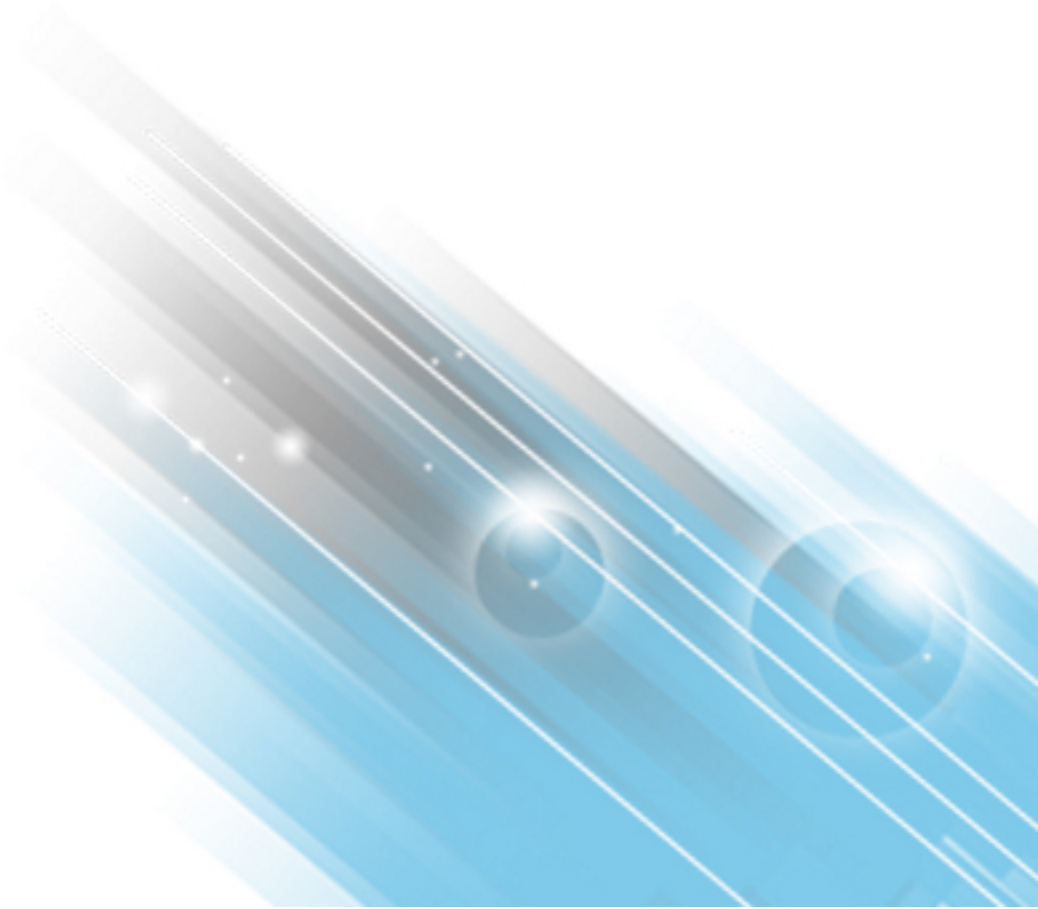
HP UFT Web Add-in Extensibility

Software Version: 12.50
Windows ® operating systems

Developer Guide



Document Release Date: June 2015
Software Release Date: June 2015

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 HP UFT Web Add-in Extensibility

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

Extensibility can also be used to enable HP Sprinter, HP's solution for efficient and effective manual testing, to learn Web objects that are not supported out-of-the-box. For more information about Sprinter, see the *HP Sprinter User Guide*.

This chapter includes:

About the UFT Web Add-in Extensibility SDK

Installing Extensibility Accelerator for HP Functional Testing also installs the UFT Web Add-in Extensibility SDK, which provides the following:

- An API that enables you to extend the UFT Web Add-in to support custom Web controls.
- The Web Add-in Extensibility Help, which includes the following:
  - A developer guide, including a step-by-step tutorial in which you develop support for a sample custom control.
  - API References.
  - A Toolkit Configuration Schema Help.
  - The UFT Test Object Schema Help.

  The Help is available from **Start > All Programs > HP Software > HP Unified Functional Testing > Extensibility > Documentation**
- A printer-friendly Adobe portable document format (PDF) version of the developer guide (available in the **<Extensibility Accelerator installation>\help** folder).
- Web Add-in Extensibility toolkit support sets that extend UFT GUI testing support for the following Web 2.0 environments:
  - ASP.NET AJAX control toolkit
  - Google Web Toolkit

- Yahoo User Interface
- Dojo

- A sample Web toolkit that includes controls named **Book** and **UsedBooksTable** (if you install Extensibility Accelerator).

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

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

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

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

About the UFT Web Add-in Extensibility Developer Guide

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

This guide assumes you are familiar with UFT functionality, and should be used together with the following documents, provided in the Web Add-in Extensibility Help (**Start > All Programs > HP Software > HP Unified Functional Testing > Extensibility > Documentation > Web Add-in Extensibility Help**):

- *API References*
- *Toolkit Configuration Schema Help*
- *Test Object Schema Help*

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

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

Note:

The information, examples, and screen captures in this guide focus specifically on working with UFT GUI tests. However, all toolkit support sets developed using the Web Add-in Extensibility can also be used to extend Sprinter's capabilities to support Web controls that are not supported out-of-the-box. All references to UFT in this guide apply to both UFT and Sprinter.

Additionally, much of the information in this guide applies equally to components.

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

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

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

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

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

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

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

Resource	Description
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

Part 1: Working with Web Add-in Extensibility

Chapter 1: Introducing UFT Web Add-in Extensibility

UFT Web Add-in Extensibility enables you to provide high-level support for third-party and custom Web controls that are not supported out-of-the-box by the UFT Web Add-in.

This chapter includes:

About UFT Web Add-in Extensibility

The UFT Web Add-in provides built-in support for a number of commonly used Web controls. You use UFT Web Add-in Extensibility to extend that support and enable UFT to recognize additional Web controls. You can also use Web Add-in Extensibility to teach UFT to recognize a group of controls or Web elements as one object. For example, if a search button and an edit box comprise a search box, and you want UFT to see them as one functional unit.

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.

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.

Alternatively, consider a table on your Web page that is made up of multiple appended html table elements. You can define rules that help UFT recognize this aggregate table as one object, and provide relevant test object methods. For example, methods that receive a row number relative to the aggregate table and know which html table actually contains that row.

By creating support for a Web control using Web Add-in Extensibility, you can direct UFT to recognize the control or group of controls as belonging to a specific test object class, and you can specify the behavior of the test object. You can also extend the list of available test object classes that UFT is able to recognize. This enables you to create tests that fully support the specific behavior of your custom Web controls.

Extensibility Accelerator

An increasing number of Web applications are making use of Web 2.0-based toolkits, such as ASP.NET AJAX, Dojo, YahooUI, GWT, and JQueryUI to add dynamic and interactive content to their sites. The controls in these toolkits are complex and require sophisticated and flexible testing capabilities.

UFT Web Add-in Extensibility enables you to extend the Web Add-in to customize how UFT recognizes and interacts with different types of controls. Until now, using Web Add-in Extensibility consisted of manually developing and maintaining toolkit support sets.

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

The Extensibility Accelerator for HP Functional Testing 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.

Identifying the Building Blocks of Web Add-in Extensibility

The sections below describe the main elements that comprise UFT object support. These elements are the building blocks of Web Add-in Extensibility. By extending the existing support of one or more of these elements, you can develop the support you need to create meaningful and maintainable tests.

Test Object Classes

In UFT, every object in an application is represented by a test object of a specific test object class. The test object class determines the list of identification properties and test object methods available in UFT for this test object. The icon used to represent the test object in UFT, for example in the Keyword View and Object Repository, is also determined by the test object class.

Test Object Names

When UFT learns an object, it creates a unique name for each test object on the page. A descriptive test object name enables you distinguish between test objects of the same class and makes it easier to identify them in your object repository and in tests.

By default, a test object is given the name of its test object class (appended with an index if there is more than one test object of the same class on the page). In many cases, this is not the ideal name for the custom control.

The test object name needs to be meaningful to the UFT user, preferably using terminology that is relevant to your toolkit. UFT displays this name in the Keyword View, in the Editor, and in the object repository.

Test Object Identification Properties

The test object class used to represent the Web control determines the list of identification properties available for the test object. It also determines which of these identification properties are used to uniquely identify the control, which identification properties are available for checkpoints and output values (in the Checkpoint Properties and Output Value Properties dialog boxes), and which are selected by default for checkpoints. However, the actual values of the identification properties are derived from the Web control. Therefore, several Web controls that are represented by test objects from the same test object class might have different definitions for the same identification property.

Test Object Methods

The test object class used to represent the Web control determines the list of test object methods for a test object. However, the same test object method might operate differently for different Web controls represented by test objects from the same test object class. This happens because depending on the specific type of Web control, UFT may have to perform the test object method differently.

Recording Events

One way to create UFT GUI tests is by recording user operations on the application. When you start a recording session, UFT listens for events that occur on objects in the application and writes corresponding test steps. The test object class used to represent a Web control determines which events UFT can listen for on the Web control and what test step to record for each event that occurs.

Deciding When to Use Web Add-in Extensibility

The UFT Web Add-in provides a certain level of support for most Web controls, but ignores controls defined as DIV or SPAN elements. Before you extend support for a custom Web control, analyze it from a UFT perspective to view the extent of this support and to decide which elements of support you need to modify.

When you analyze the custom Web control, use the Object Spy, Keyword View, Editor, and the Record option. Make sure you examine each of the elements described in " Identifying the Building Blocks of Web Add-in Extensibility".

If you are not satisfied with the existing object identification or behavior, your Web control is a candidate for Web Add-in Extensibility, as illustrated in the following situations:

- UFT might recognize the control using a test object class that does not fit your needs. You can use Web Add-in Extensibility to instruct UFT to identify the custom control as belonging to a new test object class that you create.
- The test object class that UFT uses for the control might be satisfactory, but you would like to

customize the behavior of certain test object methods or identification properties. You can use Web Add-in Extensibility to create a new test object class that extends the one UFT uses, override the implementation of these properties and methods with your own custom implementation, and instruct UFT to use the new test object class.

- You might find that the test object names UFT generates for all objects of a certain control type are identical (except for a unique counter) or that the name used for the test object does not clearly indicate the control it represents. You can use Web Add-in Extensibility to create a new test object class that extends the one UFT uses, modify how UFT names test objects of your new class, and instruct UFT to use the new test object class.
- UFT might identify individual sub-controls within your custom control, but not properly identify your main control. For example, if your main custom control is a digital clock with edit boxes containing the hour and minute digits, you might want changes in the time to be recognized as **SetTime** operations on the clock control and not as **Set** operations on the edit boxes. You can use Web Add-in Extensibility to modify how events that occur on child controls are treated.
- During a record session, when you perform operations or trigger events on your control, UFT might not record a step at all, or it might record steps that are not specific to the control's behavior. Alternatively, UFT might record many steps for an event that should be considered a single operation, or it might record a step when no step should be recorded.

  You can configure the events you want to record for each type of existing Web object by modifying the Web event configuration. For more information, see the section on configuring Web event recording in the *HP Unified Functional Testing Add-ins Guide*.

  If Web event configuration does not sufficiently enable you to customize recording, for example, if you want to modify the steps that UFT records when certain events occur, you can use Web Add-in Extensibility.

Analyzing the Default UFT Support and Extensibility Options for a Sample Custom Control

The following example illustrates how you can use Web Add-in Extensibility to improve the UFT support of a custom control.

The Book control shown below represents a book sold on the Internet. This control is not specifically supported on UFT.

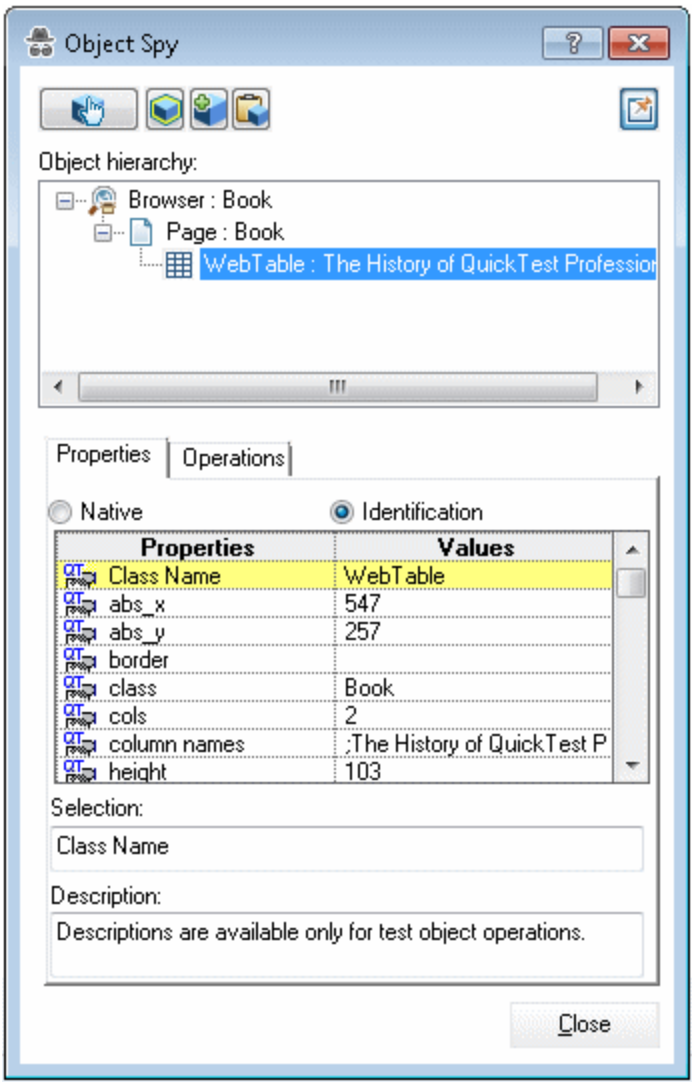
This control contains information including the title of the book, its authors, the price for a new copy of the book, and the lowest price for which a used copy can be purchased.

Clicking on the title of the book opens a page with more details about the book. Clicking on an author name opens a page with a list of books by the same author. Clicking on **Used** opens a UsedBooks page, listing all of the available used copies of the book, and their prices.

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$</strong>   <a
href=".\UsedBooks.htm">Used:</a> from <strong>29.99$</strong>
        </td>
    </tr>
</table>
```

Therefore, if you point to this control using the Object Spy, UFT recognizes it as a WebTable object named according to the title of the book. The icon used for the test object is the standard WebTable class icon.



If you record on the Book control without implementing support for it, the Keyword View looks like this:

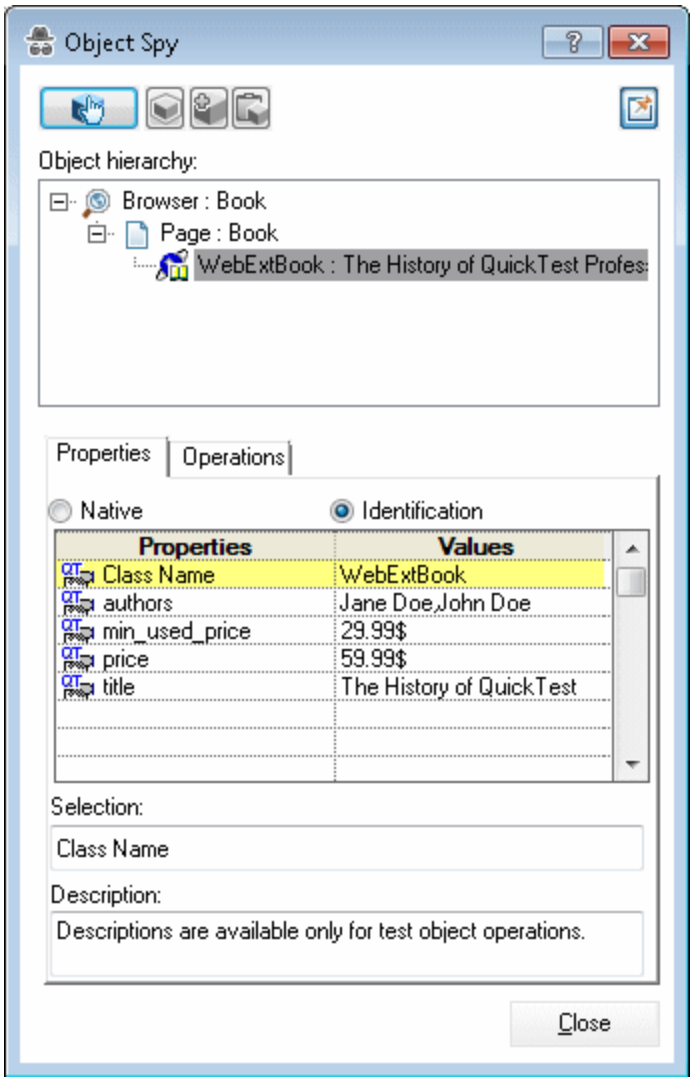
Item	Operation	Value	Documentation
Action1			
Book			
Book			
The History of Quick...	Click		Click the "The History of QuickTest" link.
Book	Click		Click the "Book" image.
Jane Doe	Click		Click the "Jane Doe" link.
Used:	Click		Click the "Used:" link.

In the Editor, the recorded test looks like this:

```
Browser("Book").Page("Book").Link("The History of QuickTest").Click
Browser("Book").Page("Book").Image("Book").Click
Browser("Book").Page("Book").Link("Jane Doe").Click
Browser("Book").Page("Book").Link("Used:").Click
```

Note that 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.

If you use Web Add-in Extensibility to support the Book control, the result is more meaningful. UFT recognizes the control as a **WebExtBook** test object (still named according to the book title) and uses a different icon. The identification properties include relevant information, such as **authors** and **min_used_price**, which provide the names of all the book's authors and the lowest price for which a used copy can be purchased.



When you are ready to create a test on the control, the **Select**, **GoToAuthorPage**, and **GoToUsedBookPage** methods are supported. These methods can be recorded or you can select them manually in the **Operation** column of the Keyword View. When recording a test, both clicking on the book's image and clicking its title result in a **Select** step being recorded.

You can also create a checkpoint to check the value of identification properties, for example, **authors** (that provides a string comprised of all the books authors).

In the Keyword View, a test created by recording the same user operations as the test shown above looks like this:

Item	Operation	Value	Documentation
Action1			
Book			
Book			
The History of QuickTest	Select		Select the "The History of QuickTest" book.
The History of QuickTest	Select		Select the "The History of QuickTest" book.
The History of QuickTest	GoToAuthorPage	"Jane Doe"	Open the Web page for "Jane Doe".
The History of QuickTest	GoToUsedBooksPage		Open the "The History of QuickTest" used books page.

In the Editor, the test looks like this:

