

Extensibility Accelerator for HP Functional Testing

Software Version: 11.00

User Guide

Document Release Date: October 2010
Software Release Date: October 2010



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.

Copyright Notices

© 1992 - 2010 Hewlett-Packard Development Company, L.P.

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

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:

<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

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 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 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 register for an HP Passport ID, go to:

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

To find more information about access levels, go to:

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

Table of Contents

Chapter 1: Welcome to Extensibility Accelerator for HP Functional Testing	7
Extensibility Accelerator Overview	7
How Do I Find the Information That I Need?	8
Extensibility Accelerator Documentation Contents.....	11
QuickTest Professional Documentation Library	11
Additional Online Resources.....	12
Chapter 2: Introducing Extensibility Accelerator	13
Concepts	
QuickTest Web Add-in Extensibility - Overview	14
What Extensibility Accelerator Helps You Do	15
Reference	
Extensibility Accelerator at a Glance	17
Chapter 3: Installing the Extensibility Accelerator	25
Concepts	
Installed Components.....	26
Installation Prerequisites.....	27
Installing on a Non-QuickTest Computer	28
Chapter 4: Supporting a Custom Toolkit	29
Concepts	
Custom Toolkit Support Sets.....	30
When Are Your Changes Applied and Saved.....	32
Tasks	
How to Create or Update Support for a Custom Toolkit.....	34
How to Import an Existing Toolkit Support Set	35

Reference

Workflow Window 37
Class View 39
Project Explorer 41
Import Toolkit Support Set Dialog Box 43
Toolkit Support Properties Designer 44
Enumerations Designer 48

Chapter 5: Supporting a Custom Control 51

Concepts

Base Class Selection 52
JavaScript Function Debugging 54

Tasks

How to Create or Update Support for a Single Control 56
How to Map a Test Object Class to Application Controls 58
How to Design Test Object Class Operations 69
How to Design Test Object Class Identification Properties 72
How to Test and Debug Your Test Object Operation Support 75
How to Test and Debug Your Property Retrieval Function 78

Reference

General Tab (Test Object Class Designer) 81
Map to Controls Tab (Test Object Class Designer) 92
Operations Tab (Test Object Class Designer) 105
Properties Tab (Test Object Class Designer) 117
Debug Test Object Operation Dialog Box 127
Debug Property Retrieval Dialog Box 129

Chapter 6: Custom Toolkit Support Deployment 133

Concepts

Deployment Objectives 134
Deployment Destinations 136
Deployment File Structure 136

Tasks

How to Deploy a Toolkit Support Set 137

Welcome to Extensibility Accelerator for HP Functional Testing

This chapter includes:

- ▶ Extensibility Accelerator Overview on page 7
- ▶ How Do I Find the Information That I Need? on page 8
- ▶ Extensibility Accelerator Documentation Contents on page 11
- ▶ QuickTest Professional Documentation Library on page 11
- ▶ Additional Online Resources on page 12

Extensibility Accelerator Overview

QuickTest Web Add-in Extensibility enables you to develop support for testing third-party and custom Web controls that are not supported out-of-the-box by the QuickTest Professional Web Add-in.

Extensibility Accelerator for HP Functional Testing is an IDE that facilitates the design, development, and deployment of this support. This IDE is powered by the Microsoft Visual Studio Shell and therefore provides the same look and feel as Visual Studio, as well as many of the Visual Studio basic IDE functionalities.

Extensibility Accelerator provides a user interface that helps you define new test object classes, map those test object classes to the controls in your application, and teach QuickTest how to identify the controls, perform operations on the controls and retrieve their properties.

This information is stored in XML files and JavaScript files, which comprise a toolkit support set that you deploy to QuickTest to extend the Web Add-in to support the custom controls. For details on toolkit support sets, see "Custom Toolkit Support Sets" on page 30.

To use Extensibility Accelerator, you should be familiar with:

- QuickTest Professional (and the Web Add-in)
- XML
- JavaScript programming


How Do I Find the Information That I Need?


Within this guide, some subject areas are organized into topics. A topic contains a distinct module of information for that subject. The topics are generally classified according to the type of information they contain.



This structure is designed to create easier access to specific information by dividing the documentation into the different types of information you may need at different times.

Three main topic types are in use: **Concepts**, **Tasks**, and **References**. The topic types are differentiated visually using icons.

Topic Types

Topic Type	Description	Usage
Concepts 	General Concepts. Background, descriptive, or conceptual information.	Learn general information about what a feature does.
	Use-case Scenario Concepts. Real-life examples of when or why to use a specific product area.	Learn why or when you may want to use the feature.

Topic Type	Description	Usage
<p>Tasks</p> 	<p>Instructional Tasks. Step-by-step guidance to help you work with the application and accomplish your goals. Some task steps include examples, using sample data. Task steps can be with or without numbering:</p> <ul style="list-style-type: none"> ▶ Numbered steps. Tasks that are performed by following each step in consecutive order. ▶ Non-numbered steps. A list of self-contained operations that you can perform in any order. 	<ul style="list-style-type: none"> ▶ Learn about the overall workflow of a task. ▶ Follow the steps listed in a numbered task to complete a task. ▶ Perform independent operations by completing steps in a non-numbered task.
	<p>Exercise Tasks. Step-by-step instructions for a task using a sample application or sample data.</p>	<p>Follow the steps in these topics to practice the workflow of a task.</p>
	<p>Use-case Scenario Tasks. Examples of how to perform a task for a specific situation.</p>	<p>Learn how a task could be performed in a realistic scenario.</p>

Topic Type	Description	Usage
Reference 	General Reference. Detailed lists and explanations of reference-oriented material.	Look up a specific piece of reference information relevant to a particular context.
	User Interface Reference. Specialized reference topics that describe a particular user interface in detail. Pressing F1 in the product area generally open the user interface reference topics.	Look up specific information about what to enter or how to use one or more specific user interface elements, such as a window, dialog box, or wizard.
Troubleshooting and Limitations 	Troubleshooting and Limitations. Specialized topics that describe commonly encountered problems and their solutions, and list limitations of a feature or product area.	Increase your awareness of important issues before working with a feature, or if you encounter usability problems in the software.

Extensibility Accelerator Documentation Contents

This user guide explains how to use Extensibility Accelerator. You can open it by selecting **Help > Extensibility Accelerator User Guide** or pressing F1 on extensibility-specific windows.

For details on Web Add-in Extensibility, see the *HP QuickTest Professional Web Add-in Extensibility Developer Guide* (**Help > Web Add-in Extensibility Developer Guide**).

This guide is also available in printer-friendly (PDF) format, in the **<Extensibility Accelerator installation>\Help** folder.

For details on the Visual Studio standard functionalities and windows in Extensibility Accelerator, see the online MSDN Visual Studio Help (<http://msdn.microsoft.com/en-us/library/aa187919.aspx>). If you are connected to the Internet while using Extensibility Accelerator, you can access this Help by selecting **Help > Contents** or pressing F1 on standard windows in the product.

QuickTest Professional Documentation Library

The QuickTest Professional Documentation Library provides a single-point of access for QuickTest Professional documentation.

You can access the QuickTest Professional Documentation Library by using the following:

- ▶ In QuickTest, select **Help > QuickTest Professional Help**.
- ▶ In the Start menu on the QuickTest computer, select **Program Files > HP QuickTest Professional > Documentation > HP QuickTest Professional Help**.
- ▶ Click in selected QuickTest windows and dialog boxes or press F1.
- ▶ View a description, syntax, and examples for a QuickTest test object, method, or property by placing the cursor on it (in QuickTest) and pressing F1.

Additional Online Resources

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/hpsupport.

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>

HP Software Web site accesses the HP 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. Choose **Help > HP Software Web site**. The URL for this Web site is www.hp.com/go/software.

1

Introducing Extensibility Accelerator

This chapter includes:

Concepts

- ▶ QuickTest Web Add-in Extensibility - Overview on page 14
- ▶ What Extensibility Accelerator Helps You Do on page 15

Reference

- ▶ Extensibility Accelerator at a Glance on page 17

Troubleshooting and Limitations - Extensibility Accelerator on page 23

Concepts

QuickTest Web Add-in Extensibility - Overview

The QuickTest Professional Web Add-in provides built-in support for a number of commonly used Web controls. The add-in provides test object classes, operations (methods), and properties that can be used when testing Web applications.

Web Add-in Extensibility enables you to develop support for testing third-party and custom Web controls that are not supported out-of-the-box by the QuickTest Professional Web Add-in.

When QuickTest learns an object in an application, it recognizes the object as belonging to a specific test object class. This determines the identification properties and test object operations of the test object that represents the application's object in QuickTest.

The type of test object that QuickTest uses might not have certain characteristics that are specific to the Web control you are testing. Therefore, when you try to create test steps with this test object, the available identification properties and test object operations might not be sufficient.

For example, consider a custom Web control that is a special type of table that QuickTest recognizes as a plain `WebElement`. `WebElement` test objects do not support **GetCellData** operations. To create a test step that retrieves the data from a cell in the table, you would need to create test objects to represent each cell in the table, and create a complex test that accesses the relevant cell's test object to retrieve the data.

To create support for Web controls using Web Add-in Extensibility, you create new test object classes, based on the Web Add-in ones. You can then direct QuickTest to recognize each control as belonging to a specific test object class, and you can specify the behavior of each test object class. This enables you to create tests that fully support the specific behavior of your custom Web controls.

For more details on Web Add-in Extensibility, see the *HP QuickTest Professional Web Add-in Extensibility Developer Guide*, available in the <Extensibility Accelerator installation>\Help folder.

What Extensibility Accelerator Helps You Do

To extend the QuickTest Web Add-in to support custom Web toolkits, you create **custom toolkit support sets** and deploy them to QuickTest. The toolkit support set is comprised of XML configuration files and JavaScript functions. For details, see "Custom Toolkit Support Sets" on page 30.

Creating support for a custom toolkit is comprised of the following stages:

1 Planning how you want QuickTest to operate on your controls.

This is a preliminary stage that you perform by using QuickTest on your application and determining what aspects of QuickTest's behavior you would like to customize.

For details, see the section on planning QuickTest support for your toolkit in the *HP QuickTest Professional Web Add-in Extensibility Developer Guide*.

2 Creating and defining the test object classes, operations, properties and settings.

The Extensibility Accelerator IDE simplifies creating and editing the XML files required for a toolkit support set, by providing designers in which you specify the relevant information. This enables you to invest your main efforts in the development of the JavaScript functions.

3 Writing and debugging JavaScript implementation functions.

The JavaScript functions that you write as part of the toolkit support set enable QuickTest to work with your custom Web controls. Extensibility Accelerator creates the necessary JavaScript files and adds stubs for the functions that you must implement. In addition, Extensibility Accelerator provides JavaScript editing capabilities and debugging tools to facilitate writing these functions.

4 Deploying the toolkit support so it can be used on QuickTest.

Extensibility Accelerator deployment capabilities enable you to automatically deploy your new toolkit support set to QuickTest or to package it so that you can share it with other QuickTest users.

For task details, see "How to Create or Update Support for a Custom Toolkit" on page 34.

Reference

Extensibility Accelerator at a Glance

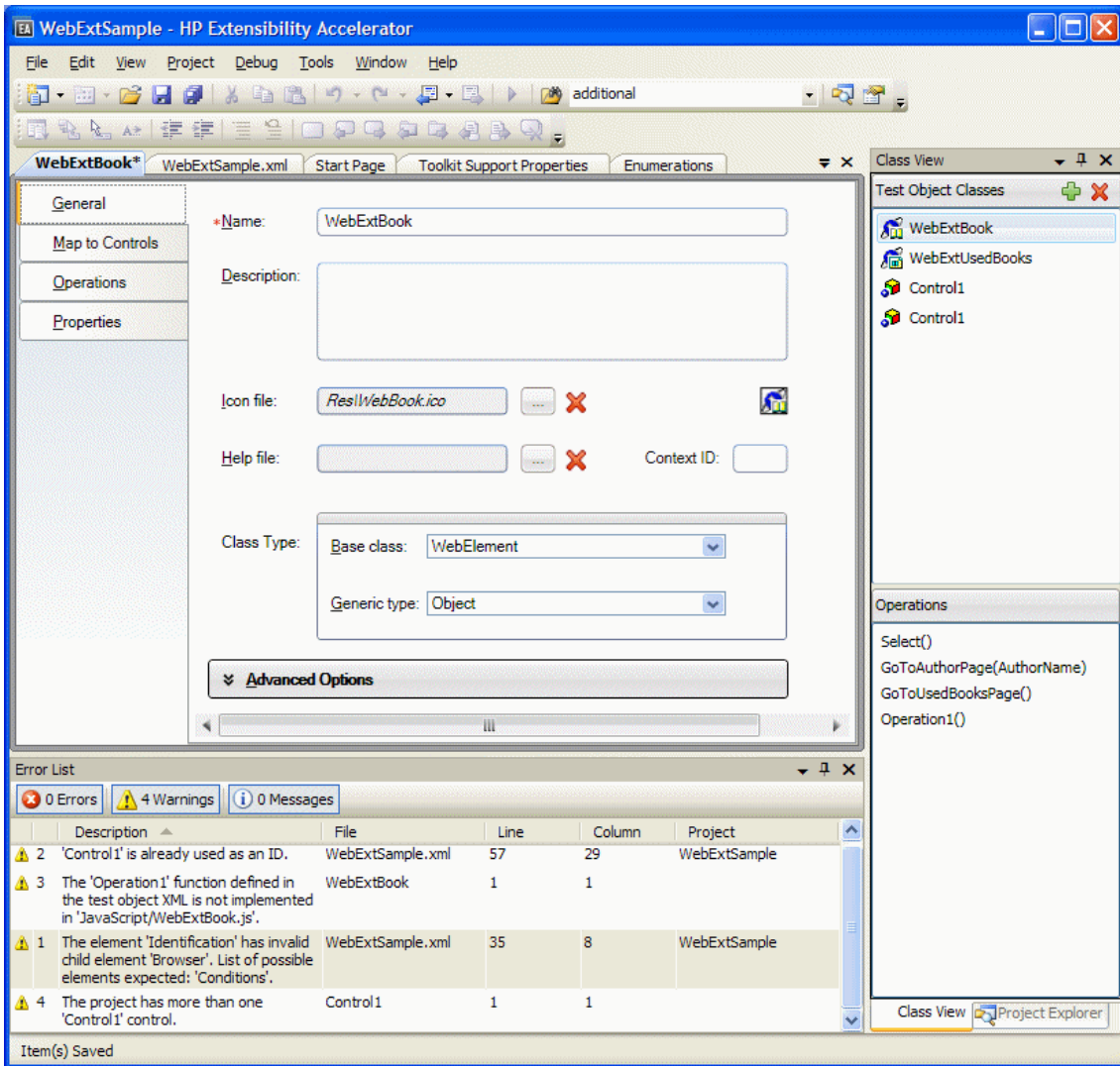
Extensibility Accelerator for HP Functional Testing is a Visual Studio-like IDE that facilitates the design, development, and deployment of Web Add-in Extensibility toolkit support sets.

You develop a toolkit support set in an extensibility project. The main Extensibility Accelerator functionalities are available only when you have a project open. You can open only one project at a time.

This section introduces the Extensibility Accelerator window and lists the main areas it comprises and how you can use them.

You can customize the appearance of this window by moving and docking the windows it contains and by customizing toolbars, in the same ways as you would in Visual Studio.

Note: If your computer's display is set to the Windows Classic style, the colors and appearance of some tabs will differ from the intended design.



The Extensibility Accelerator window contains:

- ▶ Main Area (described on page 19)
- ▶ Additional Windows (described on page 21)
- ▶ Menus and Toolbars (described on page 22)

Main Area

The main area of the Extensibility Accelerator window can display the following:

Start Pages

Extensibility Accelerator displays two Start Pages:

- ▶ **The Extensibility Accelerator Start Page.** Displayed when no projects are open. This page describes the Extensibility Accelerator product, and provides access to some basic functionalities, such as creating a new project, opening recent projects or sample projects, or viewing a movie about Extensibility Accelerator.
- ▶ **The Project Start Page.** Displayed when you create or open a project. It explains the steps that comprise defining a test object class. These steps correspond to the tabs in the test object class designer.

In addition, this Start Page provides a link that you can click to create a new test object class.

Toolkit Support Properties Designer

This designer enables you to define settings that affect how QuickTest treats this toolkit support set. The information that you define in this designer is stored in the toolkit support set's XML files.

For details, see "Toolkit Support Properties Designer" on page 44.

Test Object Class Designer

The test object class designer is the main designer in Extensibility Accelerator. It enables you to define all of the details about the test object class that you want QuickTest to use for the custom control. For example, the name of the test object class, the types of controls that it represents, and the operations and properties that it supports.

Make sure to visit all of the tabs in this designer.

For details, see:

- "General Tab (Test Object Class Designer)" on page 81.
- "Map to Controls Tab (Test Object Class Designer)" on page 92
- "Operations Tab (Test Object Class Designer)" on page 105
- "Properties Tab (Test Object Class Designer)" on page 117

Enumerations Designer

This designer enables you to define lists of values that can be used for test object operation arguments or return values. This information is stored in the test object configuration XML file.

For details, see "Enumerations Designer" on page 48.

JavaScript Editor

When you open a JavaScript file, it opens in an editor that provides standard JavaScript editing capabilities, such as full syntax highlighting and IntelliSense features. For details, see the online MSDN Visual Studio Help.

The `_elem` token represents the control or element that QuickTest is handling when the function runs. In Extensibility Accelerator, IntelliSense for the `_elem` token provides the methods and properties available for an application control that matches the identification rules defined for your test object class.

IntelliSense for the `_elem` token is available only when you are editing the default implementation file for the test object class.

For IntelliSense to be available for the `_elem` token, an application must be open and fully loaded, and a control of the type you are supporting must be visible. (You must run Extensibility Accelerator and open a project before you open the Web browser.)

To make sure that a control of the correct type is available, make sure that the rules displayed in the rule editor for your test object class correctly identify at least one control in your application. To do this, you can click **Test All Rules** in the Map to Controls Tab (Test Object Class Designer) described on page 92 and verify that a control is highlighted in the application.

XML Editor

When you open an XML file, it opens in an editor that provides standard XML editing capabilities, such as color-coded display and syntax completion features. For details, see the online MSDN Visual Studio Help.

This editor can also provide XML IntelliSense and validation based on the relevant schemas. Extensibility Accelerator provides the XML schemas that you need for editing the toolkit configuration XML file and the test object configuration XML file (<Extensibility Accelerator installation>\dat\Toolkit.xsd and <Extensibility Accelerator installation>\dat\ClassesDefinitions.xsd respectively).

Caution: In extensibility projects, there is a strong connection between file names, the location of the files in the project, and the content of different files. Therefore, if you edit XML files manually, make sure you do not create discrepancies.

Additional Windows

In addition to the main area, the following windows are available. (You can show them by selecting them in the **View** menu.)

- **Workflow.** Displays the development stages required to create and deploy support for a custom toolkit, highlighting the current stage.

