HP Extensibility Accelerator for HP Functional Testing

Software Version: 12.00

Windows ® operating systems

User Guide

Document Release Date: December 2014

Software Release Date: March 2014

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 Notice

Trademark Notices

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

Google™ and Google Maps™ are trademarks of Google Inc.

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

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

Oracle and Java are registered trademarks of Oracle and/or its affiliates.

Documentation Updates

The title page of this document contains the following identifying information:

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

To check for recent updates or to verify that you are using the most recent edition of a document, go to: https://softwaresupport.hp.com/group/softwaresupport/search-result.

This site requires an HP Passport account. If you do not have one, click the **Create an account** button on the HP Passport Sign in page.

Support

Visit the HP Software Support Online web site at: https://softwaresupport.hp.com

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: https://softwaresupport.hp.com and click **Register**.

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

HP Software Solutions & Integrations and Best Practices

Visit **HP Software Solutions Now** at https://h20230.www2.hp.com/sc/solutions/index.jsp to explore how the products in the HP Software catalog work together, exchange information, and solve business needs.

Visit the **Cross Portfolio Best Practices Library** at https://hpln.hp.com/group/best-practices-hpsw to access a wide variety of best practice documents and materials.

Contents

Welcome to Extensibility Accelerator for HP Functional Testing

This chapter includes:

Extensibility Accelerator Overview

UFT 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 Unified Functional Testing 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 UFT 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 UFT to extend the Web Add-in to support the custom controls. For details on toolkit support sets, see "Custom Toolkit Support Sets" on page 29.

To use Extensibility Accelerator, you should be familiar with:

- Unified Functional Testing (UFT) and the Web Add-in
- XML
- JavaScript programming

Extensibility Accelerator Documentation Content

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 UFT Web Add-in Extensibility Developer Guide* (**Help > Web Add-in Extensibility Developer Guide**).

Each guide also includes a step-by-step tutorial in which you develop support for a sample custom control.

These guides are also available in printer-friendly Adobe portable document format (PDF), 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.

> **Note:** The information, examples, and screen captures in this guide focus specifically on working with UFT GUI tests. However, all toolkit support sets developed using the UFT Web Add-in Extensibility can also be used to enable Sprinter's Power Mode to learn Web objects that are not supported out-of-the-box.

All references to UFT in this guide apply to both UFT and Sprinter. For more information about Sprinter, see the *HP Sprinter User Guide*.

Additional Online Resources

The following additional online resources are available:

Resource	Description
Troubleshooting & Knowledge Base	The Troubleshooting page on the HP Software Support Web site where you can search the Self-solve knowledge base. The URL for this Web site is http://h20230.www2.hp.com/troubleshooting.jsp.
HP Software Support	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. The URL for this Web site www.hp.com/go/hpsoftwaresupport. • Most of the support areas require that you register as an HP Passport user and sign in. Many also require a support contract. • To find more information about access levels, go to: http://h20230.www2.hp.com/new_access_levels.jsp • To register for an HP Passport user ID, go to: http://h20229.www2.hp.com/passport-registration.html
HP Software Web site	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. The URL for this Web site is www.hp.com/go/software

Chapter 1: Introducing Extensibility Accelerator

This chapter includes:

Concepts

UFT Web Add-in Extensibility - Overview

The UFT 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 UFT Web Add-in.

When UFT 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 UFT.

The type of test object that UFT 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 UFT 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 UFT 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 UFT Web Add-in Extensibility Developer Guide*, available in the `<Extensibility Accelerator installation>\Help` folder.

What Extensibility Accelerator Helps You Do

To extend the UFT Web Add-in to support custom Web toolkits, you create **custom toolkit support sets** and deploy them to UFT. The toolkit support set is comprised of XML configuration files and JavaScript functions. For details, see "Custom Toolkit Support Sets" on page 29.

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

1. **Planning how you want UFT to operate on your controls**

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

   For details, see the section on planning UFT support for your toolkit in the *HP UFT 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 UFT 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 UFT

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

For task details, see "How to Create or Update Support for a Custom Toolkit" on page 33.

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.

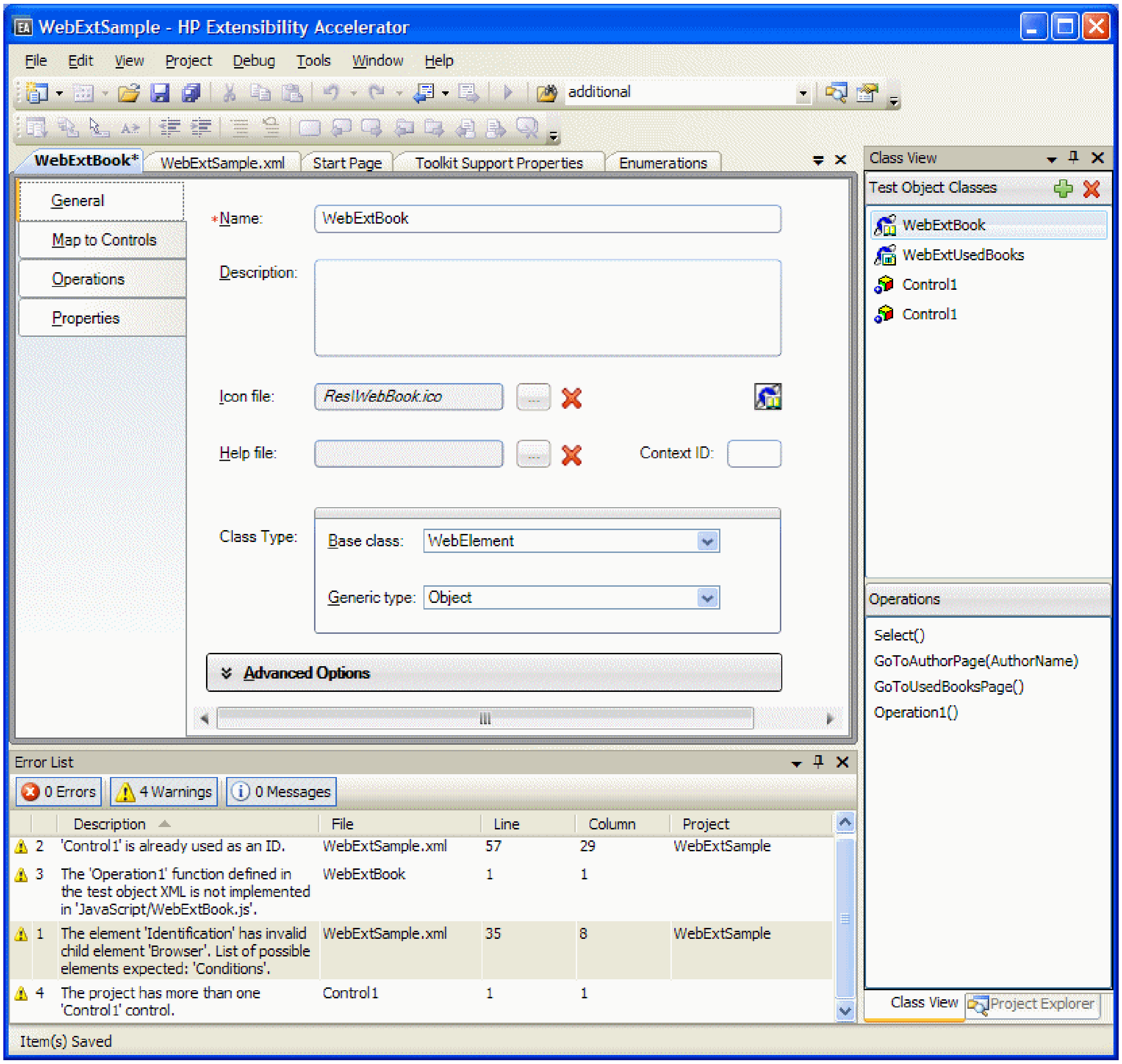
You develop a toolkit support set in an extensibility project. The main Extensibility Accelerator functionalities are available only when you have a project open. You can open only one project at a time.

This section introduces the Extensibility Accelerator window and lists the main areas in 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.

If you are working in Windows 8, you can access program files previous accessible from the **Start** menu in the Windows 8 **Start** screen. UFT Documentation and other files are accessible in Windows 8 from the **Apps** screen.



*Name: WebExtBook

Description:

Icon file: Res\WebBook.ico

Help file:

Context ID:

Class Type:

Base class: WebElement

Generic type: Object

	Description	File	Line	Column	Project
2	'Control1' is already used as an ID.	WebExtSample.xml	57	29	WebExtSample
3	The 'Operation1' function defined in the test object XML is not implemented in 'JavaScript/WebExtBook.js'.	WebExtBook	1	1	
1	The element 'Identification' has invalid child element 'Browser'. List of possible elements expected: 'Conditions'.	WebExtSample.xml	35	8	WebExtSample
4	The project has more than one 'Control1' control.	Control1	1	1	

The Extensibility Accelerator window contains:

- "Main Area" on the next page
- "Additional Windows" on page 20
- "Menus and Toolbars" on page 21

Main Area

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

Start Pages

Extensibility Accelerator displays two Start Pages:

- **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.
- **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 UFT 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 UFT 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 82.
- "Map to Controls Tab (Test Object Class Designer)" on page 95
- "Operations Tab (Test Object Class Designer)" on page 109
- "Properties Tab (Test Object Class Designer)" on page 121

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 UFT 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 95) 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 38.

  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 40.

- **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 UFT 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.

Chapter 2: Installing the Extensibility Accelerator

This chapter includes:

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 Unified Functional Testing program group (**Start > Program Files > HP Software > HP Unified Functional Testing > Extensibility Accelerator**).

  > **Note:** If you are working in Windows 8, you can access program files previous accessible from the **Start** menu in the Windows 8 **Start** screen. UFT Documentation and other files are accessible in Windows 8 from the **Apps** screen.

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

- **A demo movie.**

  You can access the demo movie from the Extensibility Accelerator Start Page or Help menu (**Help > Product Feature Movies)**.

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

- **Documentation.**

  The *HP UFT Extensibility Accelerator for HP Functional Testing User Guide*, and the *HP UFT Web Add-in Extensibility Developer Guide*, in both online Help format and printer-friendly Adobe portable document (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 4.5
- Microsoft Visual C++ 2008 SP1 Run-time Components
- Visual Studio 2008 Shell (isolated mode) with SP1 Redistributable Package

The Extensibility Accelerator for HP Functional Testing installation includes the installation programs for these prerequisites, and runs them before installing Extensibility Accelerator, if relevant. The installation is available from the **Add-in**

Extensibility and Web 2.0 Toolkits option in the Unified Functional Testing setup program.

Note:

- As part of this process, an html page opens in your browser. To complete the installation successfully, this page must be opened in Internet Explorer.
- 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-UFT Computer

UFT 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 UFT. You can use this functionality even on a computer where UFT is not installed. The debugging mechanism enables you to debug part of your JavaScript functions locally, without deploying the toolkit support set to UFT.

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

Chapter 3: Supporting a Custom Toolkit

This chapter includes:

Concepts

Custom Toolkit Support Sets

To extend the UFT Web Add-in to support custom Web toolkits, you create **custom toolkit support sets** and deploy them to UFT. 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 UFT operates on the custom controls. The JavaScript functions provide an interface between UFT 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 33.

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 UFT 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 UFT 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. UFT 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 UFT Web Add-in Extensibility Developer Guide*.

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

  The icon files contain icons used in UFT 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 40.

For details on the structure of a toolkit support set deployed to UFT, see "Deployment File Structure" on page 140.

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, 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" below
- "Open, create, or import an extensibility project" below
- "Define the toolkit support properties - Optional" on the next page
- "Create or update support for a single control" on the next page
- "Deploy the toolkit support to UFT, or package it for distribution" on the next page

1. **Prerequisites - Plan your support**

   For details, see the section on planning UFT support for your toolkit in the *HP UFT 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 the next page.

  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 57.

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

For details, see "How to Deploy a Toolkit Support Set" on page 141.

You can practice developing support for a custom toolkit by performing the tutorial. For details, see "Tutorial: Create Support For a Custom Web Control Using Extensibility Accelerator" on page 144.

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 33.

This task includes the following steps:

- "Prerequisites" below
- "Import the toolkit support set" on the next page
- "Results" on the next page

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 UFT, as described in the section on deploying toolkit support sets in the *HP UFT 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. <UFT 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" on page 42 that opens to browse to the toolkit support set and import it.

3. ## Results

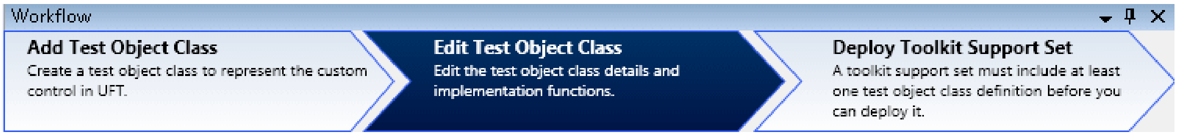
   The toolkit support set files are copied into a newly created Extensibility Accelerator project. For details on the project file structure, see "Project Explorer" on page 40.

   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.



Add Test Object Class
Create a test object class to represent the custom control in UFT.

Edit Test Object Class
Edit the test object class details and implementation functions.

Deploy Toolkit Support Set
A toolkit support set must include at least one test object class definition before you can deploy it.

To access	Select **View > Workflow**.
Relevant tasks	"How to Create or Update Support for a Custom Toolkit" on page 33

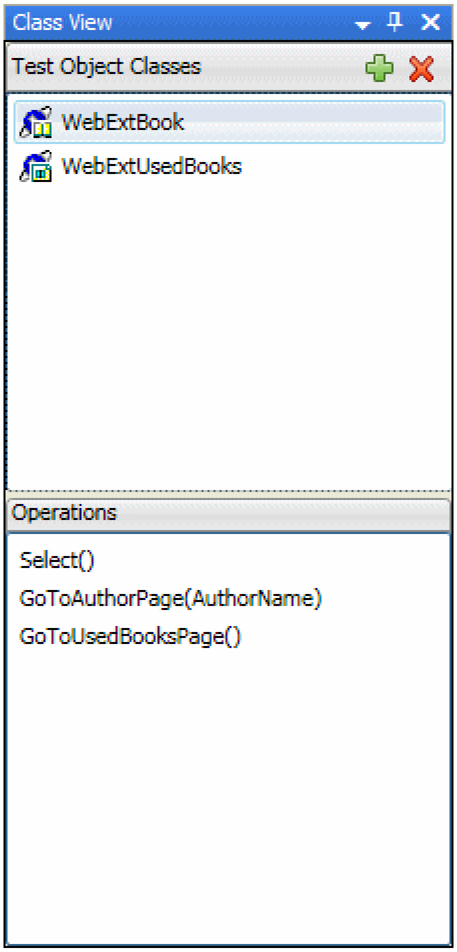
User interface elements are described below:

UI Elements	Description
Add Test Object Class	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 57. **Highlighted when:** No test object classes are defined in the project.

UI Elements	Description
Edit Test Object Class	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.
Deploy Toolkit Support	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 141. **Available when:** At least one test object class is defined in the project. **Highlighted when:** A deploy command is in progress.

Class View

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



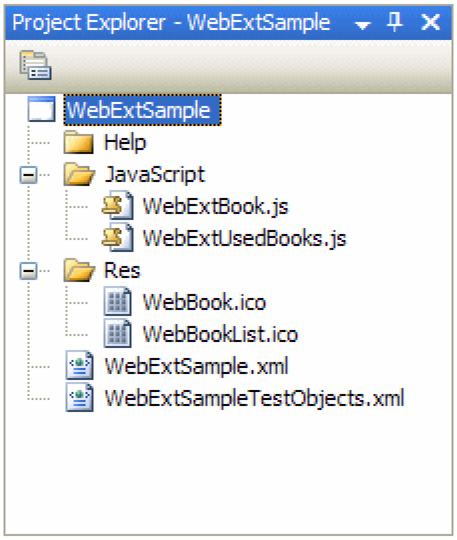
To access	When a project is open, select **View > Class View**.
Important information	This window enables you to do the following: • Add or delete test object classes. • Open the test object class designer. • Open the implementation code for a test object operation. • Start a debug session for a test object operation.

User interface elements are described below:

UI Elements	Description
Test Object Classes	The list of test object classes defined in the open project. In this area, you can: • Use the toolbar buttons to add and delete test object classes. • Double-click a test object class to open its designer.
Operations	The operations defined for the selected test object class. In this area, you can: • Double-click an operation to open the test object class designer. It opens to the Operations tab (described on page 109) 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 131). 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	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. For the same reason, you should also avoid using the **Exclude From Project** command, which is available when you right-click.

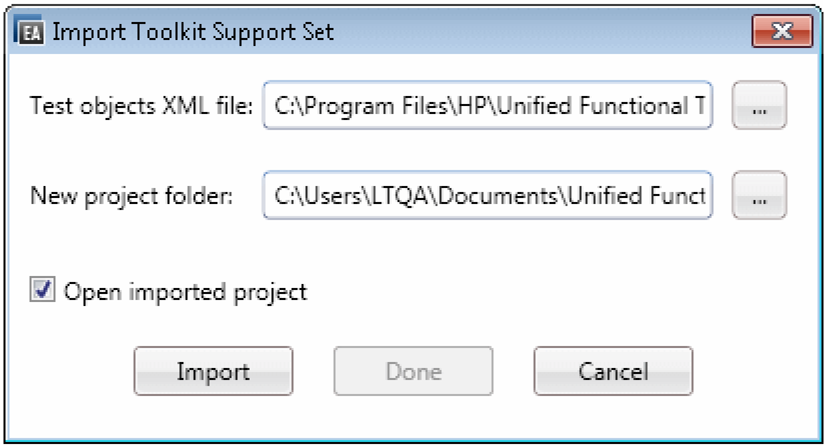
The Project Explorer tree contains the following:

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 UFT to use for context-sensitive Help for the test object classes you define. **Supported file type:** `.chm`

Tree Node	Description
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 UFT 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.



Test objects XML file: C:\Program Files\HP\Unified Functional T

New project folder: C:\Users\LTQA\Documents\Unified Funct

☑ Open imported project

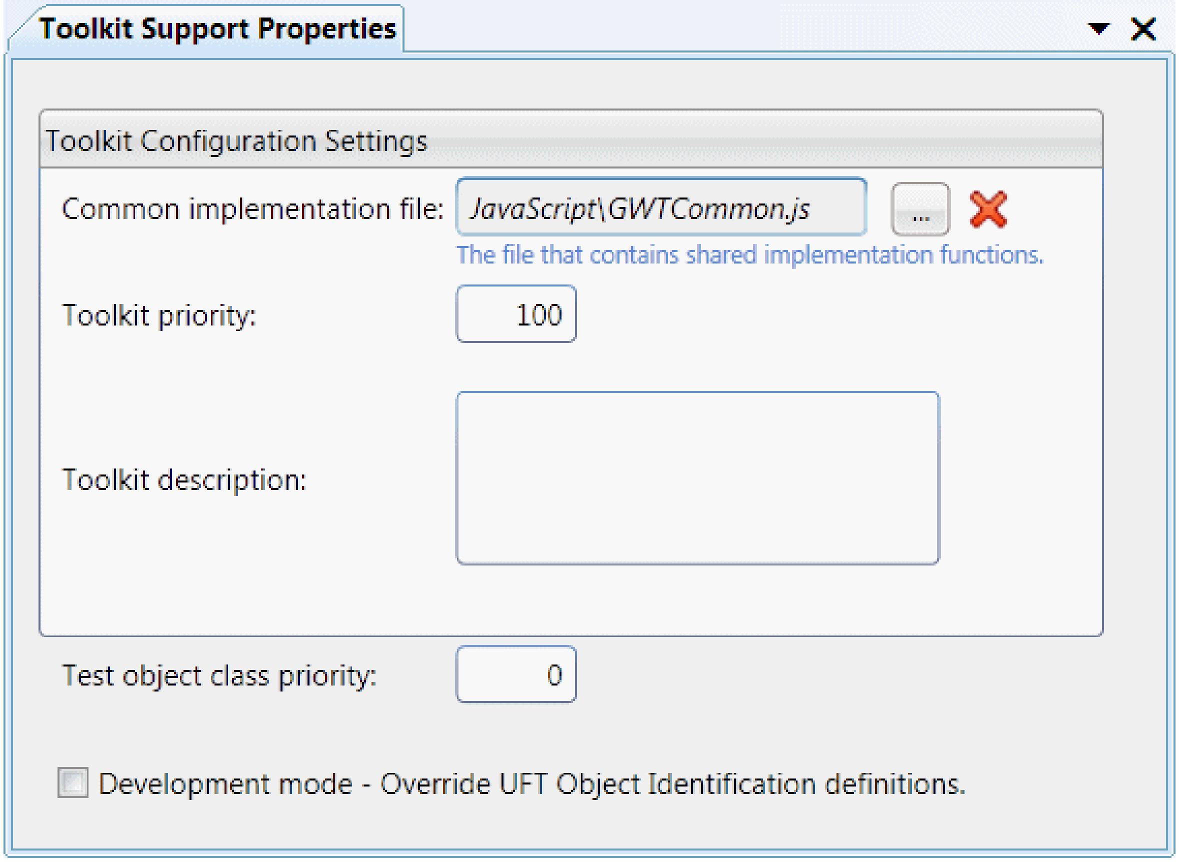
To access	Select **File > Import Toolkit Support Set**.
Important information	The toolkit support set that you want to import must have the structure described in "Prerequisites" on page 35.
Relevant tasks	"How to Import an Existing Toolkit Support Set" on page 35

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 UFT treats this toolkit support set.



Common implementation file: JavaScript\GWTCommon.js

The file that contains shared implementation functions.

Toolkit priority: 100

Toolkit description:

Test object class priority: 0

□ Development mode - Override UFT Object Identification definitions.