```
Browser("Book").Page("Book").WebExtBook("The History of QuickTest").Check CheckPoint("The History of QuickTest")
Browser("Book").Page("Book").WebExtBook("The History of QuickTest").Select
Browser("Book").Page("Book").WebExtBook("The History of QuickTest").GoToAuthorPage "Jane Doe"
Browser("Book").Page("Book").WebExtBook("The History of QuickTest").GoToUsedBooksPage
```

This test is more meaningful and relevant for the Book control's functionality.

Understanding How to Implement Web Add-in Extensibility

Using Web Add-in Extensibility, you can implement full support for all UFT features for your controls. You can implement Web Add-in Extensibility support for a set of controls (also referred to as a **toolkit** or **custom toolkit**) by developing a **toolkit support set**.

Implementing Web Add-in Extensibility consists of the following stages:

1. **Planning the Toolkit Support Set.**
   - Determine the set of Web controls that comprise your custom toolkit.
   - Define the test object model by determining which test objects and operations you want to support based on the controls and business processes supported by your toolkit.

2. **Developing the Toolkit Support Set.**

   A Web Add-in Extensibility toolkit support set is comprised of the following files:

   - One **test object configuration** file, which describes the test object model that you want UFT to use for your toolkit.
   - One **toolkit configuration** file, which describes which test object class represents each control in the toolkit and how UFT interacts with each control.
   - One or more files containing JavaScript functions that UFT can call to perform operations on the custom controls.

3. **Deploying and Testing the Toolkit Support Set.**

   To deploy your toolkit support set and enable UFT to support your controls, you need to copy the files you created to specific locations within the UFT installation folder.

   After you deploy the toolkit support set, when UFT opens, it displays your toolkit name as a child node under the Web Add-in node in the Add-in Manager. If you select the check box for your toolkit, UFT supports the controls in this toolkit using the toolkit support set that you developed.

4. **Enhancing the Toolkit Support Set.**

   After you have created and tested basic Web Add-in Extensibility support for your controls you can enhance your toolkit support set by using some of the more complex options to fine tune your support.

For more information on each of these stages, see:

- "Planning UFT Support for Your Toolkit" on page 23
- "Developing Support for Your Toolkit" on page 30
- "Deploying the Toolkit Support Set" on page 72

When you develop a Web Add-in Extensibility toolkit support set, you can start by creating a simple and basic support set and deploying it to UFT. This enables UFT to recognize your controls correctly and enables the UFT user to create and run tests on the controls. You can then enhance your support to enable more complex capabilities, such as filtering the child objects learned with a control and more advanced handling of events when recording a test.

You can learn how to develop a toolkit support set hands-on, by performing the lessons in "Tutorial: Learning to Create Web Custom Toolkit Support" on page 78.

Web Add-in Extensibility Samples

The Web 2.0 Feature Pack for HP Functional Testing provides a number of completed toolkit support sets from which you can learn more about Web Add-in Extensibility.

- Installing the Web 2.0 Toolkit Support for Unified Functional Testing from this feature pack on a UFT computer installs child add-ins under the Web Add-in that support some public Web 2.0 toolkits: ASP.NET AJAX, GWT, YahooUI, and Dojo. These add-ins are toolkit support sets that were developed using Web Add-in Extensibility. The files that comprise these toolkit support sets are available in

<UFT installation folder>\dat\Extensibility\Web folder and in the **Toolkits** subfolder within this folder.

- Installing Extensibility Accelerator installs sample Web Add-in Extensibility projects that contain the Web 2.0 toolkit support sets mentioned above.

  The sample projects are installed in the **%ALLUSERSPROFILE%\Documents\ExtAccTool\Samples** folder, and are also accessible from the Extensibility Accelerator Start Page. You can open these projects in Extensibility Accelerator 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.

- Installing Extensibility Accelerator also installs a sample Book application, and the Web Add-in Extensibility project containing the toolkit support set for this application.

  - The application is installed in **%ALLUSERSPROFILE%\Documents\ExtAccTool\Samples\WebExtSample\Application\Book.htm**
  - The Web Add-in Extensibility project containing the toolkit support set for the Book application is installed in **%ALLUSERSPROFILE%\Documents\ExtAccTool\Samples\WebExtSample**.

  In "Tutorial: Learning to Create Web Custom Toolkit Support" on page 78, you create a similar toolkit support set for the Book application and deploy it to UFT (without using Extensibility Accelerator).

  The *HP UFT Extensibility Accelerator for HP Functional Testing User Guide* includes a tutorial in which you create support for this control using Extensibility Accelerator.

Chapter 2: Planning UFT Support for Your Toolkit

Before you begin to create support for a custom toolkit, you must carefully plan the support. Detailed planning of how you want UFT to recognize the custom controls enables you to correctly build the fundamental elements of the custom toolkit support.

This chapter includes:

Note: This chapter assumes familiarity with the concepts presented in "Introducing UFT Web Add-in Extensibility" on page 12.

About Planning UFT Support for Your Toolkit

Extending the UFT Web Add-in's support to recognize custom controls is a process that requires detailed planning. To assist you with this, the sections in this chapter include sets of questions related to the implementation of support for your custom toolkit and its controls. When you create your toolkit support set, you implement it based on the answers you provide to these questions.

The first step is determining general information related to your custom toolkit, after which you will define the specific information related to each control you want to support.

Preparing to Create Support for a Custom Toolkit

Before you begin planning support for custom Web controls, make sure you have full access to the controls and understand their behavior. You must have an application or Web page in which you can view the controls in action, and view the source that implements them.

You do not need to modify any of a 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 the control comprises, what HTML properties it has, the events for which you can listen, and so on.

Determining the Toolkit Related Information

When you plan your toolkit support set, begin by deciding the general toolkit related information:

- Provide a unique name for the toolkit or environment for which you are creating support.

  UFT displays the name of your environment in all of the dialog boxes that display lists of add-ins or supported environments. For example, when UFT opens, it displays the name of your environment as a child of the Web Add-in in the Add-in Manager dialog box and the UFT user can specify whether to load support for that environment.

- Decide which controls this toolkit support set will support.
- Decide what files will contain the JavaScript functions that you write for the toolkit support set.
  - You can specify one default file for the JavaScript functions that implement support for the different UFT functionalities and the different test object classes. In addition, you can define separate files for your implementation functions for the different functionalities and test object classes.
  - You can specify one file that contains common JavaScript functions that you call from within others.
- Decide whether to use one JavaScript function for the whole toolkit to match test object classes to the custom controls. For more information, " Teaching UFT to Identify the Test Object Class to Use for a Custom Web Control" on page 49.

When you design the toolkit support set, you specify this information in the toolkit configuration file. For more information, see "Understanding the Toolkit Configuration File" on page 41.

Determining the Support Information for Each Custom Control Type

When planning custom support for a specific type of control, carefully consider how you want UFT to recognize controls of this type—what type of test object you want to represent the controls in UFT GUI tests, which identification properties and test object methods you want to use, and so on. Make these decisions based on the business processes that might be tested using this type of control and operations that users are expected to perform on these controls.

You can run an application containing the custom control and analyze the control from a UFT perspective using the Object Spy, the Keyword View, and the Record option. This enables you to see how UFT recognizes the control without custom support, and helps you to determine what you want to change.

Note: Web Add-in Extensibility can be used to create support for Web controls within Web pages and frames. You cannot develop custom support for Web pages or frames themselves.

To view an example of analyzing a custom control using UFT, see "Analyzing the Default UFT Support and Extensibility Options for a Sample Custom Control" on page 16.

This section also includes:

- "Understanding the Web Add-in Extensibility Planning Checklist" below
- "Web Add-in Extensibility Planning Checklist" on page 27

Understanding the Web Add-in Extensibility Planning Checklist

When you plan the support for a specific type of control, you must ask yourself a series of questions. These are explained below and are available in an abbreviated, printable checklist on page 27.

1. Make sure you have access to an application that runs the custom control on a computer with UFT installed.
2. Is there an existing Web test object class which can be extended to represent the custom control? If so, which one? If not, your new test object class needs to extend the WebElement class.

3. If the new test object class extends a base test object class other than WebElement, does the control contain an element of the type normally represented by the base test object class (also referred to as a base element)?
   - If not, you will need to implement test object methods inherited from the base class that not supported by WebElement. In addition, you will need to design a method that returns values for all of the test class' identification properties that are not supported by WebElement.
   - If the control includes a base element, is it the root Web element of the control?
     - If it is the root element, UFT will use its internal implementation for the inherited test object methods and identification properties that you do not override.
     - If the base element is not the root element, you need to implement a JavaScript function that returns the base element.
4. Define the details for the new test object class that will represent the custom control in UFT GUI tests.

   When you design the toolkit support set, you specify this information in the test object configuration file. For more information, see "Understanding the Test Object Configuration File" on page 33.

   a. Specify the test object class name.
   b. Do you want the new test object class to belong to a different generic type than the one to which the base class belongs? If so, specify the generic type. (For example, if your new test object class extends WebElement (whose generic type is **object**), but you would like UFT to group this test object class with the **edit** test object classes.)
   c. Do you want UFT to use a different icon for the new test object?

      If so, make sure the icon file is available in an uncompressed **.ico** format. Recommended location: **<UFT installation folder>\dat\Extensibility\<UFT add-in name>\Toolkits\<Environment name>\Res**.
   d. Specify one or more identification properties that can be used to uniquely identify the control (in addition to the test object class).
   e. Specify the default test object method to be displayed in the Keyword View and Step Generator when a step is generated for a test object of this class.
   f. Do you want to provide a Help file, which UFT will open when **F1** is pressed for test objects of this class in the Keyword View or Editor?

      If so, make sure that the Help file is available in **.chm** format. Recommended location: **<UFT installation folder>\dat\Extensibility\<UFT add-in name>\Toolkits\<Environment name>\Help**.

5. Decide how to design the process of identifying the test object class to use for this control.
   - Do you want to limit the process to HTML elements with specific HTML tags? If so, which? Make sure to include all HTML tags that can be relevant for your custom control.
   - Which HTML properties will you use to determine what test object class represents controls of this type?
   - Do you need to create different identification rules to use when the control runs on different browsers?

   For more information, see " Teaching UFT to Identify the Test Object Class to Use for a Custom Web Control" on page 49.
6. Specify the basis for naming the test object that represents the control.
7. What identification properties do you want to support? Which properties should be displayed in the Checkpoint Properties and Output Value Properties dialog boxes in UFT, and which should be selected by default in this dialog box? Which identification properties can be used for Smart Identification?
8. What test object methods do you want to support? Specify the method argument types and names, and whether the method returns a value in addition to the return code.

   Optionally, specify the location of a Help file, which UFT will open when **F1** is pressed in the Keyword View or Editor or the Operation Help button is clicked in the Step Generator for a test object method.
9. Do you want to dynamically provide a list of possible values for any test object method arguments? Which?
10. Which types of children should UFT learn with the control?
11. Should the Object Spy display test objects of this class?
12. Do you want to provide support for creating UFT GUI tests by recording?

    If so, list the events you want to record on the custom control during a UFT recording session.
13. Determine what parts of the support need to be designed in the toolkit configuration file and what parts need JavaScript functions.

Web Add-in Extensibility Planning Checklist

Use this checklist to plan the support for your custom control.

☑	Custom Control Support Planning Checklist	Specify in Toolkit XML?	Support by JavaScript function?
❑	The sources for this custom control are located in:	n/a	n/a
❑	Specify the Web test object base class that the new test object class extends: (Default—	n/a	n/a

☑	**Custom Control Support Planning Checklist**	**Specify in Toolkit XML?**	**Support by JavaScript function?**
	WebElement)		
❑	Is the base test object class WebElement? **Yes/No** If No, is there a base element (an element that matches the base test object class)? **Yes/No** If there is a base element, do you need a JavaScript function to return it? **Yes/No**	**Yes/No**	**Yes/No**
❑	Specify the New Web test object class details: • Test object class name: • Generic type (optional): • Icon file location (optional): • Identification properties for description: • Default test object method: • Help file location:	n/a	n/a
❑	Specify the basis for identifying the test object class to use for the control (consider different browsers):	**Yes/No**	**Yes/No**
❑	Specify the basis for naming the test object:	n/a	**Yes**
❑	List the identification properties to support. Mark which should be available for checkpoints and output values (and which should be selected by default in checkpoints) and which (if any) should be used for Smart Identification:	**Yes/No**	**Yes/No**
❑	List the test object methods to support (if required, include arguments, return values, Help file location and Help ID):	**Yes/No**	**Yes/No**
❑	Provide a dynamic list of values for any test object method arguments? **Yes/No** (default) If so, list the arguments:	n/a	**Yes/No**
❑	Specify the types of children that UFT should learn with the control:	**Yes/No**	**Yes/No**
❑	Display test objects of this class in the Object Spy? **Yes** (default)**/No**	**Yes/No**	n/a
❑	Provide support for recording? **Yes/No**	**Yes/No**	**Yes/No**

☑	Custom Control Support Planning Checklist	Specify in Toolkit XML?	Support by JavaScript function?
	If so, list the events that should trigger recording:		

Where Do You Go from Here?

After you finish planning the custom toolkit support, you create the toolkit support set to support the custom toolkit as per your plan. "Developing Support for Your Toolkit" on page 30 explains how to develop the toolkit support set.

Chapter 3: Developing Support for Your Toolkit

This chapter explains how to create support for a custom Web toolkit. It explains what files you have to create for the toolkit support set, the structure and content of these files, and how to use them to support the different UFT capabilities for your environment.

For information on where the toolkit support set files should be stored to activate the support you design, see "Deploying the Toolkit Support Set" on page 72.

This chapter includes:

About Custom Toolkit Support

You implement Web Add-in Extensibility by creating a **toolkit support set** for each Web toolkit you want to support. 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.

This chapter describes the different files, definitions, and functions that you must include in a custom support set. For more information, see the *Toolkit Configuration Schema Help*, the *API Reference*, the *JavaScript Function Reference*, and the *UFT Test Object Schema Help* (available in the UFT Web Add-in Extensibility Help).

Extensibility Accelerator provides sample projects containing toolkit support sets that extend UFT's support for some public Web 2.0 toolkits. These are the support sets for ASPAjax, GWT, YahooUI, and Dojo, which are installed on UFT when you install the Web 2.0 Toolkit Support for Unified Functional Testing. The samples also include Help files for the test object models that UFT uses to support testing controls from these toolkits.

The sample projects are installed under **%ALLUSERSPROFILE%\Documents\ ExtAccTool\Samples**. You can open the projects in Extensibility Accelerator and browse through the toolkit support set files to see examples of how Web Add-in Extensibility can be implemented. Some of the examples in this chapter are taken from these files.

Note: Before you actually begin to create a toolkit support set, you must plan it carefully. For more information, see "Planning UFT Support for Your Toolkit" on page 23.

Developing Browser-Independent Support

Many Web controls are implemented differently and operate differently on different browsers, because of the way different DOM properties are implemented on the different browsers.

To enable your custom support to work for Microsoft Internet Explorer, Google Chrome, and Mozilla Firefox, you must address the following:

- If the control's DOM structure is different in different browsers, UFT may need to use different logic to identify which test object class to use for the control when learning objects or running steps on different browsers. For information on how to define browser-specific identification conditions, see "Using the Conditions Elements" on page 51.
- When you write your JavaScript functions, use jQuery function calls to create code that can run smoothly on different types and versions of browsers. For more information see "Creating Browser-Independent Support" on page 44.
- Web Add-in Extensibility provides the **GetBrowserType** and **GetBrowserVersion** utility methods, which you can use to determine the type and version of the browser that is currently running a control. This enables your JavaScript code to treat a control differently, depending on the browser that is currently running the control.

Creating a Custom Toolkit Support Set

A Web Add-in Extensibility toolkit support set is comprised of the following files:

- One **test object configuration** file, which describes the test object model for your toolkit: The test object classes that UFT should use to represent controls in your toolkit, and the identification properties and test object methods that need to be supported for the test objects. For more information, see "Understanding the Test Object Configuration File" on the next page.

  Note: UFT loads of all the test object class definitions (from all of the test object configuration files) when it opens, regardless of the custom toolkit for which they were created. This enables you to use the same test object class definitions when supporting different custom toolkits.

- One **toolkit configuration** file, which describes which test object class represents each control in the toolkit and how UFT interacts with each control. For more information, see "Understanding the Toolkit Configuration File" on page 41.
- One or more files containing JavaScript functions that UFT can call to retrieve information from or perform operations on the custom controls.
- Optionally, icon files that contain icons used in UFT to represent the test object classes that you define, and Help files that describe these test object classes and their methods and properties.

"Deploying the Toolkit Support Set" on page 72 describes the names of the different files required for a toolkit support set, and the folder structure in which they are stored.

To create a custom toolkit support set:

1. Choose a unique name to represent the toolkit or environment for which you are creating support.

   You use the custom toolkit name to compose the name of the toolkit folder and the toolkit configuration file. The name must start with a letter and can contain only alphanumeric characters and underscores.

Providing unique toolkit names enables a single UFT installation to support numerous custom toolkit support sets simultaneously. For this reason, a name such as `MyToolkit` is not recommended.

2. Create a folder for your toolkit support set.

   You can choose any convenient name and location for this folder.

3. Create the following folder structure:
   - **<toolkit support set folder>\Toolkits\<toolkit environment name>\JavaScript**
   - **<toolkit support set folder>\Toolkits\<toolkit environment name>\Res**
   - **<toolkit support set folder>\Toolkits\<toolkit environment name>\Help**
4. In the toolkit support set folder, create a file named **<toolkit environment name>TestObjects.xml**. This is the test object configuration file.
5. In the **<toolkit support set folder>\Toolkits\<toolkit environment name>** folder, create a file named **<toolkit environment name>.xml**. This is the toolkit configuration file.
6. In the **<toolkit support set folder>\Toolkits\<toolkit environment name>\JavaScript** folder, create one or more files that will contain the JavaScript functions you design.

   As a best practice, create one JavaScript file for each test object class. In addition, you can create one JavaScript file that contains JavaScript functions called from JavaScript functions in the other files.
7. You can use the **<toolkit support set folder>\Toolkits\<toolkit environment name>\Res** folder to store any icons that you provide to represent your test object classes in UFT.
8. You can use the **<toolkit support set folder>\Toolkits\<toolkit environment name>\Help** folder to store any Help (**.chm**) files that describe the test objects in your environment.

Understanding the Test Object Configuration File

The first stage of developing support for a custom toolkit is to introduce the test object model that you want UFT to use to test your applications and controls. To do this, you define the test object model in the test object configuration XML file. You need to create a test object class for every type of custom control for which you want to extend or modify UFT support.

In a test object configuration XML, you define the test object classes (for example, the test object methods they support, their identification properties, and so on).

You create a **ClassInfo** element for each test object class that you want to define. In addition, you define the name of the environment or custom toolkit for which the test object classes are intended (in the **PackageName** attribute of the **TypeInformation** element), and the UFT add-in which these test object classes extend (in the **AddinName** attribute of the **TypeInformation** element).

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

For more information, see "How UFT Loads the Test Object Configuration XML" on page 37.

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

Class Name and Base Class

The name of the new test object class and its attributes, including the base class—the test object class that the new test object class extends. A new test object class extends an existing WebUFT test object class, directly or indirectly. The base class may be a class delivered with UFT or a class defined using Web Add-in Extensibility.

By default, the base class is WebElement.

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

Note:

- A test object class inherits the base class' test object operations (methods and properties), generic type, default operation, and icon. Identification properties are not inherited.
- If you create test object classes that extend test object classes defined in another toolkit support set, you create a dependency between the two toolkit support sets. Whenever you select to load the extending toolkit support set in the UFT Add-in Manager, you must also select to load the toolkit support set that it extends.

Generic Type

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

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

Test Object Operations

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

- The arguments, including the argument type (for example, `String` or `Integer`), direction (`In` or `Out`), whether the argument is mandatory, and, if not, its default value.
- Whether a dynamic list of possible values for this argument can be retrieved from the supported control and displayed in the Keyword View, Editor, and Step Generator.
- The operation description (shown in the Object Spy and as a tooltip in the Keyword View and Step Generator).
- The Documentation string (shown in the **Documentation** column of the Keyword View and in the Step Generator).
- The return value type.
- A context-sensitive Help topic to open when **F1** is pressed for the test object operation in the Keyword View or Editor, or when the **Operation Help** button is clicked for the operation in the Step Generator. The definition includes the Help file path and the relevant Help ID within the file.

Default Operation

The test object operation that is selected by default in the Keyword View and Step Generator when a step is generated for an object of this class.

Identification Properties

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

- The identification properties that are used for the object description.

  Tip: Include the **html tag** property in the object description to facilitate faster object identification in UFT.

- The identification properties that are used for **smart identification**. (This information is relevant only if smart identification is enabled for the test object class. To enable smart identification, use the Object Identification dialog box in UFT.)
- The identification properties that are available for use in checkpoints and output values.
- The identification properties that are selected by default for checkpoints (in the UFT Checkpoint Properties dialog box).

Icon File

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

Help File

A context-sensitive Help topic to open when **F1** is pressed for the test object in the Keyword View or

Editor. The definition includes the **.chm** Help file path and the relevant Help ID within the file.

For information on the structure and syntax of a test object configuration XML, see *UFT Test Object Schema Help (*available in the UFT Web Add-in Extensibility Help).

Sample Test Object Configuration XML

The following example shows the definition of the **WebExtUsedBooks** test object definition, which is part of the **WebExtSample** toolkit used for the tutorial section of this guide:

```
<ClassInfo BaseClassInfoName="WebTable" GenericTypeID="Table"
     DefaultOperationName="SelectBook" Name="WebExtUsedBooks">
    <IconInfo
        IconFile="INSTALLDIR\Dat\Extensibility\Web\Toolkits\
    WebExtSample\Res\WebBookList.ico"/>
    <TypeInfo>
        <Operation ExposureLevel="CommonUsed" Name="SelectBook"
PropertyType="Method">
            <Description>
                Selects the radio button for the specified book and clicks
Select.
            </Description>
            <Documentation>
                <![CDATA[Select the radio button for the book with index %a1 and
click Select.]]>
            </Documentation>
            <Argument Name="BookIndex" IsMandatory="true" Direction="In">
                <Type VariantType="Integer"/>
            </Argument>
        </Operation>
    </TypeInfo>
    <IdentificationProperties>
        <IdentificationProperty ForDefaultVerification="true"
ForVerification="true" ForDescription="true" Name="title"/>
    </IdentificationProperties>
</ClassInfo>
```

This example shows that the **WebExtUsedBooks** test object class extends the **WebTable** test object class. The **WebExtUsedBooks** test object class uses the **WebBookList.ico** icon file, its default test object method is **SelectBook** (which has one Integer mandatory input parameter: **BookIndex**), and it has one additional identification property: **title**.

One identification property is defined for the **WebExtUsedBooks** test object class: **title.** This identification property is used in the object description, available for checkpoints and output values, and selected by default in the Checkpoint Properties dialog box in UFT.

See also:

- "How UFT Loads the Test Object Configuration XML" below
- "Extending an Existing Test Object Class" on the next page
- "Providing a Help File for Customized Test Object Classes" on page 40

How UFT Loads the Test Object Configuration XML

Each time you run UFT, it reads all of the test object configuration XMLs and merges the information for each test object class from the different XMLs into one test object class definition. For more information, see "Understanding How UFT Merges Test Object Configuration Files" below.

The following attributes of the **Identification Property** element in the test object configuration file specify information that can be modified in UFT (using the Object Identification dialog box): **AssistivePropertyValue**, **ForAssistive**, **ForBaseSmartID**, **ForDescription**, **ForOptionalSmartID**, and **OptionalSmartIDPropertyValue**. These attributes determine the lists of identification properties used for different purposes in UFT.

Therefore, by default, UFT reads the values of these attributes from the XML only once, to prevent overwriting any changes a user makes using the Object Identification dialog box. In this way, UFT provides persistence for the user defined property lists. For more information, see "Modifying Identification Property Attributes in a Test Object Configuration File" on page 75.

Understanding How UFT Merges Test Object Configuration Files

Each time you open UFT, it reads all of the test object configuration files located in the **<UFT installation folder>\dat\Extensibility\<UFT add-in name>** folders. UFT then merges the information for each test object class from the different files into a single test object class definition, according to the priority of each test object configuration file.

UFT ignores the definitions in a test object configuration file in the following situations:

- The **Load** attribute of the **TypeInformation** element is set to `false`.
- The environment relevant to the test object configuration file is displayed in the Add-in Manager dialog box, and the UFT user selects not to load the environment.

Define the priority of each test object configuration file using the **Priority** attribute of the **TypeInformation** element.

If the priority of a test object configuration file is higher than the existing class definitions, it overrides any existing test object class definitions, including built-in UFT information. For this reason, be aware of any built-in functionality that will be overridden before you change the priority of a test object configuration file.

When multiple test object class definitions exist, UFT must handle any conflicts that arise. The following sections describe the process UFT follows when **ClassInfo**, **ListOfValues**, and **Operation** elements are defined in multiple test object configuration files. All of the **IdentificationProperty** elements for a specific test object class must be defined in only one test object configuration file.

ClassInfo Elements

- If a **ClassInfo** element is defined in a test object configuration file with a priority higher than the existing definition, the information is appended to any existing definition. If a conflict arises between **ClassInfo** definitions in different files, the definition in the file with the higher priority overrides (replaces) the information in the file with the lower priority.
- If a **ClassInfo** element is defined in a test object configuration file with a priority that is equal to or lower than the existing definition, the differing information is appended to the existing definition. If a conflict arises between **ClassInfo** definitions in different files, the definition in the file with the lower priority is ignored.

ListOfValues Elements

- If a conflict arises between **ListOfValues** definitions in different files, the definition in the file with the higher priority overrides (replaces) the information in the file with the lower priority (the definitions are not merged).
- If a **ListOfValues** definition overrides an existing list, the new list is updated for all arguments of type **Enumeration** that are defined for operations of classes in the same test object configuration file.
- If a **ListOfValues** is defined in a configuration file with a lower priority than the existing definition, the lower priority definition is ignored.

Operation Elements

- **Operation** element definitions are either added, ignored, or overridden, depending on the priority of the test object configuration file.
- If an **Operation** element is defined in a test object configuration file with a priority higher than the existing definition, the operation is added to the existing definition for the class. If a conflict arises between **Operation** definitions in different files, the definition in the file with the higher priority overrides (replaces) the definition with the lower priority (the definitions are not merged).

For more information, see the *HP UFT Test Object Schema Help* (available with the Web Add-in Extensibility SDK Help).

Extending an Existing Test Object Class

If there is an existing test object class that provides partial support for your control, but needs some modification, for example, a different naming convention for test objects in the class, or additional or modified test object methods, you create a new test object class to represent the control. When you create the new test object class, you base this test object class on the existing test object class, inheriting all of its test object methods.

You can then extend the functionality of this test object class by defining and implementing additional test object methods and identification properties. In addition, you can override existing test object methods by providing an alternate implementation for them. You define the new or changed methods

and properties in the test object configuration file, and design their implementation using JavaScript functions.

To extend an existing test object class, you define the name of the base test object class in the **ClassInfo\BaseClassInfoName** attribute in the **ClassInfo** element for the new test object class in the test object configuration file. This declares that the new test object class supports all of the test object methods of the base test object class in addition to any that you define for the new test object class.

How to Choose a Base Class

When you choose a test object class to extend consider the following:

- Your test object class inherits the operation definitions from the test object class. Choose a test object class with a set of operations functionally relevant for your control.
- To inherit the implementation for the base class operations, your control must include an element of the type that matches the base test object class.

How to Make Sure That All Methods and Properties Are Implemented

You must ensure that all of the inherited test object methods are implemented and not only declared. One way to do this is to write JavaScript functions to support each inherited test object method. Another, simpler way is to ensure that the control includes an element of the type that matches the base test object class. This element is referred to as the **base element**. UFT can then use its internal implementation for the inherited test object methods that you do not specifically implement for the custom control, communicating with the base element.

In addition, if the control includes a base element, UFT uses the base test object class implementation to retrieve the identification property values when the following conditions are met:

- In the test object configuration file, you defined identification properties for the new test object class with the same names as properties of the base test object class.
- You do not implement a JavaScript function that retrieves the values for those properties.

When You Need to Return a Base Element to UFT

If the base element is the root element of the control that you are supporting, UFT recognizes the base element and uses the base test class implementation for test object methods and identification properties that you do not implement.

If the base element is not the root element of the control, you must write a JavaScript function that returns this base element to UFT, and specify the name and location of the JavaScript function in the toolkit configuration file. You specify this information in the **Control\Settings** element in the toolkit configuration file. For example, if your control is a special kind of table, and is defined as a DIV element (which UFT normally ignores) that contains a table element (which UFT normally represents with a WebTable element), you can create a MyWebTable test object class that extends WebTable and map the DIV element to this the MyWebTable test object class.

To return the base element, you implement a JavaScript function named **get_base_table** in a file named **HPTable.js**. In the toolkit configuration **Settings** element for the MyWebTable Control element, you define the **func_to_get_base_elem** as follows:

```
<Control TestObjectClass=MyWebTable>
    <Settings>
        <variable name="default_imp_file " value="HPTable.js"/>
        <variable name="func_to_get_base_elem" value="get_base_table"/>
    </Settings>
</Control>
```

Providing a Help File for Customized Test Object Classes

As part of the UFT online Help, UFT provides an Object Model Reference for the test object classes that it defines. The reference is intended to help the UFT users use UFT test objects, methods, and properties in their tests. In addition, when F1 is pressed for a test object or test object method in the Keyword View or Editor, or when the Operation Help button is clicked for a test object method in the Step Generator, UFT opens the Object Model Reference to the relevant location.

When you modify test object classes or define new ones, using test object configuration XML definitions, you can provide similar (**.chm**) Help files for the test objects, methods, and properties that you define. Deploy these Help files as part of your toolkit support set and inform the users where they can be found. You can store the Help files in any convenient location. For example, in a **<UFT installation folder>\dat\Extensibility\<UFT Add-in name>\Toolkits\<Environment name>\help** folder. In the test object configuration XML, you define **HelpInfo** elements for test object classes and test object methods, specifying the Help file path and the relevant Help ID within the file.

Caution: The Help file name must be different from the names of the Help files provided in the **<UFT installation folder>\help** folder.

When F1 or the Operation Help button is pressed for a test object class or test object method for which you defined a **HelpInfo** element, UFT opens the context-sensitive Help topic you specified. For inherited test object methods that do not have a **HelpInfo** element, UFT opens the Help provided for the base test object class.

In the Help that you provide for your test object classes, it is helpful to specify the following information for each test object class:

- The base test object class that this test object class extends (include a note explaining that the descriptions for inherited test object methods not covered in this Help can be found in the Help for the base test object class).
- A list of the available test object methods (including the inherited methods).
- Descriptions of the test object methods that you defined, including the method's purpose, syntax, arguments, return value, and any other relevant information.
- A list of the available identification properties and descriptions for those properties.

Understanding the Toolkit Configuration File

To begin developing your toolkit support set, you define a basic toolkit configuration file. You can verify the toolkit configuration file you design against the **<Extensibility Accelerator installation folder>\dat\Toolkit.xsd** file. (This file is also located in the **<UFT installation folder>\dat\Extensibility\Web\Toolkits** folder.)

In a toolkit configuration XML, you must define a **Control** element for each test object class that you plan to use to support controls in your toolkit. Each **Control** element must include a **TestObjectClass** attribute that specifies the name of the test object class to which it applies.

The **Control** elements are contained within one **Controls** element, which represents the toolkit as a whole.

The toolkit configuration file must provide information that enables UFT to identify which test object class to use for each control in the toolkit. This information can be provided at toolkit level or at control level, as described in the following sections.

The toolkit configuration XML can also contain additional information. A brief summary of the possible content of this XML is provided below and more detail on how to design and use the toolkit configuration XML is provided in the subsequent sections of this guide.

The toolkit specific information can include:

- A toolkit description.

  UFT displays the description in the Add-in Manager when a user selects the toolkit support set's environment name in the list of available add-ins. If you are developing this toolkit support set for distribution, include a **Provided by** clause in this description, specifying the relevant person or company.
- 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 priority first).
- An identification function (and optionally the name of the file that contains the function) used to identify the test object class to use for each control in the toolkit.
- The name of the default file from which implementation functions are called if no file is specifically defined for a test object class.
- JavaScript libraries for UFT to inject into the Web page, enabling you to call the library functions from the support functions that you design.

A test object class element can include:

- HTML tags and identification conditions or an identification function (and optionally the name of the file that contains the function) used to identify the custom controls that should be represented by this test object class.
- The name of the default file from which implementation functions are called if no file is specifically defined for a function.

- A function (and optionally the name of the file that contains that function) that returns the element whose test object implements the properties and test object methods inherited from the base class and not implemented for this control.
- The functions (and optionally the name of the file containing those functions) that implement the test object methods of this test object class. UFT calls these functions to perform test steps on the control. If no functions are defined, UFT calls implementation functions with the same name as the test object methods.
- The function (and optionally the name of the file containing the function) that retrieves the identification properties from the control. If no function is defined, UFT calls a function named **get_property_value**.
- Elements that indicate whether to use the default Web Event Configuration for handling events on the control and its children, and can specify functions that UFT should run to perform customized event listening and handling.
- Elements that indicate when to learn the control and its children, and optionally, a function that specifies which children to learn.
- Elements that indicate whether to display test objects of this class in the Object Spy.
- A function (and optionally the name of the file containing the function) that retrieves the list of possible values for a test object method argument from the control. If no function is defined, UFT calls a function named **get_list_of_values**.
- A function (and optionally the name of the file containing the function) that retrieves the state of the supported toolkit, checking whether it is fully loaded on the current Web page.

  Note: When planning the order of the **Control** elements in this file, consider that UFT follows this order when searching for a test object class to match a control. The first matching test object class is used.

For information on the structure and syntax of this XML, see the *Toolkit Configuration Schema Help*, available in the UFT Web Add-in Extensibility Help.

Designing JavaScript Functions for Your Toolkit Support Set

As part of the toolkit support set, you design JavaScript functions that UFT calls. The *UFT Web Add-in Extensibility JavaScript Function Reference* (available with the Web Add-in Extensibility Help) describes the JavaScript functions you implement.

When writing your JavaScript functions, also consider the information in the following sections:

- "Conventions, Utility Objects and Global Functions You Can Use" below
- "Creating Browser-Independent Support" on the next page
- "Global Variable Scope" on page 45
- "Writing Global JavaScript Functions" on page 45
- "Calling Functions from External JavaScript Libraries" on page 45
- "Handling Exceptions" on page 46

Conventions, Utility Objects and Global Functions You Can Use

The Web Add-in Extensibility API provides the following conventions, constants, utility object methods, and global JavaScript functions for you to use in your JavaScript functions:

- **_elem.** A token that represents the Web element that UFT is currently handling. In identification functions, the _elem token represents the HTML element being learned. In all other contexts, it represents the root element of the custom control. These contexts include:
  - Event handlers and event registration functions.
  - Functions for returning base elements, property values, and lists of possible values.
  - Filter functions.

  For example, the following JavaScript function returns the value of the **id** identification property of the current Web element:

```
function get_property_value(prop)
{
    if (prop == "id")
    {
        return _elem.id;
    }
}
```

- **_util**. The utility object whose methods you can use to instruct UFT to perform different operations. For more information, see "Global JavaScript Methods and Utility Methods" on page 46.
- Global JavaScript functions and constants defined in the **common.js** file located in the **<Extensibility Accelerator installation folder>\bin\PackagesToLoad** folder. (This file is also located in the **<UFT installation folder>\dat\Extensibility\Web\Toolkits** folder.) For more information, see "Global JavaScript Methods and Utility Methods" on page 46.
- **window.** Provides access to the Window object that corresponds to the Web browser window you are testing. You can use this object to access objects and functions that reside in the browser namespace and manage the controls for which you are creating support.

  For example, suppose that the Web page contains a global JavaScript function **getTree()** that returns a **Tree** object that manages a specific tree control. This **Tree** object would have **Collapse**, **Expand**,

and **Select** methods. You could write the following JavaScript method to support running a **Collapse** test object method:

```
function Collapse (treeBranch)
{
   var tree = window.getTree(_elem);
   tree.Collapse(treeBranch);
}
```

Creating Browser-Independent Support

To enable the your toolkit support set to work smoothly on different browser types and versions, do one or both of the following in the extensibility code that you develop:

- If you need to treat a control differently in different browsers, call the **GetBrowserType** or **GetBrowserVersion** utility methods to retrieve the relevant browser information. For more information, see "Global JavaScript Methods and Utility Methods" on page 46.
- Use functions from external browser-independent JavaScript function libraries, such as **jQuery**, to create code that can run smoothly on different types and versions of browsers.

  The **jQuery** (version 1.3.2) and **jQuery.simulate** libraries are included with the UFT installation and provide browser-independent functions. You use **jQuery** function calls to retrieve information and perform operations, and **jQuery.simulate** function calls to handle dispatching events.

  For example, if you need to perform a click operation on your control, one line of code that uses **jQuery** would replace a complex block of code that sends the event differently, depending on the current browser.

  Use:

  ```
  $(_elem).simulate("click");
  ```

  Instead of:

  ```
  if ( _util.GetBrowserType()==QtpConstants.FireFox)
  {
      var myEvent = window.document.createEvent("MouseEvents")
      myEvent.initEvent("click", true, true)
      _elem.dispatchEvent(myEvent)
  }
  else
      _elem.click();
  ```

  For information on the **jQuery** JavaScript library and on **jQuery.simulate.js**, see http://jquery.com/ and http://code.google.com/p/jqueryjs/source/browse/trunk/plugins/simulate/jquery.simulate.js?r=6163

For details on using the jQuery libraries and other external JavaScript function libraries, see "Calling Functions from External JavaScript Libraries" below.

- In JavaScript functions that need to run on Mozilla Firefox, you need to explicitly use the **window** object to access any item in the global namespace. For example, `window.document` will correctly access the **document** object in both Internet Explorer and Mozilla Firefox, while `document` will only work on Internet Explorer.
- When developing for Chrome consider the following:
  - If you identify a control using the **<body>** tag, and the **<body>** is a child of an **<iframe>** that has a `blank` or `about:blank` **SRC** value, then UFT can identify the control only in Chrome 37 and later.
  - In Chrome, if a custom control is comprised of two or more sibling controls, you must include both of them in the identification of the object. For example, this may be relevant if your custom control is comprised of an edit box and its adjacent submit button or a check box and its label. For a code sample, see "Example: Using Sibling Controls to Identify an Object" on page 48.
- Even if your code is designed to be browser-independent, each browser works using different logic, so validate your scripts on all browsers.

Global Variable Scope

When you define global variables in your JavaScript function, the variables exist in the scope of the JavaScript engine running in the browser. UFT can only connect to the JavaScript engine if UFT is opened before the browser. Therefore, your global JavaScript variables are accessible until either the browser or UFT is closed.

Writing Global JavaScript Functions

As a best practice, create one JavaScript file for each test object class. However, if you want to call the same JavaScript function from JavaScript functions of multiple test objects, you can store these global functions in a separate file, which will be used as the toolkit support set's common JavaScript file. Specify the path of the common JavaScript file in the **Controls\Settings\Variable** element in the toolkit configuration file. For more information, see the Toolkit Configuration Schema Help, available in the UFT Web Add-in Extensibility Help.

Calling Functions from External JavaScript Libraries

In the JavaScript functions that you design, you can call any JavaScript function included in the Web page that UFT is testing.

Keep in mind that if your toolkit support set is loaded, UFT might call your code on any Web page that opens in a browser. For example, when learning objects on a Web page, UFT might call the identification function that you designed, even if the page does not contain any of your controls, and does not include your function library.

To enable the use of your own JavaScript libraries on pages that do not include them, you can instruct UFT to inject the JavaScript library into every page. To do this, specify the file system path to the library

in the **JSLibrariesToInject\JSLibrary** element in the toolkit configuration file. For more information, see the Toolkit Configuration Schema Help, available in the UFT Web Add-in Extensibility Help.

When UFT is open with your toolkit support set loaded, UFT injects the specified JavaScript libraries into every Web page that opens in a browser.

It is recommended to design your JavaScript library so that it will only be loaded if it is not already included in the Web page.

When you deploy the toolkit support set to UFT, make sure that the JavaScript libraries you reference are located in the paths that you specified in the toolkit configuration file.

Using the jQuery Libraries Provided with UFT

The **jQuery** (version 1.3.2) and **jQuery.simulate** libraries are included with the UFT installation. They are located in: **<UFT installation folder>\bin\JSFiles**

To call the **jQuery** library functions from your JavaScript functions, specify the library locations in your toolkit configuration file:

```
<Controls>
    <JSLibrariesToInject>
        <JSLibrary path="INSTALLDIR\bin\JSFiles\jQuery-1.3.2.js" />
        <JSLibrary path="INSTALLDIR\bin\JSFiles\jQuery.simulate.js" />
    </JSLibrariesToInject>
</Controls>
```

These jQuery libraries are designed so that they are not loaded into the Web page if the page already includes some version of the library.

Handling Exceptions

If the JavaScript function that you design throws an exception, UFT displays a run-time error message and the test step fails. The message that you provide with the exception is added to the test report details for the failed step.

If you want to add a message to the run results that the step failed, but not cause a run-time error, call the **_util.Report** method from your JavaScript function.

Note: If the message you include in the exception is a simple string, the run-time error that UFT displays includes the line of code that threw the exception. If the message uses formatting to include a variable in the string, the run-time error does not include the line of code.

Global JavaScript Methods and Utility Methods

As part of the toolkit support set, you design JavaScript functions that UFT calls. Your JavaScript functions can call global methods and utility methods that UFT provides. The global methods are provided as JavaScript functions in the **common.js** file. The utility methods are exposed by the **_util** object.

The **_util** utility object exposes the following methods that you can call in your JavaScript functions:

- **Alert (***message***).** Opens a modal message box displaying the specified message.
- **GetBrowserType / GetBrowserVersion ().** Returns the name or version of the browser that is currently running the control.
- **LogLine (***text***,** *severity, Id, category***).** Adds an entry to the Microsoft Windows event log (which you can view using the Event Viewer), or for Chrome, adds an entry in the Chrome Console. Use this method to help you analyze the performance of your support set or debug its functionality.

  For more information on using the Event Viewer to debug your toolkit support set, see "Using the Microsoft Windows Event Log" on page 58.
- **Record (***method***,** *arrParams***,** *delay***).** Adds a step to the test and adds a test object to the object repository if it is not already there. Use this method in a JavaScript function that records a step in a test after an event occurs on a control. For more information on developing support for recording, see "Implementing Support for Recording" on page 69.
- **RegisterForEvent (***element***,** *eventName***,** *handler***,** [*HandlerParam*]**).** Registers to listen for a specific event on a specific Web element. Use this method in a JavaScript function that controls listening to events to support recording. You can also invoke it from event handlers. For more information on developing support for recording, see "Implementing Support for Recording" on page 69.
- **Report (***status***,** *method***,** *arrParams***,** *details***).** Adds information about the results of a step to the run results. Use this method in a JavaScript function that performs a step on a control. For more information on developing support for running tests, see "Implementing Support for Test Object Methods" on page 61.
- **Wait (***milliseconds***).** Suspends execution for the number of specified milliseconds.

The most commonly used functions from **common.js** are:

- **getFrameElement.** Returns the frame element that contains the current element.
- **toSafeArray(inArray).** Converts a JavaScript Array object to COM SafeArray.
- **trim(inString).** Returns the string after stripping leading and trailing white spaces.

The **common.js** file also contains variables with preset values that you can use as constants in your code.

The **common.js** file is located in the **<Extensibility Accelerator installation folder>\bin\PackagesToLoad** folder and the **<UFT installation folder>\dat\Extensibility\Web\Toolkits** folder.

For more information on these methods and constants, see the UFT Web Add-in Extensibility API Reference (available with the Web Add-in Extensibility Help).

Example: Using Sibling Controls to Identify an Object

In Chrome, if a custom control is comprised of two or more sibling controls, you must include both of them in the identification of the object.

This example demonstrates this for a custom control comprised of an input and a label:

```
<input type="checkbox" role="hidden-checkbox-input" class="ui-helper-
hidden"/><label for="check" class="ui-toggle-button"
role="button">Toggle</label>
```

In another browser, you could identify your control by specifying only the **input** tag:

SampleToolkit.xml:

```
<Control TestObjectClass="SampleToggleButton">
    ...
    <Identification type="javascript" function="isSampleToggleButton">
        <Browser name="*">
            <HTMLTags>
                <Tag name="input" />
            </HTMLTags>
        </Browser>
    </Identification>
    ...
</Control>
```

SampleToggleButton.js:

```
function isSampleToggleButton ()
{
    return _elem.getAttribute("role") === "hidden-checkbox-input";
}
```

In Chrome, you should identify the control with both the input and label tag.

To do this, you add one more tag to the **SampleToolkit.xml**:

```
<Control TestObjectClass="SampleToggleButton ">
    ...
    <Identification type="javascript" function=" isSampleToggleButton ">
        <Browser name="*">
            <HTMLTags>
                <Tag name="input" />
```

```
                <Tag name="label" />
            </HTMLTags>
        </Browser>
    </Identification>
    ...
</Control>
```

Then update the **SampleToggleButton.js** function accordingly:

```
function isSampleToggleButton ()
{
    if (_elem.getAttribute("role") === "hidden-checkbox-input" && _
elem.tagName.toUpperCase() === "INPUT")
        return true;

    if (_elem.tagName.toUpperCase() === "LABEL" && _elem.getAttribute("role") ==
"button")
    {
        var sibling = _elem.previousSibling;
        if (sibling && sibling.tagName.toUpperCase() === "INPUT" &&
sibling.getAttribute("role") === "hidden-checkbox-input")
        {
            return sibling; // use the sibling input element to replace label to
serve as _elem.
        }
    }

    return false;
}
```

Teaching UFT to Identify the Test Object Class to Use for a Custom Web Control

After you define the test object classes that you want UFT to use to represent your controls, you need to map each type of control to a specific test object class.

This identification can be performed using JavaScript functions or condition elements that check the control's properties and determine the test object class that should represent it. To improve UFT's performance of learning custom controls and running steps on them, define the identification elements in such a way that JavaScript function calls are avoided as much as possible.

You can limit the identification process of custom controls to HTML elements with HTML tags you specify. This can further improve performance, and is more efficient than defining conditions that check the **tagName** property.

You provide information enabling UFT to identify which test object class to use for the different controls in the **Controls\CommonIdentification** element in the toolkit configuration file, or in the **Control\Identification** element. In these elements you can define the following:

- **A set of HTML tags per test object class.** When UFT handles each control, it continues the identification process according to the definitions in the **Identification** element, only for HTML elements with the specified HTML tags.
- **A set of conditions per test object class.** When UFT handles each control, UFT checks the control's HTML properties against the conditions you define in each **Control** element, and locates the first one whose conditions the control meets.

  Note: You can define different conditions and HTML tags for UFT to use when running on different browsers.

- **One JavaScript identification function for the whole toolkit.** When UFT handles each control, UFT calls this function, which checks the control's properties and returns the name of the appropriate test object class.

  Note: If you specify an identification function at the toolkit level (in the **Controls\CommonIdentification** element) and also conditions at the test object class level (in the **Control\Identification** element), UFT checks the conditions before calling the JavaScript function, to avoid unnecessary calls.

- **A JavaScript identification function per test object class.** When UFT handles each control, UFT calls each of these JavaScript functions, in the order in which the test objects are defined in the toolkit configuration file. Each function checks whether the test object class for which it is defined should represent the control. UFT uses the first test object class whose function returns **true**.

  This method of identification should be avoided if possible because it significantly affects performance, as it involves many calls to JavaScript functions for each control.

  Note: If you specify an identification function at the toolkit level (in the **Controls\CommonIdentification** element), UFT does not call any identification functions specified at the test object class level (in the **Control\Identification** element).

- You can also combine the use of conditions and JavaScript functions, defining conditions that limit the calls to the JavaScript function you define, based on the control's properties. For more information, see "Using the Conditions Elements" on the next page.

For more information on writing JavaScript functions for Web Add-in Extensibility, see "Designing JavaScript Functions for Your Toolkit Support Set" on page 42.

For information on the structure and syntax of the identification elements, see the Toolkit Configuration Schema Help, available in the UFT Web Add-in Extensibility Help.

After you teach UFT to identify the test object class to use for the custom control, you can test the basic functionality of your toolkit support set. For more information, see "Testing the Toolkit Support Set During Development" on page 55.

Using the Conditions Elements

You can define **Conditions** elements in the **Control\Identification\Browser** element defined for a test object class. This enables UFT to identify the controls that should be represented by this test object class, based on the control's properties, without calling an identification function.

Alternatively, you can define an identification function (per test object class or for the whole toolkit) and use the conditions to limit the times UFT calls the function. You do this by defining that the identification function be called only in cases when the control's properties meet certain conditions.

You can define different sets of conditions for UFT to use when running on different browsers, or you can define one set of conditions and specify that it is relevant for all browsers. For more information, see the **Browser** element description in the Toolkit Configuration Schema Help, available in the UFT Web Add-in Extensibility Help.

You compose the **Conditions** using a set of **Condition** elements, joined by either **and** or **or** logic. Each **Condition** element specifies a certain property of the HTML control, and the expected value for that property. The condition is met if the value in the control's property matches the specified value (you can specify in the condition whether the value must be equal or not equal to the specified value). You can nest **Conditions** elements to create complex logic.

For each set of conditions, that is for each top-level **Conditions** element, you specify one of the following types, instructing UFT how to treat the control if its properties match the conditions within it:

- **IdentifyIfPropMatch.** If the conditions in this element are met, use the current test object class to represent the control.
- **CallIDFuncIfPropMatch.** If the conditions in this element are met, call the identification function to check this control. Otherwise, do not use the current test object class to represent the control.
- **SkipIfPropMatch.** If the conditions in this element are met, do not use the current test object class to represent the control.

If an identification function is defined, and checking the conditions does not enable UFT to determine whether to use the current test object class to represent the control, it calls the identification function.

- If a Conditions element of type **IdentifyIfPropMatch** is defined, it is checked before the other types.
- If both **CallIDFuncIfPropMatch** and **IdentifyIfPropMatch** Conditions elements are defined, the Conditions of type **CallIDFuncIfPropMatch** are checked only if the Conditions of type **IdentifyIfPropMatch** are not met.
- If both **SkipIfPropMatch** and **IdentifyIfPropMatch** Conditions elements are defined, the Condition of type **SkipIfPropMatch** are checked only if the Conditions of type **IdentifyIfPropMatch** are not met.
- A Conditions element of type **SkipIfPropMatch** is not checked if a Conditions element of type **CallIDFuncIfPropMatch** is defined.
- If you nest **Conditions** elements, the type attribute of the nested elements is ignored.

Understanding How UFT Performs the Identification

For each custom control that it handles, UFT performs the following procedure to determine which test object class to use for the custom control:

1. If an identification function is defined at the toolkit level (in the **CommonIdentification** element of the toolkit configuration file), first check the **Identification\Conditions** elements in all of the **Control** elements, to determine whether calling the function can be avoided.

   For each **Control** element:

   a. If an **IdentifyIfPropMatch** condition is defined and the custom control's properties meet the condition's requirements, use this **Control** element's test object class. Otherwise, continue with the following steps.

   b. If a **CallIDFuncIfPropMatch** condition is defined and the custom control's properties do not meet the condition's requirements, do not use this **Control** element's test object class.

      If this **Control** element does not contain a **CallIDFuncIfPropMatch** condition, continue with the following step.

   c. If a **SkipIfPropMatch** condition is defined and the custom control's properties meet the condition's requirements, do not use this **Control** element's test object class.

   After checking all of the conditions, call the toolkit level identification function unless a matching **Control** element was found or the conditions in **all** of the **Control** elements indicate not to use this **Control** element's test object class.

   The identification function returns the name of the test object class to use to represent this custom control.

2. If no identification function is defined at the toolkit level, check the **Control** elements one by one. Each **Control** element is defined for one test object class.

   For each **Control** element, perform the following steps to determine whether to use this element's test object class for the custom control. Use the first matching **Control** element:

   a. If an **IdentifyIfPropMatch** condition is defined and the custom control's properties meet the condition's requirements, use this **Control** element's test object class. Otherwise, continue

with the following steps.

b. If a **CallIDFuncIfPropMatch** condition is defined and the custom control's properties meet the condition's requirements, continue with step d. If the custom control's properties do not meet the condition's requirement, do not use this **Control** element's test object class.

   If this **Control** element does not contain a **CallIDFuncIfPropMatch** condition, continue with the following steps.

c. If a **SkipIfPropMatch** condition is defined and the custom control's properties meet the condition's requirements, do not use this **Control** element's test object class. Otherwise, continue with the following step.

d. If an identification function is defined within this **Control** element, run it to determine whether to use this element's test object class for the custom control. The function returns `true` or `false`. If this **Control** element does not contain an identification function, do not use this **Control** element's test object class.

Understanding How HTML Properties are Compared for Conditions

When comparing the value of an HTML property specified in a condition with the specified expected value, the following rules apply:

- String value comparisons are not case-sensitive.
- You can instruct UFT to treat the string provided in the expected value as a regular expression. In this case, set the **is_reg_exp** attribute to **true**.
- Numeric value comparisons simply compare the numeric values.
- When comparing a Boolean value, the values **true**, **1**, and **yes** are all considered true. The values **false**, **0**, and **no**, are all considered false.
- If the HTML property that you are checking returns an object, use `valid` and `null` as the expected values. The property is considered **valid** if it successfully returns an object, and **null** if it fails to return an object.
- Set the **equal** attribute in the **Condition** element to **false** if you want to check if the property does not have a certain value.
- Set the **trim** attribute in the **Condition** element to **true** if you want UFT to remove leading and trailing spaces from the property value and the expected value before evaluating the condition.

The examples below illustrate different ways that you can use the **Conditions** element.

Example 1: Identification Function and Conditions

If the custom control has a **CalendarBehavior** or **PopupBehavior** property defined, UFT calls the **isCalendar** JavaScript function to determine whether to use a **TSGCalendar** test object to represent the control. Otherwise, it continues to the next **Control** element in the toolkit configuration file and checks whether it matches the custom control.