You can click on the relevant stage to create a new test object class or deploy the toolkit support set. For details, see "Workflow Window" on page 37.

- **Class View.** Displays the test object classes defined in the open project, and the operations defined for each test object class.

This window also provides access to common activities such as adding or editing classes and editing or debugging operations. For details, see "Class View" on page 39.

This window is available only when a project is open.

- ▶ **Project Explorer.** Displays the folders and files that make up the open extensibility project.

You can double-click files in the project tree to open them. For details, see "Project Explorer" on page 41.

- ▶ **Error List.** Displays error, warning, or information messages when mandatory data is missing in your project or if conflicts or discrepancies are found between information in the different files.

Standard Visual Studio windows are available as well, such as: Task List and Find Results, and debugging related windows such as Breakpoints and Command Window.

Menus and Toolbars

The menus and toolbars available in Extensibility Accelerator are similar to the ones in Visual Studio, and change according to the type of designer or file that you are working in. For example:

- ▶ The **File**, **Edit**, **View**, **Tools**, **Window**, and **Help** menus are always available.
- ▶ The **Project** menu is available when a project is open.
- ▶ The **XML** menu is available when an XML file is open.

Troubleshooting and Limitations - Extensibility Accelerator

This section describes troubleshooting and limitations for Extensibility Accelerator.

Why is the XML editor providing only generic XML IntelliSense?

If you imported your toolkit support set from another computer, the XML file might be referencing the schema file in an incorrect location.

Check the reference to the **.xsd** file at the beginning of your XML file (in the **xsi:noNamespaceSchemaLocation** attribute of the **TypeInformation** element).

If the reference is incorrect, do one of the following:

- ▶ Manually correct the reference to refer to the **.xsd** file in the correct location (in the **<Extensibility Accelerator installation>\dat** folder)
- ▶ Remove the reference line, save the file and reopen it. Extensibility Accelerator inserts a reference to the correct schema file.

Extensibility Accelerator supports comments in XML files only in the following situations

- ▶ The test object class designer for the test object class whose XML section you are modifying is open. (Relevant when adding comments manually. For example, if you want to add comments related to the **GWTPushButton** test object in the XML editor, you must also open the **GWTPushButton** test object class designer.)
- ▶ The comments appear directly before the opening tag of an element. However, comments before the following element types are not supported:
 - ▶ In the test object configuration file: **Description, AdditionalInfo, Documentation, IdentificationProperties.**
 - ▶ In the toolkit configuration file: **Controls, Methods, HTMLTags, Settings, Variable** (within a **Controls\Settings** element).

If you import a support set with XML files containing comments that are not supported, the comments are not included in the Extensibility Accelerator project's XML files. If you add such comments manually to an existing XML file in the XML editor, the comments are discarded.

Naming Rules in the Extensibility Accelerator Designers

In Extensibility Accelerator, **Name** edit boxes support only English letters, numeric characters, hyphens, and underscores, and must begin with a letter. **Property** names can contain spaces in addition to these characters.

If you enter unsupported characters in an edit box, they are ignored.

Description elements inside Argument elements are not supported in the test object configuration XML file

If you import a support set with an XML file that contains these, or you add such elements manually to an existing XML file in an Extensibility Accelerator project, those elements are deleted from the XML file.

Workaround: Document argument descriptions in the **Description** element inside the **Operation** element. However, keep in mind that the operation description is displayed in QuickTest tooltips.

Documentation Limitations

- Context-sensitive Help (opened by pressing F1) is not supported for generic Visual Studio IDE when you are not connected to the Internet.
- F1 is not supported for the Project Explorer window and the Control Selection dialog box.

2

Installing the Extensibility Accelerator

This chapter includes:

Concepts

- ▶ Installed Components on page 26
- ▶ Installation Prerequisites on page 27
- ▶ Installing on a Non-QuickTest Computer on page 28

Concepts

Installed Components

The Extensibility Accelerator for HP Functional Testing installation program installs the following:

► **Extensibility Accelerator.**

You can access this program from the icon installed on your desktop or from the QuickTest Professional program group (**Start > Program Files > HP QuickTest Professional > Extensibility Accelerator**).

► **The QuickTest Professional Web Add-in Extensibility API**, including XSD files, JavaScript files with global functions, and so on.

► **A demo movie.**

This movie demonstrates the basic capabilities of Extensibility Accelerator by walking you through the process of customizing QuickTest's support for a specific Web control.

You can access the demo movie from the Extensibility Accelerator Start Page or Help menu (**Help > Demo Movie**).

► **Documentation.**

The *HP Extensibility Accelerator for HP Functional Testing User Guide*, and the *HP QuickTest Professional Web Add-in Extensibility Developer Guide*, in both online Help format and printable (PDF) format.

You can access the guides directly in the <**Extensibility Accelerator installation folder**>**Help** folder or you can open the online Help from the Extensibility Accelerator Help menu.

► **Sample Web Add-in Extensibility projects.**

These projects contain completed toolkit support sets, which were developed to provide support for some public Web 2.0 toolkits.

The samples are installed in the %ALLUSERSPROFILE%\Documents\ExtAccTool\Samples folder, and are also accessible from the Extensibility Accelerator Start Page. You can open these projects and browse through the files, functions, and comments to learn more about how these support sets are designed. You can also modify these sample projects and experiment with them. Backup copies of the sample projects are installed in the <Extensibility Accelerator installation folder>\Help\Samples folder.

Installation Prerequisites

The following prerequisites must be installed before you can install the Extensibility Accelerator for HP Functional Testing:

- ▶ .NET Framework v3.5 SP1
- ▶ Microsoft Visual C++ 2008 Run-time Components
- ▶ Visual Studio 2008 Shell (isolated mode) with SP1 Redistributable Package

The Extensibility Accelerator for HP Functional Testing installation is available in two formats: A large installation package that includes the installation programs for these prerequisites and a smaller package that includes only Extensibility Accelerator.

The installation is available from:

- ▶ The **Add-in Extensibility and Web 2.0 Toolkits** option in the QuickTest Professional setup program. (Large installation package)
- ▶ www.hp.com/go/functionaltestingWeb2 (Large and small installation packages)

If you run the installation from the large package, the program runs the installation programs for any missing prerequisites before installing Extensibility Accelerator.

To run the installation from the smaller package, first ensure that you have all of the prerequisites installed.

Note: If you have Visual Studio 2008 installed on your computer, you must also have Service Pack 1 installed before you can install Extensibility Accelerator.

Installing on a Non-QuickTest Computer

QuickTest Professional does not have to be installed on the computer in order to install and use Extensibility Accelerator to create toolkit support sets for your Web controls.

Extensibility Accelerator includes a debugging mechanism for test object operations that you design, which simulates running the operations using QuickTest. You can use this functionality even on a computer where QuickTest is not installed. The debugging mechanism enables you to debug part of your JavaScript functions locally, without deploying the toolkit support set to QuickTest.

Installing the Extensibility Accelerator on a QuickTest computer enables you to automatically deploy your toolkit support set to QuickTest, making it simpler to complete the debugging and testing of your toolkit support set.

3

Supporting a Custom Toolkit

This chapter includes:

Concepts

- ▶ Custom Toolkit Support Sets on page 30
- ▶ When Are Your Changes Applied and Saved on page 32

Tasks

- ▶ How to Create or Update Support for a Custom Toolkit on page 34
- ▶ How to Import an Existing Toolkit Support Set on page 35

Reference

- ▶ Workflow Window on page 37
- ▶ Class View on page 39
- ▶ Project Explorer on page 41
- ▶ Import Toolkit Support Set Dialog Box on page 43
- ▶ Toolkit Support Properties Designer on page 44
- ▶ Enumerations Designer on page 48

Concepts

Custom Toolkit Support Sets

To extend the QuickTest Web Add-in to support custom Web toolkits, you create **custom toolkit support sets** and deploy them to QuickTest. The toolkit support set is comprised of XML configuration files and JavaScript functions.

The XML configuration files define the test object classes that you create to support the custom Web controls and map them to the controls. In addition, they define how QuickTest operates on the custom controls. The JavaScript functions provide an interface between QuickTest and the application being tested, retrieving information about the control and performing operations on it.

In Extensibility Accelerator, when you create an extensibility project, the project contains the mandatory files for a toolkit support set. A project can contain the following types of files:

- ▶ **XML files.** Extensibility Accelerator provides designers (such as the test object class designer) to guide and assist you in editing information stored in the test object configuration and toolkit configuration XML files.
- ▶ **JavaScript files.** For each test object class that you create, Extensibility Accelerator creates a corresponding JavaScript file. Within the file, Extensibility Accelerator creates function stubs for the functions that you have to implement. You can jump to these files from Extensibility Accelerator to add the code necessary for implementing these functions.
- ▶ **Additional files.** Extensibility Accelerator provides **Import** buttons, within some of the designers, to add additional files to the project.

For task details, see "How to Create or Update Support for a Custom Toolkit" on page 34.

A toolkit support set contains the following:

- **A test object configuration XML file.** This file describes the test object classes that you create to support the custom controls, and the identification properties and test object operations that need to be supported for those test objects.

For details on the structure and syntax of this XML, see the QuickTest Test Object Schema Help, available in the Extensibility Accelerator Help.

- **A toolkit configuration XML file.** This file maps the test object classes that you create to the relevant controls, and provides implementation details for how QuickTest operates on the control. Some implementation details are contained in this configuration file, others are in JavaScript files that this file references.

For details on the structure and syntax of this XML, see the Toolkit Configuration Schema Help, available in the Extensibility Accelerator Help.

- **JavaScript files.** These files contain the implementation functions referenced from the toolkit configuration XML file. QuickTest calls these functions to retrieve information from or perform operations on the custom controls.

For details, see the section on designing JavaScript functions for your toolkit support set in the *HP QuickTest Professional Web Add-in Extensibility Developer Guide*.

- **Icon and Help files (Optional).**

The icon files contain icons used in QuickTest to represent your test object classes. (Supported file types: **.ico**, **.exe**, **.dll**)

The Help files are used for context-sensitive Help for your test object classes and their methods and properties. (Supported file type: **.chm**)

For details on the structure of an extensibility project, see "Project Explorer" on page 41.

For details on the structure of a toolkit support set deployed to QuickTest, see "Deployment File Structure" on page 136.

When Are Your Changes Applied and Saved

Information that you define in the designers provided by Extensibility Accelerator is stored in different files in your toolkit support set. There is a strong connection between the information in the different files, therefore it is important that you save your changes frequently and consistently. This is especially true when you make changes that affect more than one file or designer, such as adding or renaming test object classes or operations.

Using the Save Commands

- ▶ **Save All.** Saves any changes you made in designers or file editors. All of the XML and JavaScript files are saved.
- ▶ **Save (file).** If you use the **Save** command when a file editor is selected, only the file currently open in the editor is saved. If you subsequently discard corresponding changes in another file, this can result in discrepancies within your toolkit support set. These discrepancies are reported in the Error List window.
- ▶ **Save (designer).** If you use the **Save** command when a designer is selected, the changes that you made in the designer are updated in all relevant XML and JavaScript files.

Considerations When Editing Multiple Test Object Classes

The XML information for all test object classes is stored in one XML file. If the XML file is closed while you make changes in the designers, the relevant changes are applied to the file only when you save them. This ensures that only information pertaining to the designer on which you performed the **Save** command is saved.

However, if the XML file is open while you make changes in the designers, the changes are immediately written to the XML editor (not to the file). If change more than one test object class and then save one, the XML file in the editor is saved with the changes made for all of the test object classes. If discrepancies are subsequently created, they are reported in the Error List window.

It is therefore recommended to keep the XML files closed while you work in the Extensibility Accelerator designers, or to make sure to finish changing one test object class before beginning to change another.

Changes Made Automatically to JavaScript Files

When you modify definitions in the Properties Tab or the Operations Tab (Test Object Class Designer), function stubs may be added or updated in the relevant JavaScript files. The JavaScript files are modified when you save your changes, or when you leave the designer and move the focus to another designer, file, or window.

If a JavaScript file is open when the functions in it are modified, the changes are made in the JavaScript editor. They are saved to the file system only when you use the **Save** command.

If the files are closed, the changes are made directly in the file system. If you later decide not to save the changes that you made in the test object designer, the changes made in the JavaScript files remain.

Tasks

How to Create or Update Support for a Custom Toolkit

This task describes the overall process of creating, designing, and deploying a toolkit support set using Extensibility Accelerator.

This task includes the following steps:

- "Prerequisites - Plan your support" on page 34
- "Open, create, or import an extensibility project" on page 34
- "Define the toolkit support properties - Optional" on page 35
- "Create or update support for a single control" on page 35
- "Deploy the toolkit support to QuickTest, or package it for distribution" on page 35

1 Prerequisites - Plan your support

For details, see the section on planning QuickTest support for your toolkit in the *HP QuickTest Professional Web Add-in Extensibility Developer Guide*.

2 Open, create, or import an extensibility project

Create a new extensibility project, or open an existing one.

- To open an existing project, select **File > Open > Project/Solution** and browse to the **.weproj** project file.
- To create a new empty project, select **File > New > Project**, and use the Web Add-in Extensibility template available in the New Project dialog box that opens.

In the project name, use only English letters, numeric characters, or hyphens.

- To import an existing toolkit support set and create a new extensibility project, select **File > Import Toolkit Support Set**. For details, see "How to Import an Existing Toolkit Support Set" on page 35.

Note: You can have only one project open at a time.

3 Define the toolkit support properties - Optional

For details, see "Toolkit Support Properties Designer" on page 44.

4 Create or update support for a single control

For details, see "How to Create or Update Support for a Single Control" on page 56.

5 Deploy the toolkit support to QuickTest, or package it for distribution

For details, see "How to Deploy a Toolkit Support Set" on page 137.

How to Import an Existing Toolkit Support Set

This task describes how to import an existing Web Add-in Extensibility toolkit support set. This creates a new Extensibility Accelerator project that contains the support set's files.

Note: This task is part of a higher-level task. For details, see "How to Create or Update Support for a Custom Toolkit" on page 34.

This task includes the following steps:

- "Prerequisites" on page 36
- "Import the toolkit support set" on page 36
- "Results" on page 36

1 Prerequisites

The Web Add-in Extensibility toolkit support set that you want to import must have the structure of a toolkit support set deployed to QuickTest, as described in the section on deploying toolkit support sets in the *HP QuickTest Professional Web Add-in Extensibility Developer Guide*.

This means that the XML files are in specific locations, and the locations of the rest of the files, such as JavaScript files, icon files, and Help files, are specified in the XML files.

A standard toolkit support set will have the following structure:

```

Parent folder (e.g. <QuickTest installation>\dat\Extensibility\Web)
|
|---<ToolkitName>TestObjects.xml file
|---Toolkits folder:
|
|   |---<ToolkitName> folder
|   |
|   |   |---<ToolkitName>.xml file
|   |   |---JavaScript files (optionally stored in JavaScript subfolder)
|   |   |---Res folder with icon files (optional)
|   |   |---Help folder with .chm files (optional)

```

2 Import the toolkit support set

Select **File > Import Toolkit Support Set** and use the Import Toolkit Support Set Dialog Box that opens to browse to the toolkit support set and import it.

3 Results

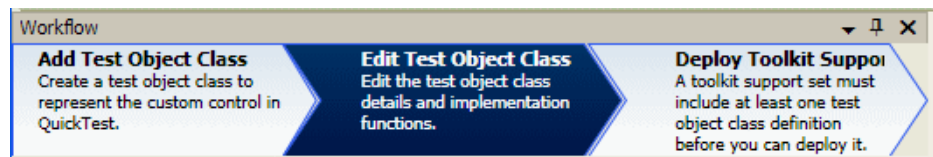
The toolkit support set files are copied into a newly created Extensibility Accelerator project. For details on the project file structure, see "Project Explorer" on page 41.

If **Open imported project** was selected in the Import Toolkit Support Set Dialog Box, the new extensibility project opens. Otherwise, the dialog box remains open, enabling you to import additional support sets.

Reference

Workflow Window

This window guides you through the workflow you need to follow when working in an Extensibility Accelerator project. It displays the development stages required to create and deploy support for a custom toolkit, highlighting the current stage.



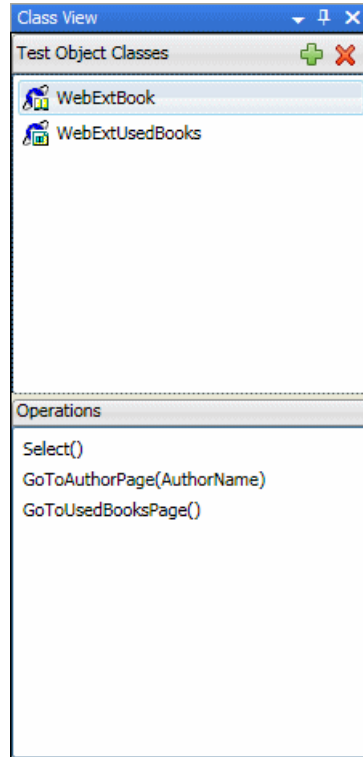
To access	Select View > Workflow
Relevant tasks	"How to Create or Update Support for a Custom Toolkit" on page 34

User interface elements are described below:

UI Elements	Description
<p>Add Test Object Class</p>	<p>Clicking in this area creates a new test object class. For more details, see "How to Create or Update Support for a Single Control" on page 56. Highlighted when: No test object classes are defined in the project.</p>
<p>Edit Test Object Class</p>	<p>When this area is highlighted, it indicates that you are in the developing stage of your project. During this stage, you can create additional test object classes, edit the test object class details, implementation files, toolkit support properties, and so on. Highlighted when: At least one test object class is defined in the project.</p>
<p>Deploy Toolkit Support</p>	<p>Clicking in this area deploys the toolkit support set to a .zip file. For more details, see "How to Deploy a Toolkit Support Set" on page 137. Available when: At least one test object class is defined in the project. Highlighted when: A deploy command is in progress.</p>

Class View

This window displays the test object classes defined in the open project, and the operations defined for each test object class.



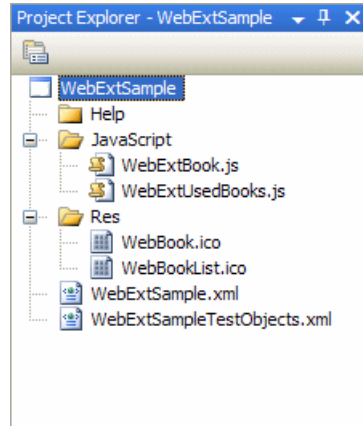
To access	When a project is open, select View > Class View
Important information	<p>This window enables you to do the following:</p> <ul style="list-style-type: none"> ▶ Add or delete test object classes. ▶ Open the test object class designer. ▶ Open the implementation code for a test object operation. ▶ Start a debug session for a test object operation.

