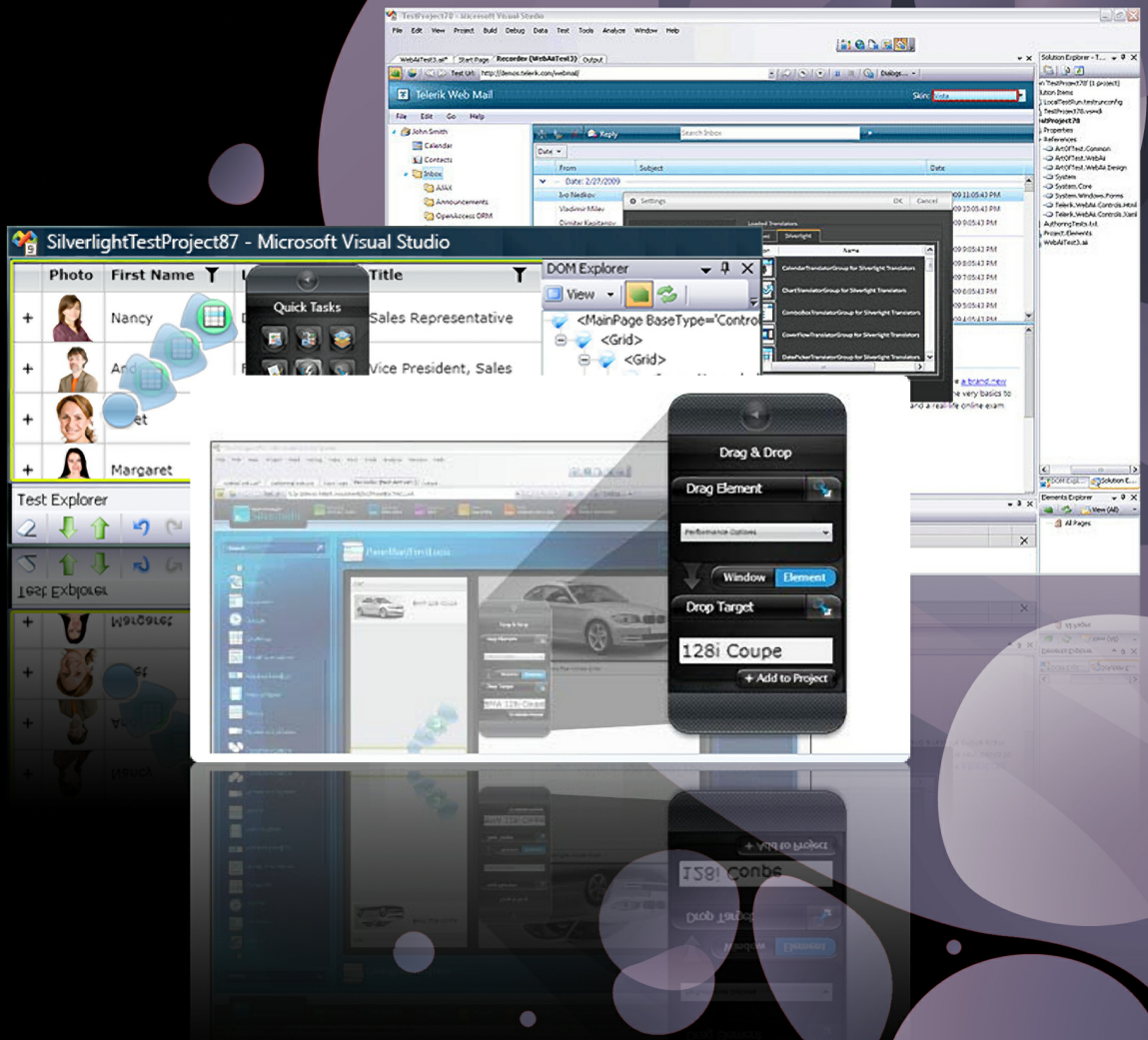
# WEBUI TEST STUDIO

## Developer Edition made easy




Written by: Noel Rice and Lino Tadros

www.falafel.com
FALAFEL Software

# WebUI Test Studio Developer Edition Made Easy

---

***by Falafel Software Inc.***

*Welcome to WebUI Test Studio Developer Edition Made Easy.*

*We hope you enjoy the book as much as we, at Falafel Software, enjoyed creating it.*

# WebUI Test Studio Developer Edition Made Easy



Printed: April 2010

**Publisher**
*Falafel Software Inc.*

**Authors**
*Noel Rice*
*Lino Tadros*

**Technical Editors**
*Lino Tadros*
*Noel Rice*

**Cover Designer**
*Matt Kurvin*

**Team Coordinator**
*Lino Tadros*

**Production**
*Falafel Software Inc.*

**Special thanks to:**

*All the team members at Telerik worldwide for creating a magnificant piece of software in WebUI Test Studio for Developers. The authors also would like to thank the Falafel team members in Colorado, Texas, Michigan, North Carolina and California for their feedback, guidance and recommendations on the subjects of the courseware*

