ClassPatterns`, 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 UFT should use to represent the control in tests and components.

The CustomServers Key Element

If you are creating support for custom grid controls, create a **Key** element whose **Name** attribute is `CustomServers`. Within this **Key** attribute, 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 UFT 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 24.

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}`
(Default)	BSTR	Required	The name of the custom Delphi test object class that you want to map.
ExtObjRecFilter	I4	Optional	Specifies which Windows messages UFT passes to the Agent Object for recording. Possible values: • **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.) If this element is not defined, all recording is handled by the inner object.
InnerMicClass	BSTR	Required	The name of the existing Standard Windows test object class that you want to use as the inner object. For a list of available test object classes, see the **Standard Windows** section of the *HPE UFT Object Model Reference for GUI Testing*.

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, UFT 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 UFT 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 UFT 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 "Instructing UFT to Treat Your Custom Grid Control as a Table " on page 25. To instruct UFT to treat this type of test object as a table test object when creating checkpoints and output values.
- The Checkpoint Properties dialog box in UFT 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 UFT 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 UFT.-->
  <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>
```

Send Us Feedback

Let us know how we can improve your experience with the Developer Guide.

Send your email to: docteam@hpe.com

Hewlett Packard Enterprise

Java™ COMPATIBLE