User interface elements are described below:

UI Elements	Description
<p>Test Object Classes</p>	<p>The list of test object classes defined in the open project.</p> <p>In this area, you can:</p> <ul style="list-style-type: none"> ▶ Use the toolbar buttons to add and delete test object classes. ▶ Double-click a test object class to open its designer.
<p>Operations</p>	<p>The operations defined for the selected test object class.</p> <p>In this area, you can:</p> <ul style="list-style-type: none"> ▶ Double-click an operation to open the test object class designer. It opens to the Operations tab (described on page 105) with the relevant operation selected. ▶ Right-click and select Implementation Code to open the file containing the JavaScript implementation function for the operation. The file opens to the relevant function. ▶ Right-click and select Debug to open the Debug Test Object Operation Dialog Box (described on page 127). The test object class and operation are automatically selected in the dialog box.

Project Explorer

This window displays the folders and files that make up the open Web Add-in extensibility project. You can double-click files in the project tree to open them.



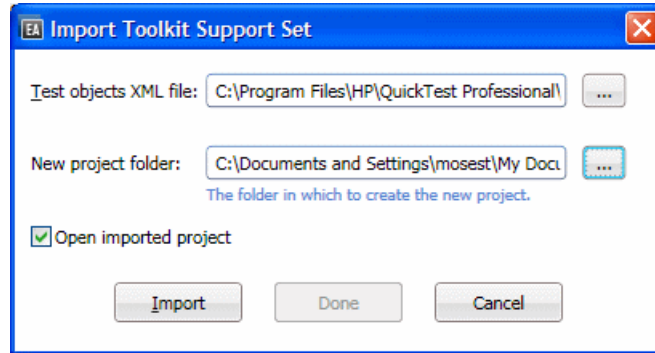
To access	Select View > Project Explorer
Important information	<p>Some standard Visual Studio Shell commands are available when you right-click items in the project tree. In extensibility projects, there is a strong connection between file names, file content, and the location of the files. Therefore, commands to add or rename files in the project are not available, as this should be done using the test object class designers.</p> <p>For the same reason, you should also avoid using the Exclude From Project command, which is available when you right-click.</p>

The Project Explorer tree contains the following:

Tree Node	Description
<Project Name>	The Web Add-in Extensibility project's top level node.
Help folder	A folder that optionally contains Help (.chm) files. The files in this folder are referenced from the project's test object configuration XML file.
<Help files>	Optional. Help files for QuickTest to use for context-sensitive Help for the test object classes you define. Supported file type: .chm
JavaScript folder	A folder that contains the project's JavaScript files. The files in this folder are referenced from the project's toolkit configuration XML file.
<Test Object Class Name>.js files	The files that contain your extensibility implementation JavaScript functions. One JavaScript file is created for each test object class that you define. Note: Any additional JavaScript implementation files that you import are also stored in the JavaScript folder.
Res	A folder that optionally contains icon files. The files in this folder are referenced from the project's test object configuration XML file.
<Icon files>	Optional. Icon files for QuickTest to use for the test object classes you define. Supported file types: .ico, .exe, .dll
<Project Name>.xml file	The project's toolkit configuration XML file.
<Project Name>TestObjects.xml file	The project's test object configuration XML file.

Import Toolkit Support Set Dialog Box

This dialog box enables you to create a new Extensibility Accelerator project based on an existing Web Add-in Extensibility toolkit support set.



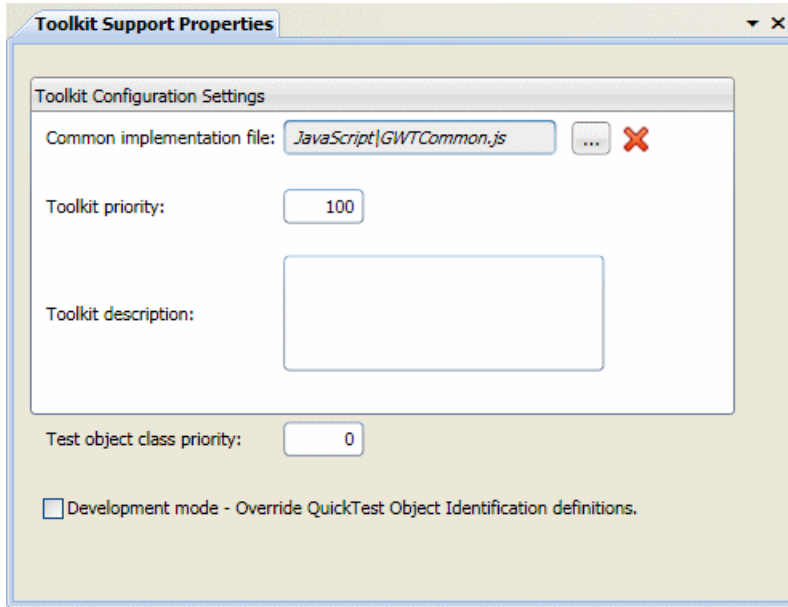
To access	Select File > Import Toolkit Support Set
Important information	The toolkit support set that you want to import must have the structure described in "Prerequisites" on page 36.
Relevant tasks	"How to Import an Existing Toolkit Support Set" on page 35

User interface elements are described below:

UI Elements	Description
Test objects XML file	The test object configuration XML file of the toolkit support set that you want to import.
New project folder	The folder in which to create the new project.
Open imported project	Specifies whether to open the new project after the import process is completed successfully. If this option is cleared, then after the import is completed, the dialog box remains open, enabling you to import and convert additional toolkit support sets to Extensibility Accelerator projects.




Toolkit Support Properties Designer

This designer enables you to define settings that affect how QuickTest treats this toolkit support set.



To access	Select View > Toolkit Support Properties
Important information	<ul style="list-style-type: none"> ▶ The information you define in this dialog box is stored in the XML files in your toolkit support set. ▶ The settings in this dialog box are optional. If you do not set them, QuickTest uses default values.
Relevant tasks	"How to Create or Update Support for a Custom Toolkit" on page 34

User interface elements are described below:

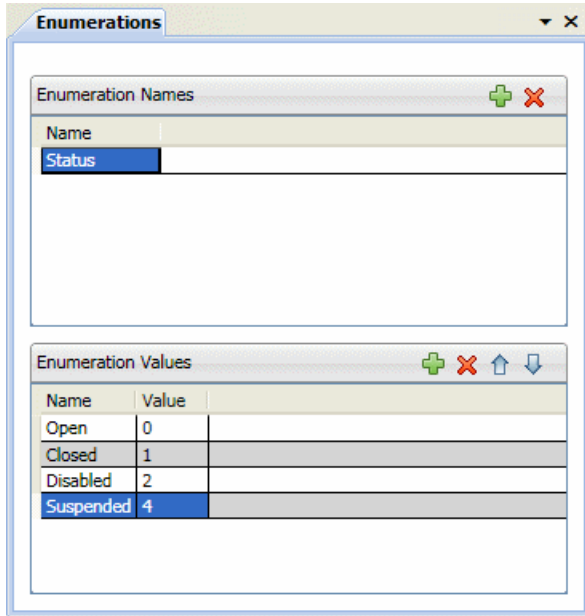
UI Elements	Description
<p>Common implementation file</p>	<p>The name of a file that contains shared JavaScript functions called from your other implementation functions. (Optional)</p> <p>You cannot modify this value directly.</p> <p>Use the Import File  button to browse to and select the relevant file.</p> <p>Use the Clear  button to clear the edit box.</p> <p>The corresponding XML attribute in the toolkit configuration XML file is cleared, but the JavaScript file is not removed from the project.</p> <p>Stored in: A Control\Settings\Variable element named common_file in the toolkit configuration XML file</p>
<p></p>	<p>Import File. Enables you to browse to and select a JavaScript file.</p> <p>If you select a file that is not located in the project's JavaScript folder, a local copy is created in that folder. The file must be located in the project's JavaScript folder to be properly deployed.</p> <p>If the file that you import has the same name as an existing file in this folder, Extensibility Accelerator appends a period (.) and a number to the imported file name (before the .js file extension).</p>
<p>Toolkit priority</p>	<p>The priority of the toolkit. When QuickTest attempts to identify the test object class mapped to a custom control, it searches in the different toolkits in the order of their priority (highest number first).</p> <p>Note: In this edit box, you can type only numeric characters.</p> <p>Default: 100</p> <p>Stored in: priority attribute of the Controls element in the toolkit configuration XML file</p>

UI Elements	Description
Toolkit description	<p>A description of the toolkit. QuickTest displays this description in the Add-in Manager dialog box when the toolkit support set's environment name is selected.</p> <p>If you are developing this toolkit support set for distribution, include a Provided by clause indicating the relevant person or company.</p> <p>Additionally, you might want to include a version number in this description.</p> <p>Stored in: Controls\Description elements in the toolkit configuration XML file</p>

UI Elements	Description
Test object class priority	<p>The priority of the test object classes defined in the test object configuration XML file. The priority is used if there are conflicts with other XML files (multiple test object classes defined with the same name).</p> <p>Note: In this edit box, you can type only numeric characters.</p> <p>Default: 0 (the lowest priority)</p> <p>Stored in: Priority attribute of the TypeInformation element in the test object configuration XML file</p>
Development mode - Override QuickTest Object Identification definitions	<p>Specifies whether the user is in development mode.</p> <ul style="list-style-type: none"> ▶ Select this option when you deploy the toolkit support set during development stages. This ensures that if you modified attributes of IdentificationProperty elements in the test object configuration XML file, QuickTest uses all of the changes you made. ▶ Make sure to clear this option before deploying the toolkit support set for regular use. This prevents the settings in the test object configuration XML file from overwriting any changes that the QuickTest user makes in the Object Identification dialog box. <p>For details, see the section on modifying deployed support in the <i>HP QuickTest Professional Web Add-in Extensibility Developer Guide</i>.</p> <p>Stored in: DevelopmentMode attribute of the TypeInformation element in the test object configuration XML file</p>



Enumerations Designer

This designer enables you to define lists of values that can be used for test object operation arguments or return values in the current project.



To access	Select View > Enumerations
Relevant tasks	"How to Design Test Object Class Operations" on page 69
See also	"Operations Tab (Test Object Class Designer)" on page 105

User interface elements are described below (unlabeled elements are shown in angle brackets>):

UI Elements	Description
Enumeration Names	<p>The name for the list of values.</p> <p>This area includes:</p> <ul style="list-style-type: none"> ▶  A toolbar that enables you to add or delete enumeration lists. ▶ Name. The name of the enumeration list. Click in this box to edit the name. <p>Stored in: ListOfValues element in the test object configuration XML file</p>
Enumeration Values	<p>The names and values of the items in the list that is currently selected in the Enumeration Names area.</p> <p>This area includes:</p> <ul style="list-style-type: none"> ▶  A toolbar that enables you to add, delete, or change the order of items in the list. Adding a value always adds it to the bottom of the list. ▶ Name. The name of the enumeration item. Click in this box to edit the name. ▶ Value. The integer value of the enumeration item. Click in this box to edit the value. <p>Default value: The value of the last item in the list + 1</p> <p>Stored in: ListOfValues\EnumValue element in the test object configuration XML file</p>

4

Supporting a Custom Control

This chapter includes:

Concepts

- ▶ Base Class Selection on page 52
- ▶ JavaScript Function Debugging on page 54

Tasks

- ▶ How to Create or Update Support for a Single Control on page 56
- ▶ How to Map a Test Object Class to Application Controls on page 58
- ▶ How to Design Test Object Class Operations on page 69
- ▶ How to Design Test Object Class Identification Properties on page 72
- ▶ How to Test and Debug Your Test Object Operation Support on page 75
- ▶ How to Test and Debug Your Property Retrieval Function on page 78

Reference

- ▶ General Tab (Test Object Class Designer) on page 81
- ▶ Map to Controls Tab (Test Object Class Designer) on page 92
- ▶ Operations Tab (Test Object Class Designer) on page 105
- ▶ Properties Tab (Test Object Class Designer) on page 117
- ▶ Debug Test Object Operation Dialog Box on page 127
- ▶ Debug Property Retrieval Dialog Box on page 129

Troubleshooting and Limitations - Supporting a Control on page 131

Concepts

Base Class Selection

When you define a test object class in the General Tab (Test Object Class Designer), described on page 81, you define a **base class**—a test object class that your new one extends. By default, all Web test object classes extend `WebElement`.

The base class that you select determines the test object class' generic type and default operation (unless you define them specifically) and provides the following:

- ▶ An initial set of test object operations, inherited from the base class. Some of these are displayed in the Operations tab (described on page 105), in which you can override them or add your own test object operations.
- ▶ A list of identification properties that you can choose to include in your test object class. Some of these are displayed in the Properties tab (described on page 117), in which you can also add or modify identification property definitions.
- ▶ If the control you are supporting contains the type of HTML element supported by the base class, your test object class also inherits the implementation that supports the inherited operations and properties. For more information, see the section on extending an existing test object class in the *HP QuickTest Professional Web Add-in Extensibility Developer Guide*.

Therefore:

- ▶ Select a base class that provides operations and properties that are relevant to the behavior of the control you are supporting.
- ▶ Make sure that the control contains an HTML element of the type supported by the base class. Otherwise, you need to provide implementation for all of the inherited test object operations and properties not supported by `WebElement`.

- ▶ If the control you are supporting contains the type of HTML element supported by the base class, but this is not the element that represents the control itself, you must implement a JavaScript function that returns the relevant base element.

Changing the Base Class

When you change the base class for a test object class, the list of inherited operations in the Operations tab and the list of base class properties in the Properties tab is automatically updated. Any operations or properties that you added, modified, or overrode remain unchanged.

Therefore, if you select a different base class after defining your lists of operations and properties, be sure to carefully reconsider these lists. Consider the following:

- ▶ You may have implemented operations or properties that are no longer relevant for the new type of class you are extending.
- ▶ Your test object class might no longer be inheriting test object operations that you wanted it to support.
- ▶ Your test object class might include identification properties that previously inherited their implementation from the old base class. You must now make sure that the new base class supports this property, remove that property from the list, or implement your **get_property_value** function to retrieve its value.
- ▶ Keep in mind that you inherit the implementation for the operations and properties of the new base class only if the control you are supporting contains the HTML element supported by that base class, and that you must implement a JavaScript function to return the relevant HTML element, if that element is not the HTML element representing the control.

JavaScript Function Debugging

After you design the JavaScript functions that implement your test object operations and property retrieval, you can test and debug them using Extensibility Accelerator. You do not need to have QuickTest installed to do this.

You can run a selected test object operation or retrieve the value of a selected property for a control that you select in your application. Extensibility Accelerator performs the test object operation or retrieves the property value by running the JavaScript function that you designed to support it, just as QuickTest would during a run session. This enables you to test and debug the support you designed.

While your JavaScript function runs, if you are running on Microsoft Internet Explorer, you can debug your JavaScript functions as you would in a regular Microsoft Visual Studio JavaScript debugging session.

For example, if you set a breakpoint in your function before running the operation, the run session will stop at the breakpoint if it is reached.

You can also add breakpoints, use step commands and other **Debug** menu commands and toolbars, use the various debugging-related windows such as Watch and Output, and so on. For more information, see the MSDN Visual Studio Help.

If you are running on Mozilla Firefox, you can add output messages to your code to help debug it. You can open message boxes from your code, or you can call `_util` methods, which result in messages being printed in the Extensibility Accelerator Output window, specifying the method called and the parameters passed.

Note: `_util` methods are relevant only when running in the QuickTest context. Therefore, if the JavaScript functions that you run include calls to `_util` methods, these calls are not carried out when they are encountered during the debugging process. Instead, a message is printed in the Extensibility Accelerator Output window specifying the method call and the parameters it passed.

For task details, see "How to Test and Debug Your Test Object Operation Support" on page 75 and "How to Test and Debug Your Property Retrieval Function" on page 78.

Tasks

How to Create or Update Support for a Single Control

A toolkit support set usually provides support for more than one type of custom control.

This task describes how to create support for one type. Perform this task for each type of control that you want to support.

When you save your changes, Extensibility Accelerator validates the information. If mandatory data is missing or if conflicts or discrepancies are found between information in the different files, the Error List window displays messages that explain the problems encountered.

See also "When Are Your Changes Applied and Saved" on page 32.

This task includes the following steps:

- "Prerequisite - Open an existing project or create a new one" on page 57
- "Design a test object class to represent your control in QuickTest" on page 57
- "Map the test object class to the relevant type of controls" on page 57
- "Design and debug the test object class operations" on page 57
- "Design the test object class's identification properties" on page 57
- "Implement support for recording on the control - Optional" on page 58
- "Deploy and test your support" on page 58

1 Prerequisite - Open an existing project or create a new one

For details, see "Open, create, or import an extensibility project" on page 34.

2 Design a test object class to represent your control in QuickTest

a Create a new test object class or open an existing one.



- ▶ To create a new test object class, click the **Add** button in the Class View.
- ▶ To open an existing test object class, double-click it in the Class View.

b The General tab of the test object class designer opens. Define a name for your test object class, specify the test object class it extends, and optionally, define additional general information.

For details, see "General Tab (Test Object Class Designer)" on page 81.

3 Map the test object class to the relevant type of controls

For details, see "How to Map a Test Object Class to Application Controls" on page 58.

4 Design and debug the test object class operations

For details, see "How to Design Test Object Class Operations" on page 69.

5 Design the test object class's identification properties

For details, see "How to Design Test Object Class Identification Properties" on page 72.

6 Implement support for recording on the control - Optional

- a In the General Tab - Advanced Options of the General Tab (Test Object Class Designer), described on page 85, set the **Record Options** according to your preferences, and specify the name of the function you implement to register for listening to events that occur on the control.
- b In the JavaScript file, implement the event registration function, and the event handlers that you want QuickTest to call when the events occur during a recording session.

For more details, see the section on implementing support for recording in the *HP QuickTest Professional Web Add-in Extensibility Developer Guide*.

7 Deploy and test your support

For details, see "How to Deploy a Toolkit Support Set" on page 137.

How to Map a Test Object Class to Application Controls

This task describes how to define and test the mapping rules for a test object class. The mapping rules indicate the types of controls for which QuickTest should use the test object class. You can create different rules to support different types and versions of browsers.

For more information on mapping rules, see the section on teaching QuickTest to identify the test object class to use for a custom Web control in the *HP QuickTest Professional Web Add-in Extensibility Developer Guide*.

Note: This task is part of a higher-level task. For details, see "How to Create or Update Support for a Single Control" on page 56.

This task includes the following steps:

- "Prerequisites" on page 59
- "Create a tab for a browser-specific rule set - Optional" on page 60
- "Expand the panel for the type of rules you want to create" on page 60
- "Create a set of mapping rules automatically" on page 61
- "Edit mapping rules manually - Optional" on page 66
- "Test your mapping rules on an application and update them if necessary" on page 66

1 Prerequisites

- a Plan your support.

For details, see the section on planning QuickTest support for your toolkit in the *HP QuickTest Professional Web Add-in Extensibility Developer Guide*.

- b Open an application that contains the relevant controls.

With an Extensibility Accelerator project open, run one or more applications that contain the controls you want to support. Make sure that the page is fully loaded and the relevant controls are visible. (You must run Extensibility Accelerator and open a project before you open the Web browsers.)

If you want to support more than one browser, open the application in different browsers.