```
<Control TestObjectClass="TSGCalendar">
```

```
    <Identification type="javascript" function="isCalendar">
      <Browser name="*">
        <Conditions type="CallIDFuncIfPropMatch" logic="or">
            <Condition prop_name="CalendarBehavior" expected_value="valid"/>
            <Condition prop_name="PopupBehavior" expected_value="valid"/>
        </Conditions>
      </Browser>
    </Identification>
</Control>
```

Example 2: Conditions Only

If the custom control has an **AccordionBehavior** property defined, UFT uses a **TSGAccordion** test object to represent the control. Otherwise, it continues to the next **Control** element in the toolkit configuration file and checks whether it matches the custom control.

```
<Control TestObjectClass="TSGAccordion">
    <Identification>
      <Browser name="*">
        <Conditions type="IdentifyIfPropMatch" logic="and">
            <Condition prop_name="AccordionBehavior" expected_value="valid"/>
        </Conditions>
      </Browser>
    </Identification>
</Control>
```

Example 3: Nested Conditions Nodes

If the value of the custom control's **className** property is equal to `tsg`, and either the value of the control's **id** property is equal to `tsg_table` or the value of the control's **myType** property is equal to `table`, UFT uses a **TSGTable** test object to represent the control.

If the value of the custom control's **className** property is equal to `tsg`, but the value of the control's **id** property is not `tsg_table` and the value of the **myType** property is not `table`, UFT calls the **isTable** JavaScript function to determine whether to use a **TSGTable** test object.

If the value of the custom control's **className** property is not equal to `tsg`, UFT continues to the next **Control** element in the toolkit configuration file without calling **isTable** identification function.

```
<Control TestObjectClass="TSGTable">
    <Identification type="javascript" function="isTable">
      <Browser name="*">
        <Conditions type="IdentifyIfPropMatch" logic="and">
            <Condition prop_name="className" expected_value="tsg"/>
            <Conditions logic="or">
```

```
                <Condition prop_name="id" expected_value="tsg_table"/>
                <Condition prop_name="myType" expected_value="table"/>
            </Conditions>
        </Conditions>
        <Conditions type=" SkipIfPropMatch" logic="and">
            <Condition prop_name="className" expected_value="tsg"
equal="false"/>
        </Conditions>
      </Browser>
    </Identification>
</Control>
```

For information on the structure and syntax of the **Conditions** and **Condition** elements, see the Toolkit Configuration Schema Help, available in the UFT Web Add-in Extensibility Help.

Testing the Toolkit Support Set During Development

After you define your test object model in the test object configuration file, and define a basic toolkit configuration file enabling UFT to identify which test object classes to use for the different controls, you can test the existing functionality of the toolkit support set. To do this, you deploy the toolkit support set and test how UFT interacts with the controls in your environment.

After you complete additional stages of developing support for your environment, you can deploy the toolkit support set again and test additional areas of interaction between UFT and your controls (learning test objects, running tests, and so on).

If you designed your support to operate on different browsers, make sure to test it accordingly.

To test your toolkit support set after defining the test object classes and mapping them to custom Web elements:

1. In the test object configuration file, set the **DevelopmentMode** attribute of the **TypeInformation** element to `true`, to ensure that UFT reads all of the test object class information from the file each time it opens. When you complete the development of the toolkit support set, be sure to set this attribute to `false`.
2. Deploy the toolkit support set on a UFT computer by copying the files of the support set to the correct locations in the UFT installation folder. For more information, see "Deploying the Toolkit Support Set" on page 72.

3. Open UFT and open a GUI test. Ensure that the environment name that you defined for the toolkit support set is displayed in the Add-in Manager dialog box as a child of the Web Add-in. (If the Add-in Manager dialog box does not open when you open UFT, see the *HP Unified Functional Testing Add-ins Guide* for instructions.)

   Tip: The environment name displayed in the add-in manager is the same as the toolkit configuration file name. If your environment is not displayed correctly in the add-in manager, make sure that the toolkit configuration file is correctly named, located in the correct folder, and that its syntax is correct.

4. Select the check box for your environment (the Web Add-in is then selected automatically) and click **OK** to load the support for your toolkit.
5. Use the Define New Test Object dialog box to create a new test object from a test object class that you defined. The name of your environment is displayed in the **Environment** list. If you select the name of your environment from the list, the test object classes that you defined in the test object configuration file are displayed in the **Class** list.

   For more information, see the section on defining new test objects in the object repository in the *HP Unified Functional Testing User Guide*.
6. Open an application with your custom controls.
7. UFT can already recognize and learn your controls, and you can create test steps with your custom test objects. (For more information on working with the options below in UFT, see the *HP Unified Functional Testing User Guide*.)

   a. Use the Object Spy  to view the identification properties and test object operations that are supported for your controls.

      - For each test object class that you defined, the Object Spy displays all of the identification properties that you defined in the test object configuration file. New identification properties are displayed without values because you have not yet implemented methods to retrieve the values from the controls.

        For identification properties that have the same names as base test object class properties, UFT uses the base class implementation to retrieve the property values if the root Web element of the control matches the base test object class. For more information, see "Implementing Support for Identification Properties" on page 65.

        Note: UFT uses only lowercase letters in identification property names. If the identification property name in the test object configuration file contains uppercase letters, they are converted to lowercase.

      - The Object Spy displays all of the test object operations available for each test object class that you defined to represent your controls. This includes test object methods and properties inherited from the base test object class, as well as the test object methods that you defined in the test object configuration file.

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

c. In the Keyword View, create a test step with a test object from a class that you defined.
   - If you defined a default operation for this class, it is displayed in the **Operation** column after you select the test object in the **Item** column.
   - The list of available operations in the **Operation** column reflects the definitions in the test object configuration file, and also includes operations inherited from the base test object class.
   - After you choose an operation, the **Value** cell is partitioned according to the number of arguments of the selected operation, and if you defined possible values for the operation (in the ListOfValues element), they are displayed in a list.
   - The descriptions and documentation strings you defined for test object methods in the test object configuration file are displayed in tooltips and in the **Documentation** column, respectively.

d. In the Editor, create a test step with a test object from a class that you defined. The statement completion feature displays all of the operations available for the test object, and possible input values for these operations, if relevant, based on the definitions in the test object configuration file. (Inherited test object methods are also displayed.)

e. In the Step Generator, create a test step with a test object from a class that you defined. The operations that you defined in the test object configuration file are displayed in the Operation list, and the descriptions you defined for the operations are displayed as tooltips. (Inherited test object methods are also displayed.)

To test your toolkit support set after developing support for additional UFT functionality:

1. Follow the steps in the previous procedure to deploy the toolkit support set, open UFT, load the support and run an application with controls from your environment.
2. Depending on the UFT functionality for which you are developing support, perform the relevant UFT operations on the application to test that support. For example, learn controls in the application, run a test on the application, record test steps on the application and so on.

   By default, UFT injects the toolkit support set JavaScript files that you develop into Web pages when they are opened. To test subsequent changes that you make in the JavaScript files, you need to close and reopen the Web page that contains the custom controls.

   Alternatively, you can define a **Controls\Settings\Variable** element in the toolkit configuration file, set its **name** attribute to `cache scripts` and its **value** attribute to `false`. This instructs UFT to re-inject the JavaScript file into the Web page every time it is refreshed. Therefore, after you modify your JavaScript functions and deploy them, if you have not changed anything in the XML files or reopened UFT, you only need to refresh the Web page to test your changes.

When you complete the development of the toolkit support set, be sure to remove this element or set the value attribute to `true`, to avoid performance degradation.

Logging and Debugging the Custom Support

When you design the JavaScript functions for your toolkit support set, you can use the **_util.LogLine** method to add log messages to the Microsoft Windows event log. When UFT runs a test or component using the support you designed, you can view these messages in the Event Viewer. For more information see, "Using the Microsoft Windows Event Log" below.

When you test the toolkit support set that you designed, you can debug your JavaScript files like you would debug any other JavaScript file, using the Microsoft Script Debugger or the Microsoft Visual Studio debugger. For more information, see " Debugging Your JavaScript Files (Internet Explorer Only)" on page 60.

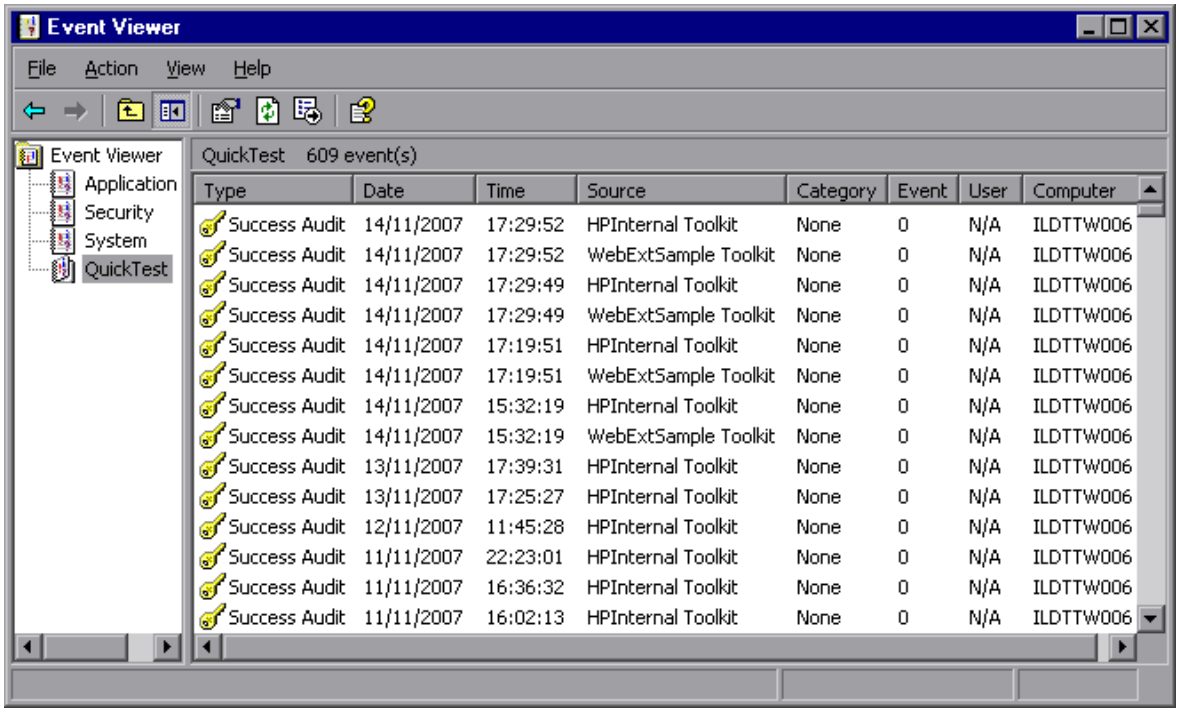
Using the Microsoft Windows Event Log

You can use the **_util.LogLine** method in your JavaScript functions to add messages to the Microsoft Windows event log. For more information, see the *UFT Web Add-in Extensibility API Reference* (available with the Web Add-in Extensibility Help).

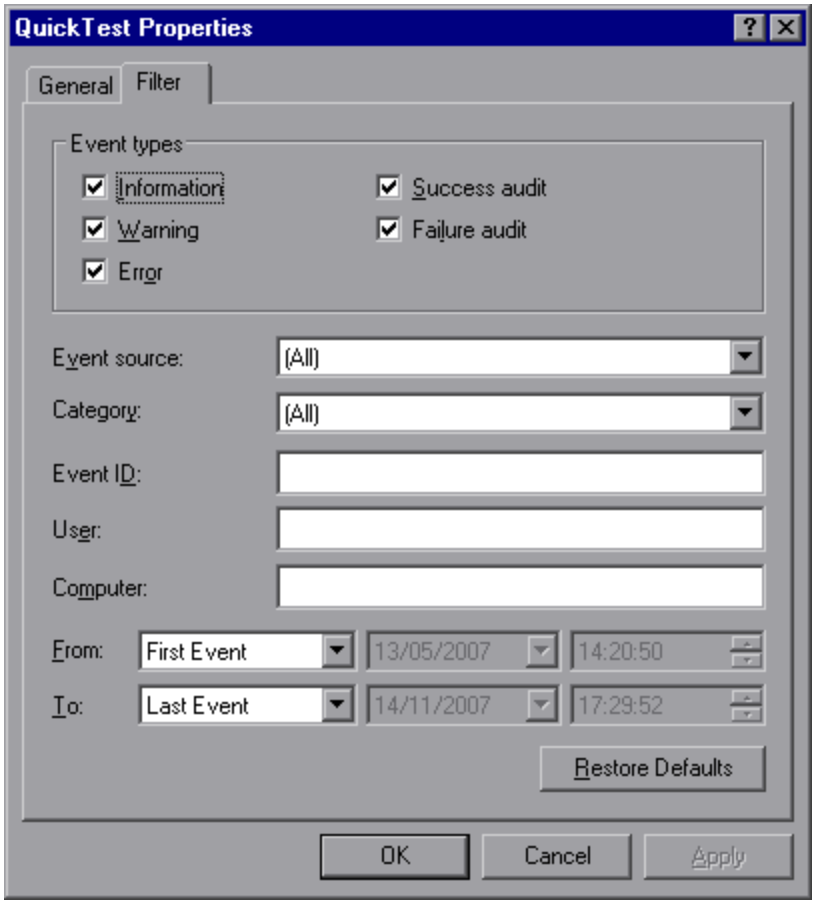
You provide the log message text and the level of severity for the log entry and, optionally, an ID and a category number. UFT adds the toolkit name and a time and date stamp to the information that you provide, and adds the entry to the event log.

In addition, while recognizing objects supported by Web Add-in Extensibility and performing tests on them, UFT also writes log and error messages to the event log.

To view the event log and analyze the performance of your toolkit support set, open the Event Viewer (in Windows XP and Windows 2000, select **Start > Settings > Control Panel > Administrative Tools > Computer Management**, expand the **Event Viewer** node in the **Computer Management** tree) and select the UFT node. Double-click a specific log entry to see its text.



You can filter the log messages displayed in the Event Viewer according to severity and other message fields. In the Computer Management toolbar, select **View > Filter**. For more information, see the Event Viewer Help (select **Action > Help** in the Event Viewer).



Debugging Your JavaScript Files (Internet Explorer Only)

You can use the Microsoft Script Debugger or the Microsoft Visual Studio debugger to debug the JavaScript files that you write for your toolkit support set.

To enable debugging you must clear the **Disable script debugging (Internet Explorer)** and **Disable script debugging (other)** options in the Internet Explorer advanced options (**Tools > Internet Options > Advanced**). After you change this option, you need to restart Internet Explorer for the change to take effect.

If you want the Just-In-Time debugger to list the Microsoft Visual Studio debugger as one of the available debuggers, you need to select the **Script** option in the Visual Studio Options dialog box (**Visual Studio > Tools > Options > Debugging > Just-In-Time**).

After these debugging options are enabled, you can use all standard methods to debug your JavaScript functions. For example:

- You can attach to the Internet Explorer process and put breakpoints in your functions.
- You can include a `debugger;` statement in your JavaScript function, to launch the Just-In-Time debugger when the function runs.
- You can use the Just-In-Time debugger when an exception occurs that causes it to open.

Implementing Support for Test Object Methods

After enabling UFT to recognize the custom controls, you must provide support for running test object methods. If you try to run a test with steps that run on custom test objects before providing implementation for these methods, the test fails and a run-time error occurs.

For each test object method that you defined in the test object configuration file, you must write a JavaScript function that UFT runs to perform the step on the control. In addition to performing the step, you can design the JavaScript function to add a line to the run results, add log messages to the event log, and display messages boxes to the UFT user, as necessary. For more information, see "Designing JavaScript Functions for Your Toolkit Support Set" on page 42.

In the toolkit configuration file, you need to specify the JavaScript file in which UFT should look for the JavaScript functions and, optionally, the name of the function to use for each test object method.

You can specify a JavaScript file and function for each test object method in the toolkit configuration file, or you can define a default JavaScript file per toolkit (**Controls** element) or per test object class (**Control** element). UFT uses the default files for all test objects methods for which you do not specify an implementation file. By default, UFT searches in the specified file for a JavaScript function with the same name as the test object method. Therefore, you do not need to specify function names in the toolkit configuration file, unless you choose to name a function something other than the corresponding test object method name. For more information, see the *Toolkit Configuration Schema Help*, available in the UFT Web Add-in Extensibility Help.

Caution: Do not create JavaScript functions named **get_property_value** or **get_list_of_values**. These names are reserved for the JavaScript functions that UFT calls (by default) to retrieve run-time values of test object identification properties and lists of possible values for test object method arguments.

After you create support for running test object methods, you can run UFT GUI tests on your custom test objects, and verify that your toolkit support set performs correctly. For more information on testing your toolkit support set, see "Testing the Toolkit Support Set During Development" on page 55.

The following example is taken from the ASPAjax toolkit support set, which includes support for a **Select** method on the ASPAjaxTabs test object. In the ASPAjax test object configuration file, this is declared as follows:

```
<ClassInfo BaseClassInfoName="WebElement" Name="ASPAjaxTabs"
           DefaultOperationName="Select" >
    <TypeInfo>
        <Operation ExposureLevel="CommonUsed" Name="Select"
                PropertyType="Method">
            <Description>Selects the specified tab.</Description>
            <Documentation>
                <![CDATA[Select the tab with index %a1.]]>
            </Documentation>
            <Argument Name="Index" IsMandatory="true" Direction="In">
                <Type VariantType="Integer"/>
                <Description>
                    The index value of the tab to select.
                </Description>
            </Argument>
        </Operation>
    </TypeInfo>
</ClassInfo>
```

In the toolkit configuration file, the **Control\Settings** element for the ASPAjaxTabs test object class is defined as follows:

```
<Control TestObjectClass="ASPAjaxTabs">
    <Settings>
        <!-- Indicates the location of the JavaScript file that contains the
implementation of the script -->
        <variable name="default_imp_file" value="Tabs.js"/>
    </Settings>
</Control>
```

No **Run** element is defined for this test object class. Therefore, when UFT runs a step with the **Select** test object method, UFT searches in the **Tabs.js** file (as defined in the toolkit configuration file) for a JavaScript function named **Select**.

In the **Tabs.js** file, the **Select** JavaScript function is implemented to accept an index and select the tab with that index.

Supporting Dynamic Lists of Values for Method Arguments

When a UFT user creates a test step with a test object method, UFT can display a set of predefined possible values available for the arguments of that method. For example, if an argument is a Boolean argument, UFT can display **true** and **false** as the possible values, or, for a **month** argument, UFT can

display a list of names of all the months. However, sometimes, a limited set of possible values for an argument exists, but depends on the specific object on which the step is performed. For example:

- The values that are actually relevant for the Integer **row** and **column** arguments of the function **Table(***<table_name>***).SetCellData (***row***,** *column***)** are limited to the number of rows and columns in the specific table.
- The relevant values for the String **path** argument of the function **Tree(***<tree_name>***).Select (***path***)** are limited to the paths that exist in the specific tree.

Using extensibility, you can enable UFT to dynamically provide a list of values for arguments of test object methods. UFT provides this list only in the Editor, when the statement completion feature is used.

To support a dynamic list of values:

1. In the test object configuration file, set the **DynamicListOfValues** attribute of the **Argument** element to **true**.
2. In the toolkit configuration file, you can specify the file name and function name of the JavaScript function that UFT must call to retrieve the list of values. By default, UFT requests the list of values by calling the **get_list_of_values** JavaScript function from the default implementation file that you specify for the test object class in the **default_imp_file** variable in the **Control\Settings** element. For more information, see the *Toolkit Configuration Schema Help*, available in the UFT Web Add-in Extensibility online Help.

   UFT calls the JavaScript function for every argument whose **DynamicListOfValues** attribute is set to **true** in the test object configuration file. The parameters provided to this function indicate the test object method and argument for which the values are being requested.
3. Write a JavaScript function that accepts the names of the test object method and argument and returns a list of values relevant for the specified argument on the current element. Return the string values concatenated to one string, each value enclosed in quotation marks.

   Note: The dynamic list of values is retrieved from the control in the application being tested. Therefore, to display the dynamic list of values, the relevant control must be visible in the application when the test is edited.

For example, in the toolkit support set for the WebExtSample environment, located in **<Web Add-in Extensibility SDK installation folder>\samples\WebExtSample** folder, a dynamic list of values is supported for the **AuthorName** argument in the **GoToAuthorPage** test object method of the test object class WebExtBook.

- In the **WebExtSampleTestObjects.xml** test object configuration file, the argument is defined as follows:

```
<Operation Name="GoToAuthorPage" PropertyType="Method">
    <Description>
        Opens the Web page for the specified author.
    </Description>
    <Argument Name="AuthorName" IsMandatory="true" Direction="In"
```

```
            DynamicListOfValues="true">
        <Type VariantType="String"/>
        <Description>The author.</Description>
    </Argument>
</Operation>
```

- In the **WebExtBook.js** file (defined as the default implementation file for the WebExtBook test object class in the **WebExtSample.xml** toolkit configuration file) the following JavaScript functions are designed to return a list of the book's authors, each enclosed in quotation marks:

```
// Dynamic list of values implementation
// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
function get_list_of_values( method, argIndex )
{
    // When creating a step with the GoToAuthorPage test object
    // method, provide a list of the authors of this book, that
    // can be used for the method's argument.
    if (method == "GoToAuthorPage")
    {
        return get_GoToAuthorPage_list_of_values(argIndex);
    }
    return null;
}
function get_GoToAuthorPage_list_of_values(argIndex)
{
    var arr = new Array();
    if( argIndex > 1 )
        return toSafeArray(arr);
    // Retrieve all authors
    var AuthorsCount = 0;
    var authors = window.$(_elem.rows[1].cells[0]).children("A");
    for( var i = 0 ; i < authors.length ; ++i )
    {
        arr[AuthorsCount] = "\"" + authors.eq(i).text() + "\"";
        AuthorsCount++;
    }
    return toSafeArray(arr);
}
```

Note: The `$` indicates that this code uses the jQuery JavaScript function library to provide browser-independent support.

Implementing Support for Identification Properties

In the test object configuration file you defined the identification properties for your test object classes. When UFT runs a test it needs to retrieve the values for these properties. UFT uses identification property run-time values in different 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.

To support retrieving the run-time values of identification properties, you need to implement a JavaScript function that accepts a **PropertyName** parameter and returns the value of any property UFT requests. (UFT uses only lowercase letters in identification property names. If the identification property name in the test object configuration file contains uppercase letters, they are converted to lowercase.)

This function must return the property value in one of the following formats: **String**, **Integer**, **Boolean**, or **array**. When returning an array, use the **toSafeArray** function to convert the array to the format that UFT expects. When you provide an identification property value in an array format, UFT converts the array to a semicolon-delimited string.

For more information on writing JavaScript functions for Web Add-in Extensibility, see "Designing JavaScript Functions for Your Toolkit Support Set" on page 42.

UFT uses the base test object class implementation to retrieve the identification property values when the following conditions are met:

- The control includes a base element (for more information, see "Extending an Existing Test Object Class" on page 38).
- The identification property is defined in the test object configuration file with the same name as a base test object class property.
- You do not provide a function that returns a value for that identification property.

Implement your JavaScript function to return a value for the identification properties defined in the test object configuration file in the following situations:

- Base test object class implementation for retrieving the value for this identification property value is not available.
- The base test object class implementation does not meet your needs.

By designing the function that returns identification property values to return a value for the **logical_name** property, you can control how UFT names test objects of this test object class. For more information, see "Customizing the Test Object Name" on the next page.

In the toolkit configuration file, you can specify the JavaScript file in which you implemented the JavaScript function that retrieves property values. You can also specify the name of the function that you implemented for this purpose (in the **Control\Run\Properties** element). However, if you do not specify a function name, UFT calls **get_property_value (***PropertyName***)** and this is the function that you must implement. If you do not specify a file name, UFT calls the function from the default JavaScript file

you specified in the **Control\Settings** element (at the test object class level) or in the **Controls\Settings** element (at the toolkit level). For more information, see the *Toolkit Configuration Schema Help*, available in the UFT Web Add-in Extensibility Help.

After you create support for retrieving the run-time values of identification properties, you can test that your toolkit support set correctly enables UFT to run checkpoints on your Web elements, output property values, display the property values in the Object Spy, and run test steps with the **GetROProperty** operation. For more information on testing your toolkit support set, see "Testing the Toolkit Support Set During Development" on page 55.

Customizing the Test Object Name

When UFT learns an object, it creates a unique name for each test object on the page. A descriptive test object name enables you distinguish between test objects of the same class and makes it easier to identify them in your object repository and in tests.

By default, a test object is given the name of its test object class (appended with an index if there is more than one test object of the same class on the page). In many cases, this is not the ideal name for the custom control.

The test object name needs to be meaningful to the UFT user, preferably using terminology that is relevant to your toolkit. UFT displays this name in the Keyword View, in the Editor, and in the object repository. The test object name is not used for object identification and therefore does not have to remain constant in the application.

For example, the test object name can be language-dependent. The UFT user can create a test with the application running in one language, creating test objects with names in that language. The user can then run the test on the same application in another language. The names of test objects in the test remain in the original language, but UFT can correctly recognize the test objects and perform operations on them, based on their description.

To control how UFT names test objects of a test object class, design the function that returns identification property values to return a value for the **logical_name** property. UFT uses this value as the test object name.

For example, **WebExtBook** test objects in the WebExtSample environment are named based on their book title. The sample toolkit support set for the WebExtSample environment is located in **%ALLUSERSPROFILE%\Documents\ExtAccTool\Samples\WebExtSample** folder. The **WebExtBook.js** file is defined as the default implementation file in the **WebExtSample.xml** toolkit configuration file. The JavaScript function for returning identification property run-time values is defined in the **WebExtBook.js** file as follows:

```
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 window.$(_elem.rows[0].cells[1]).text();
    }
}
```

Note: The `$` indicates that this code uses the jQuery JavaScript function library to provide browser-independent support.

Implementing a Filter for Learning Child Controls

When you instruct UFT to learn a Web page, the Define Object Filter dialog box opens, enabling you to determine which of the Web page's descendants should be learned with it. When you select **All object types**, instructing UFT to learn the custom control with its parent Web page, all of the controls contained within your custom control are also learned as children of that Web page (and siblings of the control itself).

In some situations, there is no need to create test objects for all of the children of a control. For example, when there are no significant operations to perform on the children and no properties to retrieve, or when, for testing purposes, operations performed on the children are viewed as operations performed on the parent control. For example, on a calculator control that contains button controls, there is no need to create test objects for the digit buttons. Pressing the digit buttons performs a **Set** operation on the calculator object itself, providing a numeric input for a calculator operation.

You can determine which controls UFT learns by defining a Learn Filter for the test object class you create. You can use the **Control\Filter\Learn** element in the toolkit configuration file to define basic filtering, or you can implement complex filters by writing a JavaScript function. If you design a filter using a JavaScript function, specify the location and name of the function in the toolkit configuration file.

In the toolkit configuration file, in the **Control\Filter\Learn** element, you can define:

- Whether to learn controls represented by this test object class. You can also specify that UFT should learn controls of this type only if they have children.
- Whether to learn the controls contained within the controls represented by this test object class. You can also specify that your JavaScript function needs to be called to determine which descendants to learn.

  If you write a JavaScript function to implement the filter, the function must return a SafeArray containing all of the descendant Web elements that you want UFT to learn. For more information, see "Designing JavaScript Functions for Your Toolkit Support Set" on page 42.

For more information, see the *Toolkit Configuration Schema Help*, available in the UFT Web Add-in Extensibility Help.

You can see an example of defining Learn Filters in the sample toolkit support set for the WebExtSample environment located in **%ALLUSERSPROFILE%\Documents\ExtAccTool\Samples\WebExtSample** folder.

- The **Filter** element for the **WebExtBook** test object class is defined (in the **WebExtSample.xml** file) as follows:

```
<Filter>
    <Learn learn_control="Yes" learn_children="No"/>
</Filter>
```

  This instructs UFT to learn **WebExtBook** test objects without their descendants.

- The **Filter** element for the **WebExtUsedBooks** test object class is defined as follows:

```
<Filter>
    <Learn learn_control="Yes" learn_children="CallFilterFunc"
           type="javascript" function="GetChildrenToLearn" />
</Filter>
```

  This instructs UFT to learn **WebExtUsedBooks** test objects, and to call the **CallFilterFunc** JavaScript function to determine which descendants to learn.

  The **GetChildrenToLearn** JavaScript function is located in the **WebExtUsedBooks.js** file, which is defined as the default implementation file in the **WebExtSample.xml** toolkit configuration file. The **GetChildrenToLearn** JavaScript function returns all of the radio button descendants of the UsedBooks table control:

```
function GetChildrenToLearn()
{
// Return all of the radio buttons in the used books table
return toSafeArray(window.$(_elem).children()[0].getElementsByTagName
("input") );
}
```

  Note: The `$` indicates that this code uses the jQuery JavaScript function library to provide browser-independent support.

After you implement a Learn Filter, you can instruct UFT to learn your custom controls, and verify that your toolkit support set correctly controls which of the control's children are learned. For more information on testing your toolkit support set, see "Testing the Toolkit Support Set During Development" on page 55.

Implementing Support for Recording

One way to add objects to the object repository and create tests in UFT is by recording. To record a test, UFT registers to listen to events on the Web elements, and, when an event occurs, UFT adds the relevant step to the test. By default, UFT uses the standard Web event configuration to determine the events to which to listen for each Web element, and the steps to record in the test when each event occurs.

If you want to customize recording on a test object class that you defined, you must specify the events that you want to record and the steps that you want UFT to add to the test when those events occur.

For each test object class on which you want to customize recording, define a **Control\Record\EventListening** element in the toolkit configuration file. In this element you can specify whether to use standard Web event configuration to handle events on controls represented by this test object class. In addition, you can specify whether to use standard Web event configuration to handle events that take place on those control's children.

In addition to specifying whether UFT should use standard Web event configuration, you can specify a JavaScript function that provides more specific event registration (and optionally, the name of the file containing the function). For more information, see the Toolkit Configuration Schema Help, available in the UFT Web Add-in Extensibility Help.

In addition to the definitions in the toolkit configuration file, you write the following types of JavaScript functions:

- One JavaScript function (named in the toolkit configuration file) that uses the **RegisterForEvent** function in the **_util** utility object 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.

  UFT calls this function after registering to listen to events according to the standard Web event configuration. The event registration performed by this function overrides any previous registrations for the same events. For events not handled by this function the standard registration is used.

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

  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). For more information on writing JavaScript functions for Web Add-in Extensibility, see "Designing JavaScript Functions for Your Toolkit Support Set" on page 42.

You can see an example of customized recording in the sample toolkit support set for the WebExtSample environment located in **%ALLUSERSPROFILE%\Documents\ExtAccTool\Samples\WebExtSample** folder.

- In the toolkit configuration file, within the **Control** element for the **WebExtBook** test object class, the following **Record\EventListening** element is defined:

```
<Record>
    <EventListening
        use_default_event_handling_for_children="false"
        use_default_event_handling="false"
        type="javascript" function="RegisterToEvents"/>
</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 **RegisterToEvents**. A JavaScript file is not defined, therefore UFT looks for the JavaScript function in the **WebExtBook.js** file that is specified at the **Control** level for the WebExtBook test object class.

- In the **WebExtBook.js** file, the following **RegisterToEvents** function is defined:

```
function RegisterToEvents( elem )
{
    // Connect to the "Select" event: When the book name or the
    // book icon is clicked, call OnSelectClicked.
    _util.RegisterForEvent( window.$(_elem.rows[0].cells[0]).
          children[0], "onclick", "OnSelectClicked" );
    _util.RegisterForEvent( window.$(_elem.rows[0].cells[0]),
          "onclick", "OnSelectClicked");
}
```

Note: The `$` indicates that this code uses the jQuery JavaScript function library to provide browser-independent support.

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

- In the **WebExtBook.js** file 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;
}
```

This function records the **Select** test object method on the WebExtBook test object when the book title or image is clicked.

After you implement support for recording, you can record a test on controls in your environment, and verify that your toolkit support set performs correctly. For more information on testing your toolkit support set, see "Testing the Toolkit Support Set During Development" on page 55.

Troubleshooting and Limitations - Developing Support

This section describes troubleshooting and limitations for developing Web Add-in Extensibility support.

- Web Add-in Extensibility can be used to create support for Web controls within Web pages and frames. You cannot develop custom support for Web pages or frames themselves. In other words, you cannot map a Web page or frame to a test object class you developed using Web Add-in Extensibility.
- For UFT to successfully use support developed using Web Add-in Extensibility, the browser's security settings must be set to allow active scripting. (In Internet Explorer 7, for example, you can find these security settings under: **Tools > Internet Options > Security > Custom Level > Scripting > Active scripting**.)
- When designing a JavaScript function, it is not possible to click a link on a Mozilla Firefox page using a simple **Click** or **FireEvent** method call. The Web Add-in Extensibility **_util** object provides a special **FireEvent** method that you can use to click links in this situation. (The method is not documented in the API Reference, as it is meant to be used only for this type of workaround).

  This is an example of how you would use this method:

```
function clickOnLink(link) {
    if (_util.GetBrowserType() == QtpConstants.IE) {
        link.click();
    }
    else {
        //Firefox.
        var evObj = window.document.createEvent("MouseEvents");
        evObj.initEvent("click", true, true);
        _util.FireEvent(link, "click", evObj);
    }
}
```

Chapter 4: Deploying the Toolkit Support Set

The final stage of extending UFT support for a custom toolkit is deploying the toolkit support set. This means placing all of the files you created in the correct locations on a computer with UFT installed, enabling UFT to recognize the controls in the toolkit and run tests on them.

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

This chapter includes:

About Deploying the Custom Toolkit Support

From the UFT user's perspective, after you deploy the toolkit support set on a computer on which UFT is installed, the toolkit support set can be used as a UFT add-in.

When UFT opens, it displays the toolkit support set's environment name in the Add-in Manager, as a child node under the Web Add-in node. Select the check box for your environment to instruct UFT to load support for the environment using the toolkit support set that you developed.

If support for your environment is loaded:

- UFT recognizes the controls in your environment and can run tests on them.
- UFT displays the name of your environment in all of the dialog boxes that display lists of add-ins or supported environments.
- UFT displays the list of test object classes defined by your toolkit support set in dialog boxes that display the list of test object classes available for each add-in. (For example: Define New Test Object dialog box, Object Identification dialog box.)

Deploying the Custom Toolkit Support

To deploy the toolkit support set that you create, you must place the files in specific locations within the UFT installation folder.

Note: Before you begin, create a folder with the name of your custom toolkit in the **<UFT Installation folder>\dat\Extensibility\Web** folder, if one does not already exist.

The following table describes the appropriate location for each of the toolkit support files:

File Name	Location
<Custom Toolkit Name>TestObjects.xml **Note:** This is the recommended file name convention. You can have more than one test object configuration XML file, and name them as you wish.	• `<UFT Installation folder>\dat\ Extensibility\ Web` • `<UFT Add-in for ALM Installation folder>\dat\Extensibilit y\ Web` (Optional. Required only if the folder exists, which means the UFT Add-in for ALM was installed independently from the ALM Add-ins page and not as part of the UFT installation.)
<Custom Toolkit Name>.xml	`<UFT Installation folder>\dat\ Extensibility\Web\Toolkits\<custom toolkit name>`
JavaScript files	Specify the location in the in **<Custom Toolkit Name>.xml**

File Name	Location
Icon files for new test object classes (optional)	The file can be a **.dll**, **.exe**, or **.ico** file, located on the computer on which UFT is installed, or in an accessible network location. Specify the location in `<Custom Toolkit Name>TestObjects.xml`
Help files for the test object classes (optional)	Must be a **.chm** file, located on the computer on which UFT is installed. Specify the location in `<Custom Toolkit Name>TestObjects.xml`
External JavaScript libraries These are not part of the toolkit support set, but when you deploy the support set you must make sure that the library files are located in the location specified in the toolkit configuration file.	A **.js** file, located on the computer on which UFT is installed, or in an accessible network location. Specify the location in `<Custom Toolkit Name>.xml`

Recommended File Locations

You specify the locations of the JavaScript, Help, and icon files in the toolkit support set's configuration files. You can specify these locations using relative paths. For more information, see the *Test Object Schema Help* and the *Toolkit Configuration Schema HelpToolkit Configuration Schema Help* (available with the Web Add-in Extensibility Help).

The recommended locations for these files are described in the following table:

File Name	Location
JavaScript files	`<UFT Installation folder>\dat\Extensibility\Web\Toolkits\<custom toolkit name>\JavaScript`
Icon files	`<UFT Installation folder>\dat\Extensibility\Web\Toolkits\<custom toolkit name>\Res`
Help files	`<UFT Installation folder>\dat\Extensibility\Web\Toolkits\<custom toolkit name>\Help`

Settings to Use During Design Stages

- If you modify attributes of **Identification Property** elements in the test object configuration file, keep the **DevelopmentMode** attribute of the **TypeInformation** element set to `true` during the design stages of the custom toolkit support. Before you deploy the custom toolkit support set for regular use, be sure to remove this attribute (or set it to `false`). For more information, see "Modifying Identification Property Attributes in a Test Object Configuration File" below.
- To instruct UFT to re-inject the JavaScript file into the Web page every time it is refreshed, define a **Controls\Settings\Variable** element in the toolkit configuration file, set its **name** attribute to `cache scripts` and its **value** attribute to `false`. If you do this, then after you modify your JavaScript functions and deploy them, if you have not changed anything in the XML files or reopened UFT, you only need to refresh the Web page containing your custom controls to test your changes. You do not have to close and reopen the Web page.

  Before you deploy the custom toolkit support set for regular use, be sure to remove this element or set its **value** attribute to `true` to avoid performance degradation.

Modifying Deployed Support

If you modify a deployed toolkit support set, you must reopen UFT and re-run the Web application for the changes to take effect.

If you change the identification property definitions that specify the functionalities for which the properties are used in UFT, see "Modifying Identification Property Attributes in a Test Object Configuration File".

Modifying Identification Property Attributes in a Test Object Configuration File

The following attributes of the **Identification Property** element in the test object configuration file specify information that can be modified in UFT (using the Object Identification dialog box): **AssistivePropertyValue**, **ForAssistive**, **ForBaseSmartID**, **ForDescription**, **ForOptionalSmartID**, and **OptionalSmartIDPropertyValue**. These attributes determine the lists of identification properties used

for different purposes in UFT. For more information, see the *UFT Test Object Schema Help*, available in the UFT Web Add-in Extensibility Help.

Therefore, by default, UFT reads the values of these attributes from the XML file only once, to prevent overwriting any changes a user makes using the Object Identification dialog box. In this way, UFT provides persistence for the user defined property lists.

If the user clicks the **Reset Test Object** button in the Object Identification dialog box, the attributes' values are reloaded from the XML.

If the XML changed since the last time it was loaded (based on the file's modification date in the system), UFT reads the attributes from the XML. UFT adds identification properties to the relevant lists (and adjusts their order if necessary) according to the values of these attributes, but does not remove any existing identification properties from the lists.

To instruct UFT to completely refresh the identification property lists according to the attributes defined in the XML each time UFT is opened, set the **DevelopmentMode** attribute of the **TypeInformation** element in this test object configuration file to `true`.

Considerations When Modifying Identification Properties Attributes

- If you modify attributes of **Identification Property** elements in the test object configuration file, keep the **DevelopmentMode** attribute of the **TypeInformation** element set to `true` during the design stages of the custom toolkit support. This ensures that UFT uses all of the changes you make to the file.
- Before you deploy the toolkit support set for regular use, be sure to remove the **DevelopmentMode** attribute of the **TypeInformation** element (or set it to `false`). Otherwise, every time UFT opens it will refresh the property lists based on the definitions in the test object configuration file. If UFT users change the property lists using the Object Identification dialog box, their changes will be lost when they reopen UFT.
- Though UFT does not remove existing properties from the property lists when reading a modified test object configuration file (unless the **DevelopmentMode** attribute is set to `true`), it does add properties and adjust the order of the lists based on the definitions in the file. If UFT users removed properties from the lists or modified their order using the Object Identification dialog box, those changes will be lost when a modified file is loaded.

  If you provide the custom toolkit support set to a third party, and you deliver an upgrade that includes a modified test object configuration file, consider informing the UFT users about such potential changes to their identification property lists.

Removing Deployed Support

When opening UFT, the UFT user can use the Add-in Manager to instruct UFT whether to load the support provided for any particular environment or toolkit.

If you want to remove support for a custom toolkit from UFT after it is deployed, you must delete its toolkit configuration file from: **UFT Installation folder>\dat\Extensibility\Web\Toolkits\<custom toolkit name>**

If none of the test object class definitions in a test object configuration file are used to represent any custom controls (meaning they are no longer needed), you can delete the file from: **UFT Installation Folder>\dat\Extensibility\Web** and
UFT Add-in for ALM Installation folder>\dat\Extensibility\Web if relevant.

Part 2: Tutorial: Learning to Create Web Custom Toolkit Support

The lessons in this tutorial are Internet-Explorer oriented. This means that the JavaScript code you create is written in Internet-Explorer style, and the support that you create for the sample custom controls will operate correctly only on Microsoft Internet Explorer. When you create your own Web Add-in Extensibility support, you can design it to run on Mozilla Firefox and Google Chrome as well. For more information, see " Developing Browser-Independent Support" on page 31.

Chapter 5: Learning to Create UFT Support for a Simple Custom Web Control

In this lesson you manually 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 creating a toolkit support set.

When you create support for your own controls, use Extensibility Accelerator to design your support. Extensibility Accelerator creates the toolkit support set files and sets up the XML files automatically, so you can focus on designing your JavaScript functions. Creating the support manually in this lesson enables you to get a better understanding of the structure and content of a Web Add-in Extensibility toolkit support set.

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.

Before you perform this lesson, ensure that you have read "Introducing UFT Web Add-in Extensibility" on page 12.

This lesson includes:

Preparing 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\Application\Book.htm**.

Open the file to run the control.

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. The UsedBooks table is a more complex control that you will learn to support in the lesson, "Learning to Create UFT Support for a Complex Custom Web Control" on page 110.

Planning 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 "Understanding the Web Add-in Extensibility Planning Checklist" on page 25, and fill in the "Web Add-in Extensibility Planning Checklist" on page 27, 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.

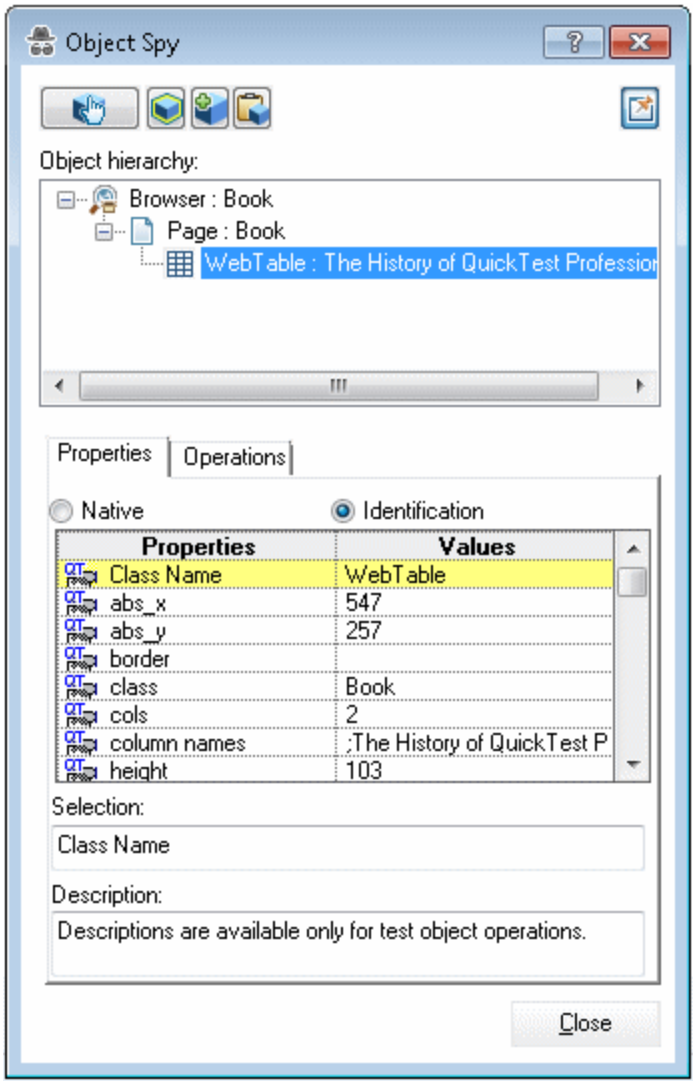
2. **Use the Object Spy to view the Book properties and operations.**

   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.

   Below, you can see the checklist, completed based on the information above.



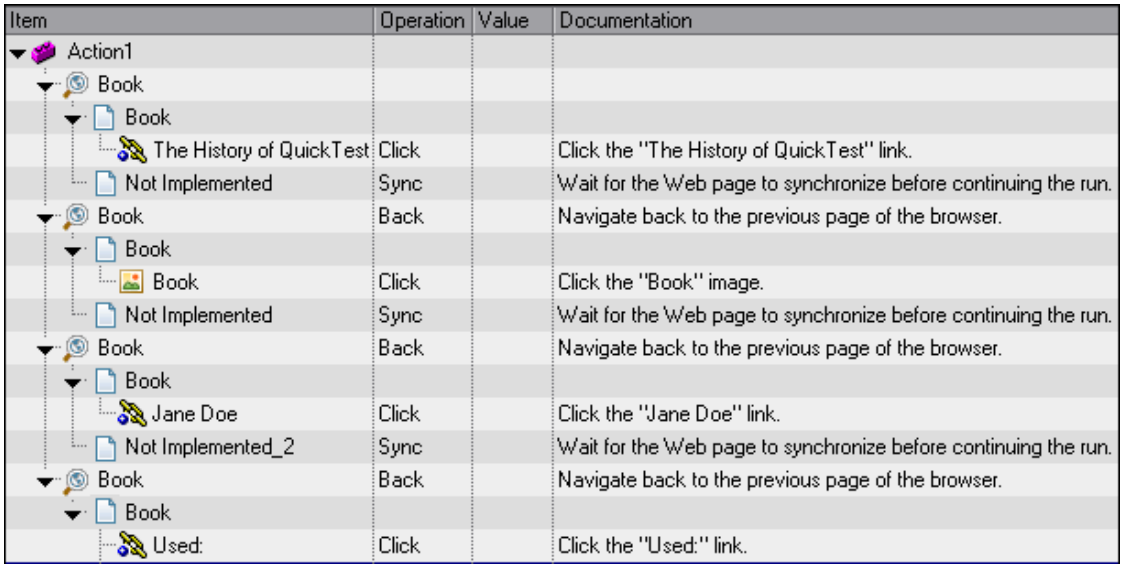
   Close the Object Spy.

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

   In UFT, select **Record > Record and Run 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 QuickTest	Click		Click the "The History of QuickTest" link.
Not Implemented	Sync		Wait for the Web page to synchronize before continuing the run.
Book	Back		Navigate back to the previous page of the browser.
Book			
Book	Click		Click the "Book" image.
Not Implemented	Sync		Wait for the Web page to synchronize before continuing the run.
Book	Back		Navigate back to the previous page of the browser.
Book			
Jane Doe	Click		Click the "Jane Doe" link.
Not Implemented_2	Sync		Wait for the Web page to synchronize before continuing the run.
Book	Back		Navigate back to the previous page of the browser.
Book			
Used:	Click		Click the "Used:" link.

   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$</strong>   <a
href=".\UsedBooks.htm">Used:</a> from <strong>29.99$</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. Define how UFT will identify which test object class to use to represent the control:

   If the control's **tagName** property is **table** and its **className** property is **Book**, use a **WebExtBook** test object to represent the control.

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. Determine whether you need to support a dynamic list of values for any method arguments:

Yes, a dynamic list of values is required for the **AuthorName** argument in the **GoToAuthorPage** method. (This requires modifying the toolkit configuration file to specify the JavaScript function that provides the values, and designing the relevant JavaScript function).

f. Define which of the control's children UFT should learn when learning the control:

For testing purposes, UFT should relate to all operations as though they are carried out on the Book control itself, even if they are technically performed on controls within it. Therefore, none of the control's children need to be learned and represented by test objects.

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

h. 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.

i. Decide what parts of the support need to be designed in the toolkit configuration file and what parts need JavaScript functions:
   - For the simple Book control, test object class identification is based simply on html property values and can therefore be supported using the toolkit configuration file without JavaScript functions.
   - Test object methods and identification properties can be supported by JavaScript functions using the default naming convention, therefore no changes are required in the toolkit configuration file.
   - Instructing UFT not to learn the control's children can be designed in the toolkit configuration file and does not require JavaScript functions.
   - To support recording, you modify the toolkit configuration file to turn off the default Web Event Configuration and specify the JavaScript function that registers UFT to listen to the correct event. In addition, design one JavaScript function that handles event registration, and additional JavaScript functions that instruct UFT to record the relevant steps when the events occur.

Below, 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
☑	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

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

☑	Provide a dynamic list of values for any test object method arguments?
	`Yes`
	If so, list the arguments:
	`AuthorName for GoToAuthorPage method`
		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 the toolkit support set for the WebExtSample toolkit and learn the basics of Web Add-in Extensibility. Developing this toolkit support set comprises the following stages:

Stage 1: Creating the Toolkit Support Set

In this section, you create the files and folders that comprise the toolkit support set for the WebExtSample toolkit.

To create the toolkit support set:

1. Create a folder for your toolkit support set.

   You can choose any convenient name and location for this folder.
2. In the toolkit support set folder, create a file named **WebExtSampleTestObjects.xml**. This is the test object configuration file.
3. In the toolkit support set folder, create a folder named **Toolkits**.
4. In the **Toolkits** folder, create a folder named **WebExtSample**.
5. In the **Toolkits\WebExtSample** folder, create the following:
   - A file named **WebExtSample.xml** (This is the toolkit configuration file.)
   - A file named **WebExtBook.js** (This is the file for all of the JavaScript functions you design to support the Book control.)
   - A folder named **Res** containing the **WebBook.ico** icon file (You can copy the icon file from **%ALLUSERSPROFILE%\Documents\ExtAccTool\Samples\WebExtSample\Res**.)

Stage 2: Introducing the WebExtSample Environment to UFT

In this section, you introduce the WebExtSample environment to UFT, using the toolkit configuration file and the test object configuration file. The first layer of Web Add-in Extensibility is introducing the environment to UFT. The toolkit configuration file informs UFT of the new environment (and its name) and the test object configuration file describes the test object model that you designed for the environment.

Designing the Toolkit Configuration File

The first role of the toolkit configuration file is informing UFT of the new supported environment.

To inform UFT that a new environment is supported, it is sufficient to create a basic toolkit configuration file, whose name is the same as the environment name. A basic toolkit configuration file must contain one **Controls** element with at least one **Control** element (describing one test object class). For more information on the elements and attributes in the toolkit configuration file, see the *UFT Java Add-in Extensibility Toolkit Configuration Schema Help* (available with the Web Add-in Extensibility Help).

To inform UFT about the WebExtSample environment:

Enter the following text in the **WebExtSample.xml** file that you created in "Stage 1: Creating the Toolkit Support Set" on the previous page:

```
<?xml version="1.0" encoding="UTF-8"?>
<Controls>
        <Control TestObjectClass="WebExtBook"/>
</Controls>
```

After you deploy this file to the correct location on a UFT computer, when UFT opens, it displays the WebExtSample environment in the Add-in Manager, as a child node beneath the Web Add-in. If you select the check box for the WebExtSample, UFT loads the support that you provide for this environment.

Later in this lesson you will add additional elements within this **Control** element, providing the location of the JavaScript functions that complete the toolkit support set and information that provides support for the following UFT abilities:

- Identifying the test object class used to represent the control (to support the Object Spy and learning controls)
- Filtering child controls when learning the control
- Listening to events on the control to record test steps

Designing the Test Object Configuration File

You use the test object configuration file to introduce the WebExtSample environment and its test object model 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.

For more information 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).

To define the WebExtSample test object model in the test object configuration file:

Enter the text below in the **WebExtSampleTestObjects.xml** file that you created in "Stage 1: Creating the Toolkit Support Set" on page 87. This defines the **WebExtSample** environment and the **WebExtBook** test object class (including its test object methods and identification properties) according to the details described in the "Web Add-in Extensibility Planning Checklist" on page 85.

```
<?xml version="1.0" encoding="UTF-8"?>
<TypeInformation Load="true" AddinName="Web"
                 PackageName="WebExtSample">
     <ClassInfo BaseClassInfoName="WebElement"
                Name="WebExtBook"
                DefaultOperationName="Select" >
         <IconInfo IconFile= "INSTALLDIR\dat\Extensibility\Web\
                  Toolkits\WebExtSample\Res\WebBook.ico"/>
         <TypeInfo>
             <Operation ExposureLevel="CommonUsed"
                        Name="Select"
                        PropertyType="Method">
                 <Description>Selects the book.</Description>
                 <Documentation>
                     <![CDATA[Select the %1 book.]]>
                 </Documentation>
             </Operation>
             <Operation ExposureLevel="CommonUsed"
                        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"
```

```
                        Direction="In"
                        DynamicListOfValues="true">
                    <Type VariantType="String"/>
                    <Description>The author.</Description>
                </Argument>
            </Operation>
            <Operation ExposureLevel="CommonUsed"
                       Name="GoToUsedBooksPage"
                       PropertyType="Method">
                <Description>
                     Opens the UsedBooks page.
                </Description>
                <Documentation>
                     <![CDATA[Open the %1 UsedBooks page.]]>
                </Documentation>
            </Operation>
        </TypeInfo>
        <IdentificationProperties>
            <IdentificationProperty ForDefaultVerification="true"
                                    ForVerification="true"
                                    ForDescription="true"
                                    Name="title"/>
            <IdentificationProperty ForDefaultVerification="true"
                                    ForVerification="true"
                                    ForDescription="true"
                                    Name="authors"/>
            <IdentificationProperty ForDefaultVerification="true"
                                    ForVerification="true"
                                    ForDescription="false"
                                    Name="price"/>
            <IdentificationProperty ForDefaultVerification="true"
                                    ForVerification="true"
                                    ForDescription="false"
                                    Name="min_used_price"/>
        </IdentificationProperties>
    </ClassInfo>
</TypeInformation>
```

Stage 3: Teaching UFT to Identify, Spy, and Learn the Book Control

To support a custom control, UFT must be able to identify which test object class should represent a given control. Therefore, the most basic element of Web Add-in Extensibility is the **Identification** element, defined within each **Control** element in the toolkit configuration file. Each **Control** element

defines a test object class. The **Identification** element specifies which controls should be represented by that test object class.

When UFT needs to recognize a control in the application being tested, it checks the **Identification** element defined for each test object class. The first test object class whose **Identification** definition matches the control is used to represent the control.

As described in "Planning Support for the Web Add-in Extensibility Book Sample Toolkit" on page 80, any control whose **tagName** property is **table** and whose **className** property is **Book** is represented by a **WebExtBook** test object. This can be defined simply in the toolkit configuration file and does not require using JavaScript functions.

To define the identification rules for the WebExtBook test object class:

Replace the text in the **WebExtSample.xml** file with the following text:

```
<?xml version="1.0" encoding="UTF-8"?>
<Controls>
    <Control TestObjectClass="WebExtBook">
        <Identification>
          <Browser name="*">
            <Conditions type="IdentifyIfPropMatch" logic="and">
                <Condition prop_name="tagName" expected_value="TABLE"/>
                <Condition prop_name="className" expected_value="Book"/>
            </Conditions>
          </Browser>
        </Identification>
    </Control>
</Controls>
```

This adds an **Identification** element to the **Control** element that defines the **WebExtBook** test object class. The Identification element includes one **Conditions** element that contains two conditions, both of which must be met for the control to qualify as a **WebExtBook**. The **Condition** elements within the **Conditions** element specify one condition each. In each condition, the value of the specified HTML property of the control must match (case-insensitive compare) the specified expected value.

This tutorial uses the definition above to illustrate the use of more than one **Condition** element within a **Conditions** element. However, if you were working with an application that had many controls on a page, or a large DOM structure, this would be a better way to define the identification rules:

```
    <Identification>
      <Browser name="*">
        <HTMLTags>
            <Tag name="TABLE"/>
        </HTMLTags>
        <Conditions type="IdentifyIfPropMatch">
            <Condition prop_name="className" expected_value="Book"/>
        </Conditions>
```

```
    </Browser>
  </Identification>
```

This definition provides improved performance because it instructs UFT to perform the identification process only on TABLE elements.

For more information on defining the **Identification** element for a test object class, see " Teaching UFT to Identify the Test Object Class to Use for a Custom Web Control" on page 49 and the *UFT Java Add-in Extensibility Toolkit Configuration Schema Help* (available with the Web Add-in Extensibility Help).

Deploying and Testing the Toolkit Support Set (for Stage 3)

After defining the **WebExtBook** test object class in the test object configuration file and the identification rules for this test object class in the toolkit configuration file, you can already test the effect of using the toolkit support set with UFT.

Note: When you develop your own toolkit support set, if you modify attributes of **Identification Property** elements in the test object configuration file, keep the **DevelopmentMode** attribute of the **TypeInformation** element set to `true` during the design stages of the custom toolkit support. Before you deploy the custom toolkit support set for regular use, be sure to remove this attribute (or set it to `false`). This is not required when performing this tutorial lesson. For more information, see "Modifying Identification Property Attributes in a Test Object Configuration File" on page 75.

To deploy the toolkit support set

1. Copy the **WebExtSampleTestObjects.xml** file to **<UFT installation folder>\dat\Extensibility\Web**.
2. In the **<UFT installation folder>\dat\Extensibility\Web\Toolkits** folder, create a folder named **WebExtSample**.
3. Copy the **WebExtSample.xml** file to the **<UFT installation folder>\dat\Extensibility\Web\Toolkits\WebExtSample** folder.
4. In the **<UFT installation folder>\dat\Extensibility\Web\Toolkits\WebExtSample** folder, create a folder named **Res**.
5. Place the **WebBook.ico** file in **<UFT installation folder>\dat\Extensibility\Web\Toolkits\WebExtSample\Res** folder.

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 UFT and open it again.

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.

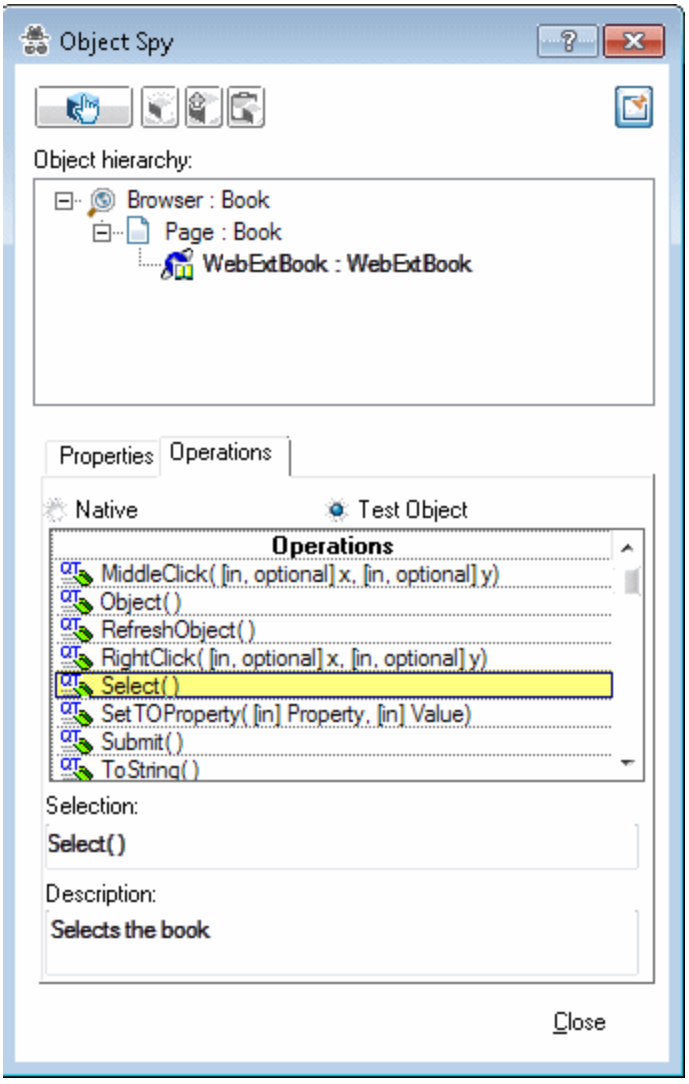
3. Use the **Define New Test Object**  button in the Object Repository dialog box to open the Define New Test Object dialog box. The WebExtSample environment is displayed in the **Environment** list. When you select the WebExtSample environment from the list, the **WebExtBook** test object class that you defined in the test object configuration file is displayed in the **Class** list, with the icon that you specified in the **WebExtSampleTestObjects.xml** file.

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 more information 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 methods that you defined in the **WebExtSampleTestObjects.xml** test object configuration file.



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

- 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 reflects the definitions in the test object configuration file.
  - After you select an operation, UFT displays the Name attribute of the operation's argument in a tooltip for the **Value** cell. If you define an operation with more than one argument, the **Value** cell is partitioned according to the number of arguments of the operation, when selected.
  - The descriptions and documentation strings you defined for test object methods in the test object configuration file are displayed in tooltips and in the **Documentation** column, respectively.
- 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, based on the definitions in the test object configuration file.

  If you defined possible values for the operation's arguments (in the ListOfValues element), they are displayed in a statement completion list as well. Later in this lesson, you learn how to enable a control to dynamically provide a list of possible argument values. You develop support for providing a list of the authors that can be used for the **AuthorName** argument of the **GoToAuthorPage** operation.

5. Run a test with a step that performs a new test object method on a WebExtBook test object. UFT searches for a JavaScript function that will run the test object method on the control. Because you have not yet implemented support for running test object methods, a run-time error occurs. In the next section, you implement this support.

Stage 4: Implementing Support for the WebExtBook's Test Object Methods

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

In the toolkit configuration file, you need to specify the JavaScript file in which UFT should look for the JavaScript functions and, optionally, the name of the function to use for each test object method.

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

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

simplest way possible. At the **Control** element level, you define one JavaScript file that UFT uses by default for all test objects methods defined within this element. As for the JavaScript function names, by default, searches in the specified file for a JavaScript function with the same name as the test object method. Therefore, you do not need to specify the function names in the toolkit configuration file, but only to create the JavaScript functions with the correct names.

To develop support for the WebExtBook test object methods:

1. In the **WebExtSample.xml** file, within the **Control** element defined for the **WebExtBook** test object class, add UFTthe following **Settings** element:

```
<Settings>
    <Variable name="default_imp_file" value="WebExtBook.js"/>
</Settings>
```

This instructs UFT to search for JavaScript functions in the **WebExtBook.js** file (in the **<UFT installation folder>\dat\Extensibility\Web\Toolkits\WebExtSample** folder).

Note: You can modify the **WebExtSample.xml** file in the toolkit support set folder and then later deploy it to UFT for testing, or you can modify **<UFT installation folder>\dat\Extensibility\Web\Toolkits\WebExtSample.xml** directly.

2. In the **WebExtBook.js** file that you created in "Stage 1: Creating the Toolkit Support Set" on page 87, paste the text below to create JavaScript functions for each test object method: **Select**, **GoToAuthorPage (***AuthorName***)**, and **GoToUsedBooksPage**.

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

```
// Run implementation
// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// This section contains the functions that carry out the
// test object methods.
function Select()
{    // Click the link in the second cell of the first row.
     _elem.rows[0].cells[1].children[0].click();
}
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 &lt; _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();
             bWasFound = true;
```

```
            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();
}
```

Deploying and Testing the Toolkit Support Set (for Stage 4)

After developing support for running the test object methods, you deploy the updated toolkit support set to UFT and test it.

To test the support for running test object methods:

1. To deploy the updated toolkit support set to UFT, copy the **WebExtBook.js** file (and **WebExtSample.xml** if necessary) to **<UFT installation folder>\dat\Extensibility\Web\Toolkits\WebExtSample**.
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.
4. Open a GUI test and use the **Add Objects to Local**  button in the Object Repository dialog box to learn the Book control.
5. Create a test with the following step and then run the test:

```
Browser("Book").Page("Book").WebExtBook("WebExtBook"). 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, UFT displays a run-time error message and the test step fails.

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

Stage 5: Implementing Support for the WebExtBook's 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.

To support retrieving the run-time values of identification properties, you need to implement a JavaScript function that accepts a **PropertyName** parameter and returns the value of any property UFT requests. You must implement this method to return a value for each identification property defined in the test object configuration file.

In the toolkit configuration file, you can specify the JavaScript file in which you implemented the JavaScript function that retrieves property values. You can also specify the name of the function that you implemented for this purpose. However, if you do not specify a function name, UFT calls **get_property_value (PropertyName)** and this is the function that you must implement. If you do not specify a file name, UFT calls the function from the JavaScript file you specified in the **Control\Settings** element. Therefore, in this lesson, you create the **get_property_value** function in the **WebExtBook.js** file.

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

Add the following lines to the **WebExtBook.js** file:

```
// Property retrieval implementation
// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// The function provides values for all of the identification
// properties defined in the test object configuration XML file.
function get_property_value(prop)
{
    if ( prop == "title" )
    // For the "title" identification 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;
    }
}
```

Note: You can modify the **WebExtBook.js** file in the toolkit support set folder and then later deploy it to UFT for testing, or you can modify **<UFT installation folder>\dat\Extensibility\Web\Toolkits\WebExtSample\WebExtBook.js** directly.

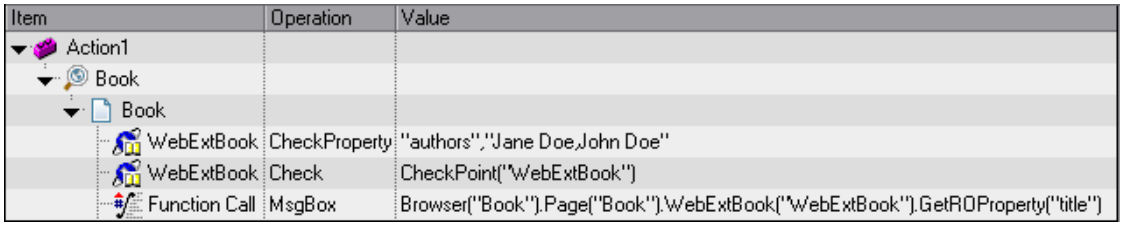
Deploying and Testing the Toolkit Support Set (for Stage 5)

After developing support for retrieving run-time values of identification properties, you deploy the updated toolkit support set to UFT and test it.