To access	Select **View > Toolkit Support Properties**.
Important information	• 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, UFT uses default values.
Relevant tasks	"How to Create or Update Support for a Custom Toolkit" on page 33

User interface elements are described below:

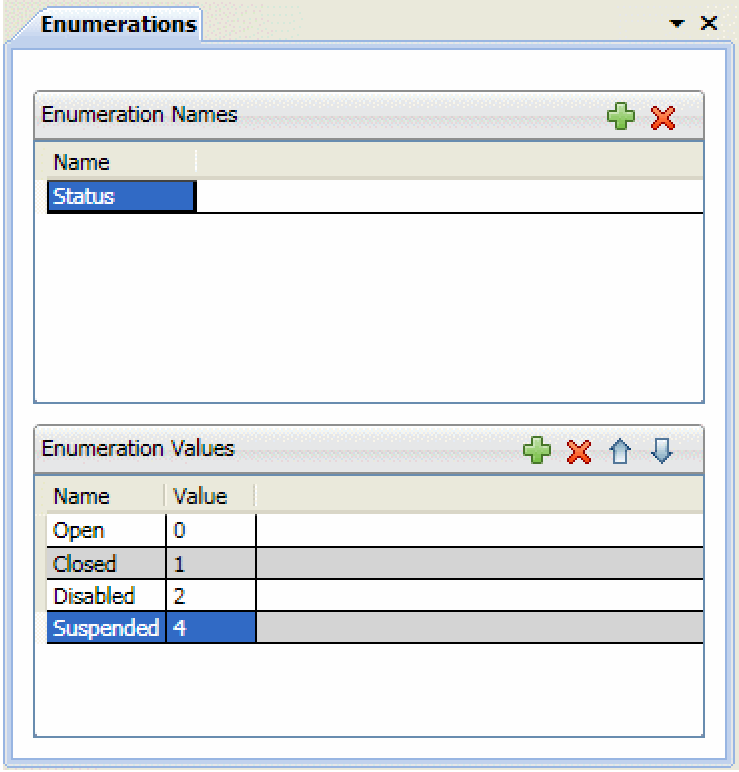
UI Elements	Description
Common implementation file	The name of a file that contains shared JavaScript functions called from your other implementation functions. (Optional) You cannot modify this value directly. Use the **Import File** ... button to browse to and select the relevant JavaScript file. Use the **Clear** ✖ button to clear the edit box. The corresponding XML attribute in the toolkit configuration XML file is cleared, but the JavaScript file is not removed from the project. **Stored in:** A **Control\Settings\Variable** element named **common_file** in the toolkit configuration XML file
...	**Import File.** Enables you to browse to and select a JavaScript file. The file must be located in the project's `JavaScript` folder to be properly deployed. Therefore, if you select a file from another location, a local copy of the file is immediately created in the `JavaScript` folder. 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).
Toolkit priority	The priority of the toolkit. When UFT 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). **Default:** 100 **Note:** In this edit box, you can type only numeric characters. **Stored in: priority** attribute of the **Controls** element in the toolkit configuration XML file

UI Elements	Description
Toolkit description	A description of the toolkit. UFT displays this description in the Add-in Manager dialog box when the toolkit support set's environment name is selected. If you are developing this toolkit support set for distribution, include a **Provided by** clause indicating the relevant person or company. Additionally, you might want to include a version number in this description. **Stored in: Controls\Description** elements in the toolkit configuration XML file
Test object class priority	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). **Default:** 0 (the lowest priority) **Note:** In this edit box, you can type only numeric characters. **Stored in: Priority** attribute of the **TypeInformation** element in the test object configuration XML file

UI Elements	Description
Development mode - Override UFT Object Identification definitions	Specifies whether the user is in development mode. • 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, UFT 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 UFT user makes in the Object Identification dialog box. For details, see the section on modifying deployed support in the *HP UFT Web Add-in Extensibility Developer Guide*. **Stored in: DevelopmentMode** attribute of the **TypeInformation** element in the test object configuration XML file

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.



Name
Status

Name	Value
Open	0
Closed	1
Disabled	2
Suspended	4

To access	Select **View > Enumerations**.
Relevant tasks	"How to Design Test Object Class Operations " on page 71
See also	"Operations Tab (Test Object Class Designer)" on page 109

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

UI Elements	Description
Enumeration Names	The name for the list of values. This area includes: • ✚ ✖ 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.\\ **Note:** In this edit box, you can enter only English letters, numeric characters, hyphens, and underscores. The first character must be a letter. **Stored in: ListOfValues** element in the test object configuration XML file

UI Elements	Description
Enumeration Values	The names and values of the items in the list that is currently selected in the **Enumeration Names** area. This area includes: • ✚ ✖ ⇧ ⇩ 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. **Note:** (In this edit box, you can enter only English letters, numeric characters, hyphens, and underscores. The first character must be a letter.) • **Value.** The integer value of the enumeration item. Click in this box to edit the value. **Default value:** The value of the last item in the list + 1 **Stored in: ListOfValues\EnumValue** element in the test object configuration XML file

Chapter 4: Supporting a Custom Control

This chapter includes:

Concepts

Base Class Selection

When you define a test object class in the "General Tab (Test Object Class Designer)", described on page 82, 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 109), 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 121), 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 details, see the section on extending an existing test object class in the *HP UFT 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.

How Extensibility Accelerator Tests Your Control Mapping

After you design the mapping rules and identification function that you want UFT to use to identify the types of controls for which a specific test object class is used, you can test how this identification works. You do this in the "Map to Controls Tab (Test Object Class Designer)", (described on page 95), and you do not need to have UFT installed.

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 UFT uses to identify the test object class to use for a custom Web control. For more details, see the section on teaching UFT to identify the test object class to use for a custom Web control in the *HP UFT Web Add-in Extensibility Developer Guide*.

Considerations when testing your JavaScript identification function:

- **_util** methods are relevant only when running in the UFT 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 (**Debug > Windows > Output**).
- 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 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.

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 UFT 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 UFT would during a run session. This enables you to test and debug the support you designed.

While your JavaScript function runs, 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 details, see the MSDN Visual Studio Help.

Note: _util methods are relevant only when running in the UFT 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 76 and "How to Test and Debug Your Property Retrieval Function" on page 79.

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 31.

This task includes the following steps:

- "Prerequisite - Open an existing project or create a new one" on the next page
- "Design a test object class to represent your control in UFT" on the next page
- "Map the test object class to the relevant type of controls" on the next page
- "Design and debug the test object class operations" on the next page
- "Design the test object class's identification properties" on the next page
- "Implement support for recording on the control - Optional" on the next page
- "Deploy and test your support" on page 59

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

For details, see the "Open, create, or import an extensibility project" on page 33 step in "How to Create or Update Support for a Custom Toolkit" on page 33.

2. Design a test object class to represent your control in UFT

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 82.

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 the next page.

4. Design and debug the test object class operations

For details, see "How to Design Test Object Class Operations " on page 71.

5. Design the test object class's identification properties

For details, see "How to Design Test Object Class Identification Properties" on page 73.

6. Implement support for recording on the control - Optional

a. In the "General Tab - Advanced Options" area of the "General Tab (Test Object Class Designer)", described on page 88, 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 UFT to call when the events occur during a recording session.

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

7. **Deploy and test your support**

   For details, see "How to Deploy a Toolkit Support Set" on page 141.

   You can practice developing support for a custom control by performing the tutorial. For details, see "Tutorial: Create Support For a Custom Web Control Using Extensibility Accelerator" on page 144.

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 UFT should use the test object class. You can create different rules to support different browser versions.

For more details on mapping rules, see the section on teaching UFT to identify the test object class to use for a custom Web control in the *HP UFT 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 57.

This task includes the following steps:

- "Prerequisites" below
- "Create a tab for a browser-specific rule set - Optional" below
- "Expand the panel for the type of rules you want to create" on the next page
- "Create a set of mapping rules automatically" on the next page
- "Edit mapping rules manually - Optional" on the next page
- "Test your mapping rules on an application and update them if necessary" on page 62

1. Prerequisites

a. Plan your support.

   For details, see the section on planning UFT support for your toolkit in the *HP UFT 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.)

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

In the "Map to Controls Tab (Test Object Class Designer)" (described on page 95), 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 UFT to use different mapping rules when working with your controls in different browser versions, click the **Add Browser-Specific Rules** ✚ to add tabs for additional sets of rules. The "Add Browser Dialog Box" (described on page 107) 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, UFT ignores the set of **Ignore Control** rules when attempting to identify the control.

For more details on how UFT uses the different types of rules, see the section on teaching UFT to identify the test object class to use for a custom Web control in the *HP UFT 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.

For details, see "How to Create a Set of Mapping Rules Automatically " on page 65.

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 95.

> **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 UFT 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:

a. In the "Map to Controls Tab (Test Object Class Designer)" (described on page 95), 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.

b. 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:

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

b. In the "Map to Controls Tab (Test Object Class Designer)" (described on page 95), 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, similarly to what happens when you click **Select Controls**.
   - A control selecting 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.

c. Continue to select or remove controls to include in the set of rules, as described in "How to Create a Set of Mapping Rules Automatically " on the next page.

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.

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.

See also "How Extensibility Accelerator Tests Your Control Mapping" on page 54.

How to Create a Set of Mapping Rules Automatically

This task describes how to use Extensibility Accelerator to automatically create one set of rules that identifies the types of controls for which the test object class is used.

Perform this task separately in the "Map to Controls Tab (Test Object Class Designer)", described on page 95, in each relevant "Rule Creation Panel" and within each relevant browser-specific tab.

Note: This task is part of a higher-level task. For details, see "How to Map a Test Object Class to Application Controls" on page 59.

This task includes the following steps:

- "Prerequisites" below
- "Click the Select Controls button to start the control selecting session" on the next page
- "Move your mouse over your open applications" on the next page
- "Click on a control of a type that you want to support with this test object class" on page 67
- "Click the Select button" on page 68
- "Select additional controls that need to be supported by the same test object class" on page 69
- "Modify your selection of controls - Optional" on page 69
- "How to Create a Set of Mapping Rules Automatically " above
- "How to Create a Set of Mapping Rules Automatically " above

1. Prerequisites

Follow the first three steps in "How to Map a Test Object Class to Application Controls" on page 59.

Make sure that running scripts or ActiveX controls is enabled in the browser running your application.

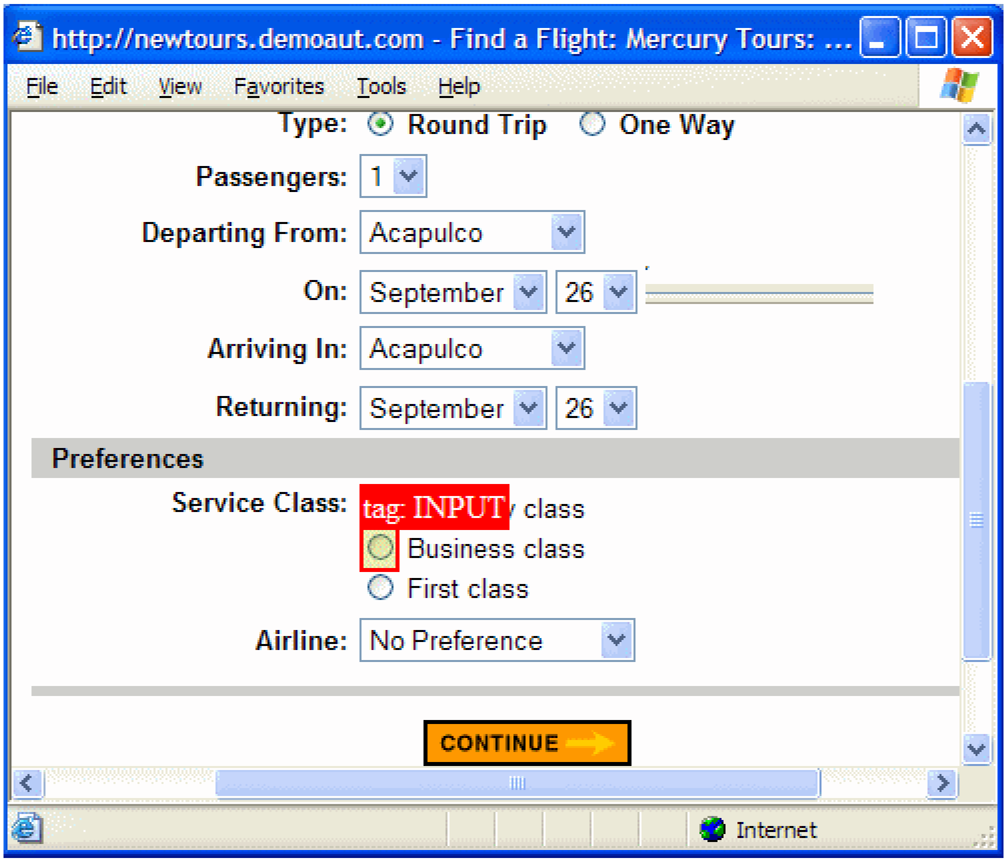
2. Click the Select Controls button to start the control selecting session

Extensibility Accelerator is hidden, and two buttons are displayed at the top of the screen: **Create Rules** and **Cancel**.

3. 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:


http://newtours.demoaut.com - Find a Flight: Mercury Tours: ...
File Edit View Favorites Tools Help
Preferences
CONTINUE
Internet

Type: ● Round Trip ○ One Way

Passengers: 1

Departing From: Acapulco

On: September 26

Arriving In: Acapulco

Returning: September 26

Service Class: tag: INPUT class
○ Business class
○ First class

Airline: No Preference

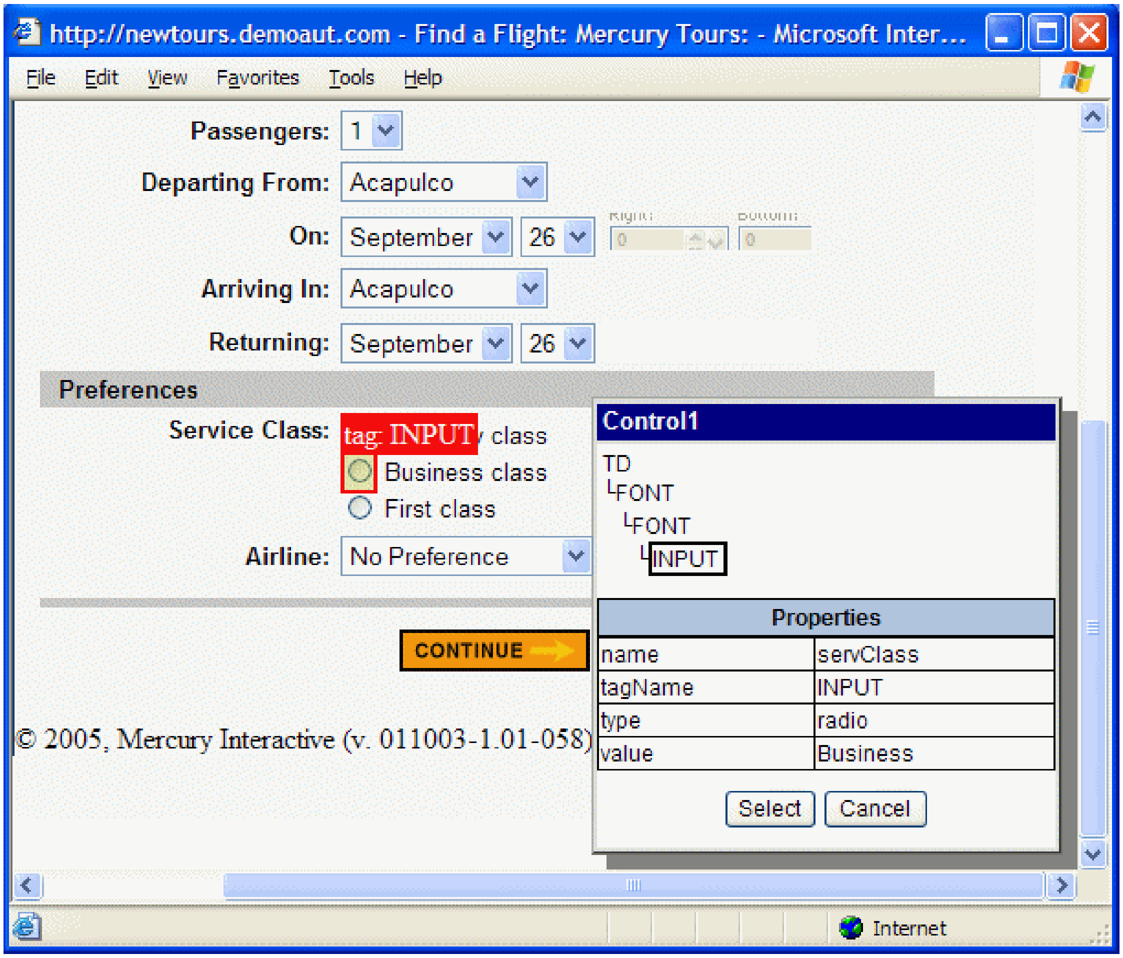
Tip: In many cases, you can hold the left CTRL key to change the pointing hand to a standard pointer and perform operations in your application, such as navigating to different Web pages, clicking links, selecting edit boxes to enter information, selecting from drop-down lists and so on. (Keep in mind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

4. Click on a control of a type that you want to support with this test object class

A"Selection Dialog Box" (described on page 105) opens on top of the browser, displaying the properties of the HTML element that represents the selected control:


http://newtours.demoaut.com - Find a Flight: Mercury Tours: - Microsoft Inter...
File
Edit
View
Favorites
Tools
Help
Right:
Bottom:
CONTINUE
Select
Cancel
Internet

Passengers: 1

Departing From: Acapulco

On: September 26

Arriving In: Acapulco

Returning: September 26

Preferences

Service Class: tag: INPUT class
○ Business class
○ First class

Airline: No Preference

© 2005, Mercury Interactive (v. 011003-1.01-058)

Control1

TD
└FONT
  └FONT
    └INPUT

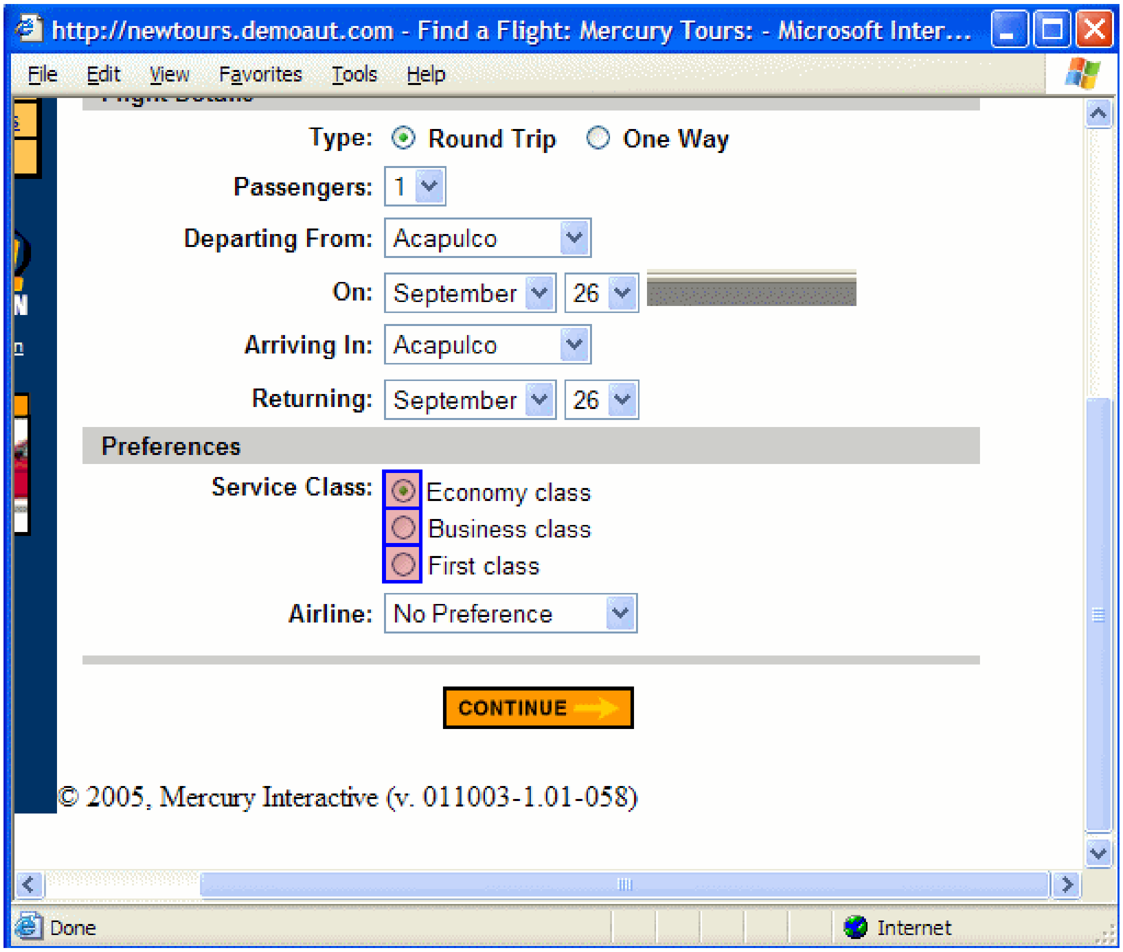
Properties	
name	servClass
tagName	INPUT
type	radio
value	Business

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

> **Note:** To view the properties of a different HTML element, or to select it to represent the control, select the element from the displayed hierarchy.

5. **Click the Select button**

The selected control is highlighted in the application. In the image below, the radio button controls are selected:



http://newtours.demoaut.com - Find a Flight: Mercury Tours: - Microsoft Inter...
File
Edit
View
Favorites
Tools
Help

Type: ● Round Trip ○ One Way

Passengers: 1

Departing From: Acapulco

On: September 26

Arriving In: Acapulco

Returning: September 26

Preferences

Service Class: ● Economy class ○ Business class ○ First class

Airline: No Preference

CONTINUE

© 2005, Mercury Interactive (v. 011003-1.01-058)

6. 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.

7. Modify your selection of controls - Optional

- To deselect a control, click the control and then click **Delete** in the "Selection

Dialog Box" on page 105.

- 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" on page 105, and click **Apply**.

8. Complete the process by clicking the Create Rules button, or click the Cancel button

The control selecting session ends.

9. Results

If you clicked **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.

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 57.

This task includes the following steps:

- "Define the list of operations supported by this test object class " below
- "Design the JavaScript functions that implement the test object operations" on the next page
- "Test and debug the operations" on page 73

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

   In the "Operations Tab (Test Object Class Designer)" (described on page 109), 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 UFT.

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 details, see the section on extending an existing test object class in the *HP UFT 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 31.

2. Design the JavaScript functions that implement the test object

operations

a. In the "Operations Tab (Test Object Class Designer)" (described on page 109), 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 17.

   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 UFT 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 76.

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 57.

This task includes the following steps:

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

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

In the "Properties Tab (Test Object Class Designer)" (described on page 121), add or remove properties or select base class properties to inherit and include in the list.

2. Specify the UFT 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 128).

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 121), click the **Implementation Code** button. The JavaScript file opens to the relevant JavaScript function in a "Extensibility Accelerator at a Glance", (described on page 15).

   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 UFT 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 UFT.

   - 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 79.

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 71.

This task includes the following steps:

- "Prerequisites" below
- "Set a breakpoint in your implementation function - Optional" on the next page
- "In the Debug Test Object Operation dialog box, select the test object class operation to run" on the next page
- "Select an application control on which to run the operation" on page 78
- "Run the operation" on page 78

1. Prerequisites

a. Enable script debugging in Microsoft Internet Explorer.

For example: In Internet Explorer 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 95) and verify that the control is highlighted in the application.

2. Set a breakpoint in your implementation function - Optional

If 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:

   - In the Class View, right-click the operation and select **Debug**. The "Debug Test Object Operation Dialog Box" (described on page 131) opens with the test object class and operation selected.

   - In the"Operations Tab (Test Object Class Designer)" (described on page 109), 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 Debug Test Object Operation 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.

You can now debug your functions using the Microsoft Visual Studio Shell debugging tools that are available in Extensibility Accelerator.