2 Create a tab for a browser-specific rule set - Optional

In the Map to Controls Tab (Test Object Class Designer) (described on page 92), if you create mapping rules only in the **Default Rules** tab, these rules are used to map your controls to a test object class, for all browsers you work with.



If you want QuickTest to use different mapping rules when working with your controls in different browser types or versions, click the **Add Browser-Specific Rules** to add tabs for additional sets of rules. The Add Browser Dialog Box (described on page 103) opens, enabling you to specify the browser details.

Perform the next steps in each tab to create the necessary set of rules in each one.

3 Expand the panel for the type of rules you want to create

- a In the Map to Controls Tab (Test Object Class Designer), select the **Default Rules** tab, or a browser-specific tab.
- b Select the type of rules you want to create, and expand the relevant panel.

Available types: **Identify Control**, **Call Identification Function**, **Ignore Control**.

If you want to create more than one type of rules, perform the next steps in each relevant panel.

Note: If you create a set of **Call Identification Function** rules and a set of **Ignore Control** rules, QuickTest ignores the set of **Ignore Control** rules when attempting to identify the control.

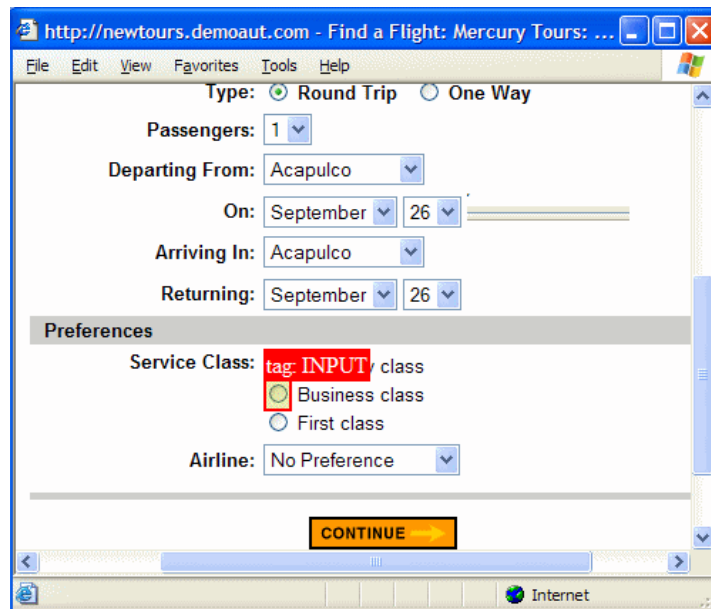
For more details on how QuickTest uses the different types of rules, see the section on teaching QuickTest to identify the test object class to use for a custom Web control in the *HP QuickTest Professional Web Add-in Extensibility Developer Guide*.

4 Create a set of mapping rules automatically

Perform this step separately in each relevant Rule Creation Panel and within each relevant browser-specific tab.

- a Click **Select Controls**. Extensibility Accelerator is hidden, and two buttons are displayed at the top of the screen: **Create Rules** and **Cancel**.
- b Move your mouse over your open applications. The mouse pointer is converted to a pointing hand.

Each control that you move over is highlighted in the application, and the name of the HTML element that represents the control is displayed. In the image below, the **INPUT** HTML element is displayed for a highlighted radio button control.

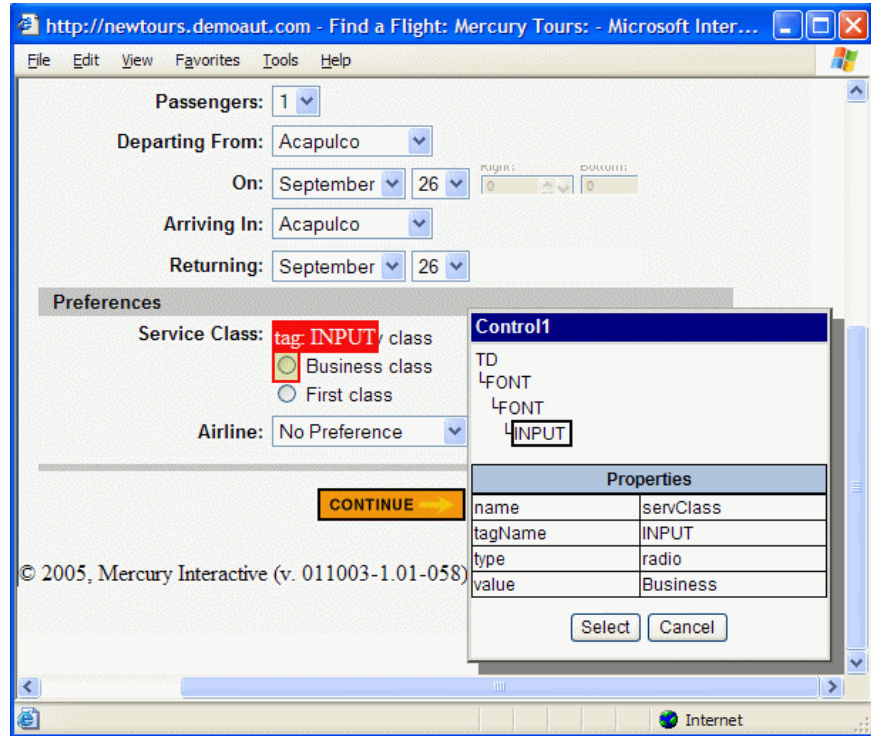


Tip: In many cases, you can hold the left CTRL key to change the pointing hand to a standard pointer and perform operations in your application, such as navigating to different Web pages, clicking links, selecting edit boxes to enter information, selecting from drop-down lists and so on. (Keep in mind that the browser behavior might be affected by the fact that the CTRL key is pressed.)

If a specific page does not load properly when navigating to it this way, load that page in an additional browser before beginning the session for selecting controls.

If you navigate to a different Web page, the highlighting process continues on the page that opens, after it is fully loaded.

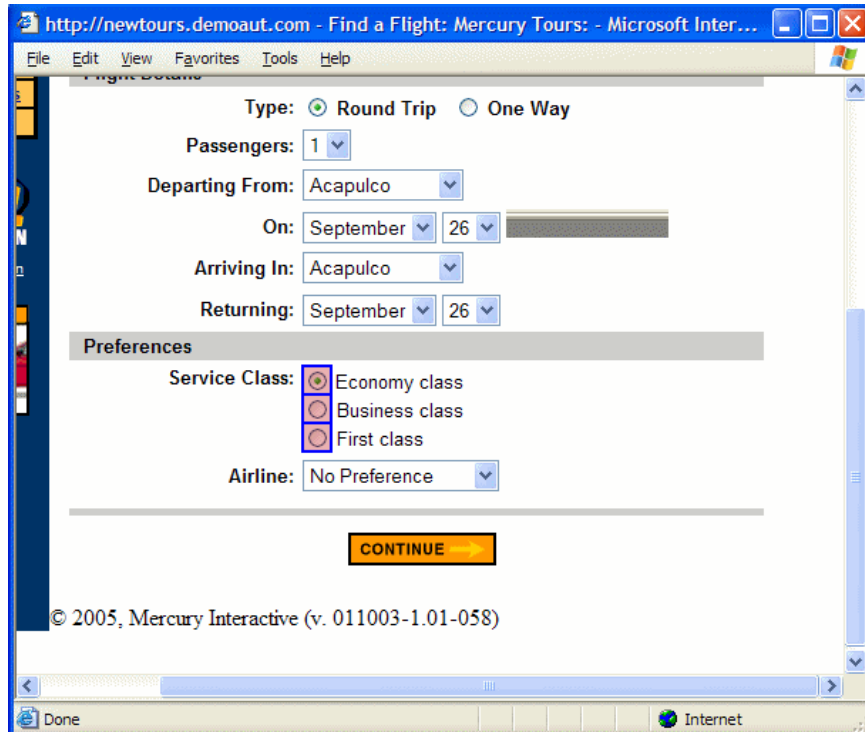
- c Click on a control of a type that you want to support with this test object class. A Selection Dialog Box (described on page 101) opens, displaying the properties of the HTML element that represents the selected control.



The top part of this dialog box displays additional elements in the control's HTML hierarchy.

- d To view the properties of a different HTML element, or to select it to represent the control, select the element from the displayed hierarchy.

- e Click **Select**. The selected control is highlighted in the application. In the image below, the radio button controls are selected.



- f Select additional controls that need to be supported by the same test object class.

Try to select several controls that need to be treated as the same type of control and share common properties, but are not identically implemented. The quality and accuracy of the rules that Extensibility Accelerator creates is affected by the number of controls you select, and their diversity.

- g To deselect a control, click the control and then click **Delete** in the Selection Dialog Box.
- h To specify a different HTML element to represent a selected control, click the control, select the appropriate element from the hierarchy displayed in the Selection Dialog Box, and click **Apply**.

- i To complete the process, click **Create Rules**.

The following happens:

- Extensibility Accelerator creates mapping rules for this test object class based on properties that are common to all of the controls you selected. If a large majority of the selected controls share common properties, the remaining controls might be ignored when creating the rules.

If appropriate, the created rules might contain regular expressions. For example, if you select two ASP.NET Ajax accordion panels, one that is selected (`className = accordionHeaderSelected`) and one that is not (`className = accordionHeader`), the created rule will include a regular expression condition: `className equal accordionHeader*`

Caution: Any rules previously contained in this panel of the Map to Controls tab are now replaced.

- If the controls do not have enough properties in common, no rules are created.

Tip: If you want to use the same test object class to support different types of controls, use this automatic process to create rules that identify one type of control. Then edit the rules manually to include additional types, for example, by adding rules with **Or** or **And NotEqual** logic.

- The highlighting is removed from the application.
- The rules are displayed in the rule editor area in the Map to Controls tab and added to the relevant **Identification** element in the toolkit configuration XML file, in **Conditions** elements.

Alternatively, click **Cancel** to end the process without creating rules.

5 Edit mapping rules manually - Optional

In the Rule Editor Area (in each rule creation panel and within each relevant browser-specific tab in the Map to Controls tab), you can make manual changes to the automatically created rules, or create your own rules.

For example, you can:

- ▶ Add and delete rules.
- ▶ Change the order or logic of rules.
- ▶ Generalize rules by defining regular expressions for property values.
- ▶ Modify automatically created regular expressions to make them more accurate.

The rules are stored in the relevant **Identification** element in the toolkit configuration XML file, in **Conditions** elements.

For information on the options in the rule editor, see the Rule Editor Area section in "Map to Controls Tab (Test Object Class Designer)" on page 92.

Tip: You can improve performance by limiting the identification process of custom controls to HTML elements with HTML tags you specify. However, you must do this manually in the toolkit configuration file, and the definitions that you add will not be displayed in Extensibility Accelerator. For details, see the section on the **HTMLTags element** in the Toolkit Configuration Schema (available in the QuickTest Professional Web Add-in Extensibility Help).

6 Test your mapping rules on an application and update them if necessary

You can test the rules in each panel separately, and test all of the rules together.

Follow one of the procedures described below:

To test one set of rules without modifying:

1 In the Map to Controls Tab (Test Object Class Designer) described on page 92, select the set of rules that you want to test and click **Test Rules**.

The following happens:

- ▶ Extensibility Accelerator is hidden.
- ▶ All of the controls that match the mapping rules are highlighted in all Web applications that are open in a browser that is relevant to this set of rules.

In many cases you can open or navigate to additional applications or Web pages at this point. Once the application or page loads successfully, the matching controls are highlighted in it as well.

If a specific page does not load properly when navigating to it at this point, load that page in an additional browser before beginning the session for testing the rules.

Note: Controls are highlighted only in browsers that are opened after you open a project in Extensibility Accelerator.

- ▶ A **Done** button is displayed at the top of the screen.

2 Click **Done**. The Map to Controls tab opens and the highlighting is removed from the applications.

To test one set of rules and update them if necessary:

1 Open the applications on which you want to test the rules, and make sure that the page is fully loaded and the relevant controls are visible. (You must run Extensibility Accelerator and open a project before you open the Web browsers.)

2 In the Map to Controls Tab (Test Object Class Designer) described on page 92, select the set of rules that you want to test and click **Test & Refine**.

The following happens:

- ▶ Extensibility Accelerator is hidden.
- ▶ The **Create Rules** and **Close** buttons are displayed at the top of the screen.
- ▶ A session for automatically creating rules begins. All of the controls that match the existing mapping rules are marked as selected in all Web applications that are open in a browser that is relevant to this set of rules.

3 Continue as described in step 4 b, above.

To test all of the rules together:

Click **Test All Rules**.

All of the controls that match the mapping rules in all open Web applications, are highlighted.

The rules from each tab are applied to the corresponding open browsers. For example, if you have a **Default Rules** tab, a **Firefox 3** tab and a **Firefox 3.5.5** tab, the default rules will be applied to any open Internet Explorer windows, and to Firefox 2, while the Firefox 3 rules will be applied to any open Firefox 3 windows and so on.

In addition, when the defined rules warrant it, the identification function that you implemented is also called to assist in identification of the relevant controls. For example, the identification function is called if a control's properties meet the rules defined in a set of **Call Identification Function** rules, or if no rules are defined at all.

The logic that Extensibility Accelerator uses when testing the rules and deciding whether to call the identification function are the same as the logic that QuickTest uses to identify the test object class to use for a custom Web control. For more details, see the section on teaching QuickTest to identify the test object class to use for a custom Web control in the *HP QuickTest Professional Web Add-in Extensibility Developer Guide*.

Considerations when testing your JavaScript identification function:

- ▶ **_util** methods are relevant only when running in the QuickTest context. Therefore, if your JavaScript identification function includes calls to **_util** methods, these calls are not carried out when testing the function. Instead, a message is printed in the Extensibility Accelerator Output window specifying the method call and the parameters it passed. You can make use of these messages to debug your function.
- ▶ The identification function is not called in debug mode, so you cannot use the Microsoft Visual Studio JavaScript debugging tools available in Extensibility Accelerator to debug it as it runs. If you are running on Internet Explorer and you want to use these debugging tools to debug your function, you can create a temporary test object operation that uses the identification function as its implementation function, and debug it as you would debug a test object operation.

How to Design Test Object Class Operations

This task describes how to define, implement, and debug the operations that your test object class supports.

Note: This task is part of a higher-level task. For details, see "How to Create or Update Support for a Single Control" on page 56.

This task includes the following steps:

- "Define the list of operations supported by this test object class" on page 69
- "Design the JavaScript functions that implement the test object operations" on page 71
- "Test and debug the operations" on page 71

1 Define the list of operations supported by this test object class

In the Operations Tab (Test Object Class Designer) described on page 105, do the following:

- Add or remove operations, or select base class operations to override.

Keep in mind that if the following conditions are met, you need to override all of the base class operations that are not supported by the `WebElement` test object class:

- The control you are supporting is not represented by the type of HTML element supported by the base class.
- You did not implement a **get base element** function that returns such an element to QuickTest.

In this case, operations that you do not override will be available when editing tests, but will not be implemented. Including these operations in test steps will result in run-time errors. For more information, see the section on extending an existing test object class in the *HP QuickTest Professional Web Add-in Extensibility Developer Guide*.

- ▶ For operations that you add or override, define the method signature and optionally, additional information.
- ▶ Specify the default operation for this test object class (optional).
If you do not select a default operation, the base class's default operation is used.

How these definitions affect the files:

- ▶ The information defined in this tab is stored in the toolkit support set XML files.
- ▶ JavaScript function stubs for new operations are added to the relevant JavaScript file.
- ▶ JavaScript function signatures for operations whose signature you modify are updated.
- ▶ JavaScript functions for deleted operations are **not** removed from the JavaScript file.

Note: If you define the **Implementation file name** or **Implementation function name** advanced options, Extensibility Accelerator does not manage the JavaScript implementation functions. This means the function stub is not added to the file, and the function's signature is not updated when you modify the operation's signature, or the **Implementation function name** option.

For more details, see "When Are Your Changes Applied and Saved" on page 32.

2 Design the JavaScript functions that implement the test object operations



- a In the Operations Tab (Test Object Class Designer) described on page 105, select the relevant operation and click the **Implementation Code** button. The JavaScript file opens to the relevant JavaScript function in a JavaScript Editor, described on page 20.

Alternatively, you can open the relevant JavaScript file by double-clicking it in the Project Explorer.

By default, the name of the JavaScript file is **<test object class name>.js**, and the name of the function you need to implement is the same as the test object operation name. You can modify these names in the advanced options in the Operations Tab (Test Object Class Designer). If you update the function name in the designer or in the JavaScript file, make sure to update it in the other location as well.

- b Implement the JavaScript functions to perform the test object operations on the control. For details, see the section on implementing support for test object methods in the *HP QuickTest Professional Web Add-in Extensibility Developer Guide*.

You must implement JavaScript functions for all new and overridden operations.

3 Test and debug the operations

For details, see "How to Test and Debug Your Test Object Operation Support" on page 75.

How to Design Test Object Class Identification Properties

This task describes how to define and implement support for the identification properties of your test object class.

Note: This task is part of a higher-level task. For details, see "How to Create or Update Support for a Single Control" on page 56.

This task includes the following steps:

- ▶ "Define the list of identification properties for your test object class" on page 72
- ▶ "Specify the QuickTest functionalities for which the properties are used" on page 73
- ▶ "Define advanced options for identification property support - Optional" on page 73
- ▶ "Implement the JavaScript function that retrieves the identification property values from the run-time object" on page 73
- ▶ "Test and debug the function that retrieves the identification property values." on page 74

1 Define the list of identification properties for your test object class

In the Properties Tab (Test Object Class Designer) described on page 117, add or remove properties or select base class properties to inherit and include in the list.

2 Specify the QuickTest functionalities for which the properties are used

Add properties from the **Properties** list on the left side of the Properties Tab to the different groups on the right. This indicates which properties are included in test object descriptions, which can be verified in checkpoints and used in output values, which should be used for Smart Identification, and so on.

3 Define advanced options for identification property support - Optional

For details, see the Advanced Options section of the Properties Tab (Test Object Class Designer) described on page 125.

4 Implement the JavaScript function that retrieves the identification property values from the run-time object



- a In the Properties Tab (Test Object Class Designer) described on page 117, click the **Implementation Code** button. The JavaScript file opens to the relevant JavaScript function in a JavaScript Editor, described on page 20.

If you selected a property before clicking the button, the file opens to the relevant section within the function.

Alternatively, you can open the relevant JavaScript file by double-clicking it in the Project Explorer.

By default, the name of the JavaScript file is **<test object class name>.js**, and the name of the function you need to implement is **get_property_value**. You can modify these names in the advanced options in the Properties Tab (Test Object Class Designer). If you update the function name in the designer or in the JavaScript file, make sure to update it in the other location as well.

- b** Implement the JavaScript function to retrieve the run-time values for the identification properties. For details, see the section on implementing support for identification properties in the *HP QuickTest Professional Web Add-in Extensibility Developer Guide*.

If the following conditions are met, the test object class inherits the **get_property_value** implementation from the base class. In that case, the function that you write does not have to retrieve a value for this property.