To test the support for retrieving run-time values of identification properties:

1. Make sure that your most updated files are located in **<**UFT **installation folder>\dat\Extensibility\Web\Toolkits\WebExtSample**.
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.
4. Create a new GUI test, add a WebExtBook test object to your object repository, and create a test step with this test object. 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.

5. 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:



   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.

6. 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 6: Changing the Name of the Test Object

In this section, you modify the toolkit support set to instruct UFT to name the WebExtBook test object according to its title, as per your plan ("Planning Support for the Web Add-in Extensibility Book Sample Toolkit" on page 80).

When UFT creates the test object that represents a control, it calls the **get_property_value** function (used for retrieving test object identification property values) with the argument **logical_name** to determine the name for the test object. You can implement the **get_property_value** function accordingly, to customize the name UFT gives the test object. 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 customize the name of the test object:

In the **get_property_value** function in the **WebExtBook.js** file, replace the lines:

```
if ( prop == "title" )
// For the "title" identification property,
```

with the lines:

```
if ( prop == "logical_name" || prop == "title" )
// For the "title" identification property, as well as
// the "logical_name" property
```

Note: You can modify the **WebExtBook.js** file in the toolkit support set folder and then later deploy it to UFT for testing, or you can modify **<UFT installation folder>\dat\Extensibility\Web\Toolkits\WebExtSample\WebExtBook.js** directly.

The **get_property_value** function now returns the same text for the **logical_name** property that UFT uses to name the test object, as it does for the **title** identification property. (Modify the comment that explains this function accordingly. At the end of the comment, add the following: `as well as the hard coded "logical_name" property that UFT uses to name the test object.`)

Deploying and Testing the Toolkit Support Set (for Stage 6)

After developing support for naming the test object that represents the control, you deploy the updated toolkit support set to UFT and test it.

To test the support for naming the WebExtBook test object:

1. Make sure that your most updated files are located in **<UFT installation folder>\dat\Extensibility\Web\Toolkits\WebExtSample.**
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.
4. 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**.

Stage 7: Implementing a Filter to Prevent Learning Child Objects

When you instruct UFT to learn a Web page, the Define Object Filter dialog box opens, enabling you to determine which of the Web page's descendants should be learned with it. When you select **All object types**, instructing UFT to learn the WebExtBook control with its parent Web page, all of the controls that the WebExtBook control contains are also learned as children of that Web page (and siblings of the WebExtBook control).

In the case of the Book control, there is no need to create test objects for all of its children, as described in "Planning Support for the Web Add-in Extensibility Book Sample Toolkit" on page 80.

To prevent UFT from learning all of the descendants of a control supported by Web Add-in Extensibility, you can define a Learn Filter. Complex filters can be implemented using a JavaScript function, in which case you specify the location and name of the function in the toolkit configuration file. Simple filters can be implemented directly in the toolkit configuration file, without using JavaScript functions.

To prevent learning the controls contained in the Book control, a simple filter is sufficient. Before you implement this filter, learn the Web page that contains the Book control with all of its descendants to see that all of the Book's children are learned as well. To do this, you can follow the procedure described in "Deploying and Testing the Toolkit Support Set (for Stage 7)" on the next page.

To prevent learning the controls contained in the Book control:

In the **WebExtSample.xml** file, within the **Control** element defined for the **WebExtBook** test object class, add the following **Filter** element:

```
<Filter>
    <Learn learn_control="Yes" learn_children="No"/>
</Filter>
```

This instructs UFT to learn WebExtBook test objects when learning their parent Web pages, but not to learn the child controls they contain.

Note: You can modify the **WebExtSample.xml** file in the toolkit support set folder and then later deploy it to UFT for testing, or you can modify **<UFT installation folder>\dat\Extensibility\Web\Toolkits\WebExtSample\.xml** directly.

Deploying and Testing the Toolkit Support Set (for Stage 7)

After defining the filter to prevent learning children, you deploy the updated toolkit support set to UFT and test it.

To test the support for learning the WebExtBook test object without its children:

1. Make sure that your most updated files are located in **<UFT installation folder>\dat\Extensibility\Web\Toolkits\WebExtSample**.
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.
4. Open a GUI test and open the Object Repository. Use the **Add Objects to Local**  button in the Object Repository dialog box to learn the Web page that contains the Book control. The Define Object Filter dialog box opens.
5. Select **All object types** and click **OK**. The WebExtBook object named **The History of QuickTest Professional** is added to the object repository, but the controls it contains are not.

Stage 8: 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 "Planning Support for the Web Add-in Extensibility Book Sample Toolkit" on page 80, 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.

  In the toolkit configuration file, you specify the name and, optionally, the location of this JavaScript function.
- 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 Java Add-in Extensibility API Reference* (available with the Web Add-in Extensibility Help).

To develop support for recording on the Book control:

Note: You can modify the **WebExtSample.xml** and **WebExtBook.js** files in the toolkit support set folder and then later deploy them to UFT for testing, or you can modify these files in **<UFT installation folder>\dat\Extensibility\Web\Toolkits\WebExtSample** directly.

1. In the toolkit configuration file, within the **Control** element add the following **Record\EventListening** 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 that you specified at the **Control** level for the WebExtBook test object class.

2. In the **WebExtBook.js** file, add the following **ListenToEvents** function:

```
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.

3. In the **WebExtBook.js** file 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 "Planning Support for the Web Add-in Extensibility Book Sample Toolkit" on page 80.

Deploying and Testing the Toolkit Support Set (for Stage 8)

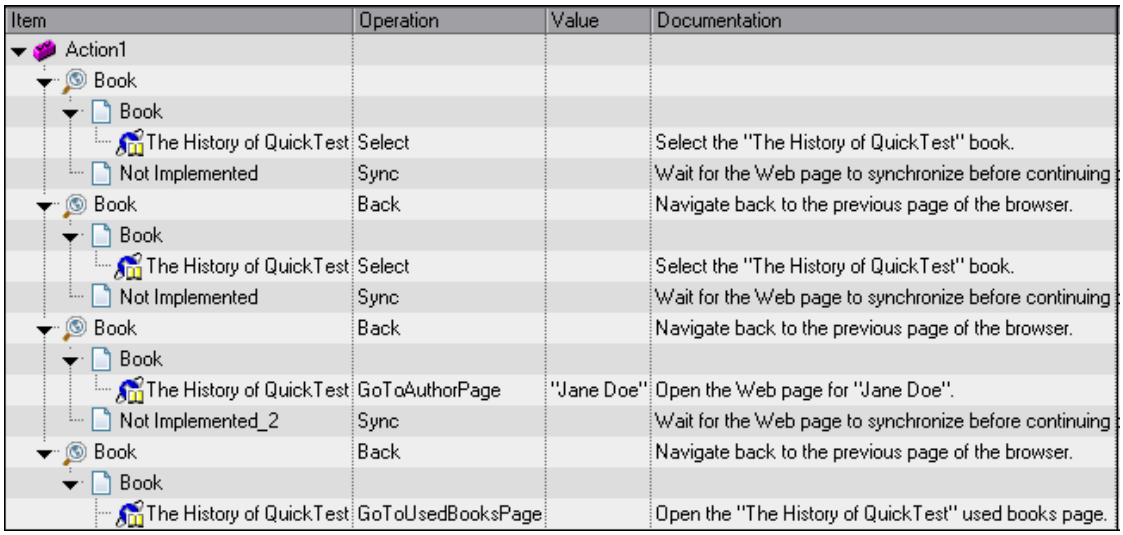
After developing the support for recording on the Book control, you deploy the updated toolkit support set to UFT and test it.

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

1. Make sure that your most updated files are located in **<UFT installation folder>\dat\Extensibility\Web\Toolkits\WebExtSample**.
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.
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:



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 9: Implementing Support for Dynamic List of Values for AuthorName

Using Web Add-in Extensibility, you can provide the UFT user a list of possible values to use for a test object method argument, based on the run-time values of the specific control. For example, the **GoToAuthorPage** test object method of the **WebExtBook** test object class receives an **AuthorName** argument. It is easier for the UFT users if they can select an author name from a list of possibilities instead of typing the name. However, this list is different for each **WebExtBook** control.

In the test object configuration file, you defined the **DynamicListOfValues** attribute for the **AuthorName** argument as **true**, instructing UFT to request the list of possible values from the control when creating a test step.

In the toolkit configuration file, you can specify the file name and function name of the JavaScript function that UFT must call to retrieve the list of values. By default, UFT requests the list of values by calling the **get_list_of_values** JavaScript function from the **WebExtBook.js** file that you specified at the **Control** level for the WebExtBook test object class. UFT calls the JavaScript function for every argument whose **DynamicListOfValues** attribute is set to **true** in the test object configuration file. The parameters provided to this function indicate the test object method and argument for which the values are being requested.

In this section, you implement the **get_list_of_values** JavaScript function, to return the author names from the Book control.

To provide a dynamic list of values for the AuthorName argument in the GoToAuthorPage test object method:

In the **WebExtBook.js** file add the JavaScript functions:

```
// Dynamic list of values implementation
// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
function get_list_of_values( method, argIndex )
{
    // When creating a step with the GoToAuthorPage test
    // object method, provide a list of the authors of this book
    // that can be used for the method's argument.
    if (method == "GoToAuthorPage")
    {
        return get_GoToAuthorPage_list_of_values(argIndex);
    }
    return null;
}
function get_GoToAuthorPage_list_of_values(argIndex)
{
    var arr = new Array();
    if( argIndex > 1 )
        return toSafeArray(arr);
    // Retrieve all authors
    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" )
        {
            arr[AuthorsCount]="\""+_elem.rows[1].cells[0].children
[i].innerText+"\"";
            AuthorsCount++;
        }
    }
    return toSafeArray(arr);
}
```

This returns a list of the book's authors, each enclosed in quotation marks.

The Book custom control is now fully supported, according to the specifications you decided on when planning your custom support.

Note: You can modify the **WebExtSample.xml** file in the toolkit support set folder and then later deploy it to UFT for testing, or you can modify **<UFT installation folder>\dat\Extensibility\Web\Toolkits\WebExtSample\.xml** directly.

Deploying and Testing the Toolkit Support Set (for Stage 9)

After implementing the **get_list_of_values** JavaScript function, you deploy the updated toolkit support set to UFT and test that the dynamic list of author names is properly provided.

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

1. Make sure that your most updated files are located in **<UFT installation folder>\dat\Extensibility\Web\Toolkits\WebExtSample**.
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.
4. Open a GUI test in the Editor and create a step with a **WebExtBook** test object and the **GoToAuthorPage** test object method. When you type a space after the method name, the names of the authors listed on the Book control are displayed in a drop-down list to be used for the method argument.

Lesson Summary

In this lesson you created a new test object class, WebExtBook, defining its identification properties and test object methods. You created support for the Book control, enabling UFT to recognize it as an WebExtBook test object.

- You learned to understand the test object configuration file.
- You learned to understand the toolkit configuration file.
- You learned to support new identification properties and test object methods.
- You learned to create a filter for preventing child controls from being learned.
- You learned to support recording and you made use of the **Record** and **RegisterForEvent** utility methods.
- You learned how to provide a dynamic list of values for a test object argument.

Where Do You Go from Here?

For more information on the structure and content of a toolkit support set, see "Developing Support for Your Toolkit" on page 30.

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

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

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

In the next lesson you learn how to create support for the UsedBooks custom control. The test object class that represents the UsedBooks control extends the existing WebTable test object class. In developing UFT support for this control you will learn to use some of the more advanced options that Web Add-in Extensibility has to offer.

Chapter 6: Learning to Create UFT Support for a Complex Custom Web Control

In this lesson you create support for the UsedBooks control in the Web Add-in Extensibility Book Sample toolkit, which is installed with Extensibility Accelerator for HP Functional Testing. The test object class that represents the UsedBooks control extends the existing WebTable test object class. Creating support for the UsedBooks control teaches you how to use some of the more advanced options of Web Add-in Extensibility.

In the lesson "Learning to Create UFT Support for a Simple Custom Web Control" on page 79, you learned to create support for a simple custom control. You are now familiar with the basics of Web Add-in Extensibility, therefore this lesson explains only the more advanced information.

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. The JavaScript code is not identical to the code you will create, because the sample support set is designed to work on Firefox as well as Internet Explorer, and uses the **jQuery** JavaScript library.

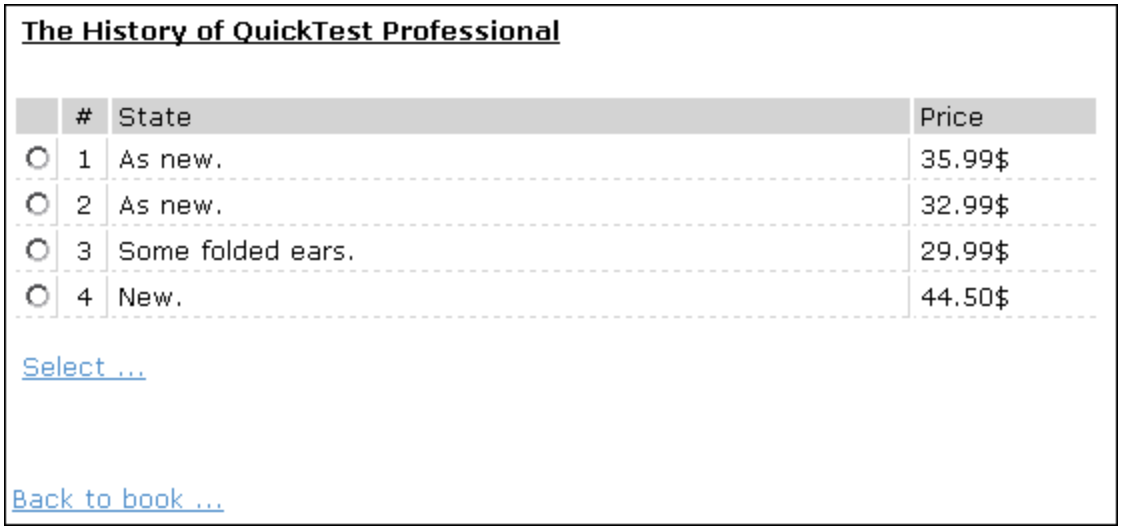
This lesson includes:

Preparing 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 may occur on this control, and so on. You use this information when you design the support.

The source file for the UsedBooks control is located in **%ALLUSERSPROFILE%\Documents\ExtAccTool\Samples\WebExtSample\Application\UsedBooks.htm**.

To run the sample application, open the **%ALLUSERSPROFILE%\Documents\ExtAccTool\Samples\WebExtSample\Application\Book.htm** file. The Book control opens. In the Book control, click **Used** to run the UsedBooks control.



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

The UsedBooks control is implemented as a **div** element that comprises a Web table containing information about the available used copies of this book and radio buttons, and a **Select** link (outside the table element) used to select a book from the list. Selecting a book and opening the page about the selected book (which is not implemented in this sample) requires selecting the radio button in the relevant row in the table and then clicking **Select**.

Planning Support for the Web Add-in Extensibility Sample UsedBooks Control

In this section, you analyze how UFT currently recognizes the UsedBooks control versus the way it should recognize it, based on your knowledge of the control. Next, you determine the answers to the questions

in "Understanding the Web Add-in Extensibility Planning Checklist" on page 25, and fill in the "Web Add-in Extensibility Planning Checklist" on page 27, accordingly.

The best way to do this is to analyze the UsedBooks control from a UFT perspective 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 UsedBooks control.**

   Open UFT and load the Web Add-in.

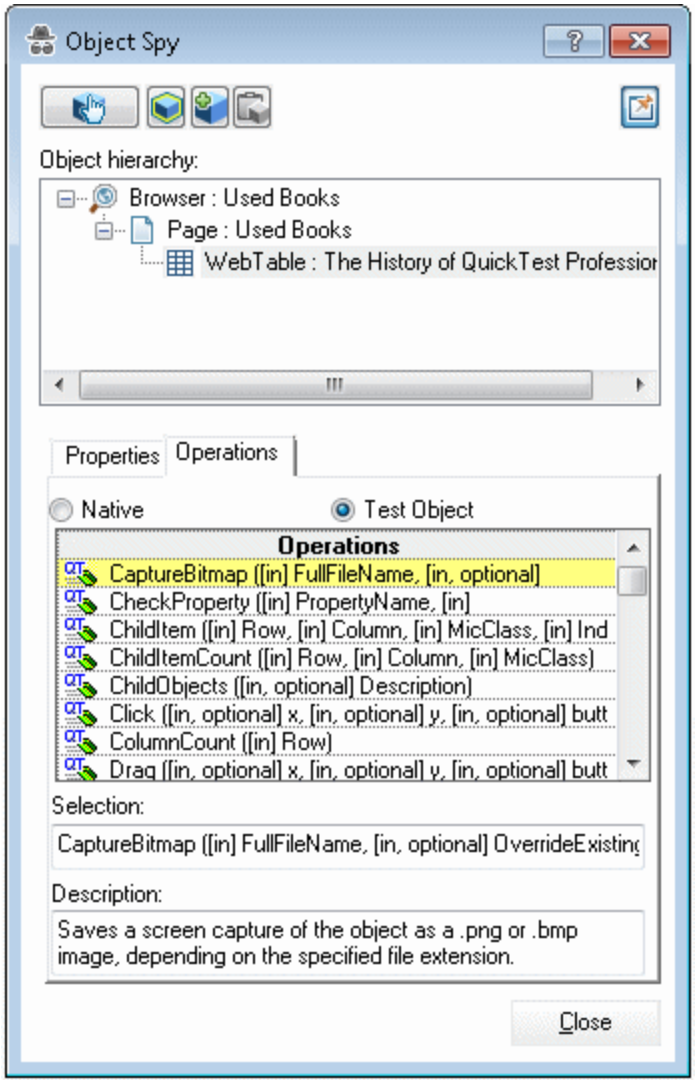
   Close any open instances of the UsedBooks control and open it by opening the **%ALLUSERSPROFILE%\Documents\ExtAccTool\Samples\WebExtSample\Application\Book.htm** file and then clicking **Used** in the Book control that opens.

2. **Use the Object Spy to view the UsedBooks test object 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 **Operations** tab and select **Test Object Operations**.

   In the Object Spy dialog box, click the pointing hand  , then click the UsedBooks table.

The UsedBooks control contains 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 table. The icon used for the test object is the standard WebTable class icon. UFT ignores the **div** element, which is actually the root of the UsedBooks control.



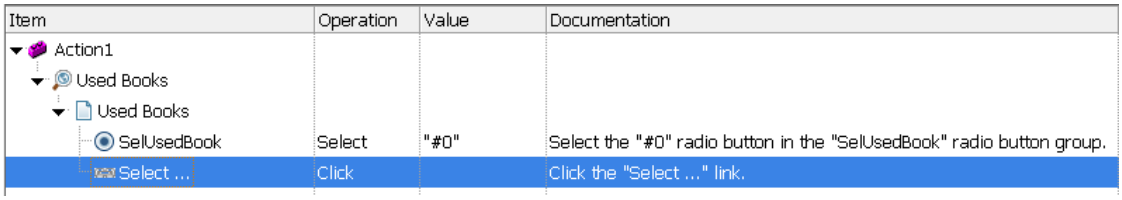
Close the Object Spy.

3. **Record operations on the UsedBooks 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**. In the UsedBooks table, select one of the radio buttons and then click **Select**.

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



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

The recorded steps reflect the selection of the radio button and the clicking of the link separately, and do not recognize these operations as related to the UsedBooks control.

4. **Determine the custom toolkit to which the UsedBooks 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 UsedBooks.

   You created the toolkit support set for this toolkit in the previous lesson. In this lesson you add support for the UsedBooks control in the WebExtSample toolkit support set.

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

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

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

      The internal content of the UsedBooks control is implemented as a Web table control because of the type of information it contains. For the purpose of performing tests on the UsedBooks control and checking the information it contains, it is appropriate that UFT recognize this control as a table. However, to optimally support the UsedBooks control, the test object that represents the control must support a **SelectBook** test object method that selects a book from the table by selecting the radio button in the correct row in the table, and clicking **Select**.

      In addition, because the first row in the UsedBooks table contains the column names, it would be helpful to replace (or override) the **RowCount** test object method supported for WebTable objects to reduce the row count and return the number of used copies available for this book. To support the **SelectBook** test object method and override the implementation of **RowCount**, you create a new test object class named **WebExtUsedBooks**, which extends **WebTable**. You then teach UFT to identify this test object class as the one that represents the UsedBooks control.

   b. Define how UFT will identify which test object class to use to represent the control:

      If the following conditions are met, use a **WebExtUsedBooks** test object to represent the control:

      - The control's **tagName** property is **div**.
      - The **tagName** property of the control's first child is **table**.
      - The **className** property of the control's first child is **UsedBooks**.

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\WebBookList.ico**
- No Help file is provided.
- The **WebExtUsedBooks** test object class needs to support a **title** identification property (used to uniquely identify the control, and selected by default in the checkpoint properties dialog box, not used for Smart Identification).
- 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 **WebExtUsedBooks** test object class needs to support all of the test object operations supported by the **WebTable** test object class. In addition, it needs to support the **SelectBook** test object method.

- All of the operations inherited from the base class, WebTable, can be supported by the **table** element contained in the UsedBooks control. However, because the **table** element is a not the root element of the UsedBooks control, UFT does not recognize this element as the base element. You must implement a JavaScript function that returns the **table** element as the base element for the UsedBooks control. This instructs UFT to use the **table** element to support the operations inherited from the base WebTable test object class.
- The **SelectBook** test object method simulates selecting the radio button for the specified book and clicking **Select**.
- The WebTable test object method **RowCount** needs to be overridden, to return the actual number of books in the table instead of the number of rows.

e. Define which of the control's children UFT should learn when learning the control:

For the purpose of this tutorial, when a **WebExtUsedBooks** test object is learned as part of a Web page, the radio buttons within in should be learned as well.

f. Decide whether the Object Spy should display **WebExtUsedBooks** test objects: Yes.

g. Decide whether to support recording, and what events to record:

Listen to mouse clicks that occur on the **Select** link. When a radio button is selected and this link is clicked, record a test step that selects the book whose radio button is selected.

h. Decide what parts of the support need to be designed in the toolkit configuration file and what parts need JavaScript functions:
   - For the UsedBooks control, test object class identification is performed by a JavaScript function, specified in the toolkit configuration file. To avoid unnecessary calls to the JavaScript function, a condition element is defined in the toolkit configuration file, instructing UFT to call the JavaScript function only if the control is defined as a **div** element.
   - The **table** base element must be returned by a JavaScript function specified in the toolkit configuration file.
   - Test object identification properties can be supported by JavaScript functions using the default naming convention, therefore no changes are required in the toolkit configuration file.
   - The WebTable's **RowCount** test object method is overridden by a new implementation, provided by a JavaScript function named **BookCount**. Therefore, the name of the function needs to be specified in the toolkit configuration file.
   - Filtering the children that are learned with the UsedBooks control is done by calling a JavaScript function that needs to be specified in the toolkit configuration file.
   - To support recording, you modify the toolkit configuration file to turn off the default Web Event Configuration and specify the JavaScript function that registers UFT to listen to the correct event. In addition, you design one JavaScript function that handles event registration, and additional JavaScript functions that instruct UFT to record the relevant steps when the events occur.

Below, 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\UsedBooks.htm`	n/a	n/a
☑	Specify the Web test object base class that the new test object class extends: (Default—WebElement) `WebTable`	n/a	n/a
☑	Is the base test object class WebElement? No If No, is there a base element (an element that matches the base test object class)? `Yes` If there is a base element, do you need a JavaScript function to return it? `Yes`	`Yes`	`Yes`
☑	Specify the New Web test object class details:	n/a	n/a

☑	Custom Control Support Planning Checklist	Specify in Toolkit XML?	Support by JavaScript function?
	• Test object class name: `WebExtUsedBooks` • Icon file location (optional): `<UFT installation folder>\dat\Extensibility\Web\Toolkits\WebExtSample\Res\WebBookList.ico` • Identification properties for description: `title` • Default test object method: `SelectBook` • Help file location: `n/a`		

☑	Specify the basis for identifying the test object class to use for the control:
	`tagName = div`
	`tagName of 1st child = table`
	`className of 1st child = UsedBooks.`
		Yes	Yes

☑	Specify the basis for naming the test object: `Use the UsedBooks table title`	n/a	Yes
☑	List the identification properties to support, and mark which should be available (and which selected by default) for checkpoints and which (if any) should be used for Smart Identification: `title (available for checkpoints and selected by default, not 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):
	`SelectBook (BookIndex)`
	`RowCount`
		Yes	Yes

☑	Provide a dynamic list of values for any test object method arguments? No If so, list the arguments.	n/a	No
☑	Specify the types of children that UFT should learn with the control: `Radio buttons`	`Yes`	Yes
☑	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:
	`Click on Select`
		`Yes`	`Yes`

Developing the Toolkit Support Set

Follow the steps in this section to develop the toolkit support set for the WebExtSample toolkit and learn more about Web Add-in Extensibility. Developing this toolkit support set comprises the following

stages:

Stage 1: Expanding the Toolkit Support Set to Support an Additional Control

To add support for the UsedBooks control, you first add the definition for the **WebExtUsedBooks** test object class to the **WebExtSampleTestObjects.xml** file.

To expand the toolkit support set to support the UsedBooks control:

1. Copy the icon file for the UsedBooks control, **WebBookList.ico**, from **%ALLUSERSPROFILE%\Documents\ExtAccTool\Samples\WebExtSample\Res** to the **<toolkit support set folder>\Toolkits\WebExtSample\Res** folder.
2. Add the following definition for the **WebExtUsedBooks** test object class to the **WebExtSampleTestObjects.xml** file (within the **TypeInformation** element, after the ClassInfo element for the WebExtBook test object class):