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 73.

This task includes the following steps:

- "Prerequisites" below
- "Set a breakpoint in your implementation function - Optional" on the next page
- "In the Debug Property Retrieval dialog box, select a property to retrieve" on the next page
- "Select the application control whose property value you want to retrieve" on the next page
- "Retrieve the property value" on page 81

1. Prerequisites

a. Enable script debugging in Microsoft Internet Explorer.

For example: In Internet Explorer 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 95) and verify that the control is highlighted in the application.

2. Set a breakpoint in your implementation function - Optional

If 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 121), 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 133) 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. Select only properties for which you designed support in your JavaScript function.

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.

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 UFT's property value retrieval behavior.)

You can now debug your functions using the Microsoft Visual Studio Shell debugging tools that are available in Extensibility Accelerator.

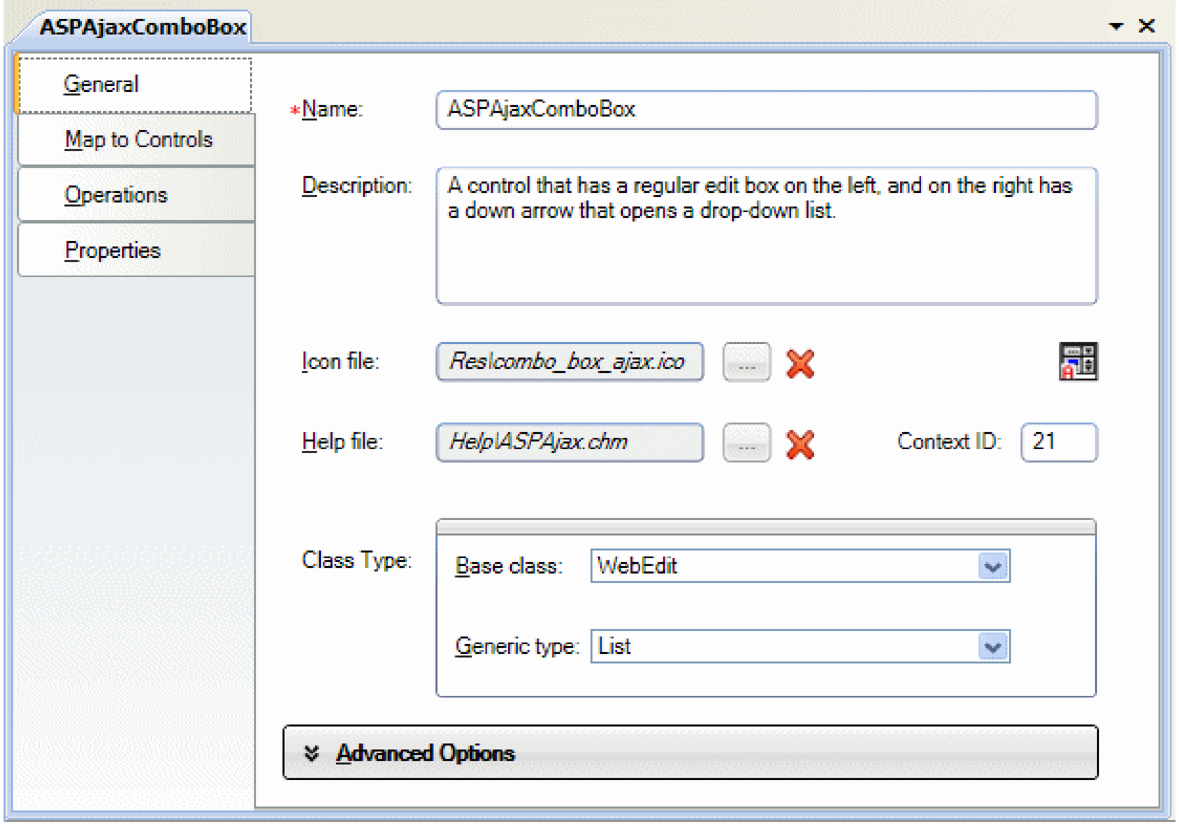
Reference

General Tab (Test Object Class Designer)

This tab enables you to define general details about the test object class that you want UFT 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.



*Name: ASPAjaxComboBox

Description: A control that has a regular edit box on the left, and on the right has a down arrow that opens a drop-down list.

Icon file: Res\combo_box_ajax.ico

Help file: Help\ASPAjax.chm

Context ID: 21

Class Type:

Base class: WebEdit

Generic type: List

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 57
See also	"Custom Toolkit Support Sets" on page 29

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

UI Elements	Description
Name	The name of the test object class that you want UFT to use to represent the custom control. This name is fundamental to the infrastructure of the support you are creating for the custom control. It determines: • 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. If you rename the test object class, all of the above are modified automatically. (It is therefore recommended to save such a change immediately.) **Note:** In this edit box, you can enter only English letters, numeric characters, hyphens, and underscores. The first character must be a letter.
Description	A description of the custom control you are supporting. This description is intended for your internal documentation purposes, it is not displayed in UFT. **Stored in: ClassInfo\Description** element in the test object configuration XML file

UI Elements	Description
Icon file	The name of the icon file that you want UFT to display for this test object class in tests, dialog boxes, and run session results. Use the **Import File** ... button to specify the relevant file. Use the **Clear** ✖ button to clear the edit box. **Default icon:** UFT's WebElement icon **Stored in: ClassInfo\IconInfo** element in the test object configuration XML file
...	**Import File.** Enables you to browse to and select the icon file. You can select an icon from an `.ico`, `.dll`, or `.exe` file. The file must be located in the project's `Res` folder to be properly deployed. Therefore, if you select a file from another location, a local copy of the file is immediately created in the `Res` folder. 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 file extension). **Note:** Avoid importing large `.exe` or `.dll` files, as these are added to your toolkit support set and deployed with it.
<icon>	An image of the icon you selected or the default icon.
Help file	The name of the `.chm` Help file that you want UFT to use for context-sensitive Help on this test object class. Use the **Import File** ... button to specify the relevant file. Use the **Clear** ✖ button to clear the edit box. **Stored in: ClassInfo\HelpInfo** element in the test object configuration XML file

UI Elements	Description
...	**Import File.** Enables you to browse to and select the `.chm` Help file. The file must be located in the project's `Help` folder to be properly deployed. Therefore, if you select a file from another location, a local copy of the file is immediately created in the `Help` folder. 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 file extension).
Context ID	The numeric value that indicates the help topic to open within the specified Help file. **Stored in: ClassInfo\HelpInfo** element in the test object configuration XML file

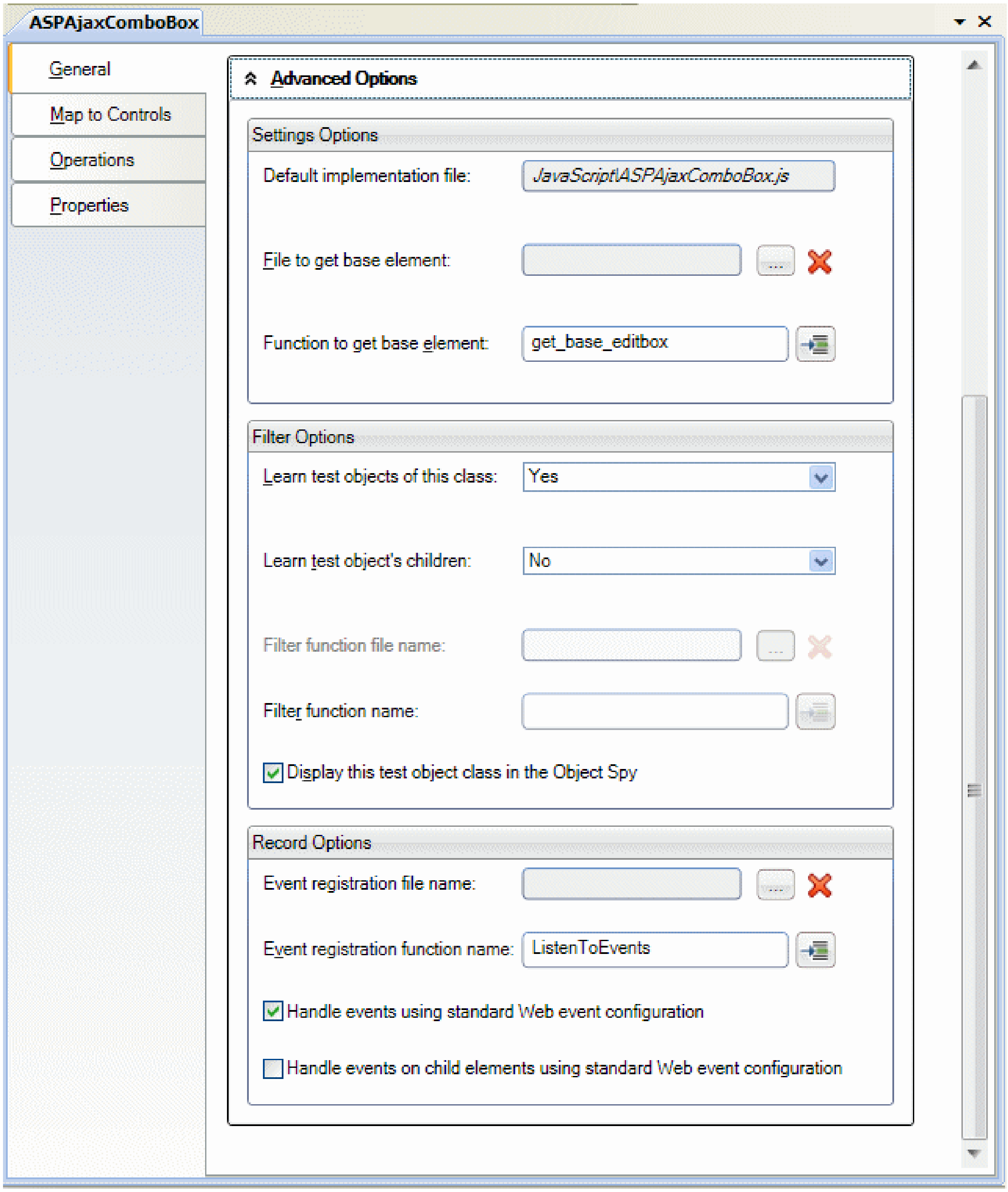
UI Elements	Description
Base class	The test object class this class extends. By default all Web test object classes extend WebElement. The base class that you select determines the default **Generic type**, the initial set of operations that your test object class includes, and a list of identification properties that you can choose to include in your test object class. 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. For more details, see "Base Class Selection" on page 52. **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. • In this edit box, you can enter only English letters, numeric characters, hyphens, and underscores. The first character must be a letter. **Stored in: BaseClassInfoName** attribute of the **ClassInfo** element in the test object configuration XML file

UI Elements	Description
Generic type	The type of control you are supporting. The generic type is used for object filtering in UFT and for creating documentation strings for the Documentation column of the Keyword View (unless you define them specifically in the test object operation definition). **Default:** The base test object class's generic type. (This value is selected automatically when you select a base class.) **Stored in: GenericTypeID** attribute of the **ClassInfo** element in the test object configuration XML file
Advanced Options	Expands to display the advanced options, described in "General Tab - Advanced Options" below. If you do not define these options, UFT uses their default values.

General Tab - Advanced Options

The **Advanced Options** area in the "General Tab (Test Object Class Designer)", (described on page 82), enables you to set advanced options for the test object class. If you do not define these options, UFT uses their default values.

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



Settings Options

Default implementation file: *JavaScript\ASPAjaxComboBox.js*

File to get base element:

Function to get base element: get_base_editbox

Filter Options

Learn test objects of this class: Yes

Learn test object's children: No

Filter function file name:

Filter function name:

☑ Display this test object class in the Object Spy

Record Options

Event registration file name:

Event registration function name: ListenToEvents

☑ Handle events using standard Web event configuration

☐ Handle events on child elements using standard Web event configuration

User interface elements are described below:

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

UI Elements	Description
...	**Import File.** Enables you to browse to and select a JavaScript file. The file must be located in the project's `JavaScript` folder to be properly deployed. Therefore, if you select a file from another location, a local copy of the file is immediately created in the `JavaScript` folder. 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).
Function to get base element	The function that you implement to return the base element. Access to the base element enables UFT to use the base class's implementation for inherited test object operations and properties. 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. 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. **Note:** In this edit box, you can enter only English letters, numeric characters, hyphens, and underscores. The first character must be a letter. **Stored in:** A **Control\Settings\Variable** element named **func_to_get_base_elem** in the toolkit configuration XML file
Filter Options	

UI Elements	Description
Learn test objects of this class	Indicates whether UFT should learn this control. Possible values: • **Yes** • **No** • **Only if has children** - learn the control only if it has children. (Stored as **IfChildren**) **Default:** Yes **Stored in: learn_control** attribute of the **Learn** element in the toolkit configuration XML file
Learn test object's children	Indicates whether UFT should learn the children of this control. Possible values: • **Yes** • **No** • **Use Filter Function** - the function specified below performs the filtering. (Stored as **CallFilterFunc**) **Default:** Yes **Stored in: learn_children** attribute of the **Learn** element in the toolkit configuration XML file

UI Elements	Description
Filter function file name	The file that contains the filter function (optional). You cannot modify this value directly. Use the **Import File** ... button to browse to and select the relevant file. (For details on using this button, see above.) Use the **Clear** ✖ button to clear the edit box. The corresponding XML attribute in the toolkit configuration XML file is cleared, but the JavaScript file is not removed from the project. **Default:** The **Default implementation file** **Stored in: file_name** attribute of the **Learn** element in the toolkit configuration XML file
Filter function name	The function that performs the filtering. You must specify and implement this function if you selected the **Use Filter Function** value for the **Learn test object's children** option. 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. **Note:** In this edit box, you can enter only English letters, numeric characters, hyphens, and underscores. The first character must be a letter. **Stored in: function** attribute of the **Learn** element in the toolkit configuration XML file
Display this test object class in the Object Spy	Indicates whether the Object Spy displays this test object class. **Default:** Yes

UI Elements	Description
Record Options	
Event registration file name	The file that contains the event registration function (optional). You cannot modify this value directly. Use the **Import File** button to browse to and select the relevant file. (For details on using this button, see above.) Use the **Clear** button to clear the edit box. The corresponding XML attribute in the toolkit configuration XML file is cleared, but the JavaScript file is not removed from the project. **Default:** The **Default implementation file** **Stored in: file_name** attribute of the **Record\EventListening** element in the toolkit configuration XML file
Event registration function name	The function that implements registering to listen for events on the elements contained in the control. You must specify and implement this function if want to customize recording on your control. 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. **Note:** In this edit box, you can enter only English letters, numeric characters, hyphens, and underscores. The first character must be a letter. **Stored in: function** attribute of the **Record\EventListening** element in the toolkit configuration XML file

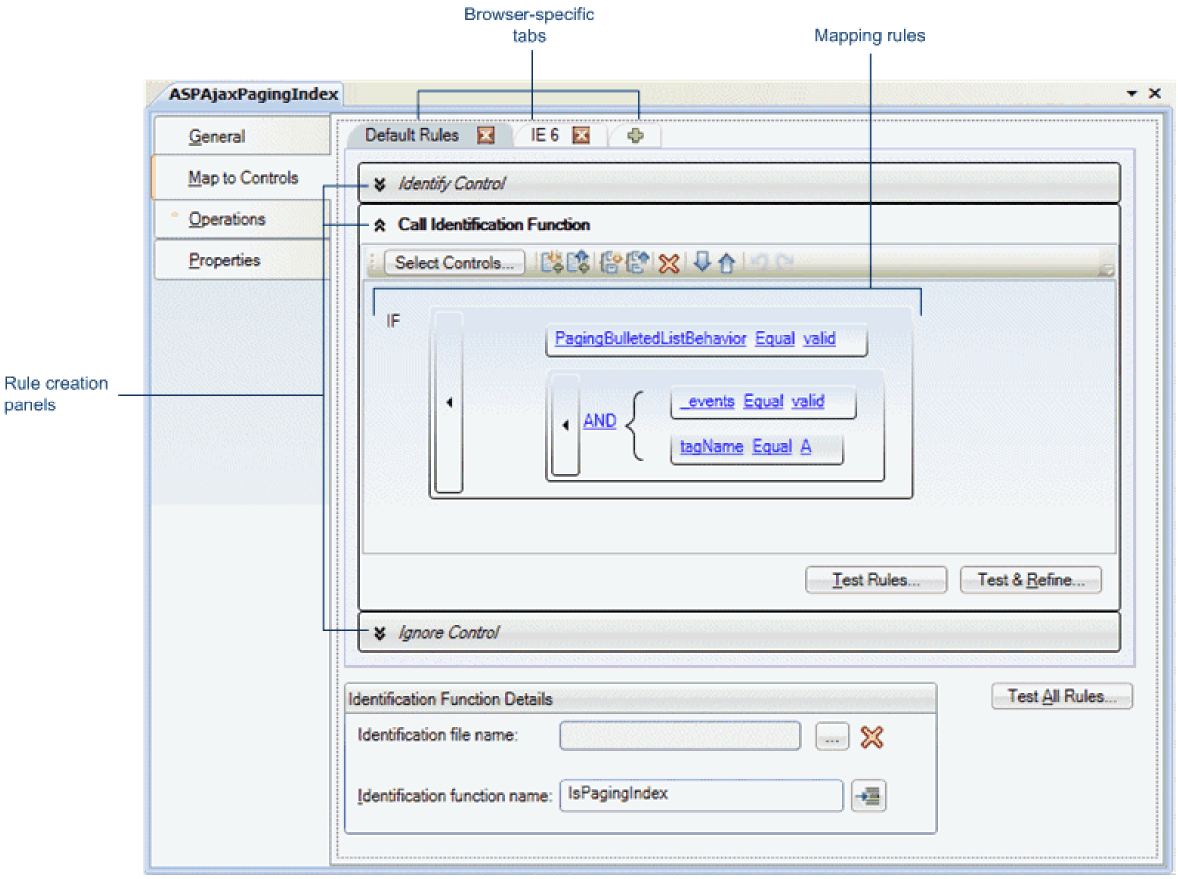
UI Elements	Description
Handle events using standard Web event configuration	Specifies whether to use standard Web event configuration during a recording session to handle events on controls represented by this test object class. **Stored in: use_default_event_handling** attribute of the **Record\EventListening** element in the toolkit configuration XML file
Handle events on child elements using standard Web event configuration	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. **Stored in: use_default_event_handling_for_children** attribute of the **Record\EventListening** element in the toolkit configuration XML file

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 UFT uses in different ways.

Each "Rule Creation Panel" (described on page 100) contains options you can use to create, edit, and test the set of rules in that panel.



PagingBulletedListBehavior Equal valid

AND
_events Equal valid
tagName Equal A

Identification file name:

Identification function name: IsPagingIndex

To access	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 **Map to Controls** tab.

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. • 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 59
See also	• The section on teaching UFT to identify the test object class to use for a custom Web control in the *HP UFT Web Add-in Extensibility Developer Guide*. • "How Extensibility Accelerator Tests Your Control Mapping" on page 54 • "Rule Creation Panel" on page 100 • "Selection Dialog Box" on page 105 • "Add Browser Dialog Box" on page 107

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

UI Elements	Description
<browser-specific tabs>	A strip of tabs, each containing the mapping rules for UFT to use when running on a specific browser type and version. To add a new tab, click the **Add Browser-Specific Rules** ✚ tab. To remove a tab, click the **Delete** ☒ button on the tab. The rules in the **Default Rules** tab are used for all supported browsers that do not have a specific set of rules defined. 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.
<rule creation panels>	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 100. **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: • **Identify Control** panel **->** **IdentifyIfPropMatch** type • **Call Identification Function** panel **->** **CallIdFuncIfPropMatch** type • **Ignore Control** panel **->** **SkipIfPropMatch** type For details on how UFT uses the different types of rules, see the section on teaching UFT to identify the test object class to use for a custom Web control in the *HP UFT Web Add-in Extensibility Developer Guide*.

UI Elements	Description
Identification file name	The file that contains the identification function (optional). You cannot modify this value directly. Use the **Import File** ... button to browse to and select the relevant file. Use the **Clear** ✖ button to clear the edit box. The corresponding XML attribute in the toolkit configuration XML file is cleared, but the JavaScript file is not removed from the project. **Default:** The **Default implementation file** defined in the General tab **Stored in: Identification** element in the toolkit configuration XML file
...	**Import File.** Enables you to browse to and select a JavaScript file. The file must be located in the project's `JavaScript` folder to be properly deployed. Therefore, if you select a file from another location, a local copy of the file is immediately created in the `JavaScript` folder. 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).

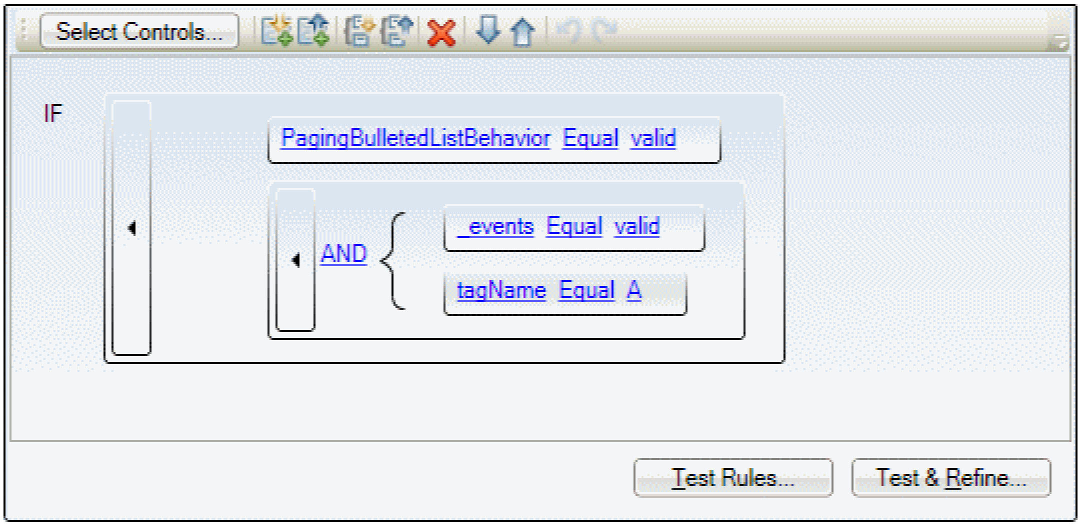
UI Elements	Description
Identification function name	The function that you implement to help identify the controls for which to use this test object class. This function is necessary only if you cannot create a set of rules that identifies the controls specifically enough. Use the **Implementation Code** 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. **Note:** In this edit box, you can enter only English letters, numeric characters, hyphens, and underscores. The first character must be a letter. **Stored in: Identification** element in the toolkit configuration XML file
Test All Rules	Highlights all of the controls that match the mapping rules in all open Web applications. The rules from each tab are applied to the corresponding open browsers, using the same logic that UFT uses to identify the test object class to use for a custom Web control. 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 details, see the step ""Test your mapping rules on an application and update them if necessary"" in "How to Map a Test Object Class to Application Controls" on page 59.