*Falafel would like to thank Faris Sweis, Chris Eyhorn, Vassil Terziev and Svetozar Georgiev for their trust and belief in the quality of Falafel Software's work.*

*Falafel would like to thank Todd Anglin and Gabe Sumner for their support.*

*Last but not least, thank you to all our families for their support and patience while we wrote the book.*

# Table of Contents

# Part I

Introduction

# 1 Introduction

## 1.1 Who Should Read This Courseware

This courseware assumes that you are familiar with VB.NET or C# code. The courseware uses Visual Studio 2008/2010 and assumes you know your way around one of these environments. You should be able to navigate the basic functional areas of the IDE (e.g. Solution Explorer, Properties, code/designer web pages etc.) and be able to run applications.

This courseware assumes some familiarity with testing and focuses on adapting your existing skills to the WebUI Test Studio product.

## 1.2 About Telerik

Telerik is a leading vendor of development, automated testing, and team productivity tools, as well as UI components and content management solutions for Microsoft .NET. Created with passion, Telerik products help software development teams every day to be more productive and to deliver reliable applications on time and under budget. Telerik was founded in 2002 by a few friends with a simple idea – to “deliver more than expected”. Nowadays, a market leader with a team of more than 220 professionals spread around the globe in 5 offices, Telerik is still true to its motto – building outstanding products and serving customers with fanatical dedication.

## 1.3 About Falafel

Founded in 2003, Falafel Software, Inc. provides the highest quality software development, consultation, and training services available. Starting initially with consulting and training, Falafel Software found itself expanding rapidly on the excellence of its engineers and the incredible sense of teamwork exhibited by everyone in the company. This common mutual respect for each other's talents has been a major asset for Falafel, causing extraordinary growth, and a level of quality that very few other IT companies can match. Employees include best-selling authors, industry speakers, technology decision makers, and former Microsoft and Borland engineers. All of Falafel engineers are Microsoft Certified Professionals, Certified Application Developers, or Most Valuable Professionals.

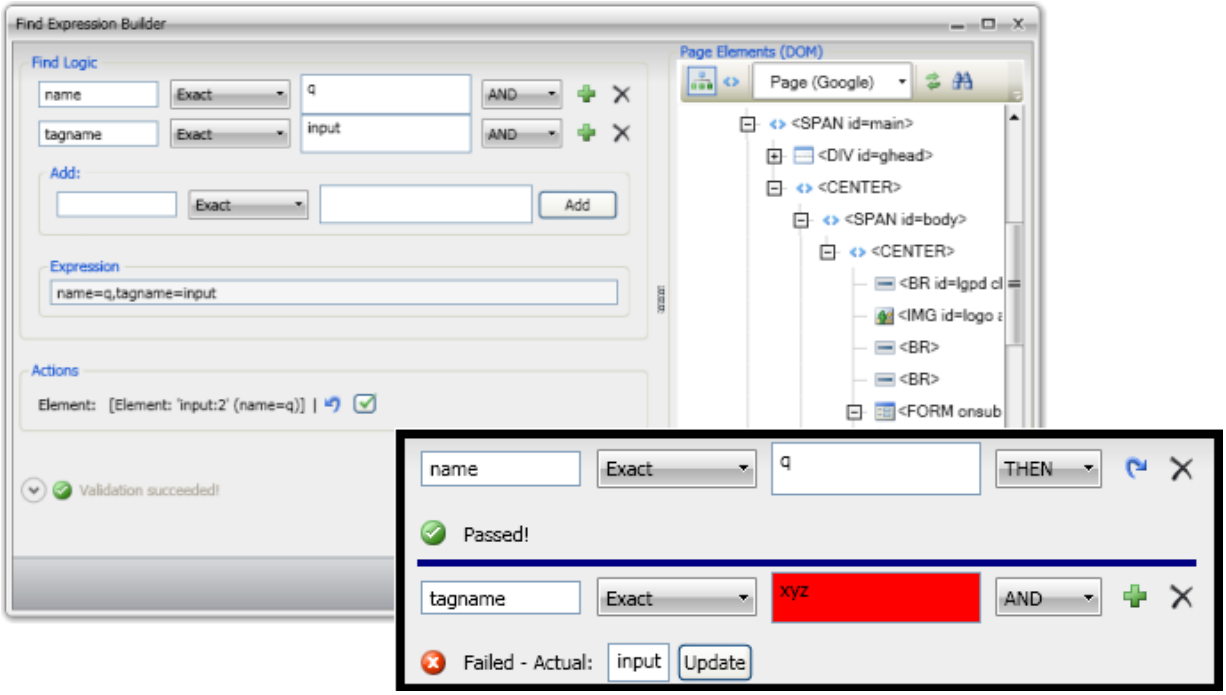
Falafel has written the following Telerik courseware:

- RadControls for ASP.NET
- RadControls for ASP.NET AJAX
- RadControls for Winforms
- RadControls for Silverlight
- Telerik Reporting
- Telerik OpenAccess ORM
- Sitefinity User Manual