- ▶ The control you are supporting is represented by the type of HTML element supported by the base class, or it contains such an element and you implemented a function that returns that element to QuickTest.
- ▶ The base class supports an identification property by the same name.

5 Test and debug the function that retrieves the identification property values.

For details, see "How to Test and Debug Your Property Retrieval Function" on page 78.

How to Test and Debug Your Test Object Operation Support

This task describes how to run your test object operations from within Extensibility Accelerator, so that you can test and debug your JavaScript implementation functions.

Note: This task is part of a higher-level task. For details, see "How to Design Test Object Class Operations" on page 69.

This task includes the following steps:

- "Prerequisites" on page 75
- "Set a breakpoint in your implementation function - Optional" on page 76
- "In the Debug Test Object Operation dialog box, select the test object class operation to run" on page 76
- "Select an application control on which to run the operation" on page 77
- "Run the operation" on page 77

1 Prerequisites

- a** If you are running on Microsoft Internet Explorer, enable script debugging in your browser.

For example: In Internet Explorer 6.0 or 7.0, select **Tools > Internet Options**. In the **Advanced** tab, clear the **Disable Script Debugging** options in the **Browsing** group.

- b** Open the application on which you want to run and debug your operation, and make sure that the page is fully loaded and the relevant control is visible. (You must run Extensibility Accelerator and open a project before you open the Web browser.)
- c** Make sure that the rules displayed in the rule editor for your test object class correctly identify the control on which you want to debug the operation.

You can click **Test All Rules** in the Map to Controls Tab (Test Object Class Designer) described on page 92 and verify that the control is highlighted in the application.

2 Set a breakpoint in your implementation function - Optional

If you are running on Microsoft Internet Explorer and you want the run session to pause when it reaches the function that you designed to support the operation, you can add a breakpoint at the beginning of the function.

Use the Microsoft Visual Studio JavaScript debugging tools available in Extensibility Accelerator to add the breakpoint.

3 In the Debug Test Object Operation dialog box, select the test object class operation to run

- a Do one of the following to open the dialog box (described on page 127):
 - ▶ In the Class View, right-click the operation and select **Debug**. The Debug Test Object Operation dialog box opens with the test object class and operation selected.
 - ▶ In the Operations Tab (Test Object Class Designer) described on page 105, select an operation from the operation list and click the **Debug Operation** button in the operation list toolbar. The Debug Test Object Operation dialog box opens with the test object class and operation selected.
 - ▶ Select **Project > Debug Test Object Operation**. In the dialog box that opens, select the test object class and the operation that you want to run.
- b If the operation you selected receives arguments, a table displays the argument names, whether they are optional, and the type of value they require. If necessary, enter the argument values to pass to the operation.



4 Select an application control on which to run the operation

- a In the Debug Test Object Operation Dialog Box, click **Select Control**. Extensibility Accelerator is hidden, all of the controls that match the mapping rules in all open Web applications, are highlighted, and a **Cancel** button is displayed at the top of the screen.

In many cases you can open or navigate to additional applications or Web pages at this point. Once the application or page loads successfully, the matching controls are highlighted in it as well. (To navigate at this point, you need to hold down the CTRL key.)

If a specific page does not load properly when navigating to it at this point, load that page in an additional browser before clicking **Select Control**.

Note: Controls are highlighted only in browsers that are opened after you open a project in Extensibility Accelerator.

- b Click the control on which you want to run the operation. You must select one of the highlighted controls.

The highlighting is removed from the application, Extensibility Accelerator opens and the Debug Test Object Operation Dialog Box is displayed.

5 Run the operation

In the Debug Test Object Operation Dialog Box, click **Run Operation**. Extensibility Accelerator begins to run the test object operation on the control you selected, calling the operation's JavaScript implementation function.

If you ran the operation on a control running in Microsoft Internet Explorer, you can now debug your functions using the Microsoft Visual Studio Shell debugging tools that are available in Extensibility Accelerator.

If you ran the operation on a control running in Mozilla Firefox, you can add output messages to your code to help debug it.

How to Test and Debug Your Property Retrieval Function

This task describes how to instruct Extensibility Accelerator to retrieve a property value from a control in your application. This enables you to test and debug the JavaScript implementation function that you designed to retrieve property values.

Note: This task is part of a higher-level task. For details, see "How to Design Test Object Class Identification Properties" on page 72.

This task includes the following steps:

- "Prerequisites" on page 78
- "Set a breakpoint in your implementation function - Optional" on page 79
- "In the Debug Property Retrieval dialog box, select a property to retrieve" on page 79
- "Select the application control whose property value you want to retrieve" on page 79
- "Retrieve the property value" on page 80

1 Prerequisites

- a If you are running on Microsoft Internet Explorer, enable script debugging in your browser.

For example: In Internet Explorer 6.0 or 7.0, select **Tools > Internet Options**. In the **Advanced** tab, clear the **Disable Script Debugging** options in the **Browsing** group.

- b Open the application from which you want to retrieve property values, and make sure that the page is fully loaded and the relevant control is visible. (You must run Extensibility Accelerator and open a project before you open the Web browser.)
- c Make sure that the rules displayed in the rule editor for your test object class correctly identify the control whose property you want to retrieve.

You can click **Test All Rules** in the Map to Controls Tab (Test Object Class Designer) described on page 92 and verify that the control is highlighted in the application.

2 Set a breakpoint in your implementation function - Optional

If you are running on Microsoft Internet Explorer and you want the run session to pause when it reaches the function that you designed to retrieve property values, you can add a breakpoint to the function.

Use the Microsoft Visual Studio JavaScript debugging tools available in Extensibility Accelerator to add the breakpoint.

3 In the Debug Property Retrieval dialog box, select a property to retrieve

Do one of the following:



- ▶ In the Properties Tab (Test Object Class Designer) described on page 117, select an identification property from the property list and click the **Debug Property Retrieval** button in the property list toolbar.

The Debug Property Retrieval Dialog Box (described on page 129) opens with the test object class and the property selected.

- ▶ Select **Project > Debug Property Retrieval**.

In the Debug Property Retrieval dialog box that opens, select the test object class and the property that you want to retrieve.

4 Select the application control whose property value you want to retrieve

- a In the Debug Property Retrieval Dialog Box, click **Select Control**. Extensibility Accelerator is hidden, all of the controls that match the mapping rules in all open Web applications, are highlighted, and a **Cancel** button is displayed at the top of the screen.

In many cases you can open or navigate to additional applications or Web pages at this point. Once the application or page loads successfully, the matching controls are highlighted in it as well. (To navigate at this point, you need to hold down the CTRL key.)

If a specific page does not load properly when navigating to it at this point, load that page in an additional browser before clicking **Select Control**.

Note: Controls are highlighted only in browsers that are opened after you open a project in Extensibility Accelerator.

- b** Click the control whose property value you want to retrieve. You must select one of the highlighted controls.

The highlighting is removed from the application, Extensibility Accelerator opens and the Debug Property Retrieval Dialog Box is displayed. If Extensibility Accelerator does not come back into focus, activate it from the Task Bar.

5 Retrieve the property value

In the Debug Property Retrieval Dialog Box, click **Retrieve Value**. Extensibility Accelerator attempts to retrieve the property value from the control you selected, by calling the JavaScript function that you implemented to retrieve property values, passing the selected property name as a parameter. (The property name is passed in lowercase letters, simulating QuickTest's property value retrieval behavior.)

If you retrieve the property value from a control running in Microsoft Internet Explorer, you can now debug your functions using the Microsoft Visual Studio Shell debugging tools that are available in Extensibility Accelerator.

If you retrieve the property value from a control running in Mozilla Firefox, you can add output messages to your code to help debug it.

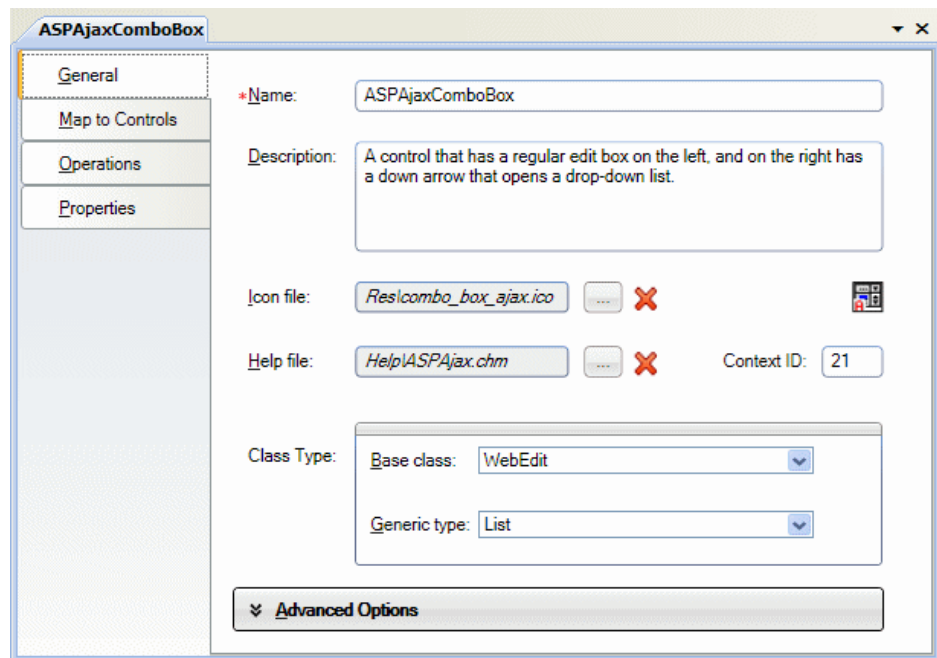
Reference

General Tab (Test Object Class Designer)

This tab enables you to define general details about the test object class that you want QuickTest to use for a custom control.



The information you define in this tab is stored in the XML files in your toolkit support set. The options in the main part of this tab are stored in the test object configuration XML file. The advanced options are stored in the toolkit configuration XML file.





The image below displays the basic options available in the General tab of the Test Object Class designer.



To access	In the Class View, add a new test object class or double-click an existing one.
Important information	Make sure to visit each tab in the test object class designer to ensure that all of the test object class details are defined correctly. A red asterisk next to the name of the tab acts as a reminder that you have not yet visited this tab.
Relevant tasks	"How to Create or Update Support for a Single Control" on page 56
See also	"Custom Toolkit Support Sets" on page 30

User interface elements are described below (unlabeled elements are shown in angle brackets>):

UI Elements	Description
Name	<p>The name of the test object class that you want QuickTest to use to represent the custom control.</p> <p>This name is fundamental to the infrastructure of the support you are creating for the custom control. It determines:</p> <ul style="list-style-type: none"> ▶ The name of the JavaScript file created for this test object's implementation functions. This file name is displayed in the Default implementation file advanced option. It is stored in the relevant Settings\Variable element in the toolkit configuration XML file. ▶ The Name attribute of the ClassInfo element in the test object configuration XML file. ▶ The TestObjectClass attribute of the Control element in the toolkit configuration XML file. <p>If you rename the test object class, all of the above are modified automatically. (It is therefore recommended to save such a change immediately.)</p>
Description	<p>A description of the custom control you are supporting.</p> <p>This description is intended for your internal documentation purposes, it is not displayed in QuickTest Professional.</p> <p>Stored in: ClassInfo\Description element in the test object configuration XML file</p>
Icon file	<p>The name of the icon file that you want QuickTest to display for this test object class in tests, dialog boxes, and run session results.</p> <p>Use the Import File  button to specify the relevant file.</p> <p>Use the Clear  button to clear the edit box.</p> <p>Default icon: QuickTest's WebElement icon</p> <p>Stored in: ClassInfo\IconInfo element in the test object configuration XML file</p>

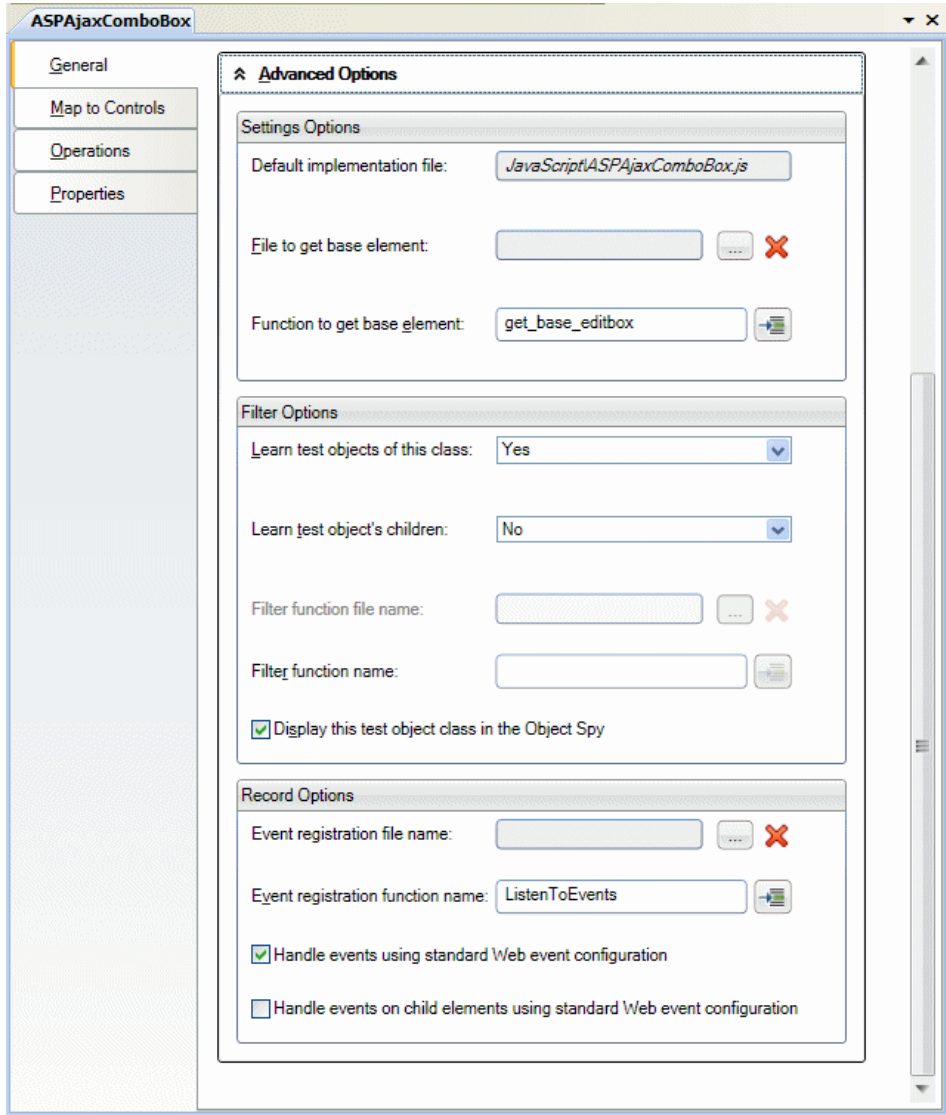
UI Elements	Description
	<p>Import File. Enables you to browse to and select the icon file. You can select an icon from an .ico, .dll, or .exe file.</p> <p>If the icon you select is not currently stored in the project's Res folder, a local copy is created in that folder. The file must be located in the project's Res folder to be properly deployed.</p> <p>Note: Avoid importing large .exe or .dll files, as these are added to your toolkit support set and deployed with it.</p>
<icon>	An image of the icon you selected or the default icon.
Help file	<p>The name of the .chm Help file that you want QuickTest to use for context-sensitive Help on this test object class.</p> <p>Use the Import File  button to specify the relevant file.</p> <p>Use the Clear  button to clear the edit box.</p> <p>Stored in: ClassInfo\HelpInfo element in the test object configuration XML file</p>
	<p>Import File. Enables you to browse to and select the .chm Help file.</p> <p>If the file you select is not currently stored in the project's Help folder, a local copy is created in that folder. The file must be located in the project's Help folder to be properly deployed.</p>
Context ID	<p>The numeric value that indicates the help topic to open within the specified Help file.</p> <p>Stored in: ClassInfo\HelpInfo element in the test object configuration XML file</p>

UI Elements	Description
Base class	<p>The test object class this class extends. By default all Web test object classes extend WebElement.</p> <p>The base class that you select determines the default Generic type, the initial set of operations that your test object class includes, and a list of identification properties that you can choose to include in your test object class.</p> <p>If the control you are supporting contains the type of HTML element supported by the base class, the test object class also inherits the implementation that supports the inherited operations and properties.</p> <p>For more information, see "Base Class Selection" on page 52.</p> <p>Note: If the control contains an HTML element of the type supported by the base class, but this is not the element that represents the control itself, be sure to define the Function to get base element in the advanced options.</p> <p>Stored in: BaseClassName attribute of the ClassInfo element in the test object configuration XML file</p>
Generic type	<p>The type of control you are supporting.</p> <p>The generic type is used for object filtering in QuickTest and for creating documentation strings for the Documentation column of the Keyword View (unless you define them specifically in the test object operation definition).</p> <p>Default: The base test object class's generic type. (This value is selected automatically when you select a base class.)</p> <p>Stored in: GenericTypeID attribute of the ClassInfo element in the test object configuration XML file</p>
Advanced Options	<p>Expands to display the advanced options, described in "General Tab - Advanced Options" on page 85. If you do not define these options, QuickTest uses their default values.</p>



General Tab - Advanced Options



The **Advanced Options** area in the General Tab (Test Object Class Designer), described on page 81, enables you to set advanced options for the test object class. If you do not define these options, QuickTest uses their default values.



The image below displays the advanced options available in the General tab of the Test Object Class designer.