Rule Creation Panel

In the "Map to Controls Tab (Test Object Class Designer)", (described on page 95), each browser-specific tab contains three rule creation panels, in which you can create a set of rules that UFT 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 59.
- To edit rules manually, use the toolbar and UI elements within the rule editor area.



To access	1. In the "Map to Controls Tab (Test Object Class Designer)" (described on page 95), create a tab for a browser-specific rule set, or select an existing one. 2. Expand the rule creation panel for the type of rules you want to create.
Relevant tasks	• "How to Map a Test Object Class to Application Controls" on page 59 • "How to Create a Set of Mapping Rules Automatically " on page 65

Each rule creation panel contains:

- "Buttons" on the next page
- "Rule Editor Area" on the next page

Buttons

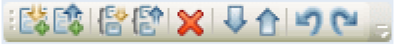
UI Elements	Description
Select Controls	Starts a session for automatically creating mapping rules. You create the rules by pointing to controls of the relevant type in your application. For task details, see "How to Create a Set of Mapping Rules Automatically " on page 65.
Test Rules	Highlights all of the controls that match the mapping rules in all open Web applications. For task details, see the step ""Test your mapping rules on an application and update them if necessary"" in "How to Map a Test Object Class to Application Controls" on page 59.
Test & Refine	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. For task details, see the step ""Test your mapping rules on an application and update them if necessary"" in "How to Map a Test Object Class to Application Controls" on page 59.

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>	 This toolbar contains the following buttons: • 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.

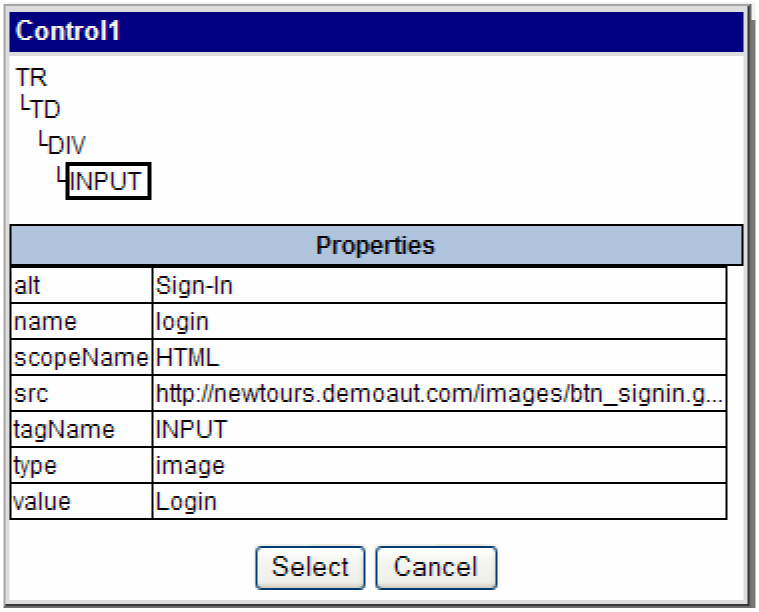
UI Elements	Description
<rules>	A single rule is made up of the following elements: • **<property name>.** The name of the HTML property checked in this rule. Click to select from a list of common property names or edit this value. • **Equal / Not Equal.** Indicates whether the value of the property 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 A.* 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: ■ **RegExp.** Indicates whether the expected value should be treated as a regular expression. **Default:** false ■ **Trim.** Indicates whether UFT should remove leading and trailing spaces from the property value and the expected value before evaluating the rule. **Default:** true Click to switch between true and false values for these rule attributes. (A toggle button that is on indicates the value `true`.)
AND/OR	Indicates whether to use And or Or logic for the set of rules in the group. Click to switch between **AND** and **OR**.

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.

The following image shows an example of the Selection dialog box that opens when clicking a non-selected control for the **Control1** test object class. The buttons on the dialog box differ slightly when clicking a previously selected control.



Control1

```
TR
└TD
  └DIV
    └INPUT
```

Properties	
alt	Sign-In
name	login
scopeName	HTML
src	http://newtours.demoaut.com/images/btn_signin.g...
tagName	INPUT
type	image
value	Login

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 59

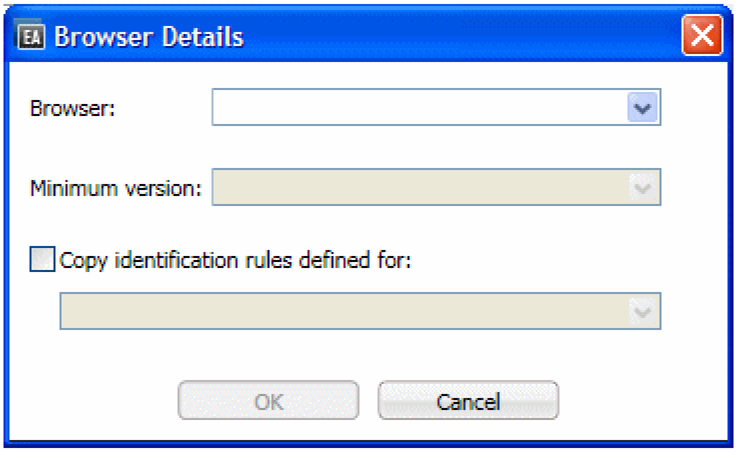
See also	"Map to Controls Tab (Test Object Class Designer)" on page 95

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

UI Elements	Description
<HTML element tree>	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. You can select a different HTML element in the hierarchy to represent the application control.
Properties	The selected HTML element's property names and values.
Select	Selects this control and include it in the set of controls that determines the rules that are created in this session. The dialog box closes and the control is highlighted in the application. **Available when:** The control is not currently selected.
Apply	Updates the set of selected controls, to use the HTML element selected in the hierarchy to represent this control. The dialog box closes and the control is highlighted in the application. **Available when:** The control is currently selected.
Delete	Removes this control from the set of controls that determines the rules that are created in this session. The dialog box closes and the control is not highlighted in the application. **Available when:** The control is currently selected.
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 59
See also	"Map to Controls Tab (Test Object Class Designer)" on page 95

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

UI Elements	Description
Browser	The type of browser to which the rules in the new tab will apply. Possible values: • **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 the Default Rules pane was deleted) • **Internet Explorer** **Stored in: name** attribute of the **Identification\Browser** element in the toolkit configuration XML file
Minimum version	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. 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 7 and another for Internet Explorer 9, the former is used when running on Internet Explorer 8, and the latter is used when running on Internet Explorer 11. **Stored in: min_version** attribute of the **Identification\Browser** element in the toolkit configuration XML file
Copy identification rules defined for	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.

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.

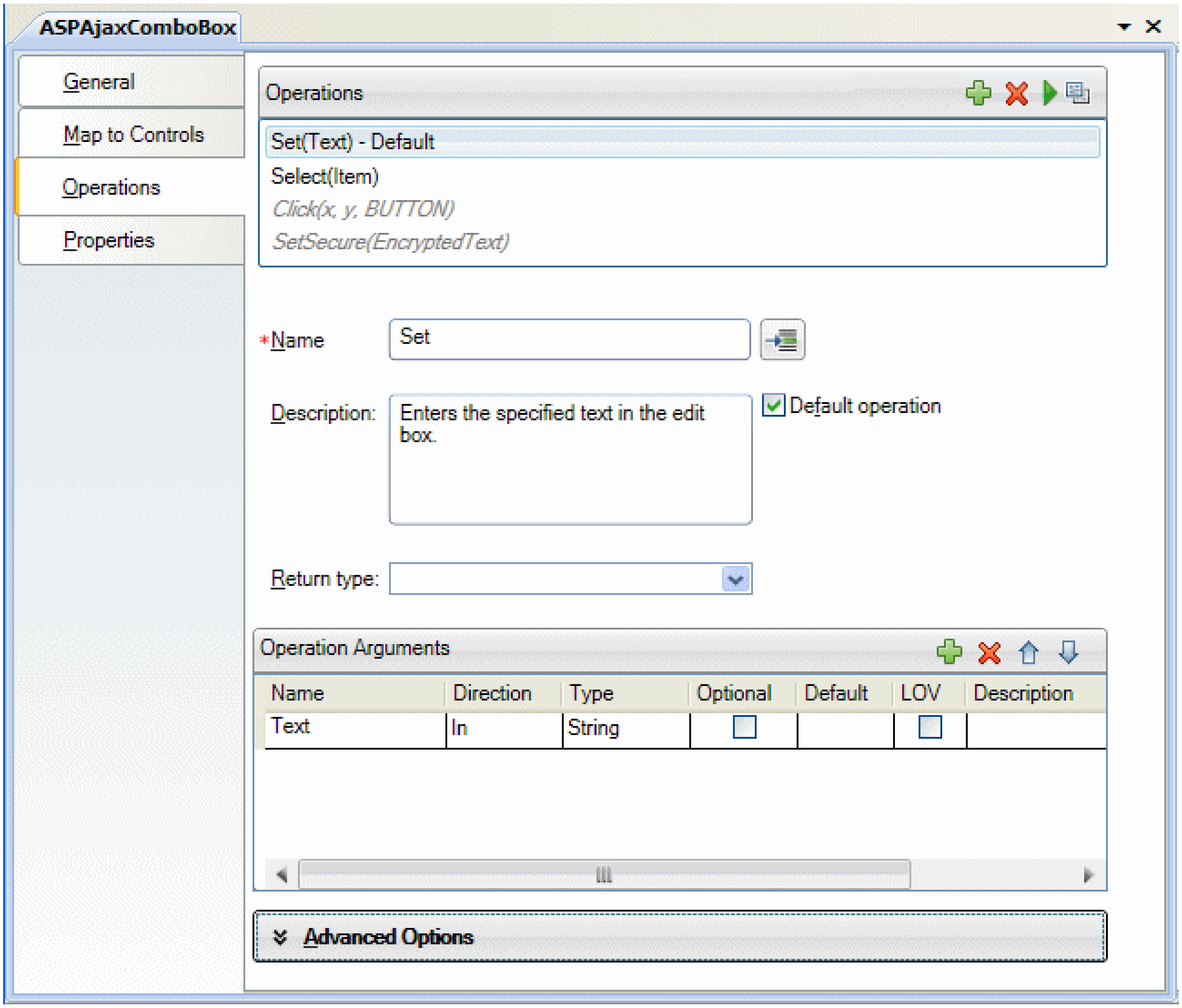
- Specify the default operation for this test object class (optional).

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.



Set(Text) - Default
Select(Item)
Click(x, y, BUTTON)
SetSecure(EncryptedText)

*Name: Set

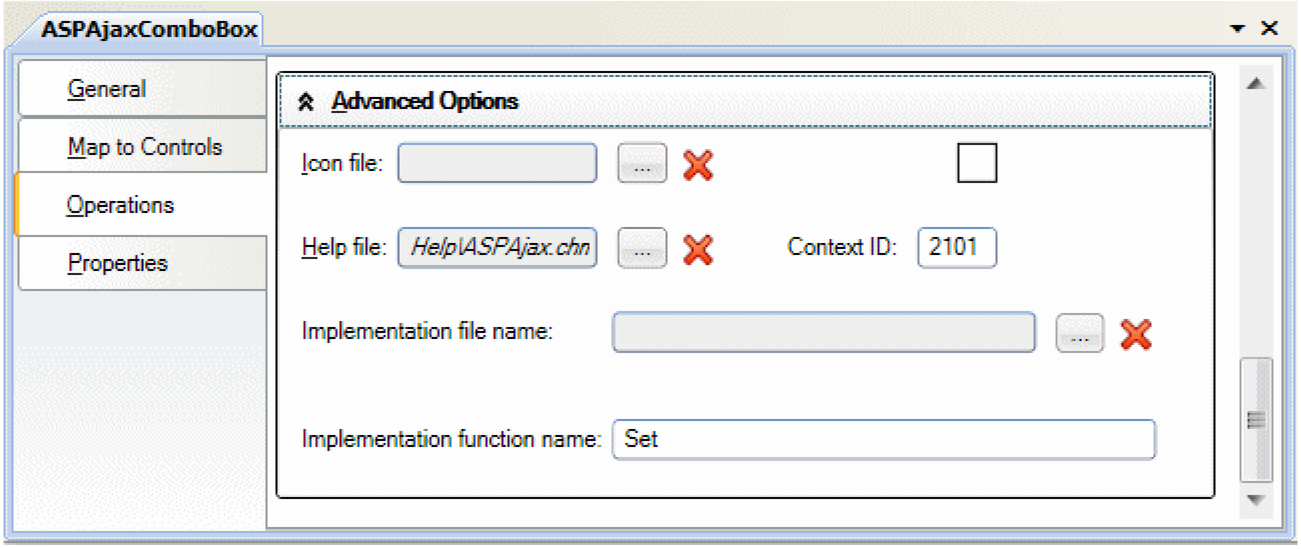
Description: Enters the specified text in the edit box.

☑ Default operation

Return type:

Name	Direction	Type	Optional	Default	LOV	Description
Text	In	String	□		□	

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



Icon file:

Help file: *Help\ASPAjax.chm*

Context ID: 2101

Implementation file name:

Implementation function name: Set

To access	To access the Operations tab: 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. To access the advanced options, click **Advanced Options**.
Important information	- 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. 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 Operations " on page 71
See also	The section on implementing support for test object methods in the *HP UFT Web Add-in Extensibility Developer Guide*.

The Operations tab contains the following areas:

- "Operation List Area" below
- "Operation Details Area" on the next page
- "Operation Arguments Area" on page 115
- "Advanced Options Area" on page 117

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
✚	**Add.** Adds a new operation to the test object class definition. In addition, a JavaScript function stub is added to the default implementation file. For details on when this takes place, see "When Are Your Changes Applied and Saved" on page 31. **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.
✖	**Delete.** Deletes the selected operation from the test object definition in the test object configuration XML file. The corresponding JavaScript functions are not deleted. If you delete an overriding operation, its signature appears in italic font again.
▶	**Debug Operation.** Opens the "Debug Test Object Operation Dialog Box" (described on page 131), enabling you to run and debug the JavaScript code that you designed to implement the test object operation.
⧉	**Override Operation.** Creates a new operation that will override the one inherited from the base class. The operation signature font changes to regular text and the operation details in this tab become editable. 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).

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	The test object operation's name. 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.) If you rename an overridden operation, its signature appears in italic font again, and a new operation is created with the new name. **Note:** In this edit box, you can enter only English letters, numeric characters, hyphens, and underscores. The first character must be a letter. **Stored in:** • **Operation** element in the test object configuration XML file • **Method** element in the toolkit configuration XML file - mapped to the JavaScript implementation function
Description	A description of the operation. This description is displayed in UFT tooltips. **Stored in: Operation\Description** element in the test object configuration XML file

UI Elements	Description
Return type	The type of value that the operation returns. 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: Operation\ReturnValueType\Type** element in the test object configuration XML file
⇥	**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.

UI Elements	Description
Name	The argument name. **Note:** In this edit box, you can enter only English letters, numeric characters, hyphens, and underscores. The first character must be a letter. **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
Optional	Specifies whether the argument is optional. If you define optional arguments, the must come after any mandatory arguments that the operation has. **Stored in: IsMandatory** attribute of the **Argument** element
Default	The default value for the argument. Only relevant if the argument is optional. **Stored in: DefaultValue** attribute of the **Argument** element

UI Elements	Description
LOV	Indicates whether UFT dynamically displays a list of possible values for this argument when editing tests. 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, UFT calls this function from the default implementation file defined for this test object class (General tab, advanced options). **Stored in: DynamicListOfValues** attribute of the **Argument** element
Description	A description of the argument. This description is intended for your internal documentation purposes, it is not displayed in UFT. **Stored in: Description** attribute of the **Argument** element

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, UFT uses their default values.

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

User interface elements are described below:

UI Elements	Description
Icon file	The name of the icon file that you want UFT to display for this operation in the run session results. Use the **Import File** ... button to specify the relevant file. Use the **Clear** ✖ button to clear the edit box. **Default:** A UFT default icon **Stored in: Operation\IconInfo** element in the test object configuration XML file 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.
...	**Import File.** Enables you to browse to and select the icon file. You can select an icon from an `.ico`, `.dll`, or `.exe` file. The file must be located in the project's `Res` folder to be properly deployed. Therefore, if you select a file from another location, a local copy of the file is immediately created in the `Res` folder. 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 file extension). **Note:** Avoid importing large `.exe` or `.dll` files, as these are added to your toolkit support set and deployed with it.
<icon>	An image of the icon you selected or the default icon.

UI Elements	Description
Help file	The name of the `.chm` Help file that you want UFT to use for context-sensitive Help on this test object class. Use the **Import File** ... button to specify the relevant file. Use the **Clear** ✖ button to clear the edit box. **Stored in: Operation\HelpInfo** element in the test object configuration XML file.
...	**Import File.** Enables you to browse to and select the `.chm` Help file. The file must be located in the project's `Help` folder to be properly deployed. Therefore, if you select a file from another location, a local copy of the file is immediately created in the `Help` folder. 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 file extension).
Context ID	The numeric value that indicates the help topic to open within the specified Help file. **Stored in: Operation\HelpInfo** element in the test object configuration XML file

UI Elements	Description
Implementation file name	The file that contains the implementation function (optional). You cannot modify this value directly. Use the **Import File** ... button to browse to and select the relevant file. Use the **Clear** ✖ button to clear the edit box. The corresponding XML attribute in the toolkit configuration XML file is cleared, but the JavaScript file is not removed from the project. **Default:** The **Default implementation file** defined in the General tab **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. **Stored in: Method** element in the toolkit configuration XML file
...	**Import File.** Enables you to browse to and select a JavaScript file. The file must be located in the project's `JavaScript` folder to be properly deployed. Therefore, if you select a file from another location, a local copy of the file is immediately created in the `JavaScript` folder. 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).

UI Elements	Description
Implementation function name	The name of the function that you implement to perform the test object operation on the control. **Default:** The operation name **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. • If you modify the name in this option, you must update the function name in the JavaScript file as well. • In this edit box, you can enter only English letters, numeric characters, hyphens, and underscores. The first character must be a letter. **Stored in: Method** element in the toolkit configuration XML file

Properties Tab (Test Object Class Designer)

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

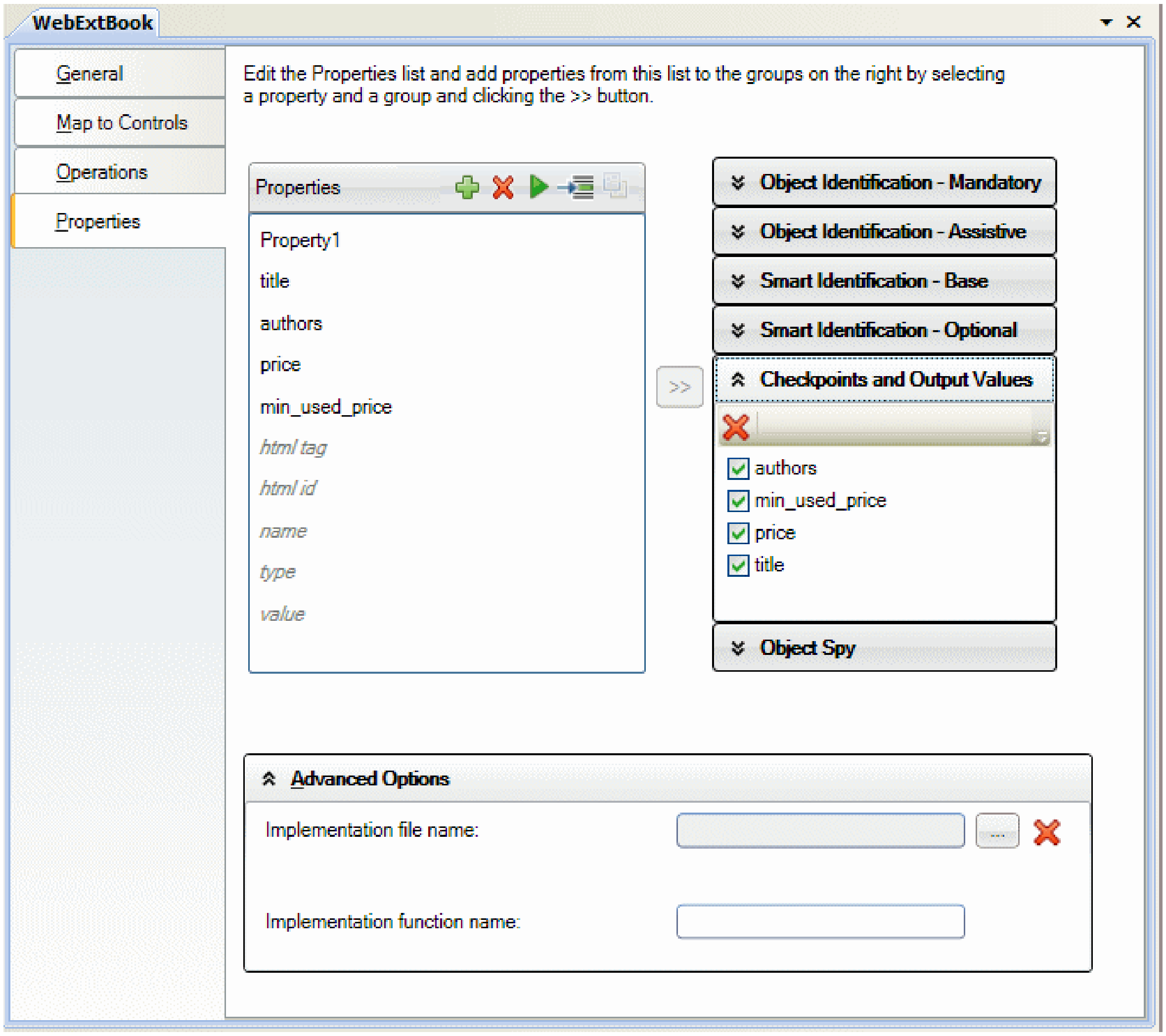
- Define the list of identification properties for your test object class.
- Add or remove properties or select base class properties to inherit and include in the list.
- Decide which properties are included in test object descriptions, which can be verified in checkpoints, and used in output values, 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 74.

Note: You must also implement a JavaScript function that retrieves the values of the properties from the control. For details, see the step ""Implement the JavaScript function that retrieves the identification property values from the run-time object"" in "How to Design Test Object Class Identification Properties" on page 73.



Edit the Properties list and add properties from this list to the groups on the right by selecting a property and a group and clicking the >> button.

WebExtBook
General
Map to Controls
Operations
Properties
Properties
Property1
title
authors
price
min_used_price
html tag
html id
name
type
value
>>
Object Identification - Mandatory
Object Identification - Assistive
Smart Identification - Base
Smart Identification - Optional
Checkpoints and Output Values
☑ authors
☑ min_used_price
☑ price
☑ title
Object Spy
Advanced Options
Implementation file name:
...
Implementation function name:

To access	To access the Properties tab: 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. To access the advanced options, click **Advanced Options**.