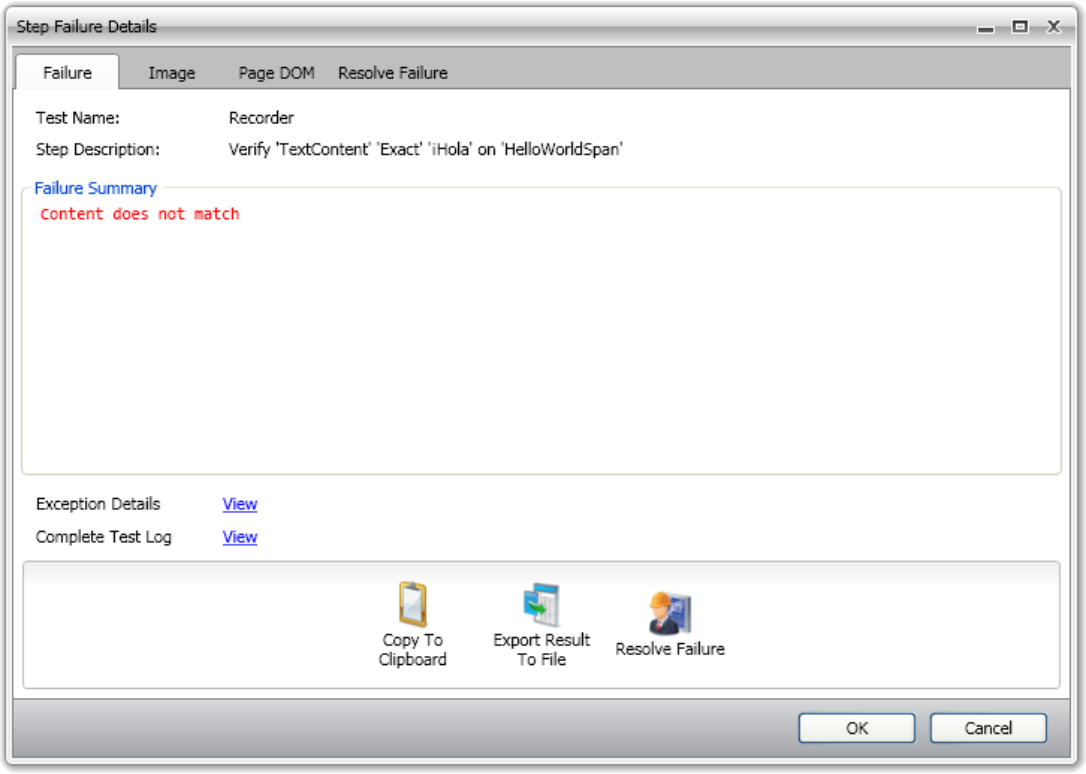
## 1.4 Introducing WebUI Test Studio

Telerik WebUI Test Studio delivers dramatically better productivity than code-based frameworks thanks to its robust test recording surface. WebUI Test Studio requires no coding. Navigate, point and click is all it takes to generate most automated tests. Telerik automated testing tools come with proprietary tools to further enhance your productivity:

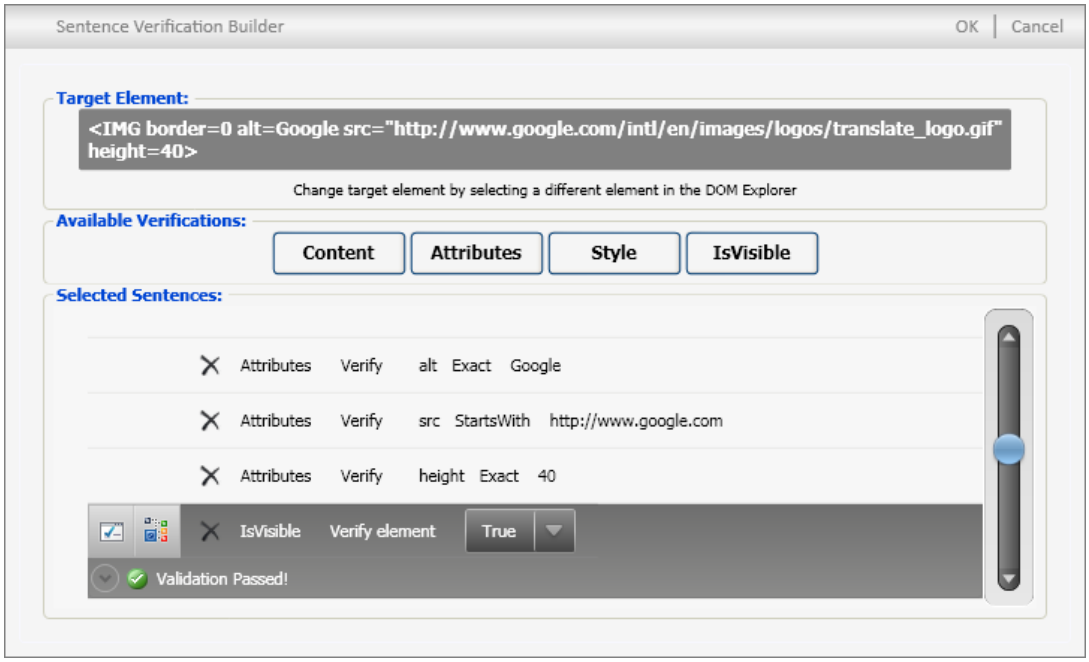
- No more wasted time on determining coordinates. A unique configurable algorithm is used to automatically determine the best parameters to use to locate a specific element on a page. Fine tune the find criteria for special cases. Intelligent tools help you correct problem situations.