User interface elements are described below:

UI Elements	Description
Settings Options	
Default implementation file	<p>The file from which QuickTest calls implementation functions for this test object class by default.</p> <p>This is a read only option, set by Extensibility Accelerator to: JavaScript\<test object class name>.js.</p> <p>If you modify the name of the test object class, this option is automatically updated to match the new name.</p> <p>If when the file is created or renamed, a file by that name already exists in the file system, Extensibility Accelerator appends a period (.) and a number to the new file name (before the .js file extension).</p> <p>Stored in: A Control\Settings\Variable element named default_imp_file in the toolkit configuration XML file</p>
File to get base element	<p>The file that contains the function that returns the base element (optional).</p> <p>You cannot modify this value directly.</p> <p>Use the Import File  button to browse to and select the relevant file.</p> <p>Use the Clear  button to clear the edit box.</p> <p>The corresponding XML attribute in the toolkit configuration XML file is cleared, but the JavaScript file is not removed from the project.</p> <p>Default: The Default implementation file</p> <p>Stored in: A Control\Settings\Variable element named file_for_func_to_get_base_elem in the toolkit configuration XML file</p>

UI Elements	Description
	<p>Import File. Enables you to browse to and select a JavaScript file.</p> <p>If you select a file that is not located in the project's JavaScript folder, a local copy is created in that folder. The file must be located in the project's JavaScript folder to be properly deployed.</p> <p>If the file that you import has the same name as an existing file in this folder, Extensibility Accelerator appends a period (.) and a number to the imported file name (before the .js file extension).</p>
<p>Function to get base element</p>	<p>The function that you implement to return the base element. Access to the base element enables QuickTest to use the base class's implementation for inherited test object operations and properties.</p> <p>You need to specify and implement this function if the control you are supporting contains an HTML element of the type supported by the base class, but this is not the element that represents the control itself. If you do not provide this function, you need to provide implementation for any inherited test object operations and properties that are not supported by WebElement and you want to support.</p> <p>Use the Implementation Code  button to open the relevant JavaScript file to the specified JavaScript function. If the function does not exist, a JavaScript function stub is added to the file.</p> <p>Stored in: A Control\Settings\Variable element named func_to_get_base_elem in the toolkit configuration XML file</p>

UI Elements	Description
Filter Options	
Learn test objects of this class	<p>Indicates whether QuickTest should learn this control.</p> <p>Possible values:</p> <ul style="list-style-type: none"> ➤ Yes ➤ No ➤ Only if has children - learn the control only if it has children. (Stored as IfChildren) <p>Default: Yes</p> <p>Stored in: learn_control attribute of the Learn element in the toolkit configuration XML file</p>
Learn test object's children	<p>Indicates whether QuickTest should learn the children of this control.</p> <p>Possible values:</p> <ul style="list-style-type: none"> ➤ Yes ➤ No ➤ Use Filter Function - the function specified below performs the filtering. (Stored as CallFilterFunc) <p>Default: Yes</p> <p>Stored in: learn_children attribute of the Learn element in the toolkit configuration XML file</p>
Filter function file name	<p>The file that contains the filter function (optional).</p> <p>You cannot modify this value directly.</p> <p>Use the Import File  button to browse to and select the relevant file. (For details on using this button, see above.)</p> <p>Use the Clear  button to clear the edit box.</p> <p>The corresponding XML attribute in the toolkit configuration XML file is cleared, but the JavaScript file is not removed from the project.</p> <p>Default: The Default implementation file</p> <p>Stored in: file_name attribute of the Learn element in the toolkit configuration XML file</p>

UI Elements	Description
<p>Filter function name</p>	<p>The function that performs the filtering.</p> <p>You must specify and implement this function if you selected the Use Filter Function value for the Learn test object's children option.</p> <p>Use the Implementation Code  button to open the relevant JavaScript file to the specified JavaScript function. If the function does not exist, a JavaScript function stub is added to the file.</p> <p>Stored in: function attribute of the Learn element in the toolkit configuration XML file</p>
<p>Display this test object class in the Object Spy</p>	<p>Indicates whether the Object Spy displays this test object class.</p> <p>Default: Yes</p>
<p>Record Options</p>	
<p>Event registration file name</p>	<p>The file that contains the event registration function (optional).</p> <p>You cannot modify this value directly.</p> <p>Use the Import File  button to browse to and select the relevant file. (For details on using this button, see above.)</p> <p>Use the Clear  button to clear the edit box.</p> <p>The corresponding XML attribute in the toolkit configuration XML file is cleared, but the JavaScript file is not removed from the project.</p> <p>Default: The Default implementation file</p> <p>Stored in: file_name attribute of the Record\EventListenering element in the toolkit configuration XML file</p>

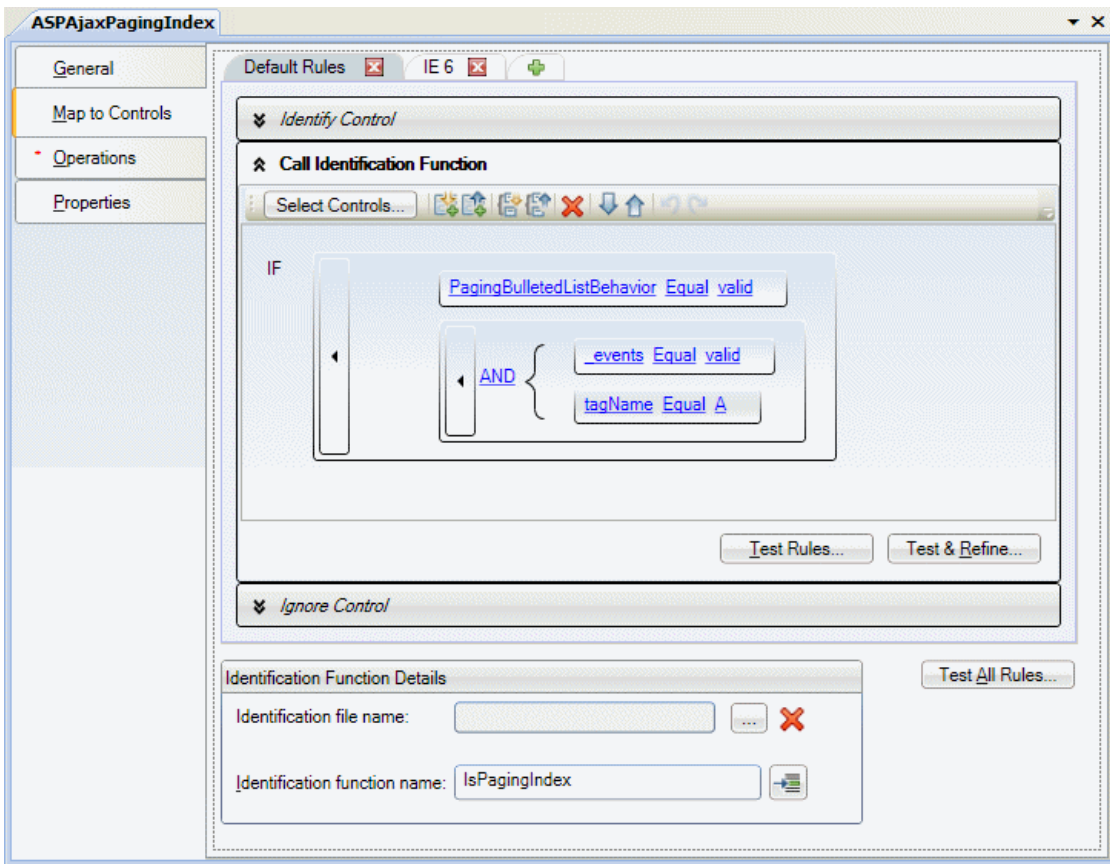
UI Elements	Description
Event registration function name	<p>The function that implements registering to listen for events on the elements contained in the control.</p> <p>You must specify and implement this function if you want to customize recording on your control.</p> <p>Use the Implementation Code  button to open the relevant JavaScript file to the specified JavaScript function. If the function does not exist, a JavaScript function stub is added to the file.</p> <p>Stored in: <code>function</code> attribute of the <code>Record\EventListening</code> element in the toolkit configuration XML file</p>
Handle events using standard Web event configuration	<p>Specifies whether to use standard Web event configuration during a recording session to handle events on controls represented by this test object class.</p> <p>Stored in: <code>use_default_event_handling</code> attribute of the <code>Record\EventListening</code> element in the toolkit configuration XML file</p>
Handle events on child elements using standard Web event configuration	<p>Specifies whether to use standard Web event configuration during a recording session to handle events that take place on the child elements of controls represented by this test object class.</p> <p>Stored in: <code>use_default_event_handling_for_children</code> attribute of the <code>Record\EventListening</code> element in the toolkit configuration XML file</p>

Map to Controls Tab (Test Object Class Designer)

This tab enables you to define rules that indicate the types of controls this test object class supports. It also enables you to test the rules that you create.



You can create browser-specific tabs with different rules to support different types and versions of browsers. Each browser-specific tab contains three rule creation panels, in which you can create a set of rules that QuickTest uses in different ways.




Each Rule Creation Panel (described on page 97) contains options you can use to create, edit, and test the set of rules in that panel.




To access	<p>1 In the Class View, add a new test object class or double-click an existing one. The test object class designer opens.</p> <p>2 In the test object class designer, select the Map to Controls tab.</p>
Important information	<ul style="list-style-type: none"> ▶ Make sure to visit each tab in the test object class designer to ensure that all of the test object class details are defined correctly. A red asterisk next to the name of the tab acts as a reminder that you have not yet visited this tab. ▶ If you need to create HTMLTags elements to improve your Web Add-in Extensibility performance, you must define these manually in the XML files. If the toolkit configuration XML file contains HTMLTags elements they are not displayed in this tab.
Relevant tasks	"How to Map a Test Object Class to Application Controls" on page 58
See also	<ul style="list-style-type: none"> ▶ The section on teaching QuickTest to identify the test object class to use for a custom Web control in the <i>HP QuickTest Professional Web Add-in Extensibility Developer Guide</i>. ▶ "Rule Creation Panel" on page 97 ▶ "Selection Dialog Box" on page 101 ▶ "Add Browser Dialog Box" on page 103

User interface elements are described below (unlabeled elements are shown in angle brackets):

UI Elements	Description
<p><browser-specific tabs></p>	<p>A strip of tabs, each containing the mapping rules for QuickTest to use when running on a specific browser type and version.</p> <p>To add a new tab, click the Add Browser-Specific Rules  tab. To remove a tab, click the Delete  button on the tab.</p> <p>The rules in the Default Rules tab are used for all supported browsers that do not have a specific set of rules defined.</p> <p>The rules in other tabs are used for the browser specified on the tab. If a browser version is specified, the rules are used when running on browsers of the specified type, whose version is the same or later.</p>
<p><rule creation panels></p>	<p>A set of panels in which you can create sets of mapping rules. For details on creating and testing the rules, see "Rule Creation Panel" on page 97.</p> <p>Stored in: Conditions elements in the toolkit configuration XML file. The type attribute of the element is determined by the panel in which you create the rules:</p> <ul style="list-style-type: none"> ➤ Identify Control panel -> IdentifyIfPropMatch type ➤ Call Identification Function panel -> CallIdFuncIfPropMatch type ➤ Ignore Control panel -> SkipIfPropMatch type <p>For details on how QuickTest uses the different types of rules, see the section on teaching QuickTest to identify the test object class to use for a custom Web control in the <i>HP QuickTest Professional Web Add-in Extensibility Developer Guide</i>.</p>

UI Elements	Description
Identification file name	<p>The file that contains the identification function (optional).</p> <p>You cannot modify this value directly.</p> <p>Use the Import File  button to browse to and select the relevant file.</p> <p>Use the Clear  button to clear the edit box.</p> <p>The corresponding XML attribute in the toolkit configuration XML file is cleared, but the JavaScript file is not removed from the project.</p> <p>Default: The Default implementation file defined in the General tab</p> <p>Stored in: Identification element in the toolkit configuration XML file</p>
	<p>Import File. Enables you to browse to and select a JavaScript file.</p> <p>If you select a file that is not located in the project's JavaScript folder, a local copy is created in that folder. The file must be located in the project's JavaScript folder to be properly deployed.</p> <p>If the file that you import has the same name as an existing file in this folder, Extensibility Accelerator appends a period (.) and a number to the imported file name (before the .js file extension).</p>

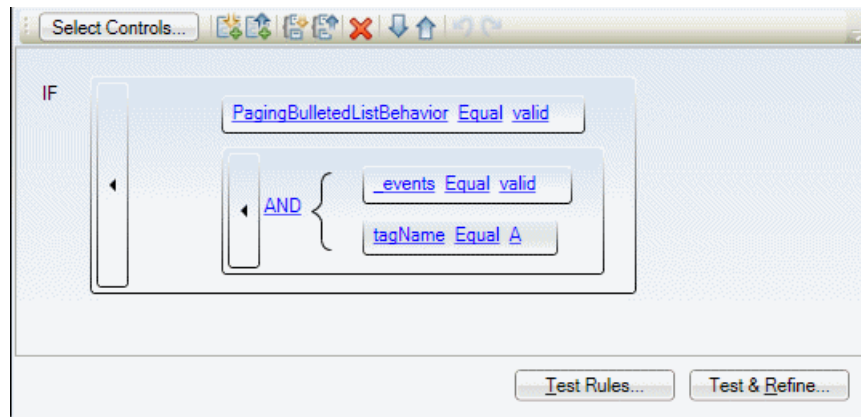
UI Elements	Description
<p>Identification function name</p>	<p>The function that you implement to help identify the controls for which to use this test object class. This function is necessary only if you cannot create a set of rules that identifies the controls specifically enough.</p> <p>Use the Implementation Code  button to open the relevant JavaScript file to the specified JavaScript function. If the function does not exist, a JavaScript function stub is added to the file.</p> <p>Stored in: Identification element in the toolkit configuration XML file</p>
<p>Test All Rules</p>	<p>Highlights all of the controls that match the mapping rules in all open Web applications.</p> <p>The rules from each tab are applied to the corresponding open browsers, using the same logic that QuickTest uses to identify the test object class to use for a custom Web control.</p> <p>In addition, when the defined rules warrant it, the identification function that you implemented is also called to assist in identification of the relevant controls.</p> <p>For details, see "Test your mapping rules on an application and update them if necessary" on page 66.</p>

Rule Creation Panel

In the Map to Controls Tab (Test Object Class Designer), described on page 92, each browser-specific tab contains three rule creation panels, in which you can create a set of rules that QuickTest uses in different ways.

Each rule creation panel contains a rule editor area and buttons that you can use to create rules automatically and to test the rules on an application.

- To create rules automatically and to test rules, follow the process described in "How to Map a Test Object Class to Application Controls" on page 58.
- To edit rules manually, use the toolbar and UI elements within the rule editor area.



Each rule creation panel contains:

- "Buttons" on page 98
- "Rule Editor Area" on page 98

Buttons


UI Elements	Description
Select Controls	<p>Starts a session for automatically creating mapping rules. You create the rules by pointing to controls of the relevant type in your application.</p> <p>For task details, see "Create a set of mapping rules automatically" on page 61.</p>
Test Rules	<p>Highlights all of the controls that match the mapping rules in all open Web applications.</p> <p>For task details, see "Test your mapping rules on an application and update them if necessary" on page 66.</p>
Test & Refine	<p>Starts a session for automatically creating rules. All of the controls that match the currently defined rules are marked as selected in all open Web applications.</p> <p>For task details, see "Test your mapping rules on an application and update them if necessary" on page 66.</p>


Rule Editor Area

This area displays the mapping rules and enables you to edit them manually. For example, you can:

- Add and delete rules.
- Change the order or logic of rules.
- Generalize the rules by defining regular expressions for the property values.

User interface elements are described below (unlabeled elements are shown in angle brackets):

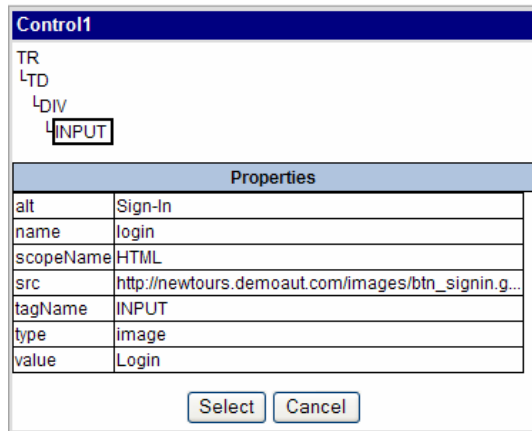
UI Elements	Description
<edit toolbar>	 <p>This toolbar contains the following buttons:</p> <ul style="list-style-type: none"> ➤ Add Single Condition Below ➤ Add Single Condition Above ➤ Add Grouped Conditions Below ➤ Add Grouped Conditions Above ➤ Delete Selected Element ➤ Move Selected Element Down ➤ Move Selected Element Up ➤ Undo ➤ Redo
<rule containers>	Rectangles that contain single rules or grouped rules. To select a rule or group of rules, click its container.

UI Elements	Description
<rules>	<p>A single rule is made up of the following elements:</p> <ul style="list-style-type: none"> ▶ <property name>. The name of the HTML property checked in this rule. Click to select from a list of common property names or edit this value. ▶ Equal / Not Equal. Indicates whether the value of the property must be equal or not equal to the expected value to conform to the rule. Click to switch between Equal and Not Equal. ▶ <expected value>. The value to compare to the value of the control's HTML property in the application. A regular expression icon  is displayed if you specified that this value should be treated as a regular expression. Click to edit this value. When you edit the expected value, additional options are displayed: <ul style="list-style-type: none"> ▶ RegExp. Indicates whether the expected value should be treated as a regular expression. Default: false ▶ Trim. Indicates whether QuickTest should remove leading and trailing spaces from the property value and the expected value before evaluating the rule. Default: true <p>Click to switch between true and false values for these rule attributes. (A toggle button that is on indicates the value true.)</p>
AND/OR	<p>Indicates whether to use And or Or logic for the set of rules in the group. Click to switch between AND and OR.</p>

Selection Dialog Box

This dialog box opens when you click a Web control during a session for selecting controls to automatically create mapping rules for a test object class. It enables you to specify whether to include the control in the set of controls that determines the rules that are created.

The dialog box displays the HTML details for the control. You can specify a different HTML element to represent this control by selecting it in the displayed hierarchy.



To access	Click Select Controls in the Map to Controls tab of the test object class designer, and then click on a control in a Web application.
Important information	The title bar of the dialog box displays the name of the test object class for which you are selecting controls.
Relevant tasks	"How to Map a Test Object Class to Application Controls" on page 58
See also	"Map to Controls Tab (Test Object Class Designer)" on page 92

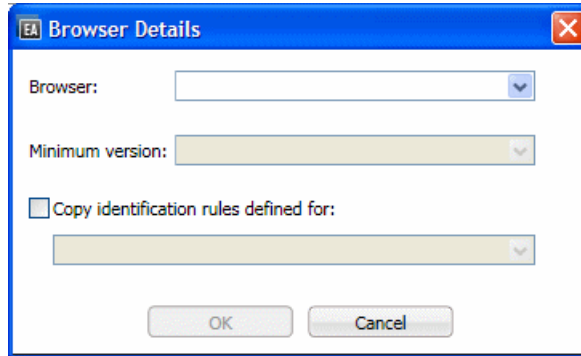
User interface elements are described below (unlabeled elements are shown in angle brackets):


UI Elements	Description
<HTML element tree>	<p>The name of the HTML element that represents the selected application control (highlighted). If relevant, additional elements in the HTML hierarchy are displayed as well.</p> <p>You can select a different HTML element in the hierarchy to represent the application control.</p>
Properties	The selected HTML element's property names and values.
Select	<p>Selects this control and include it in the set of controls that determines the rules that are created in this session.</p> <p>The dialog box closes and the control is highlighted in the application.</p> <p>Available when: The control is not currently selected.</p>
Apply	<p>Updates the set of selected controls, to use the HTML element selected in the hierarchy to represent this control.</p> <p>The dialog box closes and the control is highlighted in the application.</p> <p>Available when: The control is currently selected.</p>
Delete	<p>Removes this control from the set of controls that determines the rules that are created in this session.</p> <p>The dialog box closes and the control is not highlighted in the application.</p> <p>Available when: The control is currently selected.</p>
Cancel	Closes this dialog box without changing the set of selected controls.