Important information	- To prevent the property grouping that you define in this tab from overwriting changes that the UFT 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 *HP UFT Web Add-in Extensibility Developer Guide*. - 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 73
See also	The section on implementing support for identification properties in the *HP UFT Web Add-in Extensibility Developer Guide*.

The Properties tab contains the following:

- "Property List" below
- "Property Usage Groups" on page 126
- "Advanced Options" on page 128

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
➕	**Add.** Adds a new editable identification property to the test object class definition. 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. If you rename the property, the property name used in the code segment is updated. For details on when this takes place, see "When Are Your Changes Applied and Saved" on page 31. **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. • In the property name, you can enter only English letters, numeric characters, hyphens, underscores, and spaces. The first character must be a letter.
❌	**Delete.** Deletes the selected property from the test object definition in the test object configuration XML file. • 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.
▶	**Debug Property Retrieval.** Opens the " Debug Property Retrieval Dialog Box" (described on page 133), enabling you to run and debug the JavaScript code that you implement to retrieve the property value from the control.

Buttons	Description
⇥	**Implementation Code.** Opens the relevant JavaScript file to the JavaScript function that retrieves the property values from the control. If you select a property before clicking this button, the file opens to the relevant section within the function.
⧉	**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. • 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. 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. Implement this segment to retrieve the value for the property, if the implementation cannot be inherited from the base class. For details about inheriting base class implementation, see the section about extending an existing test object class in the *HP UFT Web Add-in Extensibility Developer Guide*.

Property Usage Groups

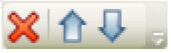
Add identification properties to the different groups in this area, to inform UFT of the purposes for which the properties should be used.

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
	Each group has a toolbar. • 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.
Object Identification - Mandatory Group	Properties that UFT always learns as part of the description for test objects of this class. **Note:** You cannot include the same property in both **Object Identification** lists. **Indicated by: ForDescription** attribute set to `true`
Object Identification - Assistive Group	Additional properties that UFT can learn for a test object of the selected class to create a unique test object description. When UFT learns an object, and assistive properties are necessary to create a unique object description, UFT 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. **Indicated by: ForAssistive** attribute set to `true`, **AssistivePropertyValue** attribute set to the position of the property within the group

UI Elements	Description
Smart Identification - Base Group	Properties that UFT 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. **Note:** You cannot include the same property in both **Smart Identification** lists. **Indicated by: ForBaseSmartID** attribute set to `true`
Smart Identification - Optional Group	Properties that UFT 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. When UFT 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. **Indicated by: ForOptionalSmartID** attribute set to `true`, **OptionalSmartIDPropertyValue** attribute set to the position of the property within the group
Checkpoints and Output Values Group	Properties available in the Checkpoint Properties and Output Value Properties dialog boxes in UFT. If the check box for a property if selected, it is selected by default in the Checkpoint Properties dialog box when creating a checkpoint. **Indicated by: ForVerification** and, if selected, **ForDefaultVerification** attribute set to `true`
Object Spy Group	**Indicated by: ForSpy** attribute set to `true`

Advanced Options

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

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

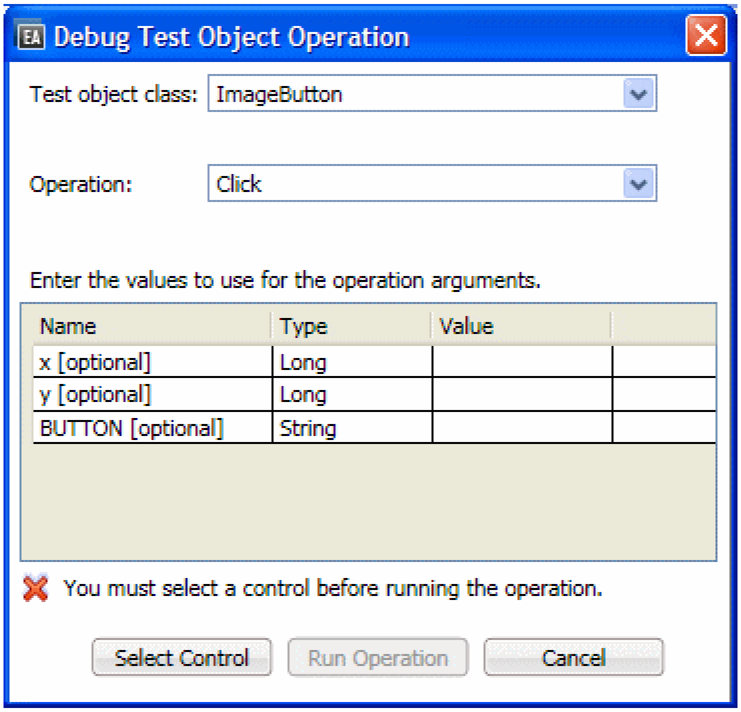
User interface elements are described below:

UI Elements	Description
Implementation file name	The file that contains the implementation function (optional). You cannot modify this value directly. Use the **Import File** … button to browse to and select the relevant file. Use the **Clear** ✖ button to clear the edit box. The corresponding XML attribute in the toolkit configuration XML file is cleared, but the JavaScript file is not removed from the project. **Default:** The **Default implementation file** defined in the General tab **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. **Stored in: Property** element in the toolkit configuration XML file

UI Elements	Description
...	**Import File.** Enables you to browse to and select a JavaScript file. The file must be located in the project's `JavaScript` folder to be properly deployed. Therefore, if you select a file from another location, a local copy of the file is immediately created in the `JavaScript` folder. 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).
Implementation function name	The name of the function that you implement to retrieve the values of the identification properties from the control. **Default: get_property_value** **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. • If you modify the name in this option, you must update the function name in the JavaScript file as well. • In this edit box, you can enter only English letters, numeric characters, hyphens, and underscores. The first character must be a letter. **Stored in: Property** element in the toolkit configuration XML file

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.



Debug Test Object Operation

Test object class: ImageButton

Operation: Click

Enter the values to use for the operation arguments.

Name	Type	Value
x [optional]	Long	
y [optional]	Long	
BUTTON [optional]	String	

You must select a control before running the operation.

To access	Do one of the following: - Select **Project > Debug Test Object Operation** - In the "Operations Tab (Test Object Class Designer)", described on page 109, click the **Debug Operation** ▶ button in the operation list toolbar. - In the Class View, right-click the operation that you want to run and select **Debug**. **Tip:** If the operations are not displayed in the Class View, first select the test object class.

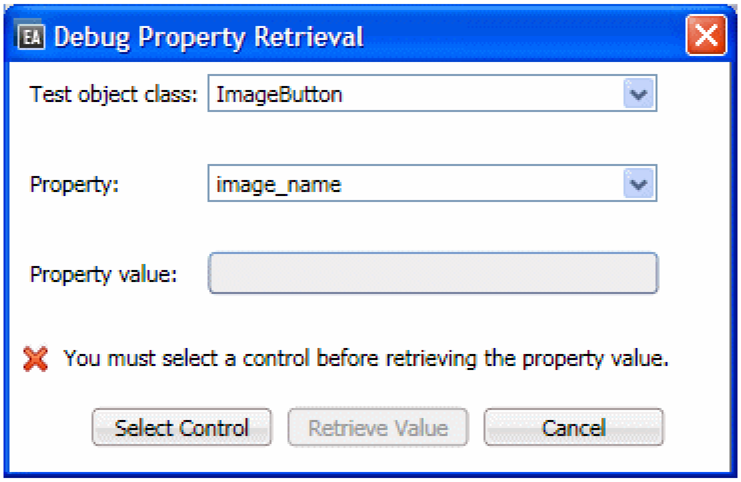
Important information	Make sure that you select the relevant control in the application before you run the operation.
Relevant tasks	"How to Test and Debug Your Test Object Operation Support" on page 76
See also	"JavaScript Function Debugging" on page 55

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>	The arguments for the operation. Includes the following for each argument: **Name.** The name of the argument and whether it is optional. (Read only) **Type.** The type of value the operation expects for this argument. (Read only) **Value.** The value to use for the argument when running the operation. You must enter values for all mandatory arguments.
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 on which you want to run the operation. **Available when:** A **Test object class** is selected in the dialog box.
Run Operation	Runs the operation on the selected control, calling the JavaScript implementation that you designed. **Available when:** A control is selected in the application.
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.



Test object class: ImageButton

Property: image_name

Property value:

✗ You must select a control before retrieving the property value.

To access	Do one of the following: - Select **Project > Debug Property Retrieval** - In the "Properties Tab (Test Object Class Designer)", described on page 121, 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 79.
See also	"JavaScript Function Debugging" on page 55

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 UFT'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**

- To enable Extensibility Accelerator to run on a custom control, running scripts must be enabled in the browser. If the browser automatically blocks running scripts when you run your application, make sure to allow blocked content before using Extensibility Accelerator its the controls.

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.

Chapter 5: Custom Toolkit Support Deployment

This chapter includes:

Concepts

Deployment Objectives

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

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 UFT, and the operations are displayed correctly in the Keyword View and when using the statement completion feature in the Editor.
You define the mapping rules for a test object class, but do not define any operations.	UFT can learn objects of this type.
You implement the JavaScript function for an operation.	UFT 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 the next page.

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, UFT 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 UFT opens, it will refresh the property lists based on the definitions in the test object configuration XML file. If UFT users change the property lists using the Object Identification dialog box, their changes will be lost when they reopen UFT.

For more details, see the section on modifying deployed support in the *HP UFT 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 UFT or the Unified Functional Testing Add-in for ALM, 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 UFT 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 UFT computer, unzip it in `<UFT installation folder>\dat\Extensibility\Web`.

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

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 UFT or the Unified Functional Testing Add-in for ALM 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 138.

3. Depending on your deployment destination, do one of the following:

   - To deploy to UFT (if it is installed on this computer), select **Project > Deploy > Deploy to UFT**.

   - To deploy to the Unified Functional Testing Add-in for ALM (if it is installed on this computer), select **Project > Deploy > Deploy to UFT Add-in For ALM**.

   - 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 140.

If you deploy to UFT or the Unified Functional Testing Add-in for ALM, Extensibility Accelerator locates the UFT or Add-in installation folder and deploys the files to the `<Extensibility Accelerator for HP Functional Testing` 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 UFT or the Unified Functional Testing Add-in for ALM installed, unzip it in the `<Extensibility Accelerator for HP Functional Testing` or `Add-in installation folder>\dat\Extensibility\Web>` folder.

> **Note:** After deploying Web support created by Web Add-in Extensibility, when you open Internet Explorer 9 or later, Internet Explorer prompts you to enable the WebExHookIE Class add-on. You must enable this add-on in order for Extensibility Accelerator features to work properly.
>
> After you enable the WebExHookIE Class add-on, Internet Explorer may prompt you to disable the add-on if it is affecting the browser performance. You can disable it

temporarily, but some Extensibility Accelerator features will not function until you enable it again.

Chapter 6: Tutorial: Create Support For a Custom Web Control Using Extensibility Accelerator

In this lesson you use Extensibility Accelerator to create support for the Book control in the Web Add-in Extensibility Book Sample toolkit, which is installed with Extensibility Accelerator for HP Functional Testing. Creating support for the Book control requires only minimal customization, allowing you to learn the basics of working with Extensibility Accelerator and developing a Web Add-in Extensibility toolkit support set.

Perform this tutorial on a computer with UFT and Extensibility Accelerator for HP Functional Testing installed. This simplifies the process of deploying the toolkit support set to UFT. When you develop your own support, you can develop it in on a computer without UFT and then deploy the support to other computers.

The `%ALLUSERSPROFILE%\Documents\ExtAccTool\Samples\WebExtSample` folder contains a complete toolkit support set for this sample to which you can refer while you perform this lesson (although the sample is not identical to the support you create).

The *HP UFT Web Add-in Extensibility Developer Guide* includes a similar tutorial, where you design support for this control manually, without the help of Extensibility Accelerator.

This lesson includes:

Web Add-in Extensibility Overview

The UFT Web Add-in provides built-in support for a number of commonly used Web controls. When UFT 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 UFT.

When UFT learns the controls on a Web page without Extensibility, it ignores certain types of elements and does not create test objects to represent the controls they define.

For other Web controls that are not supported out-of-the-box by the Web Add-in, UFT creates a generic WebElement test object. This type of test object 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.

Using Extensibility Accelerator for HP Functional Testing, you can develop UFT Web Add-in Extensibility toolkit support sets that extend the Web Add-in's basic capabilities and enable UFT to recognize additional Web controls.

Extensibility Accelerator makes it faster and easier to create the required extensibility XML files so that you can invest your main efforts in the development of the JavaScript functions that will enable UFT to work with your custom Web controls.

The Extensibility Accelerator user interface helps you define new test object classes, operations, and properties. It also provides a point-and-click mechanism you can use to map the test object classes you defined to controls in your application. Extensibility Accelerator deployment capabilities enable you to automatically deploy your new toolkit support set to UFT or to package it so that you can share it with other UFT users.

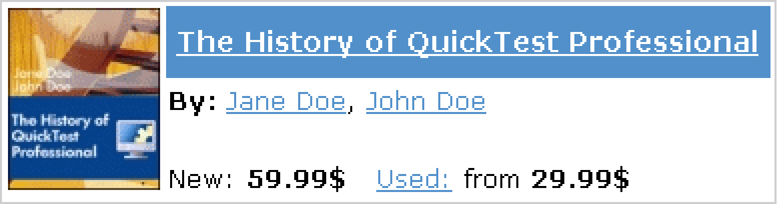
Prepare for This Lesson

Before you extend UFT support for a custom control, you must have access to its source file. You do not need to modify any of the custom control's sources to support it in UFT, but you do need to be familiar with them. Make sure you know what elements and attributes comprise the control, the events that might occur on this control, and so on. You use this information when you design the support class.

The source file for the Book control is located in:

```
%ALLUSERSPROFILE%\Documents\ExtAccTool\Samples\WebExtSample\Appli
cation\Book.htm
```

Open the file to run the control.



The History of QuickTest Professional

By: Jane Doe, John Doe

New: 59.99$ Used: from 29.99$

Run the control, open the source file for it and study the control's behavior and implementation.

The Book control contains information including the title of the book, its authors, the price for a new copy of the book, and the lowest price for which a used copy can be purchased.

Clicking on the title or the image of the book opens a page that can display more details about the book (but is not implemented in this sample). Clicking on an author name opens a page that can provide a list of books by the same author (but is not implemented in this sample). Clicking on **Used** opens a UsedBooks page, listing all of the available used copies of the book, and their prices.

Plan Support for the Web Add-in Extensibility Book Sample Toolkit

In this section, you analyze how UFT currently recognizes the Book control versus the way it should recognize it, based on your knowledge of the control. Next, you determine the answers to the questions in the section on understanding the Web Add-in Extensibility planning checklist in the *HP UFT Web Add-in Extensibility Developer Guide*, and fill in the checklist accordingly.

The best way to do this is to analyze how UFT recognizes the Book control on the one hand, using the Object Spy, Keyword View, and Record option, and on the other hand, to consider how the control is implemented and the purposes for which it is used.

1. **Open UFT and Run the Book control.**

   Open UFT and load the Web Add-in.

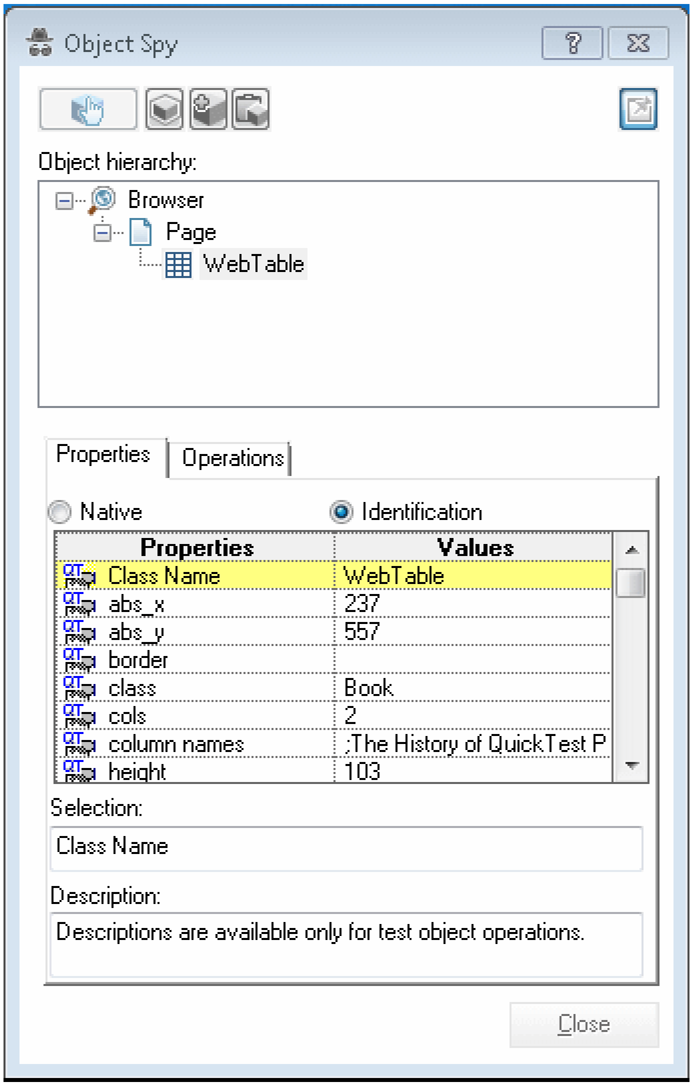
   Close any open instances of the Book control and open it by opening the `%ALLUSERSPROFILE%\Documents\ExtAccTool\Samples\WebExtSample\Application\Book.htm` file.

2. **Use the Object Spy to view the Book properties and operations.**

   In UFT, open a GUI test and select **Tools > Object Spy** or click the **Object Spy** toolbar button to open the Object Spy dialog box. Click the **Properties** tab and select **Identification Properties**.

   In the Object Spy dialog box, click the pointing hand , then click the Book control.

The Book control is implemented as a Web table, for which UFT support is built in, therefore it recognizes the control as a **WebTable**, named according to the title of the book. The icon used for the test object is the standard WebTable class icon.



Object hierarchy:

- Browser
  - Page
    - WebTable

○ Native ● Identification

Properties	Values
Class Name	WebTable
abs_x	237
abs_y	557
border	
class	Book
cols	2
column names	;The History of QuickTest P
height	103

Selection:

Class Name

Description:

Descriptions are available only for test object operations.

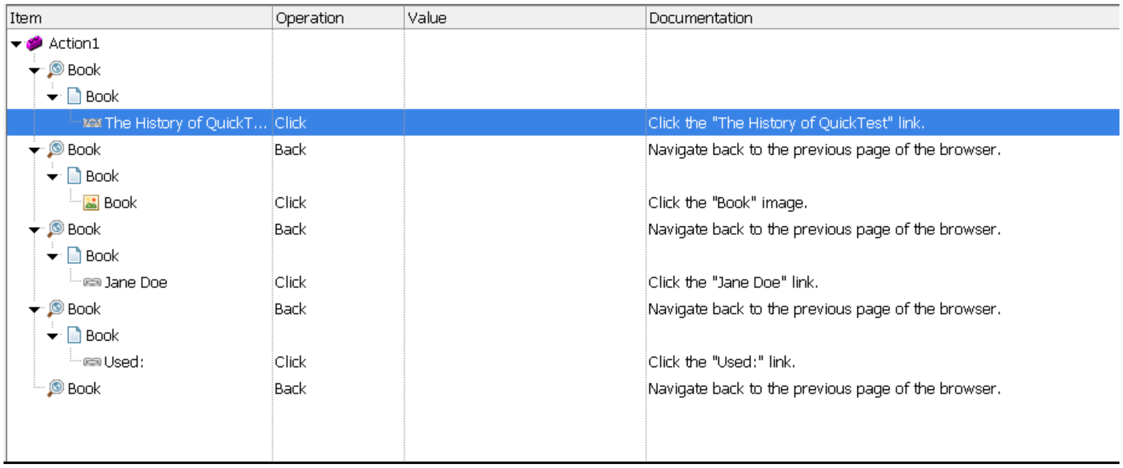
Close the Object Spy.

3. **Record operations on the Book control.**

   In UFT, select **Run > Run Settings** or **Record > Record Settings** to open the Record and Run Settings dialog box. In the Web tab, select **Record and run test on any open browser**. Click **OK**.

Click the **Record** button or select **Record > Record**. Click on different links in the Book control (you must return to the previous page after each click, to return to the Book control): the book title, the image in the control, an author name, and the **Used** link.

With each click, a new step is added to the test:



Item	Operation	Value	Documentation
Action1			
Book			
Book			
The History of QuickT...	Click		Click the "The History of QuickTest" link.
Book	Back		Navigate back to the previous page of the browser.
Book			
Book	Click		Click the "Book" image.
Book	Back		Navigate back to the previous page of the browser.
Book			
Jane Doe	Click		Click the "Jane Doe" link.
Book	Back		Navigate back to the previous page of the browser.
Book			
Used:	Click		Click the "Used:" link.
Book	Back		Navigate back to the previous page of the browser.

Click the **Stop** button or select **Record > Stop** to end the recording session.

Only simple **Click** steps are recorded, each attributed to a different object defined within the book control. **Click** operations are recorded independently on Web Link test objects with different names, or on the Book image test object. These steps are not helpfully meaningful in the context of this control.

4. **Determine the custom toolkit to which the Book control belongs.**

   When you extend UFT support for a control you always do so in the context of a toolkit. For the purpose of this tutorial, two custom Web controls are grouped to form the custom toolkit named WebExtSample: Book and UsedBooksTable.

   In this lesson you create support for the WebExtSample toolkit, initially supporting only the Book control.

5. **Complete the custom class support planning checklist.**

The Book control is implemented as a Web table, as follows:

```
<table class="Book">
    <tr>
        <td class="BookImageCell" rowspan="4">
            <a href=".\QtpHistory.htm">
                <img class="BookImage" alt="Book" src=".\Res\Book.jpg"/>
            </a>
        </td>
        <td class="BookCell">
            <a class="BookTitle" href=".\QtpHistory.htm" > The History of
QuickTest Professional </a>
        </td>
    </tr>
    <tr>
        <td class="BookCell">
             By: <a href=".\JaneDoe.htm">Jane Doe</a>, <a
href=".\JohnDoe.htm">John Doe</a>
        </td>
    </tr>
    <tr>
        <td class="BookCell">
        </td>
    </tr>
    <tr>
        <td class="BookCell">
             New: <strong>59.99$</strong> &nbsp; <a
href=".\UsedBooks.htm">Used:</a> from <strong>29.99$</strong>
        </td>
    </tr>
</table>
```

This section describes the decisions you need to make when planning your support for the Book control, and then summarizes the information in the support planning checklist.

a. Choose a test object class to represent the custom control:

   The Book control is implemented as a Web table control to assist in its appearance. For the purpose of performing tests on this control, there is no need to for UFT to recognize the Book control as a table. On the other hand, the basic support that UFT provides a generic Web element, using the WebElement object, is not specific enough for the Book control. Therefore, you create a new test object class named **WebExtBook**, which extends **WebElement**, and teach UFT to identify this test object class as the one that represents the Book control.

b. Consider how UFT can identify which test object class to use to represent the control. Keep in mind, however, that Extensibility Accelerator includes a point-and-click mechanism that you can use to automatically create rules for mapping test object classes to controls. Therefore, you do not have to make a final decision regarding these rules at the planning stage.