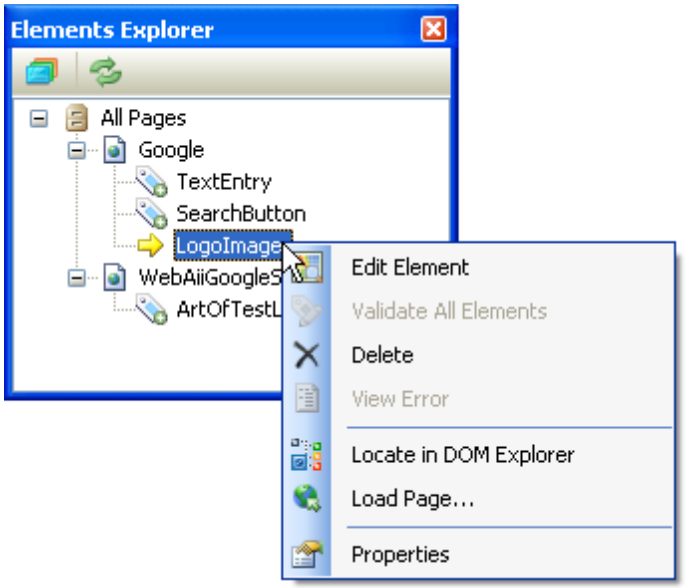
- Telerik WebUI Test Studio handles failure resolution of tests on a completely new level. Its capture feature allows you to see the DOM in its recorded and executed versions and quickly spot the reason for the failure. Additionally, when changing elements in a test, you can simply check if the change is valid without running the whole test.



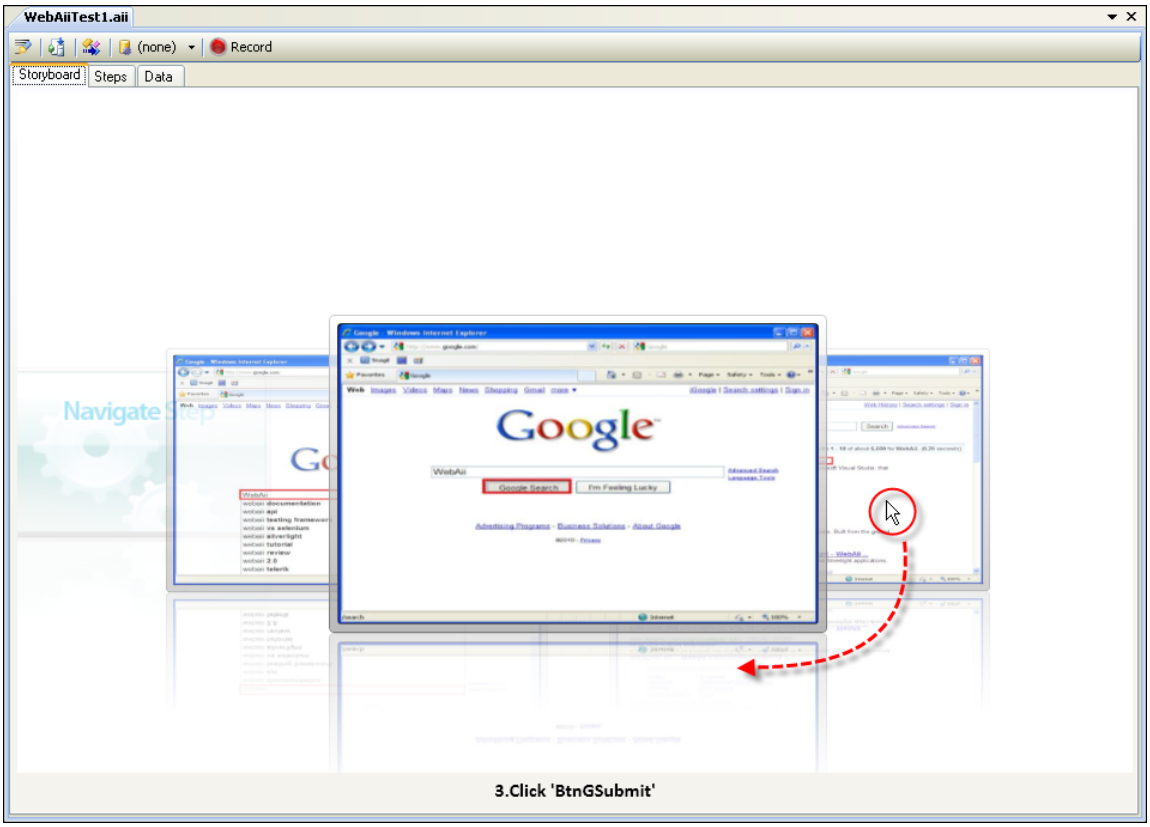
- You don't need to duplicate your tests for different browsers anymore. You can have your test recorded just once and played on multiple browsers without re-recording.
- To make crafting verification and synchronization as simple as possible, we present an innovative adaptive wizard that guides you through crafting verifications and test synchronization with elements. Using the "Sentence Based" UI you can craft a wide range of verification types.