Add Browser Dialog Box



This dialog box opens when you click the **Add Browser-Specific Rules** tab in the Map to Controls Tab (Test Object Class Designer). In this dialog box you provide the details of the browser for which you are creating the new set of rules.



To access	In the Map to Controls tab of the test object class designer, click the Add Browser-Specific Rules  tab.
Relevant tasks	"How to Map a Test Object Class to Application Controls" on page 58
See also	"Map to Controls Tab (Test Object Class Designer)" on page 92

User interface elements are described below (unlabeled elements are shown in angle brackets>):

UI Elements	Description
Browser	<p>The type of browser to which the rules in the new tab will apply.</p> <p>Possible values:</p> <ul style="list-style-type: none"> ▶ Default Rules. The set of default rules is used for all supported browsers that do not have a specific set of rules defined. (This value is displayed in the list only if no default rules are defined) ▶ Internet Explorer ▶ Firefox <p>Stored in: <code>name</code> attribute of the <code>Identification\Browser</code> element in the toolkit configuration XML file</p>
Minimum version	<p>The lowest version of the browser to which the rules in the new tab apply. You can type a value, or select from the list.</p> <p>You can define different sets of rules for different versions of the same browser. For example, if you define a set of rules for Internet Explorer 5 and another for Internet Explorer 7, the former is used when running on Internet Explorer 6, and the latter is used when running on Internet Explorer 8.</p> <p>Stored in: <code>min_version</code> attribute of the <code>Identification\Browser</code> element in the toolkit configuration XML file</p>
Copy identification rules defined for	<p>If you select this option, select one of the existing rule sets from the list. A copy of this set of rules is created for the browser you specified in this dialog box. You can then modify these rules as necessary for this type of browser.</p>

Operations Tab (Test Object Class Designer)

This tab enables you to design the operations your test object class supports. You can:

- ▶ Define the list of operations supported by this test object class.
You can add or remove operations or select base class operations to override.
- ▶ For operations that you add or override, you can edit the method signature and define additional information.
- ▶ Specify the default operation for this test object class (optional).

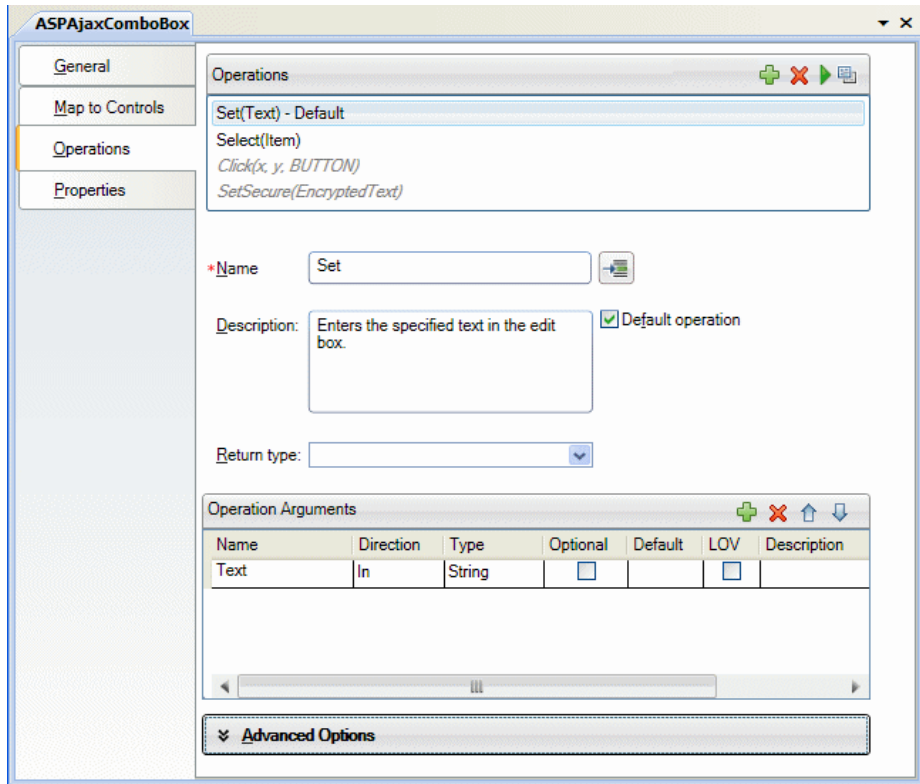
The information you define in this tab is stored in the XML files in your toolkit support set.

JavaScript function stubs are added to the relevant JavaScript file for each operation that you add or override (unless you use the advanced options to customize the name of the implementation function or file).

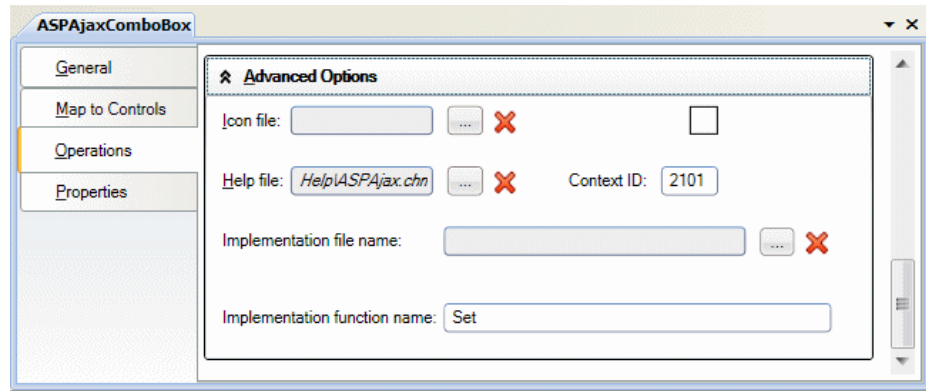


Click the **Implementation Code** button to access the function and implement it to support the operation.

The image below displays the basic options available in the Operations tab of the Test Object Class designer.



The image below displays the advanced options available in the Operations tab of the Test Object Class designer.



<p>To access</p>	<p>To access the Operations tab:</p> <ol style="list-style-type: none"> 1 In the Class View, add a new test object class or double-click an existing one. The test object class designer opens. 2 In the test object class designer, select the Operations tab. <p>To access the advanced options, click Advanced Options.</p>
<p>Important information</p>	<ul style="list-style-type: none"> ▶ You can select an inherited operation as the test object class's default operation. ▶ You cannot modify any definitions of an inherited operation. ▶ Advanced options are not available for inherited operations. ▶ Make sure to visit each tab in the test object class designer to ensure that all of the test object class details are defined correctly. <p>A red asterisk next to the name of the tab acts as a reminder that you have not yet visited this tab.</p>

Relevant tasks	"How to Design Test Object Class Operations" on page 69
See also	The section on implementing support for test object methods in the <i>HP QuickTest Professional Web Add-in Extensibility Developer Guide</i> .

The Operations tab contains the following areas:

- "Operation List Area" on page 108
- "Operation Details Area" on page 110
- "Operation Arguments Area" on page 111
- "Advanced Options Area" on page 113





Operation List Area

Displays operations you add to the test object class, and some base class operations you can choose to override.

- Inherited operations appear in italic font and cannot be modified.
- New or overridden operations appear in regular font. You can edit the methods signature in other areas of this tab.

Stored in: **Operation** elements in the test object configuration XML file. A JavaScript function stub is added to the relevant JavaScript file for each new or overridden operation. You must implement this function to support the operation.

This area also contains a toolbar with the following buttons:

UI Elements	Description
	<p>Add. Adds a new operation to the test object class definition.</p> <p>In addition, a JavaScript function stub is added to the default implementation file. For details on when this takes place, see "Changes Made Automatically to JavaScript Files" on page 33.</p> <p>Note: If you specified an Implementation file name or an Implementation function name in the advanced options, Extensibility Accelerator does not add the function stub to the JavaScript file, and you must add it manually.</p>
	<p>Delete. Deletes the selected operation from the test object definition in the test object configuration XML file.</p> <p>The corresponding JavaScript functions are not deleted.</p> <p>If you delete an overriding operation, its signature appears in italic font again.</p>
	<p>Debug Operation. Opens the Debug Test Object Operation Dialog Box (described on page 127), enabling you to run and debug the JavaScript code that you designed to implement the test object operation.</p>
	<p>Override Operation. Creates a new operation that will override the one inherited from the base class.</p> <p>The operation signature font changes to regular text and the operation details in this tab become editable.</p> <p>In addition, a JavaScript function stub is added to the default implementation file as it is when you add a new operation (for details, see above).</p>


Operation Details Area

In this area you define (or view) the name, description, and return type of the operation selected in the operation list area.

In addition, you can select the default operation and access the JavaScript function that implements the operation.

User interface elements are described below:

UI Elements	Description
Name	<p>The test object operation's name.</p> <p>If you rename an operation, and you did not customize the implementation file or function name in the advanced options, the name of the operation's JavaScript implementation function is changed as well. (It is therefore recommended to save this change immediately.)</p> <p>If you rename an overridden operation, its signature appears in italic font again, and a new operation is created with the new name.</p> <p>Stored in:</p> <ul style="list-style-type: none"> ▶ Operation element in the test object configuration XML file ▶ Method element in the toolkit configuration XML file - mapped to the JavaScript implementation function
Description	<p>A description of the operation. This description is displayed in QuickTest tooltips.</p> <p>Stored in: Operation\Description element in the test object configuration XML file</p>
Return type	<p>The type of value that the operation returns.</p> <p>This option displays a list of possible types from which you can choose. The list also includes any enumerations that you define in the Enumerations Designer (described on page 48).</p> <p>Stored in: Operation\ReturnValueType\Type element in the test object configuration XML file</p>


UI Elements	Description
	Implementation Code. Opens the relevant JavaScript file to the relevant JavaScript function.
Default operation	Indicates whether the operation is the default operation for this test object class. Default: The base class's default operation Stored in: DefaultOperationName attribute of ClassInfo element in the test object configuration XML file

Operation Arguments Area

In this area you define (or view) the arguments of the operation selected in the operation list area.

Stored in: **Argument** element in the test object configuration XML file

User interface elements are described below:

UI Elements	Description
	This toolbar contains Add and Delete buttons, and Up and Down buttons, used to set the order of the arguments.
Name	The argument name. Stored in: Name attribute of the Argument element
Direction	Specifies whether this argument is an input argument or an output argument. Stored in: Direction attribute of the Argument element
Type	The type of the argument's value. This option displays a list of possible types from which you can choose. The list also includes any enumerations that you define in the Enumerations Designer (described on page 48). Stored in: Type element within the Argument element




UI Elements	Description
Optional	<p>Specifies whether the argument is optional.</p> <p>If you define optional arguments, they must come after any mandatory arguments that the operation has.</p> <p>Stored in: IsMandatory attribute of the Argument element</p>
Default	<p>The default value for the argument. Only relevant if the argument is optional.</p> <p>Stored in: DefaultValue attribute of the Argument element</p>
LOV	<p>Indicates whether QuickTest dynamically displays a list of possible values for this argument when editing tests.</p> <p>If you select this option, you must implement a get_list_of_values JavaScript function to return the possible values from the control. By default, QuickTest calls this function from the default implementation file defined for this test object class (General tab, advanced options).</p> <p>Stored in: DynamicListOfValues attribute of the Argument element</p>
Description	<p>A description of the argument.</p> <p>This description is intended for your internal documentation purposes, it is not displayed in QuickTest Professional.</p> <p>Stored in: Description attribute of the Argument element</p>




Advanced Options Area



This area enables you to set advanced options for the operation currently selected in the operation list. If you do not define these options, QuickTest uses their default values.


To access: Click the **Advanced Options** panel.

User interface elements are described below:

UI Elements	Description
Icon file	<p>The name of the icon file that you want QuickTest to display for this operation in the run session results.</p> <p>Use the Import File  button to specify the relevant file.</p> <p>Use the Clear  button to clear the edit box.</p> <p>Default: A QuickTest default icon</p> <p>Stored in: Operation\IconInfo element in the test object configuration XML file</p> <p>If you select an icon within a .dll or .exe file, the index of the icon's location in the file is also stored in the IconInfo element.</p>
	<p>Import File. Enables you to browse to and select the icon file. You can select an icon from an .ico, .dll, or .exe file.</p> <p>If the icon you select is not located in the project's Res folder, a local copy is created in that folder. The file must be located in the project's Res folder to be properly deployed.</p> <p>Note: Avoid importing large .exe or .dll files, as these are added to your toolkit support set and deployed with it.</p>
<icon>	<p>An image of the icon you selected or the default icon.</p>

UI Elements	Description
<p>Help file</p>	<p>The name of the .chm Help file that you want QuickTest to use for context-sensitive Help on this test object class.</p> <p>Use the Import File  button to specify the relevant file.</p> <p>Use the Clear  button to clear the edit box.</p> <p>Stored in: Operation\HelpInfo element in the test object configuration XML file</p>
<p></p>	<p>Import File. Enables you to browse to and select the .chm Help file.</p> <p>If the file you select is not located in the project's Help folder, a local copy is created in that folder. The file must be located in the project's Help folder to be properly deployed.</p>
<p>Context ID</p>	<p>The numeric value that indicates the help topic to open within the specified Help file.</p> <p>Stored in: Operation\HelpInfo element in the test object configuration XML file</p>

UI Elements	Description
Implementation file name	<p>The file that contains the implementation function (optional).</p> <p>You cannot modify this value directly.</p> <p>Use the Import File  button to browse to and select the relevant file.</p> <p>Use the Clear  button to clear the edit box.</p> <p>The corresponding XML attribute in the toolkit configuration XML file is cleared, but the JavaScript file is not removed from the project.</p> <p>Default: The Default implementation file defined in the General tab</p> <p>Note: If you specify a name in this option, Extensibility Accelerator does not create a stub for the function in the relevant JavaScript file. You must create the function manually. Additionally, the function signature is not updated automatically when the operation signature is modified.</p> <p>Stored in: Method element in the toolkit configuration XML file</p>

UI Elements	Description
	<p>Import File. Enables you to browse to and select a JavaScript file.</p> <p>If you select a file that is not located in the project's JavaScript folder, a local copy is created in that folder. The file must be located in the project's JavaScript folder to be properly deployed.</p> <p>If the file that you import has the same name as an existing file in this folder, Extensibility Accelerator appends a period (.) and a number to the imported file name (before the .js file extension).</p>
<p>Implementation function name</p>	<p>The name of the function that you implement to perform the test object operation on the control.</p> <p>Default: The operation name</p> <p>Note:</p> <ul style="list-style-type: none"> ▶ If you specify a name in this option, Extensibility Accelerator does not create a stub for the function in the relevant JavaScript file. You must create the function manually. Additionally, the function signature is not updated automatically when the operation signature is modified. ▶ If you modify the name in this option, you must update the function name in the JavaScript file as well. <p>Stored in: Method element in the toolkit configuration XML file</p>

Properties Tab (Test Object Class Designer)

This tab enables you to design the identification properties of your test object class. You can:

- Define the list of identification properties for your test object class.
Add or remove properties or select base class properties to inherit and include in the list.
- Decide which properties are included in test object descriptions, which can be verified in checkpoints, and used in output values, which should be used for Smart Identification, and so on.
- Optionally, define advanced options.

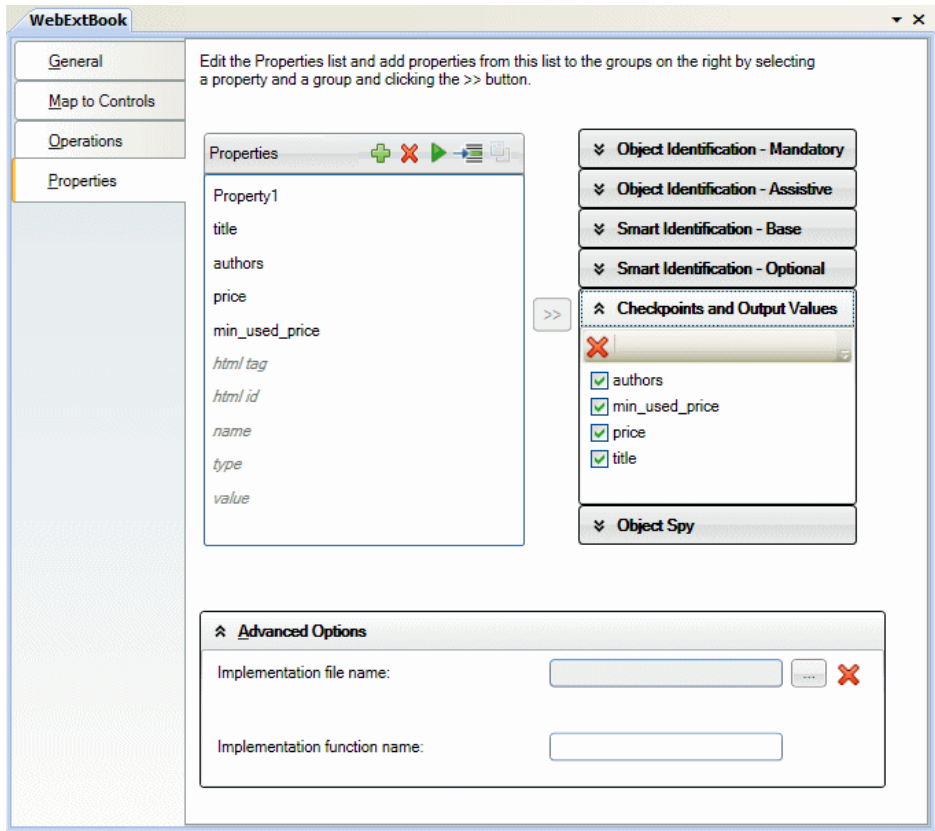
The information you define in this tab is stored in the XML files in your toolkit support set.

For each property that you add or inherit, JavaScript code segments are added in the relevant JavaScript file to the function that you implement to retrieve the property values from the control. (If you use the advanced options to customize the name of the implementation function or file, the code segments are not added).



Click the **Implementation Code** button to access the function and implement it to retrieve the property values. For details, see "Implement the JavaScript function that retrieves the identification property values from the run-time object" on page 73.

Note: You must also implement a JavaScript function that retrieves the values of the properties from the control. For details, see "Implement the JavaScript function that retrieves the identification property values from the run-time object" on page 73.



To access	<p>To access the Properties tab:</p> <ol style="list-style-type: none"> 1 In the Class View, add a new test object class or double-click an existing one. The test object class designer opens. 2 In the test object class designer, select the Properties tab. <p>To access the advanced options, click Advanced Options.</p>
Important information	<ul style="list-style-type: none"> ▶ To prevent the property grouping that you define in this tab from overwriting changes that the QuickTest user makes in the Object Identification dialog box, clear the Development mode option in the Toolkit Support Properties Designer (described on page 44) before deploying the toolkit support set for regular use. For a more detailed explanation, see the <i>HP QuickTest Professional Web Add-in Extensibility Developer Guide</i>. ▶ Make sure to visit each tab in the test object class designer to ensure that all of the test object class details are defined correctly. A red asterisk next to the name of the tab acts as a reminder that you have not yet visited this tab.
Relevant tasks	"How to Design Test Object Class Identification Properties" on page 72
See also	The section on implementing support for identification properties in the <i>HP QuickTest Professional Web Add-in Extensibility Developer Guide</i> .