```
<ClassInfo BaseClassInfoName="WebTable"
           GenericTypeID="Table"
           Name="WebExtUsedBooks"
           DefaultOperationName="SelectBook">
 <IconInfo IconFile="INSTALLDIR\dat\Extensibility\Web\Toolkits\
           WebExtSample\Res\WebBookList.ico"/>
   <TypeInfo>
      <Operation ExposureLevel="CommonUsed"
                 Name="SelectBook"
                 PropertyType="Method">
         <Description>
             Selects the radio button for the specified book and clicks
Select.
         </Description>
         <Documentation>
            <![CDATA[Select the radio button for the book with index %a1 and
click Select.]]>
         </Documentation>
```

```
        <Argument Name="BookIndex"
                  IsMandatory="true"
                  Direction="In">
           <Type VariantType="Integer"/>
        </Argument>
      </Operation>
   </TypeInfo>
   <IdentificationProperties>
      <IdentificationProperty  ForDefaultVerification="true"
                               ForVerification="true"
                               ForDescription="true"
                               Name="title"/>
   </IdentificationProperties>
</ClassInfo>
```

This defines the **WebExtUsedBooks** test object class according to the details described in the "Web Add-in Extensibility Planning Checklist" on page 116.

For more information 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).

Stage 2: Teaching UFT to Identify, Spy, and Learn the UsedBooks Control

After you define the new test object class you must enable UFT to identify the Web controls for which to use this test object class.

As described in "Planning Support for the Web Add-in Extensibility Sample UsedBooks Control" on page 111, a **WebExtUsedBooks** test object is used to represent a control whose **tagName** property is **div**, if the **tagName** and **className** properties of the control's first child are **table** and **UsedBooks** respectively.

For the **WebExtUsedBooks** test object class, identification is carried out by a combination of **Condition** elements in the toolkit configuration file and a JavaScript function.

To define the identification rules for the WebExtUsedBooks test object class:

1. In the **WebExtSample.xml** file, within the **Controls** element, add the following **Control** element for this test object type:

```
<Control TestObjectClass="WebExtUsedBooks">
    <Settings>
        <variable name="default_imp_file"    value="WebExtUsedBooks.js"/>
    </Settings>
    <Identification type="javascript"
```

```
                function="IsWebExtUsedBooks">
      <Browser name="*">
        <Conditions type="CallIDFuncIfPropMatch" logic="and">
            <Condition prop_name="tagName"    expected_value="div"/>
        </Conditions>
      </Browser">
    </Identification>
</Control>
```

This defines that UFT will look for JavaScript functions in the file **WebExtUsedBooks.js** unless another file is specified. The **Identification** element includes one **Conditions** element that specifies that if the **tagName** property of the control being handled is **div** (case-insensitive compare), the JavaScript function **IsWebExtUsedBooks** is called to identify whether to use this test object class to represent the control.

This tutorial uses the definition above to illustrate the use of the `CallIDFuncIfPropMatch` value for the **Conditions** element's **Type** attribute. However, if you were working with an application that had many controls on a page, or a large DOM structure, a better way to define these identification rules would be to use the following text:

```
    <Identification type="javascript"
                function="IsWebExtUsedBooks">
        <HTMLTags>
            <Tag name="div"/>
        </HTMLTags>
    </Identification>
```

This provides the same functionality, instructing UFT to call the **IsWebExtUsedBooks** identification function only for div elements, but it provides better performance when learning custom controls and running steps on them.

2. In the toolkit support set folder, in the **Toolkits\WebExtSample** folder, create a file named **WebExtUsedBooks.js** (This is the file for all of the JavaScript functions you design to support the UsedBooks control).
3. In **WebExtUsedBooks.js**, add the following JavaScript function:

```
function IsWebExtUsedBooks()
{
    // Verify that the tagName property is "div" and the
    // className property of the first child (a TABLE element)
    // is "UsedBooks".
    var firstChild = _elem.children[0];
    if    ( _elem.tagName == "DIV" &&
        firstChild.tagName == "TABLE" &&
        firstChild.className == "UsedBooks" )
```

```
        return true;
    return false;
}
```

This JavaScript function checks whether the control meets the conditions that determine that a control should be represented by a **WebExtUsedBooks** test object.

Deploying and Testing the Toolkit Support Set (for Stage 2)

After defining the WebExtUsedBooks test object class in the test object configuration file and the identification rules for this test object class in the toolkit configuration file and JavaScript functions, you can test the effect of using the toolkit support set with UFT.

To test the toolkit support set:

1. Deploy the test object configuration file, toolkit configuration file, icon file, and JavaScript file to their correct locations within the UFT installation folder.
2. Open UFT and load the **WebExtSample** support (select it in the Add-in Manager dialog box).
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. Select the WebExtSample environment from the **Environment** list to see that the **WebExtUsedBooks** test object class you defined in the test object configuration file is displayed in the **Class** list.
4. Run the sample control by opening the **%ALLUSERSPROFILE%\Documents\ExtAccTool\Samples\WebExtSample\Application\Book.htm** file and clicking **Used**.

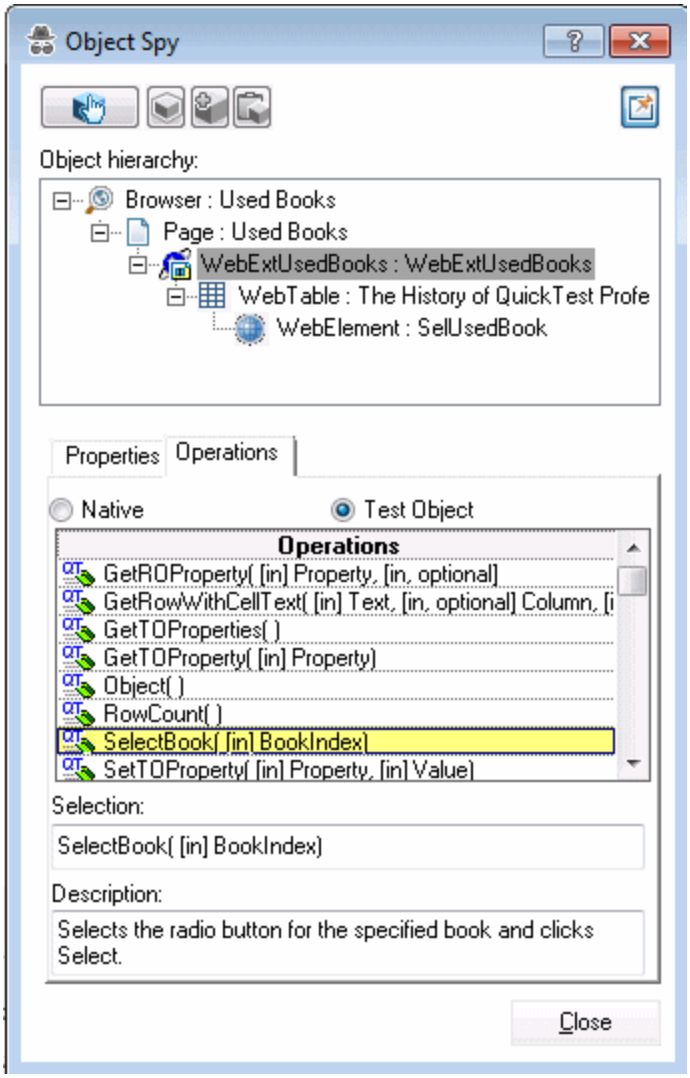
   Note: UFT establishes its connection with an application when the application opens. Therefore, if the UsedBooks control is open, you must close it and run it again.

5. In UFT, perform the following activities on the UsedBooks control to see how UFT recognizes the control. (For more information 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 UsedBooks control. No value is displayed for the **title** property because you have not yet implemented a JavaScript function that returns its value.

     The test object created for the UsedBooks control is given the name of its test object class, and uses the custom icon you defined. Later in this lesson, you customize your toolkit support set to provide a more specific name.

The **WebExtUsedBooks** test object includes all of the test object operations of a WebTable test object, as well as the SelectBook method that you defined in the test object configuration file.



- Use the **Add Objects to Local**  button in the Object Repository dialog box to learn the UsedBooks control. The custom icon is used to represent the test object in the object repository.
- In the Keyword View, create a test step choosing the **WebExtUsedBooks** object from the object repository in the **Item** column.
  - The list of available operations in the **Operation** column reflects the definitions in the test object configuration file. All of the test object operations supported by WebTable test objects are available, because in the test object configuration file, you defined that the WebExtUsedBooks test object extends (and therefore inherits from) the WebTable test object class.
  - After you choose an operation, the **Value** cell is partitioned according to the number of

arguments of the selected operation. For example, when you create a step with the operation **SelectBook**, the value cell requires one argument and displays the argument's Name attribute in a tooltip.

- The descriptions and documentation strings you defined for test object methods in the test object configuration file are displayed in tooltips and in the **Documentation** column, respectively.

- 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 inherited from WebTable.

6. Run a test with a step that performs the SelectBook test object method on a WebExtUsedBooks test object. UFT searches for a JavaScript function that will run the test object method on the control. Because you have not yet implemented support for running test object methods, a run-time error occurs. In the next section, you implement this support.

Stage 3: Implementing Support for the WebExtUsedBooks Test Object Methods

In the test object configuration file you defined the test object methods available for WebExtUsedBooks test objects. For UFT to run these test object methods, the methods must actually be implemented.

You must provide implementation for different types of test object methods:

- Test object methods inherited from the WebTable base test object class
- Test object methods added for the new test object class
- Test object methods inherited from the base class that need to be implemented differently

Implementing Test Object Methods Inherited from WebTable

In the test object configuration file, you defined that the WebExtUsedBooks test object class extends the base class WebTable. For the inherited WebTable test object methods that you do not override, UFT can use its internal implementation by interacting with the **table** base element defined within the UsedBooks control. Because the **table** element is not the root level of the UsedBooks control, you must inform UFT that the **table** element is the base element. To do this you must write a JavaScript function that returns the base element, and specify its name in the toolkit configuration file.

To instruct UFT to use the table Web element as the base element:

1. In the **WebExtSample.xml** file, within the **Settings** element that you defined in the **Control** element for the WebExtUsedBooks test object class, add the following **Variable** element:

```
<Control TestObjectClass="WebExtUsedBooks">
    <Settings>
        <variable name="func_to_get_base_elem"
                  value="GetTableElem"/>
```

```
    </Settings>
</Control>
```

This instructs UFT to call a JavaScript named **GetTableElem** (in the file **WebExtUsedBooks.js**) to return the base element that supports the inherited WebTable test object methods.

2. In the **WebExtUsedBooks.js** file, add the following JavaScript function:

```
function GetTableElem()
{
    // Get the <table> element (the first child of the <div>
    // element which is the root of the UsedBooks control)
    return _elem.children[0];
}
```

This JavaScript function returns the **table** element, which is the first element within the **div** element that defines the UsedBooks control. This element supports the test object methods inherited from WebTable that are not implemented by WebExtUsedBooks.

Other JavaScript functions that you write in this file can also use the **GetTableElem()** function to access the table element in the UsedBooks control.

Implementing the New Test Object Method SelectBook

To support the **SelectBook** test object method for the **WebExtUsedBooks** test object class, write the **SelectBook** JavaScript function in **WebExtUsedBooks.js**. This is the function that UFT calls to run the **SelectBook** test object method. It simulates selecting the radio button for the specified book, and clicking **Select**.

Add the following JavaScript function to the **WebExtUsedBooks.js** file:

```
function SelectBook( BookIndex )
// Select the radio button for the specified index
// and clicks the "Select" link.
{
    if( BookIndex > BookCount() )
        throw "Book index is out of range !"
    //Select the radio button corresponding to the specified index
    GetTableElem().rows[1+BookIndex].cells[0].children[0].click();
    //Click the "Select" link (the 3rd child of the <div> element)
    _elem.children[2].click();
    //Add a log message to the event log to assist in debugging
    _util.LogLine("Book Selected",1);
}
```

Overriding the Implementation of the Inherited Test Object Method RowCount

1. In the **WebExtSample.xml** file, add the following **Run** element within the **Control** element that defines the **WebExtUsedBooks** support.

```
<Run>
    <Methods>
        <Method name="RowCount" type="javascript"
               function="BookCount" />
    </Methods>
</Run>
```

   This defines that the **RowCount** test object method is implemented by the JavaScript function **BookCount**.

2. In the **WebExtUsedBooks.js** file, add the **BookCount** JavaScript function, which decreases the row count of the UsedBooks control, to return the number of books in the table:

```
function BookCount()
// This function overrides the RowCount test object method
// inherited from WebTable, so that it counts only book rows.
{
    var table = GetTableElem();
    if( table.rows.length < 2 )
        return 0;
    return table.rows.length - 2;
}
```

Deploying and Testing the Toolkit Support Set (for Stage 3)

After you develop support for running the test object methods, you deploy the updated toolkit support set to UFT and test it.

To test the support for running test object methods:

1. To deploy the updated toolkit support set to UFT, copy the **WebExtUsedBooks.js** file (and **WebExtSample.xml** if necessary) to **<UFT installation folder>\dat\Extensibility\Web\Toolkits\WebExtSample.**
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 **UsedBooks** sample control.
4. Open a GUI test and use the **Add Objects to Local**  button in the Object Repository dialog box to learn the UsedBooks control.
5. Create a test that runs the **SelectBook** and **RowCount** methods and make sure they perform correctly. You can also open the Microsoft Windows Event Viewer and view the log message added

by the **SelectBook** method (for more information, see "Using the Microsoft Windows Event Log" on page 58).

Stage 4: Implementing Support for the WebExtUsedBooks Identification Properties and the Test Object Name

In the **WebExtUsedBooks.js** file, implement the **get_property_value** as follows:

```
function get_property_value(prop)
// The function provides values for all of the identification
// properties defined in the test object configuration XML file, as
// well as the hard coded "logical_name" property that UFT
// uses to name the test object.
{
    if ( prop == "logical_name" || prop == "title" )
    // For the "title" identification property, as well as the
    // "logical_name" property, return the inner text of the
    // first cell in the first row
    {
        return GetTableElem().rows[0].cells[0].innerText;
    }
}
```

This function returns the title of the object for the **title** property, as well as for the test object name.

Note: You can modify the **WebExtUsedBooks.js** file in the toolkit support set folder and then later deploy it to UFT for testing, or you can modify **<UFT installation folder>\dat\Extensibility\Web\Toolkits\WebExtSample\WebExtUsedBooks.js** directly.

Deploying and Testing the Toolkit Support Set (for Stage 4)

After you develop support for retrieving run-time values of identification properties, you deploy the updated toolkit support set to UFT and test it.

To test the support for retrieving run-time values of identification properties:

1. Make sure that your most updated files are located in **<UFT installation folder>\dat\Extensibility\Web\Toolkits\WebExtSample**.
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 **UsedBooks** sample control.
4. Create a new GUI test, add a WebExtUsedBooks test object to your object repository, and create a test step with this test object. Make sure that the test object name is based on the table's title. Right-click the object and select **Insert Standard Checkpoint**. The Checkpoint Properties dialog box

opens. Make sure that the **title** identification property you defined in the test object configuration file is included in the list of properties and selected.

5. Create and run a test that retrieves each identification property and checks its value, or displays it in a message box.

Stage 5: Implementing a Filter to Prevent Learning Child Objects

In this section, you create a filter to prevent UFT from learning all of the UsedBooks control's children along with the control.

You implement this in the toolkit configuration file and in the JavaScript file.

To filter the children learned with the UsedBooks control:

1. In the **WebExtSample.xml** file, within the **Control** element defined for the **WebExtBook** test object class, add the following **Filter** element:

```
<Filter>
    <Learn learn_control="Yes" learn_children="CallFilterFunc"
          type="javascript" function="GetChildrenToLearn" />
</Filter>
```

This instructs UFT to learn **WebExtUsedBooks** test objects when learning their parent Web pages, and to call the JavaScript function **GetChildrenToLearn** to determine which children to learn. The JavaScript function returns a SafeArray of the controls descendants that should be learned with the control.

Note: You can modify the **WebExtSample.xml** file in the toolkit support set folder and then later deploy it to UFT for testing, or you can modify **<UFT installation folder>\dat\Extensibility\Web\Toolkits\WebExtSample\.xml** directly.

2. In the **WebExtUsedBooks.js** file, add the following functions:

```
// Learn filtering
// This function instructs UFT which child objects of a
// UsedBooksTable should be learned with the object is learned.
function GetChildrenToLearn()
{
    // Return all of the radio buttons in the UsedBooks table
    return toSafeArray(GetTableElem().getElementsByTagName("input") );
}
```

This ensures that only the radio buttons are learned, as planned in the outset of this lesson.

Deploying and Testing the Toolkit Support Set (for Stage 5)

After defining the filter to customize learning children, you deploy the updated toolkit support set to UFT and test it.

To test the support for learning the WebExtUsedBooks test object without its children:

1. Make sure that your most updated files are located in **<UFT installation folder>\dat\Extensibility\Web\Toolkits\WebExtSample**.
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 **UsedBooks** sample control.
4. Open a GUI test and open the Object Repository. Use the **Add Objects to Local**  button in the Object Repository dialog box to learn the Web page that contains the UsedBooks control. The Define Object Filter dialog box opens.
5. Select **All object types** and click **OK**. The WebExtUsedBooks object named **The History of QuickTest Professional** is added to the object repository, as is the SelUsedBook radio button group. However, none of the other elements contained in the control are learned.

Stage 6: Implementing Support for Recording on the UsedBooks Control

In this section, you implement support for recording on the UsedBooks control.

1. In the **WebExtSample.xml** file, and the following **Record** element within the **Control** element that defines the WebExtUsedBooks class:

```
<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 to record events on the UsedBooks control, but to call the **ListenToEvents** JavaScript function instead.

   In the **WebExtUsedBooks.js** file add the **ListenToEvents** JavaScript function:

```
function ListenToEvents( elem )
{
    // Connect to the "Select" event:
    //When "Select" is clicked, call OnSelectUsedBooksClicked.
```

```
    _util.RegisterForEvent(_elem.children[2], "onclick",
                           "OnSelectUsedBooksClicked" );
    return true;
}
```

This function registers UFT to listen to clicks on the Select link, and call the appropriate event handler when the event occurs.

2. In the **WebExtUsedBooks.js** file add the **OnSelectUsedBooksClicked** event handling JavaScript function:

```
function OnSelectUsedBooksClicked( handlerParam , eventObj )
{
    var arr = new Array();
    var booksCount = BookCount();
    // Find the index of the selected radio button and record
    // a step that runs the SelectBook test object method
    // with that index.
    var BookIndex = -1;
    for( var i = 0 ; i < booksCount ; i++ )
    {
        if( _elem.rows[2+i].cells[0].children[0].status == true )
        {
            // This is the selected item
            arr[0] = i+1;
            _util.Record( "SelectBook", toSafeArray(arr) , 0 );
            _util.LogLine("SelectBook Recorded",1);
            break;
        }
    }
    return true;
}
```

This function checks which book's radio button is selected, and instructs UFT to record a step selecting that book (and add the relevant log message to the event log).

Deploying and Testing the Toolkit Support Set (for Stage 6)

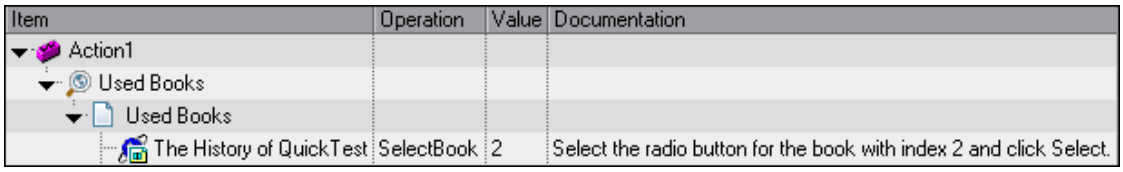
After developing the support for recording on the UsedBooks control, you deploy the updated toolkit support set to UFT and test it.

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

1. Make sure that your most updated files are located in **<UFT installation folder>\dat\Extensibility\Web\Toolkits\WebExtSample**.
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 **UsedBooks** sample control.
4. Open a GUI test and click the **Record** button or select **Record > Record**. In the UsedBooks table, select one of the radio buttons and then click **Select**.

   A new step is added to the test only after you click **Select**:



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

The Book custom control is now fully supported, according to the specifications you decided on when planning your custom support.

Lesson Summary

In this lesson you:

- created a test object class, WebExtUsedBooks, that extends the WebTable test object class.
- created support for the UsedBooks control, enabling UFT to recognize it as a WebExtUsedBooks test object.
- learned to understand more options in the toolkit configuration file.
- learned to implement the support using more complex JavaScript functions and specifying their location in the toolkit configuration file.

Where Do You Go from Here?

Now that you have performed the lessons in this tutorial, you are ready to apply the Web Add-in Extensibility concepts and the skills you learned to creating your own custom toolkit support.

For more information on the structure and content of a toolkit support set, see "Developing Support for Your Toolkit" on page 30.

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

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

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

Send Us Feedback

Can we make this Developer Guide better?

Tell us how: sw-doc@hp.com


Java™
COMPATIBLE