- All web page elements targeted for automation in your tests are abstracted out and filed in the "Elements Explorer". If there are multiple actions that use the same element, the element is referenced from the "Elements Explorer" instead of being duplicated in the test. This enables you to maintain only one unique element and update it in the event of a required change instead of having to modify multiple duplicate elements.



- As you record your test, a screenshot of your action on the target element is captured to the Visual Storyboard. This gives you a visual flow of how your test has progressed. It is a great time saver in helping others understand the state of the test at the time of recording along with what the target elements of the tests were.



# Part II

Installation

# 2 Installation

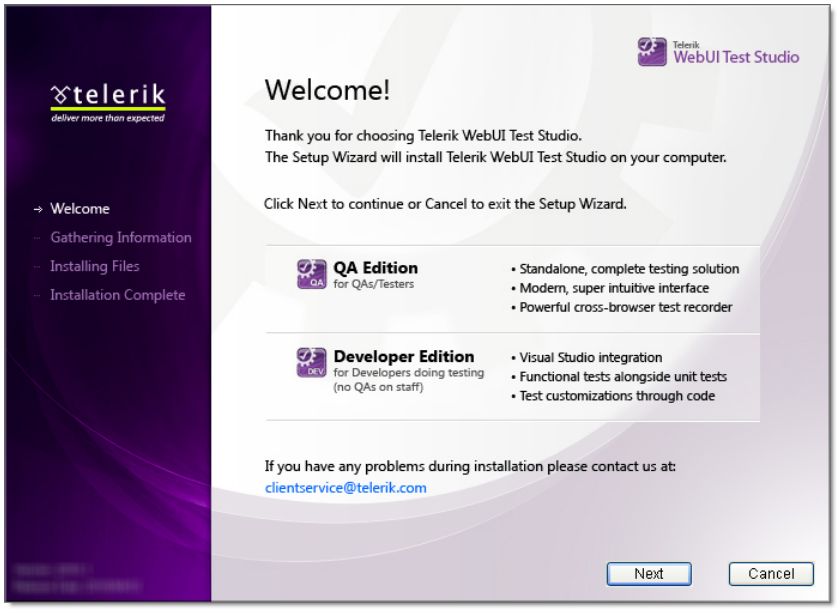
## 2.1 Objectives

In this chapter you will learn how to install Telerik WebUI Test Studio Developer Edition.

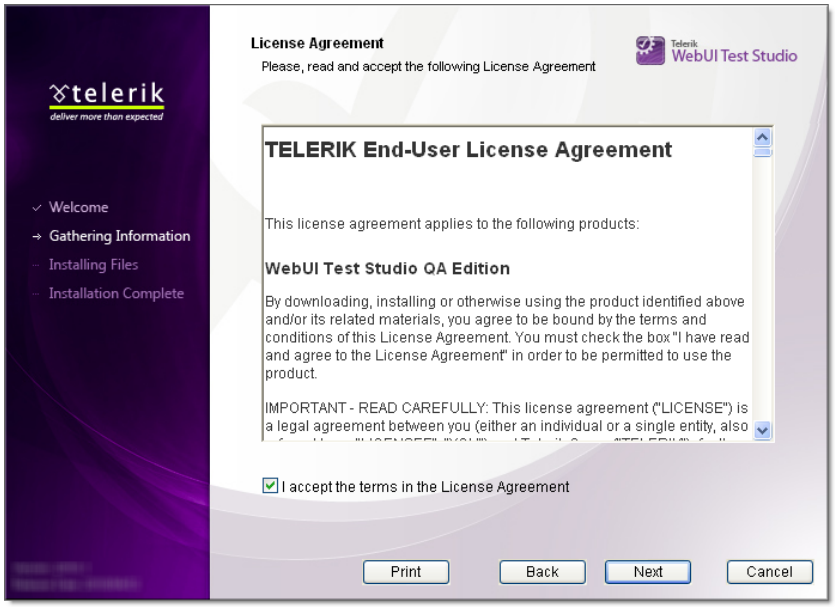
## 2.2 Install WebUI Test Studio

1) To install Telerik WebUI Test Studio Developer Edition, run the installation executable and follow the prompts in the wizard.