The Properties tab contains the following:




- "Property List" on page 120
- "Property Usage Groups" on page 122
- "Advanced Options" on page 125



Property List

Displays properties that you add to the test object class, and some base class properties you can choose to inherit and include in the list.

- Base class properties appear in italic font and are not supported by your test object class unless you choose to inherit them.
- New or inherited properties appear in regular font and are editable.
Stored in: **IdentificationProperty** elements in the test object configuration XML file

This area also contains a toolbar with the following buttons:

Buttons	Description
	<p>Add. Adds a new editable identification property to the test object class definition.</p> <p>In addition, a segment of code is added to the JavaScript function that retrieves property values from the control. Implement this segment to retrieve the value for the new property.</p> <p>If you rename the property, the property name used in the code segment is updated. For details on when this takes place, see "Changes Made Automatically to JavaScript Files" on page 33.</p> <p>Note: If you specified an Implementation file name or an Implementation function name in the advanced options, Extensibility Accelerator does not add the code segment to the JavaScript function, and you must add it manually. The same is true for renaming the property.</p>
	<p>Delete. Deletes the selected property from the test object definition in the test object configuration XML file.</p> <ul style="list-style-type: none"> ▶ When you delete a property, if the section that supports this property in the JavaScript implementation function is empty, it is deleted as well. ▶ If you delete an inherited property, it appears in italic font again.
	<p>Debug Property Retrieval. Opens the Debug Property Retrieval Dialog Box (described on page 129), enabling you to run and debug the JavaScript code that you implement to retrieve the property value from the control.</p>

Buttons	Description
	<p>Implementation Code. Opens the relevant JavaScript file to the JavaScript function that retrieves the property values from the control.</p> <p>If you select a property before clicking this button, the file opens to the relevant section within the function.</p>
	<p>Inherit from Base Class. Adds the selected base class property to the list of the test object class's properties. In some cases, the property is also added by default to specific property groups.</p> <ul style="list-style-type: none"> ▶ The property name font changes to regular text and becomes editable. You can then add or remove the property to or from the different groups as needed. ▶ If you rename an inherited property, it appears in italic font again, and a new property is created with the new name. <p>In addition, a segment of code is added to the JavaScript function that retrieves property values from the control, as it is when you add a new property.</p> <p>Implement this segment to retrieve the value for the property, if the implementation cannot be inherited from the base class. For more information about inheriting base class implementation, see the section about extending an existing test object class in the <i>HP QuickTest Professional Web Add-in Extensibility Developer Guide</i>.</p>

Property Usage Groups


Add identification properties to the different groups in this area, to inform QuickTest of the purposes for which the properties should be used.

To add a property from the list of properties on the left to a group on the right, select the group to open it, and then double-click the property or select it and click the >> button.

Base class property grouping cannot be modified unless the property is new or inherited.

Stored in: The properties' group associations are stored in the test object configuration XML file. They are indicated by attributes of the **IdentificationProperty** elements.

User interface elements are described below (unlabeled elements are shown in angle brackets>):

UI Elements	Description
	<p>Each group has a toolbar.</p> <ul style="list-style-type: none"> ▶ The toolbars all include a Delete button, to remove the selected property from the group. ▶ The groups in which the order of the properties is significant have Up and Down buttons to enable moving the selected property within the list.
<p>Object Identification - Mandatory Group</p>	<p>Properties that QuickTest always learns as part of the description for test objects of this class.</p> <p>Note: You cannot include the same property in both Object Identification lists.</p> <p>Indicated by: ForDescription attribute set to true</p>
<p>Object Identification - Assistive Group</p>	<p>Additional properties that QuickTest can learn for a test object of the selected class to create a unique test object description.</p> <p>When QuickTest learns an object, and assistive properties are necessary to create a unique object description, QuickTest adds the assistive properties to the description one at a time until it has enough information to create a unique description, according to the order you set in this group.</p> <p>Indicated by: ForAssistive attribute set to true, AssistivePropertyValue attribute set to the position of the property within the group</p>



UI Elements	Description
<p>Smart Identification - Base Group</p>	<p>Properties that QuickTest learns as base filter properties for this test object class. The Smart Identification mechanism uses these properties to create a list of possible candidate objects.</p> <p>Note: You cannot include the same property in both Smart Identification lists.</p> <p>Indicated by: ForBaseSmartID attribute set to true</p>
<p>Smart Identification - Optional Group</p>	<p>Properties that QuickTest learns as optional filter properties for this test object class. The Smart Identification mechanism uses these properties in the specified order to narrow down the object candidate list to one object.</p> <p>When QuickTest uses Smart Identification, it creates a list of possible candidate objects according the base filter properties, and then checks the values of the optional filter properties one by one according to the order you set, until it narrows down the candidate list to one object.</p> <p>Indicated by: ForOptionalSmartID attribute set to true, OptionalSmartIDPropertyValue attribute set to the position of the property within the group</p>
<p>Checkpoints and Output Values Group</p>	<p>Properties available in the Checkpoint Properties and Output Value Properties dialog boxes in QuickTest.</p> <p>If the check box for a property is selected, it is selected by default in the Checkpoint Properties dialog box when creating a checkpoint.</p> <p>Indicated by: ForVerification and, if selected, ForDefaultVerification attribute set to true</p>
<p>Object Spy Group</p>	<p>Indicated by: ForSpy attribute set to true</p>


Advanced Options

This area enables you to set advanced options for implementing property support. If you do not define these options, QuickTest uses default values.

To access: Click the **Advanced Options** panel.

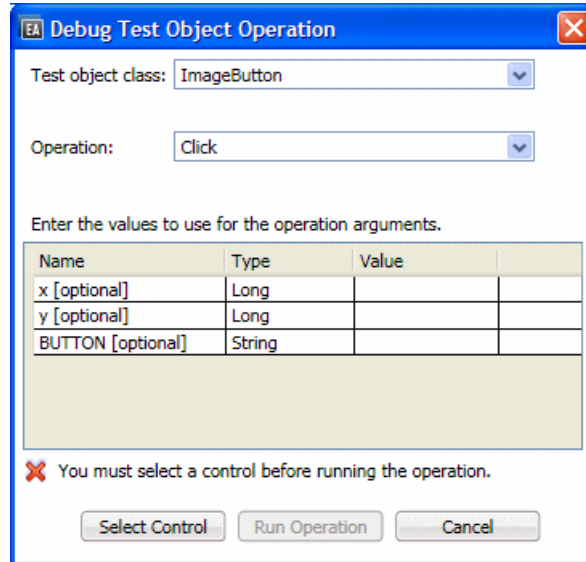
User interface elements are described below:


UI Elements	Description
Implementation file name	<p>The file that contains the implementation function (optional). You cannot modify this value directly.</p> <p>Use the Import File  button to browse to and select the relevant file.</p> <p>Use the Clear  button to clear the edit box.</p> <p>The corresponding XML attribute in the toolkit configuration XML file is cleared, but the JavaScript file is not removed from the project.</p> <p>Default: The Default implementation file defined in the General tab</p> <p>Note: If you specify a name in this option, Extensibility Accelerator does not create a stub for the function in the relevant JavaScript file. You must create the function manually. Additionally, code segments are not added or updated automatically when you add or modify properties.</p> <p>Stored in: Property element in the toolkit configuration XML file</p>

UI Elements	Description
	<p>Import File. Enables you to browse to and select a JavaScript file.</p> <p>If you select a file that is not located in the project's JavaScript folder, a local copy is created in that folder. The file must be located in the project's JavaScript folder to be properly deployed.</p> <p>If the file that you import has the same name as an existing file in this folder, Extensibility Accelerator appends a period (.) and a number to the imported file name (before the .js file extension).</p>
<p>Implementation function name</p>	<p>The name of the function that you implement to retrieve the values of the identification properties from the control.</p> <p>Default: <code>get_property_value</code></p> <p>Stored in: Property element in the toolkit configuration XML file</p> <p>Note:</p> <ul style="list-style-type: none"> ▶ If you specify a name in this option, Extensibility Accelerator does not create a stub for the function in the relevant JavaScript file. You must create the function manually. Additionally, code segments are not added or updated automatically when you add or modify properties. ▶ If you modify the name in this option, you must update the function name in the JavaScript file as well.

Debug Test Object Operation Dialog Box

This dialog box enables you to run your test object operations from within Extensibility Accelerator, so that you can test and debug your JavaScript implementation functions.



To access	<p>Do one of the following:</p> <ul style="list-style-type: none"> ▶ Select Project > Debug Test Object Operation ▶ In the Operations Tab (Test Object Class Designer) described on page 105, click the Debug Operation . ▶ In the Class View, right-click the operation that you want to run and select Debug. <p>Tip: If the operations are not displayed in the Class View, first select the test object class.</p>
Important information	<p>Make sure that you select the relevant control in the application before you run the operation.</p>

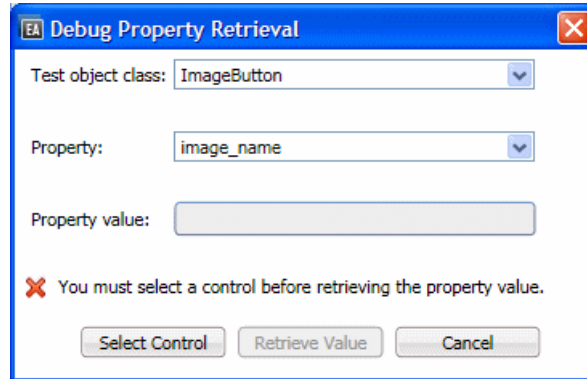
Relevant tasks	"How to Test and Debug Your Test Object Operation Support" on page 75
See also	"JavaScript Function Debugging" on page 54


User interface elements are described below (unlabeled elements are shown in angle brackets):

UI Elements	Description
Test object class	The test object class of the operation that you want to run.
Operation	The test object operation that you want to run.
<operation arguments>	<p>The arguments for the operation. Includes the following for each argument:</p> <p>Name. The name of the argument and whether it is optional. (Read only)</p> <p>Type. The type of value the operation expects for this argument. (Read only)</p> <p>Value. The value to use for the argument when running the operation. You must enter values for all mandatory arguments.</p>
Select Control	<p>Highlights all of the controls that match the test object class' mapping rules in all open Web applications, and enables you to select the control on which you want to run the operation.</p> <p>Available when: A Test object class is selected in the dialog box.</p>
Run Operation	<p>Runs the operation on the selected control, calling the JavaScript implementation that you designed.</p> <p>Available when: A control is selected in the application.</p>
Cancel	Close this dialog box without running the operation.

Debug Property Retrieval Dialog Box

This dialog box enables you to retrieve the value of a test object's identification property using Extensibility Accelerator, so that you can test and debug the JavaScript function that you wrote to retrieve the values.



To access	Do one of the following: <ul style="list-style-type: none"> ▶ Select Project > Debug Property Retrieval ▶ In the Properties Tab (Test Object Class Designer) described on page 117, click the Debug Property Retrieval  button in the property list toolbar.
Important information	Make sure that you select the relevant control in the application before you click Retrieve Value .
Relevant tasks	"How to Test and Debug Your Property Retrieval Function" on page 78.
See also	"JavaScript Function Debugging" on page 54

User interface elements are described below:

UI Elements	Description
Test object class	The test object class of the property that you want to retrieve.
Property	The property that you want to retrieve.
Property value	The value that your property retrieval implementation function returns. This value is displayed (in read only format) after you click Retrieve Value and the run session is completed.
Select Control	Highlights all of the controls that match the test object class' mapping rules in all open Web applications, and enables you to select the control from which you want to retrieve the property value. Available when: A Test object class is selected in the dialog box.
Retrieve Value	Retrieves the value of the property from the selected control by running the JavaScript implementation function that you designed, passing the selected property name as the parameter. (The property name is passed in lowercase letters, simulating QuickTest's property value retrieval behavior.) Available when: A control is selected in the application.
Cancel	Close this dialog box without running the property retrieval function.

Troubleshooting and Limitations - Supporting a Control

This section describes troubleshooting and limitations for supporting a custom control.

Selecting controls when mapping rules automatically

- ▶ You cannot select controls inside an **object** element on a Web page.

Workaround: Copy the HTML source code from inside the **object** element to another Web page and select the relevant controls from that page.

- ▶ In some specific cases, when you click a control to select it, the application responds to the click (for example, by closing a drop-down menu) instead of, or in addition to, Extensibility Accelerator recognizing the selection. This sometimes makes it difficult to select a control to create rules.

Workaround: In some cases, you can click a higher HTML element in the hierarchy and then select this control from within the displayed hierarchy. In other cases you might have to manually create a rule for the control.

- ▶ The Extensibility Accelerator rule editor (on the Map to Controls tab of the test object class designer) does not support selection of controls in a dialog box.
- ▶ To select controls in a Web page, make sure that in **Tools > Internet Options** the options specified below are enabled. (Note that in some operating systems these options may be disabled by default.)
 - ▶ **Security (Internet Zone) > Custom Level > ActiveX controls and plug-in -> Binary and script behaviors**
 - ▶ **Security (Internet Zone) > Custom Level > Scripting-> Active scripting**

Toolkit configuration files

- ▶ When the **Identification** element in the toolkit configuration XML file includes **HTMLTags** elements, they are not displayed in the rule editor on the Map to Controls tab of the test object class designer.

Note: This is relevant for some of the controls in the Extensibility Accelerator sample Web 2.0 support projects.

5

Custom Toolkit Support Deployment

This chapter includes:

Concepts

- ▶ Deployment Objectives on page 134
- ▶ Deployment Destinations on page 136
- ▶ Deployment File Structure on page 136

Tasks

- ▶ How to Deploy a Toolkit Support Set on page 137

Concepts

Deployment Objectives

You can deploy your toolkit support set at various stages of development to test how it works on QuickTest.

Example

If you deploy after ...	You can verify that...
You define a test object class and its operations, but do not implement the JavaScript function for the operation.	You can use this test object class when editing steps in QuickTest, and the operations are displayed correctly in the Keyword View and in IntelliSense.
You define the mapping rules for a test object class, but do not define any operations.	QuickTest can learn objects of this type.
You implement the JavaScript function for an operation.	QuickTest can run steps that perform the operation.
You implement the JavaScript function that retrieves the identification property values from the control.	You can see the identification property values in the Object Spy.

When you complete the development of the toolkit support set, you can deploy it for regular use, or package it for distribution. For details, see "Deployment Destinations" on page 136.

Setting the Development Mode Option

When you deploy the toolkit support set during the design stages, keep the **Development mode** option in the Toolkit Support Properties Designer (described on page 44) selected. This ensures that if you modified attributes of **IdentificationProperty** elements in the test object configuration XML file, QuickTest uses all of the changes you made.

Be sure to clear this option before you deploy the toolkit support set for regular use. Otherwise, every time QuickTest opens, it will refresh the property lists based on the definitions in the test object configuration XML file. If QuickTest users change the property lists using the Object Identification dialog box, their changes will be lost when they reopen QuickTest.

For more details, see the section on modifying deployed support in the *HP QuickTest Professional Web Add-in Extensibility Developer Guide*.

Validating the Toolkit Support Set

Before deploying the toolkit support set, Extensibility Accelerator saves all of your changes and validates the information. If mandatory data is missing, or if conflicts or discrepancies are found between information in the different files, the Error List window displays messages that explain the problems encountered.

When you deploy the toolkit support set during the design stages, many of the issues reported in the Error List can be ignored.

Before you deploy the toolkit support set for regular use, make sure to address these issues.

Deployment Destinations

You can deploy a toolkit support set locally, making it immediately available for use with QuickTest or the QuickTest Add-in for ALM/QC, if those are installed on the same computer as Extensibility Accelerator.

Alternatively, you can automatically package the toolkit support set in a **.zip** file, which you can then distribute and unzip for use on other QuickTest computers. In the **.zip** file, the files are stored in the same structure as a deployed toolkit support set. Therefore, in order to use this toolkit support set on a QuickTest computer, unzip it in **<QuickTest installation folder>\dat\Extensibility\Web**.

For task details, see "How to Deploy a Toolkit Support Set" on page 137.

Deployment File Structure

The toolkit support set files are deployed in the following structure:

```

Parent folder or zip file
|
|---<ToolkitName>TestObjects.xml file
|---Toolkits folder:
|   |
|   |---<ToolkitName> folder
|       |
|       |---<ToolkitName>.xml file
|       |---JavaScript folder containing JavaScript files
|       |---Res folder containing icon files (optional)
|       |---Help folder containing .chm files (optional)

```

If the toolkit support set was previously deployed in a different structure, files that are not overwritten remain in their original locations but are no longer used. For example, if you imported a toolkit support set developed for QuickTest 9.5 or 10.00, where the JavaScript files were not stored in a **JavaScript** subfolder, and then re-deployed this toolkit support set. You might want to delete the unused files to avoid future confusion.

Tasks

How to Deploy a Toolkit Support Set

This task describes how to deploy a toolkit support set. You can deploy it directly to QuickTest or the QuickTest Add-in for ALM/QC if they are installed on the local computer. Alternatively, you can package the toolkit support set for distribution to other computers.

You can deploy a toolkit support set:

- ▶ During the design stages, to test its functionality.
- ▶ When the design is complete, to begin using the toolkit support.

To deploy the toolkit support set:

- 1 Open an extensibility project that contains at least one test object class.
- 2 If the toolkit support set design is complete, and you are distributing it for regular use:
 - ▶ Clear the **Development mode** option in the Toolkit Support Properties Designer (described on page 44).
 - ▶ Save all your changes and make sure that you addressed all issues listed in the Error List Window.

For details, see "Deployment Objectives" on page 134.

- 3 Depending on your deployment destination, do one of the following:
 - ▶ To deploy to QuickTest (if it is installed on this computer), select **Project > Deploy > Deploy to QuickTest Professional**.
 - ▶ To deploy to the QuickTest Add-in for ALM/QC (if it is installed on this computer), select **Project > Deploy > Deploy to QuickTest Add-in For Quality Center**.
 - ▶ To package the toolkit support set for distribution, select **Project > Deploy > Deploy to Zip File**. In the **Save Zip File As** dialog box that opens, specify the file path for the **.zip** file that you want to create.

Note: If you have unsaved changes in your project, you will be prompted to save them before the Deploy command is carried out. The **Save Zip File As** dialog box opens only after you complete the saving process.

The toolkit support set files are deployed in the structure described in "Deployment File Structure" on page 136.

If you deploy to QuickTest or the QuickTest Add-in for ALM/QC, Extensibility Accelerator locates the QuickTest or Add-in installation folder and deploys the files to the **<QuickTest or Add-in installation folder>\dat\Extensibility\Web** folder.

If you deploy to a **.zip** file, Extensibility Accelerator creates the same file structure within the zip file. To use this toolkit support set with QuickTest or the QuickTest Add-in for ALM/QC installed, unzip it in the **<QuickTest or Add-in installation folder>\dat\Extensibility\Web** folder.