   A **WebExtBook** test object can be used to represent the control if the control's **tagName** property is **table** and its **className** property is **Book**.

c. Decide the details for the new test object class:

   - The new test object class is represented by the icon file: `<UFT installation folder>\Dat\Extensibility\Web\Toolkits\WebExtSample\Res\WebBook.ico`

   - No Help file is provided.

   - The new identification properties to support are: **title**, **authors**, **price**, and **min_used_price**. They should all be displayed in the UFT Checkpoint Properties and Output Value Properties dialog boxes, and be selected by default in checkpoints. None are used for Smart Identification.

     The identification properties that uniquely define the object are the book's title and the names of its authors.

   - The name of the test object itself should be the same as its title identification property.

d. Decide which test object methods to support for the custom control:

   The **WebExtBook** test object class provides the following test object methods:

   - **Select.** Simulates clicking the book's title or image. This is the default test object method.

   - **GoToAuthorPage.** Simulates clicking the specified author name (the available author names should be retrieved from the specific control during run-time).

   - **GoToUsedBooksPage.** Simulates clicking the **Used** link.

e. Decide whether the Object Spy should display **WebExtBook** test objects: Yes.

f. Define whether to support recording, and what events to record:

   Listen to mouse clicks that occur on the following elements in the control: title, image, authors, and Used. When a click occurs on one of these elements, record the relevant step in the test.

On page 152, you can see the checklist, completed based on the information above.

Web Add-in Extensibility Planning Checklist

☑	**Custom Control Support Planning Checklist**	**Specify in Toolkit XML?**	**Support by JavaScript function?**
☑	The sources for this custom control are located in: `%ALLUSERSPROFILE%\Documents\ExtAccTool\Samples\WebExtSample\Application\Book.htm`	n/a	n/a
☑	Specify the Web test object base class that the new test object class extends: (Default—WebElement) `WebElement`	n/a	n/a
☑	Is the base test object class WebElement? `Yes` If No, is there a base element (an element that matches the base test object class)? `n/a` If there is a base element, do you need a JavaScript function to return it? `n/a`	`No`	`No`

☑	**Custom Control Support Planning Checklist**	**Specify in Toolkit XML?**	**Support by JavaScript function?**
☑	Specify the New Web test object class details: • Test object class name: `WebExtBook` • Icon file location (optional): `<UFT installation folder>\Dat\Extensibility\Web\Toolkits\WebExtSample\Res\WebBook.ico` • Identification properties for description: `title, authors` • Default test object method: `Select` • Help file location: `n/a`	n/a	n/a
☑	Specify the basis for identifying the test object class to use for the control: `tagName = table, className = Book.`	Yes	No
☑	Specify the basis for naming the test object: `Use the book title`	n/a	Yes

☑	List the identification properties to support. Mark which should be available (and which selected by default) for checkpoints and which (if any) should be used for Smart Identification:
	`title, authors, price, min_used_price`
	`(all available for checkpoints and selected by default, none used for Smart Identification)`
		No	Yes

☑	List the test object methods to support (if required, include arguments, return values, Help file location and Help ID):
	`Select ()`
	`GoToAuthorPage (AuthorName)`
	`GoToUsedBooksPage ()`
		No	Yes

☑	**Custom Control Support Planning Checklist**	**Specify in Toolkit XML?**	**Support by JavaScript function?**

☑	Provide a dynamic list of values for any test object method arguments?
	`No`
	If so, list the arguments:
	`N/A`
		n/a	`Yes`

☑	Specify the types of children that UFT should learn with the control: `None`	`Yes`	No
☑	Display test objects of this class in the Object Spy? `Yes`	`No`	n/a

☑	Provide support for recording?
	`Yes`
	If so, list the events that should trigger recording:
	`Clicks on title, image, author names, and Used`
		`Yes`	`Yes`

Developing the Toolkit Support Set

Follow the steps in this section to develop support for the WebExtSample toolkit and learn the basics of working with Extensibility Accelerator and Web Add-in Extensibility. Developing this support comprises the following stages:

- "Stage 1: Create a Web Add-in Extensibility Project" (described on page 156)
- "Stage 2: Create a New Test Object Class" (described on page 157)
- "Stage 3: Create Rules to Map the Test Object Class to the Control" (described on page 161)
- "Stage 4: Define the WebExtBook's List of Operations and Identification Properties" (described on page 167)
- "Stage 5: Implement Support for the Test Object Operations" (described on page 177)
- "Stage 6: Test and Debug the Support You Designed for the Test Object Operations" (described on page 179)
- "Stage 7: Implement Support for the Identification Properties" (described on page 182)
- "Stage 8: Test and Debug the Support You Designed for the Identification Properties" (described on page 185)
- "Stage 9: Implementing Support for Recording on the Book Control" (described on page 189)
- "Stage 10: Package Your Toolkit Support Set For Distribution" (described on page 194)

Extensibility Accelerator provides tools for testing the support that you develop, without having to deploy it to UFT. In this tutorial you deploy the support at various stages of the development, to see the effect of your design on UFT's behavior.

Stage 1: Create a Web Add-in Extensibility Project

In this section, you create a new Web Add-in Extensibility project, and become familiar with the Extensibility Accelerator interface. When you create the project, Extensibility Accelerator creates the files that comprise the toolkit support set that you need to develop.

1. Open Extensibility Accelerator for HP Functional Testing

Extensibility Accelerator opens, displaying the Start Page and the Project Explorer.

The Start Page provides an explanation about Extensibility Accelerator, links to recently opened projects, sample projects, and a movie about the product, and a link that you can use to create a new project.

The Project Explorer will display the project's files, after it is created.

2. Create a project named WebExtSample

a. Click New Project .

b. In the New Project dialog box that opens, enter **WebExtSample** as the project name. Note the location for the new project and click **OK**.

   Extensibility Accelerator creates the new project and displays the following:

   - **Project Start Page.** This page provides an explanation about developing test object classes in the project, and a link you can use to create a new test object class.

   - **Workflow Window.** This window guides you through the workflow you need to follow when working in an Extensibility Accelerator project. You can click in this area to create new test object classes or deploy the support you develop.

   - **Class View.** The class view displays the test object classes that you design in your project and their operations.

3. View the project files that were created

Click the Project Explorer tab beneath the Class View to open the "Project Explorer" (described on page 40).

The project now contains the following files (you will learn more about their content later):

- A file named `WebExtSample.xml`. This is the toolkit configuration file.
- A file named `WebExtSampleTestObjects.xml`. This is the test object configuration file.
- A `JavaScript` folder that will contain files with the JavaScript functions you design to support the custom control.
- A `Res` file, for icon files that you use to represent the test object classes in UFT.
- A `Help` folder, for Help (`.chm`) files that you want UFT to use for context-sensitive Help on your test object classes. (In this tutorial you will not include Help files in your project.)

Stage 2: Create a New Test Object Class

In this section you create the WebExtBook test object class that you want UFT to use to represent the Book control, and define its basic characteristics.

1. ## Create a new test object class

   Click **Add Test Object Class** in the Workflow window.

   - A test object class designer opens in a new tab in the main document area. The designer opens to the "General Tab (Test Object Class Designer)", described on page 82.

   - In addition, a new JavaScript file is created, as you can see in the Project Explorer. This file will contain the JavaScript functions that you design to support the sample Book control.

   - In the Workflow window, the **Edit Test Object Class** area is now highlighted, indicating the current stage of the project design.

2. ## Define the name, description, and icon for the test object class

   a. In the **Name** box (in the General tab of the test object class designer), replace the default test object class name with `WebExtBook`.

      The name in the tab of the designer changes as you type, and an asterisk (*) in the tab indicates that you have made changes that have not yet been saved.

   b. Click **Save** 💾 .

      The JavaScript file name is modified, to match the name of the test object class, the name of the test object class is updated in the configuration (XML) files, and the files are saved.

   c. In the **Description** box, enter `The test object class used to represent a Book control.`

      This description is for your own records; it is not displayed in UFT.

   d. Replace the default icon used to represent WebExtBook test objects, which you can see displayed beneath the description:

      Click the browse button ... to the right of the **Icon File** box, then browse to and select

%ALLUSERSPROFILE%\Documents\ExtAccTool\Samples\WebExtSample\Res.

The icon file is imported into the Res folder in your project, as you can see the file in the Project Explorer. The file name is displayed in the **Icon File** box, and the icon image is displayed as well.

e. Click **Save All**.

3. Consider the rest of the test object class definitions that are available in this tab

- **Help File.** In this tutorial, you do not include Help files in your project.
- **Class Type.** Based on the plan in "Plan Support for the Web Add-in Extensibility Book Sample Toolkit" on page 147, accept the default WebElement base class and the default Object generic type.
- **Advanced.** This area contains advanced options that you do not need to use at this point.

4. View the content of the configuration files that were created

a. Double click the WebExtSample.xml file in the Project Explorer.

The file opens in an XML Editor in a separate tab (the editor is described on page 17). It contains the basic structure of a toolkit configuration file, which introduces your toolkit support set to UFT:

A root **Controls** element, containing a **Control** element for the WebExtBook test object class, specifying the JavaScript file that will contain the implementation of the support.

In addition, the file contains a browser independent **Identification** element, which will be updated when you perform "Stage 3: Create Rules to Map the Test Object Class to the Control" on page 161:

```
<Control TestObjectClass="WebExtBook">
```

```
    <Settings>
        <variable name="default_imp_file"
value="JavaScript\WebExtBook.js" />
    <Settings>
    <Identification>
        <Browser name="*" />
    </Identification>
</Control>
```

For details on the elements and attributes in the toolkit configuration file, see the *UFT Web Add-in Extensibility Toolkit Configuration Schema Help* (available with the Web Add-in Extensibility Help).

b. Double click the `WebExtSampleTestObjects.xml` file in the Project Explorer to open it another XML editor tab. The file contains the basic structure of a test object configuration file, which introduces your WebExtSample environment and its test object classes to UFT.

   The **PackageName** attribute in the **TypeInformation** element associates this test object configuration file (and the test objects defined in it) with the WebExtSample environment. If, when UFT opens, you do not select the WebExtSample environment, UFT ignores the test object class definitions in this file.

   Additional elements contain the test object class specifications that you defined in the designer:

```
<TypeInformation PackageName="WebExtSample" AddinName="Web">
    <ClassInfo GenericTypeID="Object" Name="WebExtBook"
BaseClassInfoName="WebElement">
        <Description>The test object class used to represent a
Book control.
        </Description>
        <HelpInfo />
        <IconInfo IconFile="INSTALLDIR\Dat\Extensibility\Web\
Toolkits\WebExtSample\Res\WebBook.ico" IconIndex="0"
        <TypeInfo />
        <IdentificationProperties />
    </ClassInfo>
</TypeInformation>
```

For details on the elements and attributes in the test object configuration file, see the *UFT Test Object Schema Help* (available with the Web Add-in Extensibility Help).

c. Close the XML files.

Stage 3: Create Rules to Map the Test Object Class to the Control

To support a custom control, UFT must be able to identify which test object class to use for a specific control. Therefore, a crucial stage in developing support for a custom control, is creating rules that enable UFT to correctly map the control to the test object class that represents it.

Extensibility Accelerator provides a point-and-click mechanism that helps you automate this process. You click on controls that you want to be represented by the test object class, and Extensibility Accelerator generates mapping rules for the test object class based on the properties of the selected controls.

These rules are stored in the toolkit configuration file (`WebExtSample.xml`), in **Identification** elements defined within each **Control** element. Each **Control** element defines a test object class. The **Identification** element specifies which controls should be represented by that test object class.

In this section, you use the mechanism provided in the "Map to Controls Tab (Test Object Class Designer)" (described on page 95), to create mapping rules for the sample Book control.

1. Open the Map to Controls Tab of the test object class designer

a. In the main document area of Extensibility Accelerator, select the tab for the WebExtBook test object class designer.

If you closed the designer, open it from the Class View by double-clicking the WebExtBook in the list of test object classes for your project.

b. Select the Map to Controls tab from the tabs on the left side of the test object class designer.

2. Run the sample Book application

Run the sample application by opening the `%ALLUSERSPROFILE%\Documents\ExtAccTool\Samples\WebExtSample\Application\Book.htm` file in a supported version of Microsoft Internet Explorer.

Make sure that running scripts in the browser is enabled.

3. Study the Map to Controls tab to understand its structure

In some cases, different rules for mapping the test object class to a control must be used, depending on the version of the browser running the control. In those cases, click the **Add Browser-Specific Rules** button ✚ on the tab next to the Default Rules tab, to create additional sets of rules.

Additionally, in some cases you want to create rules that instruct UFT to ignore specific controls, or to call a JavaScript function to perform more complex mapping. In those cases, select the **Ignore Control** or **Call Identification Function** panel instead of the **Identify Control** panel, before you begin the process of automatically creating rules.

For the book control, you only need one set of rules for all supported browser versions, and the rule is used to identify the control.

4. Start a control selecting session

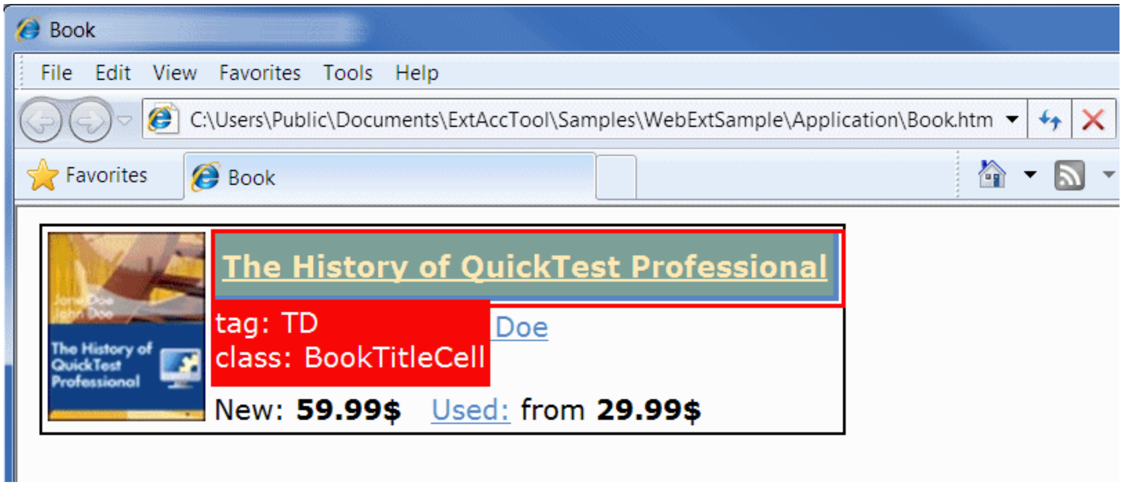
Click **Select Controls** in the Identify Control panel in the Default Rules tab.

Extensibility Accelerator is hidden, and two buttons are displayed at the top of the screen: **Create Rules** and **Cancel**.

5. **Hover over different areas in the application**

   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 **TD** HTML element is displayed for the area that contains the book title:

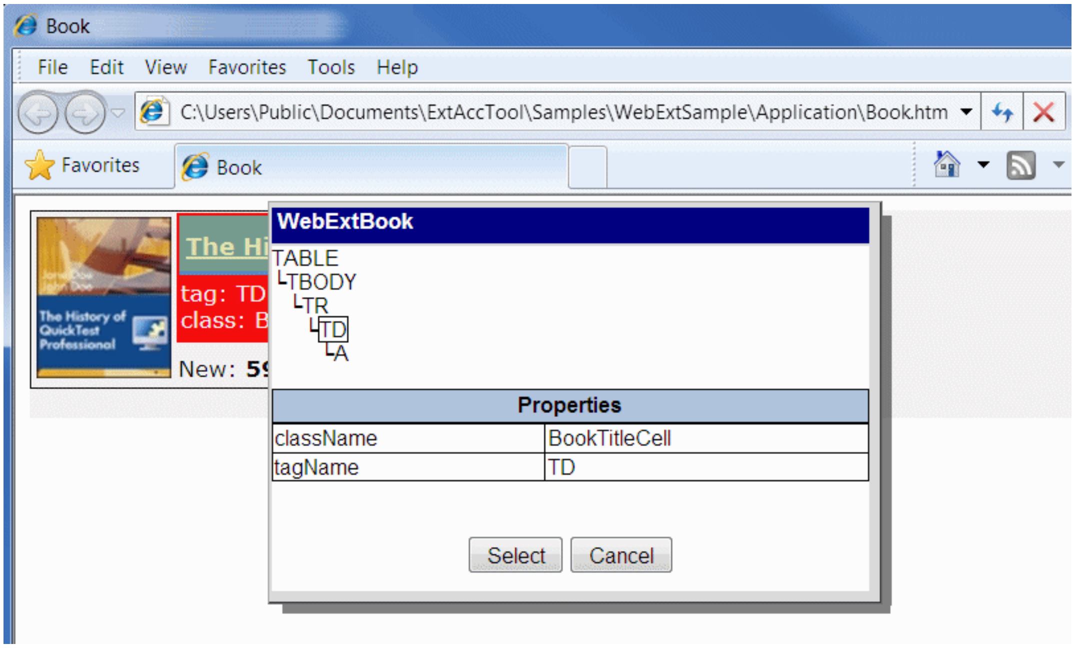


Book
File Edit View Favorites Tools Help
C:\Users\Public\Documents\ExtAccTool\Samples\WebExtSample\Application\Book.htm
Favorites
Book
The History of QuickTest Professional
tag: TD
class: BookTitleCell
Doe
New: 59.99$ Used: from 29.99$

6. **Select the HTML element that represents the Book control**

   a. Click on the area that contains the book title, selecting the **TD** element displayed above.

The "Selection Dialog Box" (described on page 105) opens on top of the browser, displaying the HTML properties of the element:



C:\Users\Public\Documents\ExtAccTool\Samples\WebExtSample\Application\Book.htm

```
TABLE
└TBODY
 └TR
  └TD
   └A
```

className	BookTitleCell
tagName	TD

This is not the element that represents the Book control itself. The top part of the dialog box displays the HTML hierarchy of the **TD** element, including the **TABLE** element that represents the Book control.

b. Click on **TABLE** in the HTML element hierarchy.

The properties of the TABLE element are displayed.

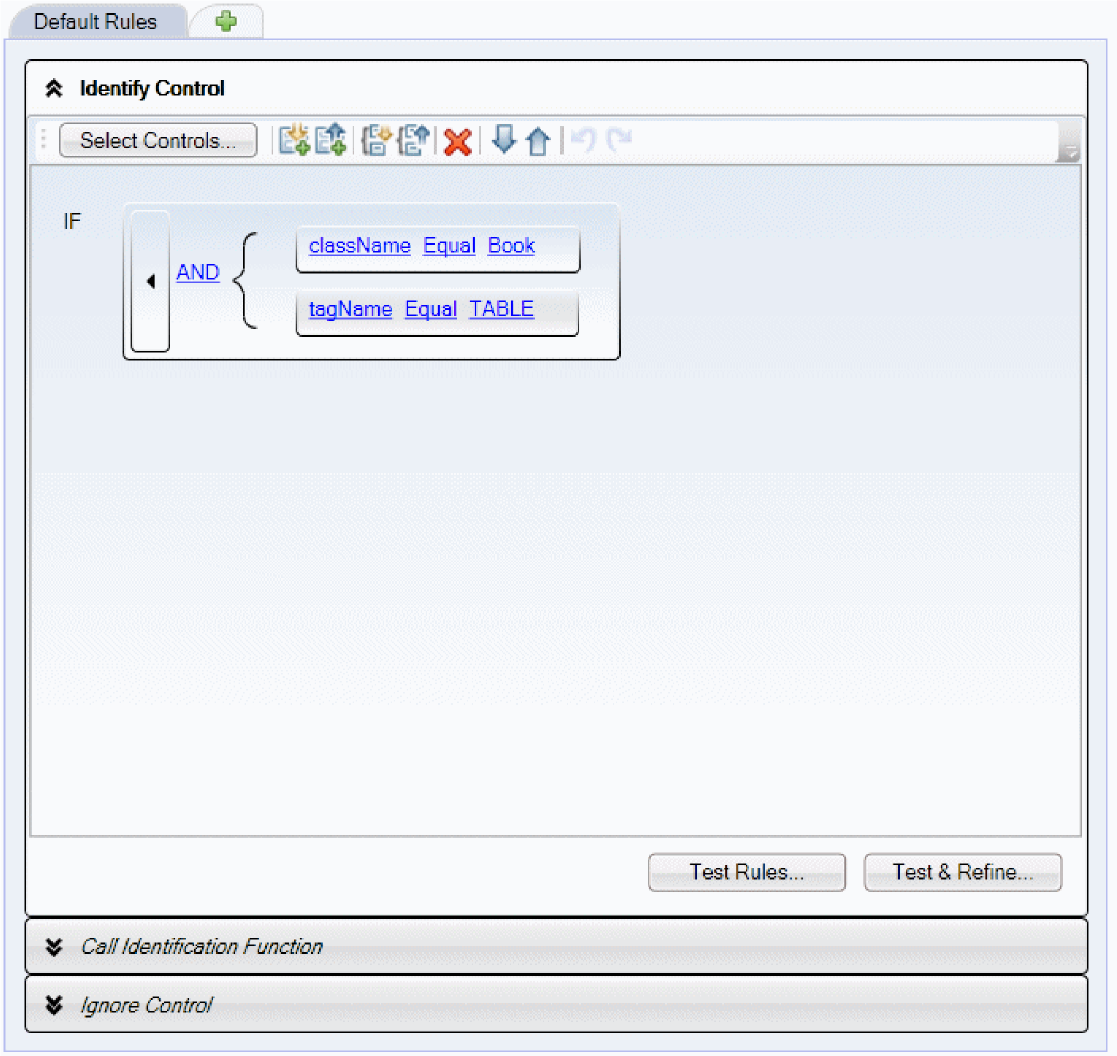
c. Click **Select**.

7. End the control selecting session

Click **Create Rules** to end the control selecting session.

Based on the properties of the TABLE element, Extensibility Accelerator creates mapping rules for the test object class in the "Rule Creation Panel" (described on page 100).

The rules match the conditions you planned in "Plan Support for the Web Add-in Extensibility Book Sample Toolkit" on page 147: `tagName = table, className = Book`.



IF AND
className Equal Book
tagName Equal TABLE

> **Tip:** When you develop support for your own controls, you should select more than one control during a session. This enables Extensibility Accelerator to create more accurate rules, which identify all controls of the relevant type and not only the specific control that you selected.