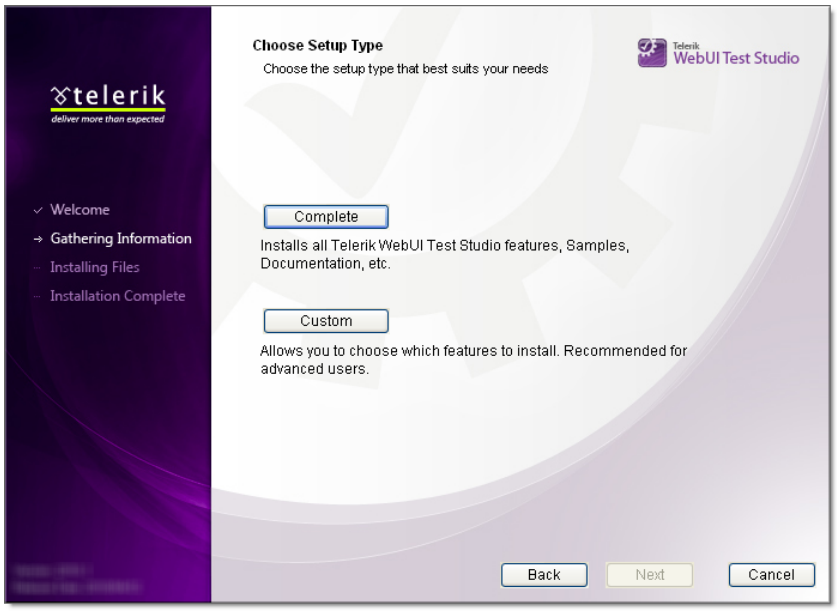
2) When you first run the installation, the setup wizard dialog displays a welcome message. Click the **Next** button to continue.




3) The next page of the setup wizard displays the "End-User License Agreement". Review the license agreement and if you approve the agreement, click the **"Accept the terms in the License Agreement"** checkbox. Click the **Next** button to continue.

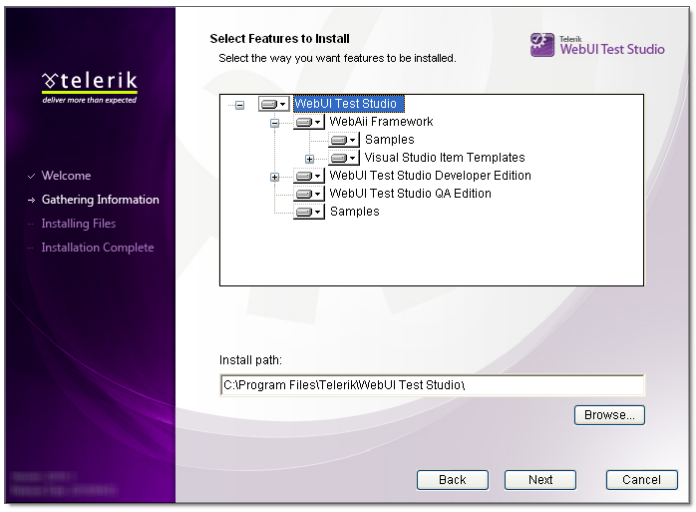


4) The next page of the setup wizard displays the "Choose Setup Type" page. Click either the **Complete** button to install all features and additional material. Click the **Custom** button if you want to choose which features you want installed.

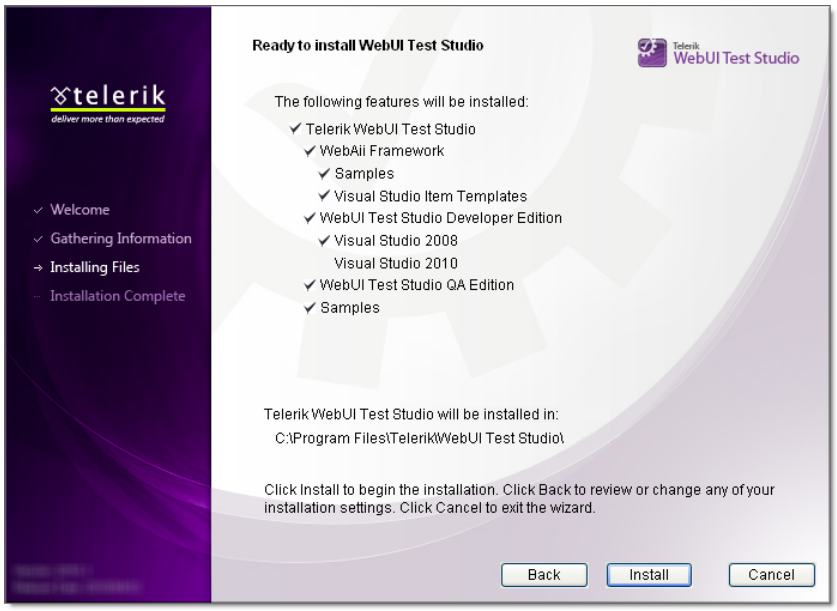



### *Notes*

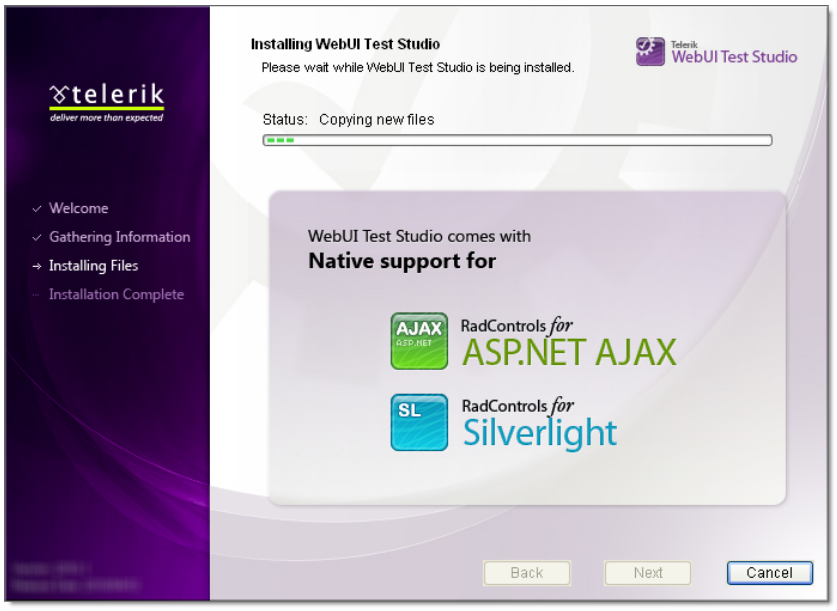
If you choose the **Custom** button, an additional page, "Select Features to Install", will appear. Use the drop down list next to each feature to tailor your installation. You can also specify an **Install Path** that is different from the default. You can leave the default path, enter a new path directly or click the **Browse...** button to select a new path. Click the **Next** button to continue.



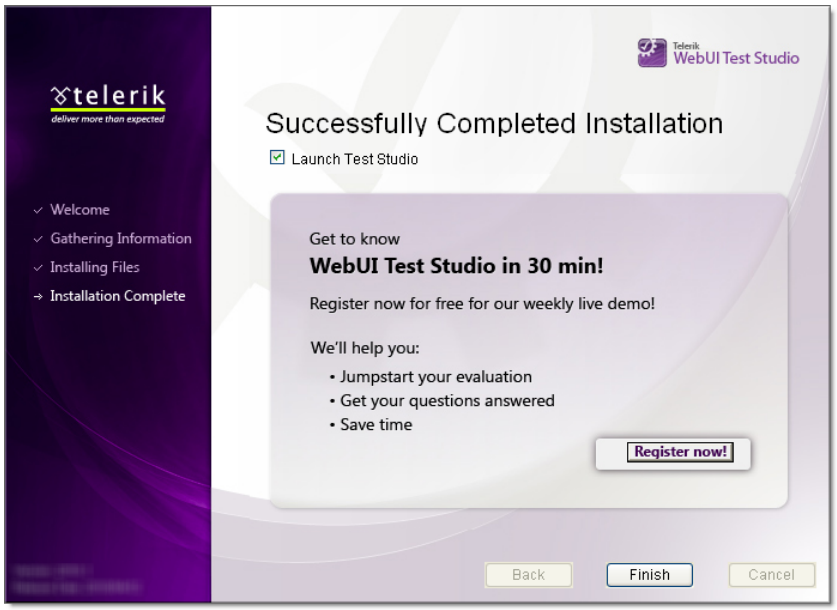
5) The next page of the setup wizard displays the "Ready to install WebUI Test Studio" page. This page lists the features that will be installed and is your last chance to click the **Back** button to make any changes before actually installing the product. Click the **Install** button to install Telerik WebUI Test Studio Developer Edition to your computer.




6) The status of the installation will display on the "Installing WebUI Test Studio" page. Depending on the resources for your computer, this step make take some time to complete.



7) When Telerik WebUI Test Studio Developer Edition has been installed to your computer, the last page of the wizard displays a completion message. If you want to run the WebUI Test Studio *QA version,* leave the "Launch Test Studio" checkbox selected. Click the **Register Now** button to both register the product and get access to a free weekly demonstration webinar. Click the **Finish** button to close the setup wizard.




Telerik WebUI Test Studio Developer Edition will now be available from the Windows Start menu, from a shortcut placed on the desktop and also from inside the Visual Studio environment.

## 2.3 Wrap Up

In this chapter you learned how to install Telerik WebUI Test Studio Developer Edition.

# Part III

Getting Started

# 3 Getting Started

## 3.1 Objectives

In this chapter you will learn how to create a new Visual Studio test project and a WebUI Test Studio test. You will learn how to interactively record a test, how to run the test and how to look at the results. You will run the test at full speed and also learn how to enable "Annotations" in order to watch the test at a slower speed, with notes displayed as the test runs. You will learn how to modify the test and see both failing and succeeding test steps and drill down to greater detail provided for failing steps.