8. Test the mapping rules that Extensibility Accelerator created

In this step, you verify that the automatically created rules identify the Book control in Internet Explorer.

a. Click the **Test Rules** button, located beneath the rules.

   The Book control is highlighted in the browser, and a **Done** button is displayed at the top of the screen. (You may need to bring the browser into focus manually.)

b. Click **Done** and then save your changes in Extensibility Accelerator.

9. View the changes that Extensibility Accelerator made in the toolkit configuration file

In the Project Viewer, double-click the `WebExtSample.xml` file. In the file, you can see the **Identification** element defined in the WebExtBook's **Control** element. It contains a browser independent **Conditions** elements that contains two conditions, both of which must be met for the control to qualify as a **WebExtBook**:

```
<Control TestObjectClass="WebExtBook">
    <Identification>
        <Browser name="*">
            <Conditions type="IdentifyIfPropMatch">
                <Condition prop_name="className" expected_
value="Book" />
                <Condition prop_name="tagName" expected_
value="TABLE" />
            </Conditions>
        </Browser>
    </Identification>
</Control>
```

For details on defining the **Identification** element for a test object class, see the section on teaching UFT to identify the test object class to use for a custom Web control in the *HP UFT Web Add-in Extensibility Developer Guide*, and the *UFT Web Add-in Extensibility Toolkit Configuration Schema Help* (available with the Web Add-in Extensibility Help).

Stage 4: Define the WebExtBook's List of Operations and Identification Properties

In this section, you complete the definition of your test object class by specifying the operations (methods) and identification properties you want to be available for WebExtBook test objects. These definitions are stored in the test object configuration file (`WebExtSampleTestObjects.xml`).

In later stages, you will add the JavaScript implementation necessary to supports these operations and properties.

1. Open the Operations tab of the test object class designer

Select the Operations tab on the left side of the WebExtBook test object class designer.

2. Add the Select and GoToUsedBooksPage operations

a. In the Operations toolbar, click **Add** ✚ .

   A new operation, named **Operation1** is added to the list of operations, and selected.

b. In the **Name** box, enter `GoToUsedBooksPage`.

c. In the **Description** box, enter `Opens the UsedBooks page.`

d. Repeat the steps above to add an operation named `Select`, with the description `Selects the book`.

e. Select the **Default operation** check box (for the Select operation).

f. Click **Save**.

g. View the remaining options in this tab to learn what is available. For details, see "Operations Tab (Test Object Class Designer)" on page 109.

The Select and GoToUsedBooksPage operations do not receive any arguments or return a value. Therefore, you do not need to make any additional changes.

3. Add the GoToAuthorPage operation

a. Add a new operation, named `GoToAuthorPage`, with the description `Opens the Web page for the specified author.`

b. In the Operation Arguments toolbar, click **Add** ✚ .

c. In the **Name** column in the Operation Arguments table, enter `AuthorName`.

d. Clear the check box in the **Optional** column, because this is a mandatory argument.

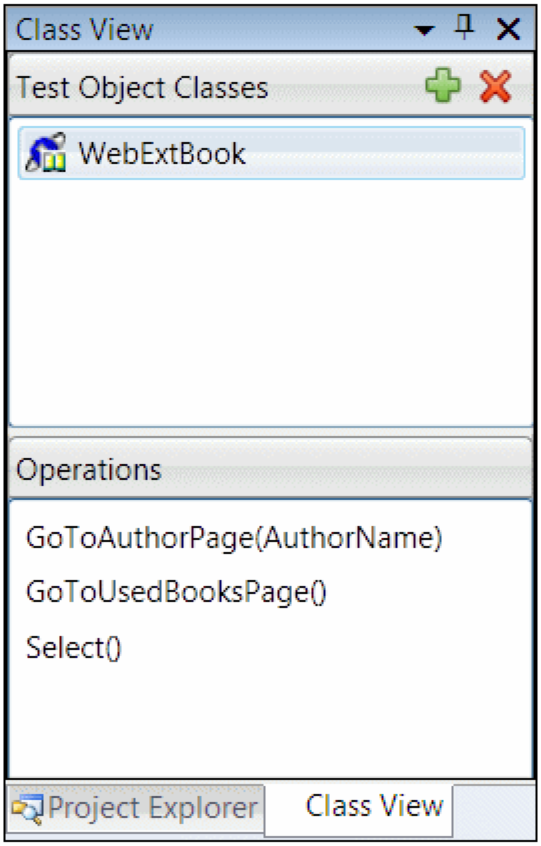
e. In the **Description** column, enter `The author.`

f. Accept the default values in the **Direction** and **Type** columns. These indicate that this is a string value passed to the operation.

g. Click **Save**.

4. View the new test object class in the Class View

Click the Class View tab beneath the Project Explorer to open the "Class View" (described on page 38):



The WebExtBook test object class and the icon you specified for it are displayed in the list of test object classes. The operations that you defined are displayed in the list of operations.

5. Open the Properties tab of the test object class designer

Select the "Properties Tab (Test Object Class Designer)" (described on page 121).

6. Specify the properties to inherit from WebElement

a. Select the **html tag** property and click **Inherit From Base Class** in the Properties toolbar.

b. Do the same thing for the **html id** property.

7. Add the properties: title, authors, price, min_used_price

a. In the Properties toolbar, click **Add** ✚ .

A new property, named **Property1**, is added to the list of properties and selected.

b. Click on the property name and enter `title`.

c. Add additional properties named `authors`, `price`, and `min_used_price`.

8. Add the properties to the relevant groups

The lists of properties on the right side of this tab specify the functionalities for which UFT uses the properties.

a. Select the **Object Identification - Mandatory** group.

Adding properties to this group instructs UFT to use these properties for the test object description. (The test object description is a set of properties used to uniquely identify a specific test object.)

html tag and **html id** were added to this group automatically.

b. Double-click on the properties `title` and `authors` to add them to the group.

c. Select the **Checkpoints and Output Values** group.

Adding properties to this group makes the properties available for checkpoints and output values in UFT.

d. Double-click each of the properties that you created to add them to this group.

The check box for each of the properties you add is selected by default. This instructs UFT to select the properties by default when creating checkpoints and output values.

e. Select the **Object Spy** group.

Adding properties to this group makes the properties available in the UFT Object Spy. All of the properties were added to this group by default.

9. View the changes that Extensibility Accelerator made in the test object configuration file

Open the Project Viewer and double-click the `WebExtSampleTestObjects.xml` file. In the file, you can see that the operations and identification properties of the **WebExtBook** test object class are now defined according to the details described in the "Web Add-in Extensibility Planning Checklist" on page 152.

The operations are defined, with their descriptions, and, where relevant, their arguments. The **Select** operation is specified as the default operation for the test object class. The properties are defined, with XML attributes indicating the groups to which they belong:

```
<ClassInfo DefaultOperationName="Select" GenericTypeID="Object"
Name="WebExtBook" BaseClassInfoName="WebElement">
    <Description>The test object class used to represent a Book control.
    </Description>
    <HelpInfo />
    <IconInfo IconFile="INSTALLDIR\Dat\Extensibility\Web\Toolkits\
WebExtSample\Res\WebBook.ico" IconIndex="0" />
    <TypeInfo>
        <Operation Name="GoToAuthorPage" PropertyType="Method">
            <Description>Opens the Web page for the specified
author.</Description>
            <Argument Name="AuthorName" IsMandatory="true">
                <Type VariantType="String" />
                <AdditionalInfo xsi:type="xsd:string">The
author.</AdditionalInfo>
            </Argument>
        </Operation>
        <Operation Name="GoToUsedBooksPage" PropertyType="Method">
            <Description>Opens the UsedBooks page.</Description>
        </Operation>
        <Operation Name="Select" PropertyType="Method">
            <Description>Selects the book.</Description>
        </Operation>
    </TypeInfo>
    <IdentificationProperties>
        <IdentificationProperty Name="authors" ForVerification="true"
ForDefaultVerification="true" />
        <IdentificationProperty Name="min_used_price" ForVerification="true"
ForDefaultVerification="true" />
```

```
        <IdentificationProperty Name="price" ForVerification="true"
ForDefaultVerification="true" />
        <IdentificationProperty Name="title" ForVerification="true"
ForDefaultVerification="true" />
    </IdentificationProperties>
</ClassInfo>
```

For details on the structure and content of the test object configuration file, see the *UFT Test Object Schema Help* (available with the Web Add-in Extensibility Help).

Deploy the Project to UFT and Test Your Toolkit Support Set

By this point, you have defined all the basic settings for your toolkit support set. Your XML configuration files contain the main elements required for UFT to display your toolkit in the list of supported environments, to display the WebExtBook test object as available in this environment, and to recognize the controls for which to use this test object. In addition, you have already defined the operations and properties available for WebExtBooks.

Though you have not designed the implementation necessary to enable the operations to run, or to enable the retrieval of the property values, in this section you already use your toolkit support set with UFT to spy on the Book control, add it to the object repository, and create test steps on the control.

To deploy the toolkit support set:

1. Select **View > Toolkit Support Properties**. In the "Toolkit Support Properties Designer" that opens (described on page 44), select the **Development mode** option and save your changes.

   When deploying during design stages, always use this option, which ensures that if you modified attributes of **IdentificationProperty** elements in the toolkit configuration XML file, UFT uses all of the changes you made. Clear this option when the design is complete, before distributing your support for regular use.

2. Select **Project > Deploy > Deploy to UFT**.

   The toolkit support set files are deployed to the `<UFT` or `Add-in installation folder>\dat\Extensibility\Web` folder in the structure described in "Deployment File Structure" on page 140.

   When you use Extensibility Accelerator to develop support on a computer without UFT, you can select **Project > Deploy > Deploy to Zip File** (or click the **Deploy Toolkit Support Set** button in the Workflow window) and specify the file path for the `.zip` file that you want to create.

   Extensibility Accelerator creates the necessary file structure within the zip file which you can then unzip it in the `<UFT` or `Add-in installation folder>\dat\Extensibility\Web` folder on a UFT computer.

To test the toolkit support set:

1. After you deploy the toolkit support set, open UFT.

   > **Note:** UFT reads toolkit support files when it opens. Therefore, if UFT is open, you must close and reopen it.

   The Add-in Manager dialog box displays the **WebExtSample** as a child of the Web environment in the list of available add-ins. (If the Add-in Manager dialog box does not open, see the *HP Unified Functional Testing Add-ins Guide* for instructions.)

2. Select the check box for **WebExtSample** and click **OK**. UFT opens and loads the support you designed.

3. Open a GUI test and use the **Define New Test Object** ✳ button in the Object Repository dialog box to open the Define New Test Object dialog box. The WebExtSample environment is displayed in the **Environment** list. When you select the WebExtSample environment from the list, the **WebExtBook** test object class

that you defined in the test object designer is displayed in the **Class** list, with the icon that you specified.

4. Run the sample control by opening the `%ALLUSERSPROFILE%\Documents\ExtAccTool\Samples\WebExtSample\Application\Book.htm` file.

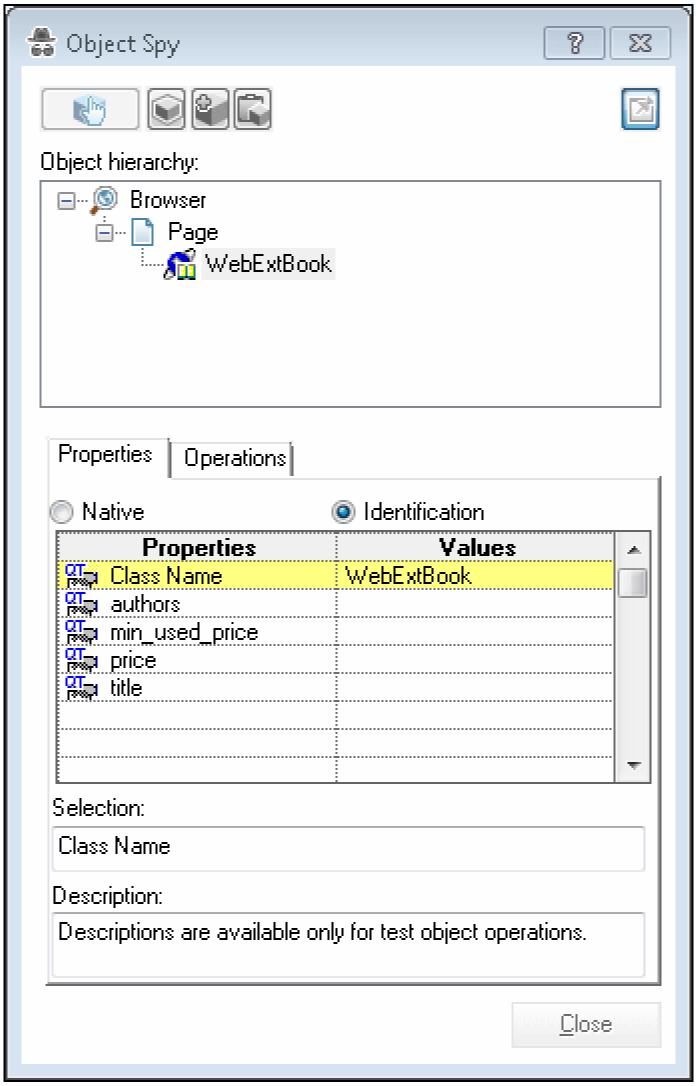
   > **Note:** UFT establishes its connection with an application when the application opens. Therefore, if the Book control is open, you must close it and run it again.

   In UFT, perform the following activities on the Book control, to see how UFT recognizes the control. (For details on working in UFT, see the *HP Unified Functional Testing User Guide*.)

   - Use the Object Spy to view the identification properties and test object operations that are supported for the Book control:

- The **WebExtBook** test object created for the Book control is given the name of its test object class. Later in this lesson, you customize your toolkit support set to provide a more specific name.

- The list of test object operations includes all of the operations (methods and properties) inherited from the **WebElement** base class, as well as all of the operations that you defined in the test object class designer.



○ Native ● Identification

Properties	Values
Class Name	WebExtBook
authors	
min_used_price	
price	
title	

Selection:
Class Name

Description:
Descriptions are available only for test object operations.

- The list of identification properties includes all of the properties that you defined. The property values are not displayed because you have not yet implemented a function that returns property values from the application (and the **WebElement** base class does not support these properties). You will implement such a function later in this tutorial.

- Use the **Add Objects to Local** button in the Object Repository dialog box to learn the Book control. Ensure that the correct icon is used to represent the test object in the object repository.

- In the Keyword View, create a test step choosing the **WebExtBook** object from the object repository in the **Item** column.

  - The list of available operations in the **Operation** column includes the operations that you defined.

  - Select the **GoToAuthorPage** operation. UFT displays the **AuthorName** argument name in a tooltip for the **Value** cell. If you defined an operation with more than one argument, the **Value** cell would be partitioned according to the number of arguments of the operation, when that operation was selected.

  - The tooltips displayed for each operation reflect the descriptions that you defined.

  - In the **Documentation** column, no documentation summary is available for steps with the operations that you defined. Extensibility Accelerator's user interface does not support documentation summaries, but you can add them manually to the test object configuration file in the XML editor. You will do that later in this lesson.

- In the Editor, create a test step with a WebExtBook test object. The statement completion feature displays all of the operations available for the test object, including the ones that you defined.

  If you had defined possible values for the operation's arguments they would be displayed in a statement completion list as well. You define possible values for an argument by defining an enumeration in the "Enumerations Designer", (described on page 48), and selecting it as the argument type in the "Operations Tab (Test Object Class Designer)", (described on page 109).

5. Run a test with a step that performs a new test object operation on a WebExtBook

test object. Because you have not yet implemented support for running test object operations, the step runs without performing anything on the application. However, Extensibility Accelerator created an empty JavaScript function stub for each new operation. Therefore, when UFT searches for a JavaScript function that will run the test object operation on the control, it finds the empty stub and does not produce a run-time error.

Stage 5: Implement Support for the Test Object Operations

After enabling UFT to recognize the custom controls, you must provide support for running test object operations. For each test object operation that you defined, you must write a JavaScript function that UFT runs to perform the step on the control.

In this section, you provide support for the WebExtBook's test object methods: **Select**, **GoToAuthorPage (***AuthorName***)**, and **GoToUsedBooksPage**.

> **Note:** All of the JavaScript code that you write in this tutorial is designed to run on Microsoft Internet Explorer. When you develop support for your own controls, you can design it to be browser-independent and enable UFT to support the controls on other browsers as well. For details, see the section on developing browser-independent support in the *HP UFT Web Add-in Extensibility Developer Guide*.

1. **Open the WebExtBook.js JavaScript file**

   It is possible to specify a JavaScript file and function for each test object method that you define. However, in this lesson, you develop support for running test object methods in the simplest way possible.

   Do one of the following:

- In the Operations tab in the test object designer, select the **Select** operation and click the **Implementation Code** button next to the **Name** box.
- In the Class View, right-click the **Select** operation, and select **Implementation Code**.

The `WebExtBook.js` file opens to the stub created for the **Select** function in a JavaScript Editor (described on page 17).

2. Add the implementation code for the operations you defined

By default, UFT searches in the JavaScript file for a function with the same name as the test object operation. The JavaScript functions must also have the same signature as the test object operations they run. Therefore, you need to implement the following JavaScript functions: **Select**, **GoToAuthorPage (**`AuthorName`**)**, and **GoToUsedBooksPage**.

Extensibility Accelerator already created stubs for these functions, with commented out suggestions of code lines that you might want to include.

Locate the stub for each function and replace it with the relevant implementation provided below.

> **Note:** The _elem object is a reserved object that UFT uses to refer to the HTML control currently being handled.

```
function Select()
{// Click the link in the second cell of the first row.
    _elem.rows[0].cells[1].children[0].click();
    _util.LogLine("Running Select test object method.",1);
}
```

```
function GoToAuthorPage(AuthorName)
{// Look for the specified author name among the children of the first cell
// in the second row and click it.
    var bWasFound = false;
    for( var i = 0 ; i < _elem.rows[1].cells[0].children.length ; ++i )
    {
        if( _elem.rows[1].cells[0].children[i].innerText == AuthorName )
        {
            _elem.rows[1].cells[0].children[i].click();
            _util.LogLine("Running GoToAuthorPage test object method.",1);
bWasFound = true;
            // Update the report
            var params = new Array();
            params[0] = AuthorName;
            _util.Report(micPass, "GoToAuthorPage", toSafeArray(params),
"Author '" + AuthorName + "' was found :-)");
            break;
        }
    }
    if( bWasFound == false )
    throw ("Author name not found !");
}
```

```
function GoToUsedBooksPage()
{// Click the link in the first cell of the third row.
    _elem.rows[3].cells[0].children[1].click();
    _util.LogLine("Running GoToUsedBooksPage test object
method.",1);
}
```

Stage 6: Test and Debug the Support You Designed for the Test Object Operations

Using the **Debug Test Operation** option, you can test and debug the implementation you designed for the operation without deploying the toolkit support set to UFT. Extensibility Accelerator runs your JavaScript functions on the control just as UFT would.

While your JavaScript function runs, you can debug your JavaScript functions as you would in a regular Microsoft Visual Studio JavaScript debugging session.

1. Prepare for testing the operations

a. Enable script debugging in your Internet Explorer browser.

   For example: In Internet Explorer 7.0, select **Tools > Internet Options**. In the Advanced tab, clear the **Disable script debugging** options in the **Browsing** group. (You must close and reopen the browser after making changes to these options.)

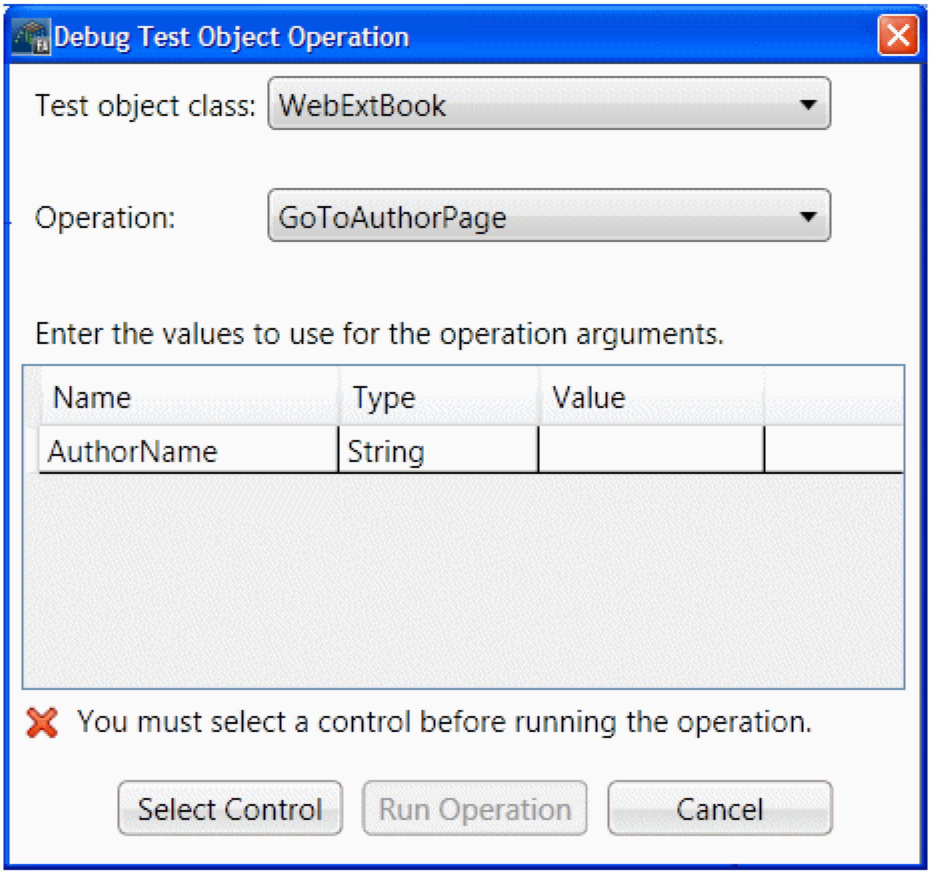
b. Run the sample application in Internet Explorer. (Make sure to allow blocked content).

c. In Extensibility Accelerator, open the Output window (**Debug > Windows > Output**). If necessary, right-click in the window and select **Clear All**.

2. Select the GoToAuthorPage test object operation for testing

In the Operations tab in the test object designer, select the **GoToAuthorPage** operation and click the **Debug Operation** button ▶ in the Operations toolbar.

The Debug Test Object Operation dialog box opens, with the **WebExtBook** test object class and the **GoToAuthorPage** operation selected.



Test object class: WebExtBook

Operation: GoToAuthorPage

Enter the values to use for the operation arguments.

Name	Type	Value
AuthorName	String	

✖ You must select a control before running the operation.

3. Select the Book control as the control on which to run the operation

a. Click **Select Control**.

   Extensibility Accelerator is hidden, the Book control is highlighted in the browser, and a **Cancel** button is displayed at the top of the screen. (This might take a moment.)

b. Click the highlighted Book control.\

   The Debug Test Object Operation dialog box comes back into focus.

4. Enter the argument to use for the operation

In the argument value table, in the cell for the AuthorName value, enter `Jane Doe`.

5. Run the operation

Click **Run Operation**.

Extensibility Accelerator runs the **GoToAuthorPage** operation with the argument "`Jane Doe`".

The sample application opens to the `JaneDoe.htm` page.

In the Output window, you can see the _util.Logline and _util.Report functions that are called during this process.

> **Note:** In this step, you tested the function using a valid value for the argument. If you use an incorrect author name, the JavaScript function throws an exception and Extensibility Accelerator displays an error message. When you run the operation in a UFT run session, the exception results in a run-time error message, displaying the string provided by the exception.

6. Run the GoToUsedBooksPage operation with a breakpoint in

the function

a. In the browser, click **Back to sample**.

b. In the `WebExtBook.js` file in the JavaScript editor, insert a breakpoint by clicking on the left margin at the first line of code in the **GoToUsedBooksPage** function.

c. In the Class View, right-click the **GoToUsedBooksPage** operation and select **Debug** to open the Debug Test Object Operation dialog box.

d. Click **Run Operation**.

   Extensibility Accelerator runs the **GoToUsedBooksPage** function and stops at the breakpoint.

e. Many debugging options are now available in the Debug toolbar and menu. You can use **Step Into**, **Continue**, or **Stop Debugging** to complete this session.

7. Test the Select method

Use the tools that you learned in the previous steps to test the remaining test object operation you implemented.

Stage 7: Implement Support for the Identification Properties

In this section you implement support for retrieving the values of identification properties during a test run. UFT uses identification property run-time values in different standard test object methods, such as **GetROProperty**. Identification property run-time values are also required for different basic capabilities, such as creating checkpoints and outputting values.

The WebExtBook's identification properties you support are: **title**, **price, min_used_price**, and **authors**. You do not have to implement support for **html tag** and **html id**, as this implementation is inherited from WebElement.

In addition, you provide a means for retrieving a value for the **logical_name** property. This is a property that UFT uses internally, when creating a test object, to determine the name for the test object. By customizing the value of this property, you instruct UFT to name the WebExtBook test object according to its title.

If the **get_property_value** function does not support the **logical_name** property, the test object is given the name of the test object class (followed by an index, if there is more than one object of the same test object class on the same page).

To support retrieving the run-time values of the WebExtBook's identification properties:

It is possible to specify a JavaScript file and function name for the retrieval of the test object's identification properties. However, in this lesson, you develop your support in the default file and function. Extensibility Accelerator created a stub for the function, with sections to implement for each property you defined.

1. In the Properties tab in the test object designer, select a property and click the **Implementation Code** button in the Properties toolbar.

   The `WebExtBook.js` file opens in a JavaScript Editor (described on page 17). The cursor is placed in the stub provided for the **get_property_value** function, at the beginning of the section designed to retrieve the selected property's value.

2. In the **get_property_value** function, locate the code snippet intended for each of the following properties and edit it to retrieve the property value based on the code below: **logical_name**, **title**, **price, min_used_price**, and **authors**.

   Alternatively, replace the whole **get_property_value** function with this implementation:

```
function get_property_value(prop)
{
    if ( prop == "logical_name" || prop == "title" )
    // For the "title" identification property, as well as the
    // "logical_name" property, return the inner text of the
    // second cell in the first row.
    {
        return _elem.rows[0].cells[1].innerText;
    }
    if ( prop == "authors" )
    // To return a list of all the authors, look for all the
    // children of the first cell in the second row.
    {
        var BookAuthors = "";
        var AuthorsCount = 0;
        for( var i = 0 ; i < _elem.rows[1].cells
[0].children.length ; ++i )
        {
            if( _elem.rows[1].cells[0].children[i].tagName == "A"
)
            {
                if( AuthorsCount > 0 )
                    BookAuthors += ",";
                BookAuthors += _elem.rows[1].cells[0].children
[i].innerText;
                AuthorsCount++;
            }
        }
        return BookAuthors;
    }
    if ( prop == "price" )
    // To return the price of the book, return the innerText
    // property of the first cell in the fourth row.
    {
        return _elem.rows[3].cells[0].children[0].innerText;
    }
    if ( prop == "min_used_price" )
    // To return the lowest price available for a used copy of
    // the book, return the innerText property of the second
    // child of the first cell in the fourth row.
    {
        if( _elem.rows[3].cells[0].children.length > 2 )
            return _elem.rows[3].cells[0].children[2].innerText;
    }
}
```

Stage 8: Test and Debug the Support You Designed for the Identification Properties

Using the **Debug Property Retrieval** option, you can test and debug the implementation you designed for the properties similarly to the way you tested the operations.

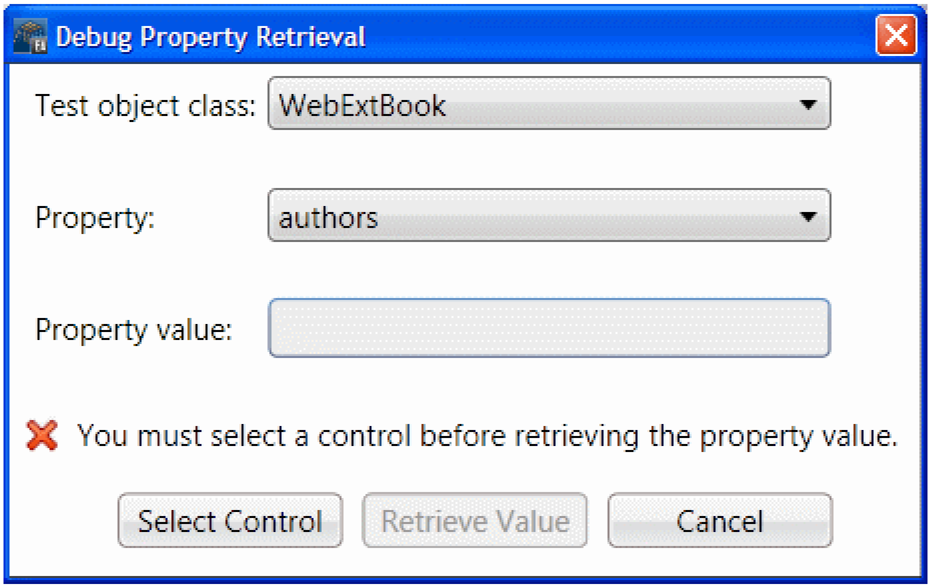
1. **Prepare for testing the properties**

   Open the sample application in a supported version of Internet Explorer. (Reminder: The WebExtBook project must be open in Extensibility Accelerator before you open the browser, and the browser must allow blocked content.)

2. **Open the Debug Property Retrieval dialog box**

   In the Properties tab in the test object designer, select the **authors** property and click the **Debug Property Retrieval** button ▶ in the Properties toolbar.

   The Debug Property Retrieval dialog box opens, with the **WebExtBook** test object class and the **authors** property selected.:



Debug Property Retrieval

Test object class: WebExtBook

Property: authors

Property value:

You must select a control before retrieving the property value.

3. Select the Book control as the control whose property value you want to retrieve

a. Click **Select Control**.

Extensibility Accelerator is hidden, the Book control is highlighted in the browser, and a **Cancel** button is displayed at the top of the screen. (This might take a moment.)

b. Click the highlighted Book control.

The Debug Test Property Retrieval dialog box comes back into focus.

4. Retrieve the property value

Click **Retrieve Value**.

Extensibility Accelerator runs the **get_property_value** function on the Book control with the argument "`authors`".

The Debug Test Property Retrieval dialog box reopens and displays `Jane Doe,John Doe` in the **Property value** box.

5. Test the rest of the properties

Test the retrieval of the values for **min_used_price**, **price**, and **title** properties the same way as you tested the first property.

Deploy the Support to UFT and Test It Again

You have now completed the design required to support running steps on the Book control, performing operations, and retrieving property values.

In this section, you deploy the toolkit support set to UFT again, and test running steps on the Book control. Before deploying, you make the changes that are necessary to support the **Documentation** column in the Keyword View.

1. Manually add Documentation elements to the test object configuration file

Open the `WebExtSampleTestObjects.xml` file from the Project Explorer in Extensibility Accelerator and add **Documentation** elements to the operations you defined. The TypeInfo element should now look like this:

```
    <TypeInfo>
        <Operation Name="GoToAuthorPage" PropertyType="Method">
            <Description>Opens the Web page for the specified
author.</Description>
            <Documentation><![CDATA[Open the Web page for %a1.]]>
</Documentation>
            <Argument Name="AuthorName" IsMandatory="true">
                <Type VariantType="String" />
                <AdditionalInfo xsi:type="xsd:string">The
author.</AdditionalInfo>
            </Argument>
        </Operation>
        <Operation Name="GoToUsedBooksPage" PropertyType="Method"<
            <Description>Opens the UsedBooks page.</Description>
            <Documentation><![CDATA[Open the %1 UsedBooks page.]]
></Documentation>
        </Operation>
        <Operation Name="Select" PropertyType="Method">
            <Description>Selects the book.</Description>
            <Documentation><![CDATA[Select the %1 book.]]
></Documentation>
        </Operation>
    </TypeInfo>
```

2. Deploy your support to UFT

Select **Project > Deploy > Deploy to UFT**.

3. Open UFT and run the Book application

a. Close and reopen UFT. Select the check box for **WebExtSample** in the Add-in

Manager dialog box and click **OK**. UFT opens and loads the support you designed.

b. Close and rerun the sample control.

4. Test the support for naming the test object

Open a GUI test, open the Object Repository, and use the **Add Object to Local** ➕ button to learn the Book control. Make sure that the test object that UFT creates is named **The History of QuickTest Professional**.

5. Test the support for running test object operations

a. Create a test with the following step and then run the test:

```
Browser("Book").Page("Book").WebExtBook("The History of
QuickTest Professional").
GoToAuthorPage "Jane Doe"
```

> **Note:** If you run the **GoToAuthorPage** test object method with an author name that does not exist in the control, the JavaScript function throws an exception. If your browser displays an error message about the unhandled exception, asking if you want to debug, click **No**.
>
> UFT displays a run-time error message with the string provided by the exception, and the test step fails.

b. Create and run similar tests to test the **Select** and **GoToUsedBooksPage** test object methods.

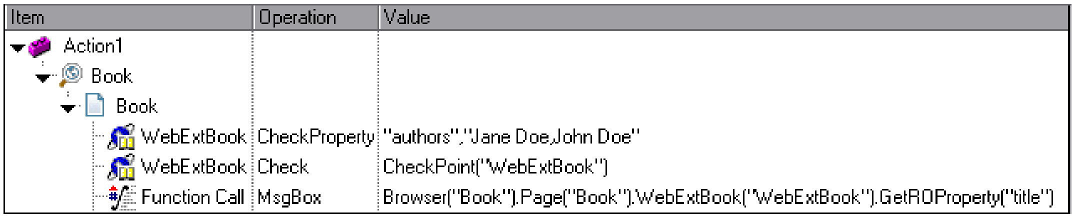
c. Verify that the documentation strings displayed for the test object operations in the Keyword view are correct.

6. Test the support for retrieving run-time values of identification properties

a. Create a step with a WebExtBook test object.

b. Right-click the object and select **Insert Standard Checkpoint**. The Checkpoint Properties dialog box opens. Make sure that the identification properties you defined in the test object configuration file (**title**, **authors**, **price**, and **min_used_price**) are included in the list of properties and are selected.

c. Create and run a test that retrieves each identification property and checks its value, or displays it in a message box. For example, you can run the following test:



Item	Operation	Value
Action1		
Book		
Book		
WebExtBook	CheckProperty	"authors","Jane Doe,John Doe"
WebExtBook	Check	CheckPoint("WebExtBook")
Function Call	MsgBox	Browser("Book").Page("Book").WebExtBook("WebExtBook").GetROProperty("title")

The first step checks the value of the authors property, the checkpoint in the second step checks the properties selected in the checkpoint (in this case price and min_used_price) and the third step displays the book's title in a message box.

d. Click **OK** to close the message box. The test run is completed and the run results are displayed. Expand the run results tree to view the step details.

Stage 9: Implementing Support for Recording on the Book Control

By this point in the tutorial, your toolkit support set already enables full UFT functionality. UFT recognizes the Book control, can learn it and can run tests on it.

An additional, optional way to create tests in UFT is by recording operations that a user performs on the application. As you can see in "Plan Support for the Web Add-in Extensibility Book Sample Toolkit" on page 147, by default UFT records plain **Click** operations on the various Web link and image objects within the Book control. It would be more helpful to record **Select**, **GoToAuthorPage**, and **GoToUsedBooksPage** operations on the Book control itself, in response to those same clicks.

To support customized recording on your control, you must instruct UFT to listen to the relevant events and inform UFT what test steps to record in response to each event.

To do this, you write two types of JavaScript functions:

- One JavaScript function that uses the **RegisterForEvent** function in the **_util** utility object that UFT exposes in the Web Add-in Extensibility SDK to register for listening to the correct events on the correct elements. The arguments of this function also determine what JavaScript functions UFT calls when each event occurs.

  You specify the name and, optionally, the location of this JavaScript function in the General tab of the test object class designer. Extensibility Accelerator saves this information in the toolkit configuration file.

- One or more JavaScript functions that handle the events by calling the **Record** function in the **_util** utility object to inform UFT what step to add to the test.

  > **Note:** The **Record** function, and other utility object functions, require a **SafeArray** type argument. To convert an array to a **SafeArray**, you can use the toSafeArray (array) function that Web Add-in Extensibility provides. This function is defined in `<Extensibility Accelerator installation folder>\bin\PackagesToLoad\common.js`. (This file is also located in the `<UFT installation folder>\dat\Extensibility\Web\Toolkits folder`.)

For information on the syntax of the utility object functions, see the _util section in the *UFT Web Add-in Extensibility API Reference* (available with the Web Add-in Extensibility Help).

To develop support for recording on the Book control:

1. Set the advanced options for recording in the General tab of the test object class designer

a. Open the WebExtBook "General Tab (Test Object Class Designer)".

   If you closed the designer, open it from the Class View by double-clicking **WebExtBook** in the list of test object classes for your project.

b. Click **Advanced Options**, and scroll down to the **RecordOptions**.

c. In the Event registration function name box enter **ListenToEvents**.

d. Clear the check boxes for the options that instruct UFT to use standard Web event configuration to handle events on the control and its children.

e. Save your changes, and open the `WebExtBook.js` file from the Project Explorer to see how your definitions are reflected in the file.

   The following **Record** element is added within the WebExtBook **Control** element:

```
<Record>
    <EventListening use_default_event_handling_for_children =
"false"
                    use_default_event_handling = "false"
                    type = "javascript" function =
"ListenToEvents"/>
</Record>
```

   This instructs UFT not to use the default Web Event Configuration for handling events on the Book control and its children, but to call the JavaScript function **ListenToEvents**. Because you did not specify a JavaScript file, UFT looks for the JavaScript function in the `WebExtBook.js` file specified at the **Control** level for the WebExtBook test object class.

2. Open the WebExtBook.js file

In the General tab in the test object class designer, in the advanced options for recording, click **Implementation Code** button near the **Event registration function name** box.

The `WebExtBook.js` file opens to the stub created for the **ListenToEvents** function.

3. Add the JavaScript implementation required to support

recording

a. Replace the stub created for the **ListenToEvents** function with the following code:

```
function ListenToEvents( elem )
{
    // Connect to the "Select" event: When the book name or the book
    // icon is clicked, call OnSelectClicked.
    _util.RegisterForEvent( _elem.rows[0].cells[0].children[0],
"onclick", "OnSelectClicked");
    _util.RegisterForEvent( _elem.rows[0].cells[1].children[0],
"onclick" , "OnSelectClicked" );
    // Connect to the "Author" event: When an author name is clicked,
    // call OnAuthorClicked.
    for( var i = 0 ; i < _elem.rows[1].cells[0].children.length ; ++i )
    {
        if( _elem.rows[1].cells[0].children[i].tagName == "A" )
        {
            _util.RegisterForEvent( _elem.rows[1].cells[0].children[i],
"onclick", "OnAuthorClicked" );
        }
    }
    // Connect to the "UsedBooks" event: When "Used" is clicked,
    // call OnUsedBooksClicked.
    if(  _elem.rows[3].cells[0].children.length > 1 )
        _util.RegisterForEvent( _elem.rows[3].cells[0].children[1],
"onclick", "OnUsedBooksClicked" );
    return true;
}
```

This function registers UFT to listen to click events on the book's title, image, and authors, and on the **Used** link. When registering for an event, this function specifies what JavaScript function UFT must call when the event occurs.

b. Add the following event handler JavaScript functions:

```
function OnSelectClicked( handlerParam , eventObj )
{
    // Record the "Select" step
    var arr = new Array();
    _util.Record( "Select", toSafeArray(arr) , 0 );
    return true;
}
```

```
function OnAuthorClicked( handlerParam , eventObj )
{
    // Record the "GoToAuthorPage" step
    var arr = new Array();
    arr[0] = eventObj.srcElement.innerText;
    _util.Record( "GoToAuthorPage", toSafeArray(arr) , 0 );
    return true;
}
```

```
function OnUsedBooksClicked( handlerParam , eventObj )
{
    // Record the "GoToUsedBooksPage" step
    var arr = new Array();
    _util.Record( "GoToUsedBooksPage", toSafeArray(arr) , 0 );
    return true;
}
```

These functions record **Select**, **GoToAuthorPage**, and **GoToUsedBooksPage** on the WebExtBook test object, as planned in "Plan Support for the Web Add-in Extensibility Book Sample Toolkit" on page 147.

Deploy the Support to UFT and Test Recording

After developing the support for recording on the Book control, you deploy the updated toolkit support set to UFT and test it.

To test the support for recording operations performed on the Book control:

1. Make sure to save all your changes, and then select **Project > Deploy > Deploy to UFT**.

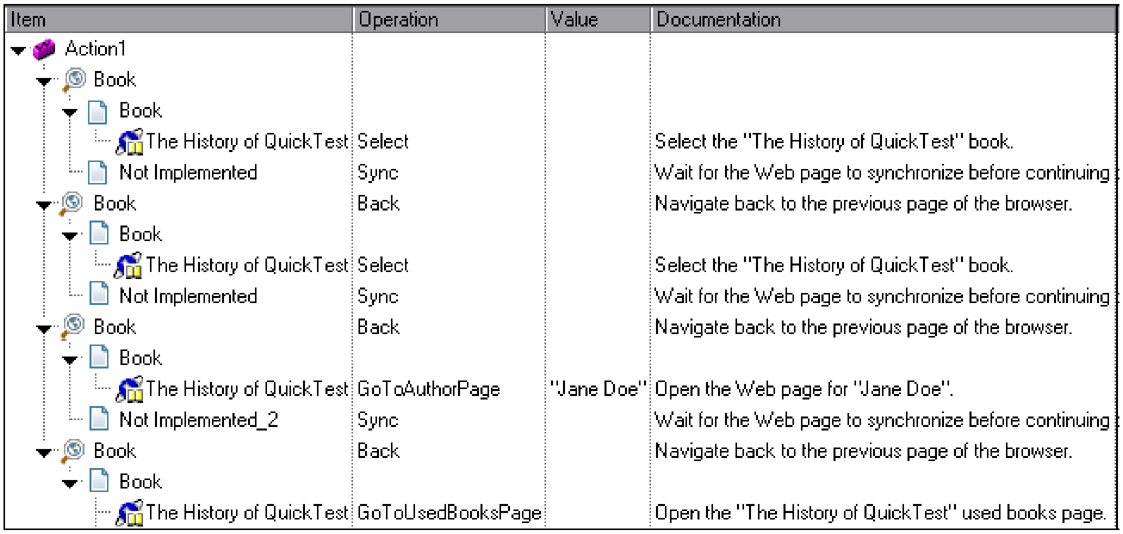
2. Close and reopen UFT. Select the check box for **WebExtSample** in the Add-in Manager dialog box and click **OK**. UFT opens and loads the support you designed.

3. Close and rerun the sample control in Internet Explorer.

4. Open a GUI test and click the **Record** button or select **Record > Record**. Click on different links in the Book control (you must return to the previous page after each

click, to return to the Book control): the book title, the image in the control, an author name, and the **Used** link.

With each click, a new step is added to the test, using the operations that you designed:



Item	Operation	Value	Documentation
Action1			
Book			
Book			
The History of QuickTest	Select		Select the "The History of QuickTest" book.
Not Implemented	Sync		Wait for the Web page to synchronize before continuing
Book	Back		Navigate back to the previous page of the browser.
Book			
The History of QuickTest	Select		Select the "The History of QuickTest" book.
Not Implemented	Sync		Wait for the Web page to synchronize before continuing
Book	Back		Navigate back to the previous page of the browser.
Book			
The History of QuickTest	GoToAuthorPage	"Jane Doe"	Open the Web page for "Jane Doe".
Not Implemented_2	Sync		Wait for the Web page to synchronize before continuing
Book	Back		Navigate back to the previous page of the browser.
Book			
The History of QuickTest	GoToUsedBooksPage		Open the "The History of QuickTest" used books page.

Click the **Stop** button or select **Record > Stop** to end the recording session.

Stage 10: Package Your Toolkit Support Set For Distribution

Now that your toolkit support set for the Book control is complete, you can deploy it to any UFT computer on which you want to run and test the Book application.

1. Select **View > Toolkit Support Properties**. In the "Toolkit Support Properties Designer" that opens (described on page 44), clear the **Development mode** option and save your changes.

2. Click **Deploy Toolkit Support Set** in the Workflow window, or select **Project > Deploy > Deploy to Zip File** and specify the file path for the `.zip` file that you want to create.

3. Unzip the file in the `<UFT` or `Add-in installation folder>\dat\Extensibility\Web` folder on any UFT computer.

Summary

In this tutorial you learned to develop a Web Add-in Extensibility toolkit support set using Extensibility Accelerator.

- You learned the recommended workflow for developing Web Add-in Extensibility projects, and practiced the required steps.
- You created a new test object class, WebExtBook, defining its identification properties and test object methods.
- You learned to create mapping rules for a Book control, enabling UFT to recognize it as an WebExtBook test object.
- You learned to support new identification properties and test object methods.
- You learned to understand the toolkit support set's configuration files.
- You learned to support recording and you made use of the **Record** and **RegisterForEvent** utility methods.

Where Do You Go from Here?

For more details on using Extensibility Accelerator, see the relevant task and reference topics in this guide.

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

For more details on the structure and content of the test object configuration file, see the *UFT Test Object Schema Help* (available with the Web Add-in Extensibility Help).

For more details on the structure and content of the toolkit configuration file, see the *UFT Web Add-in Extensibility Toolkit Configuration Schema Help* (available with the Web Add-in Extensibility Help).

For more details on the **_util** utility object and global JavaScript methods, see the *UFT Web Add-in Extensibility API Reference* (available with the Web Add-in Extensibility Help).

Send Us Feedback

✉ Can we make this User Guide better?

Tell us how: sw-doc@hp.com

Java™
COMPATIBLE