**Find the projects for this chapter at...**

\Courseware\Projects\<CS|VB>\GettingStarted\GettingStarted.sln

## 3.2 Walk Through

### 3.2.1 Create a New Visual Studio Test Project

1) From the Visual Studio choose **File > New > Project...** This will display the "New Project" dialog.

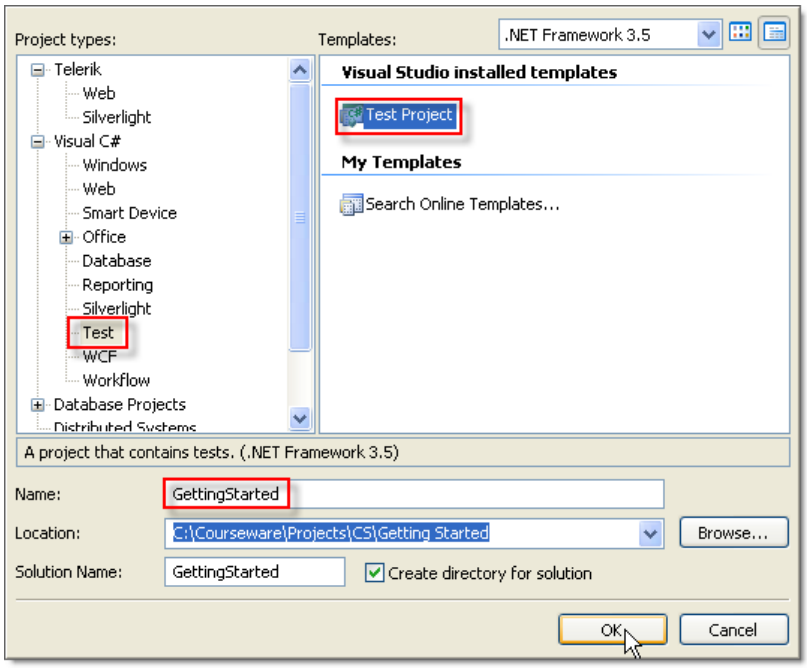
2) Define a new test project:

a) In the "New Project" dialog, select the "Test" project type for your language (C# or VB).

b) Select the "Test Project" template.

c) Enter a descriptive name for the test project, a location path and a Solution name.

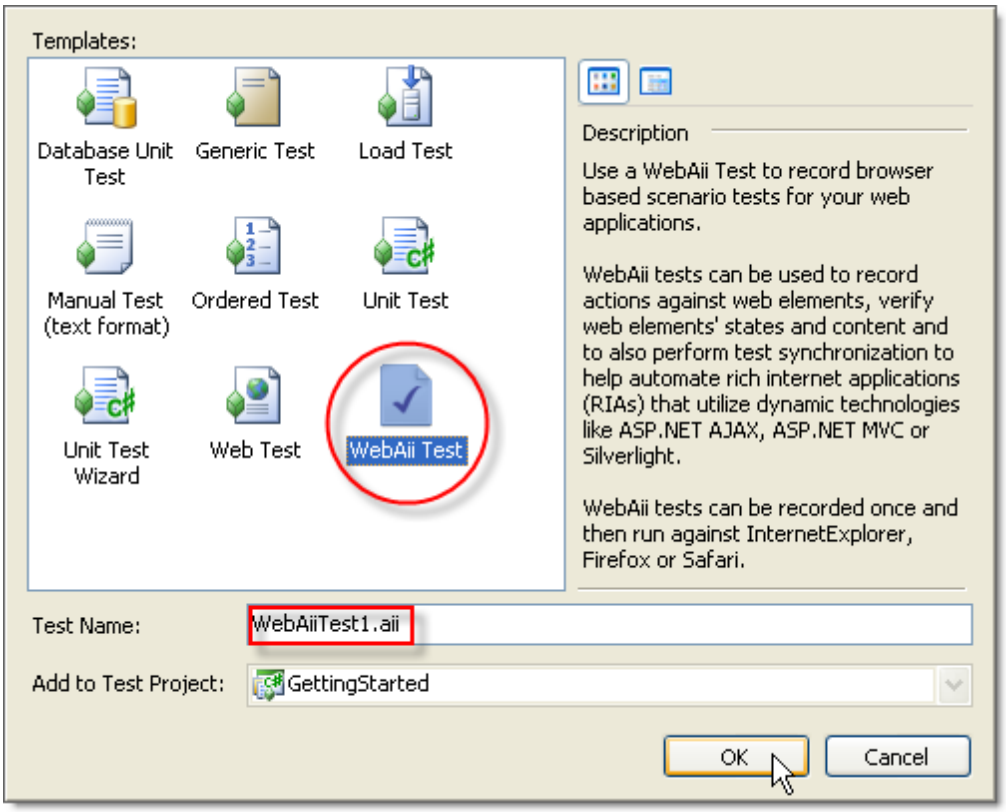
d) Click **OK** to create the new test project.



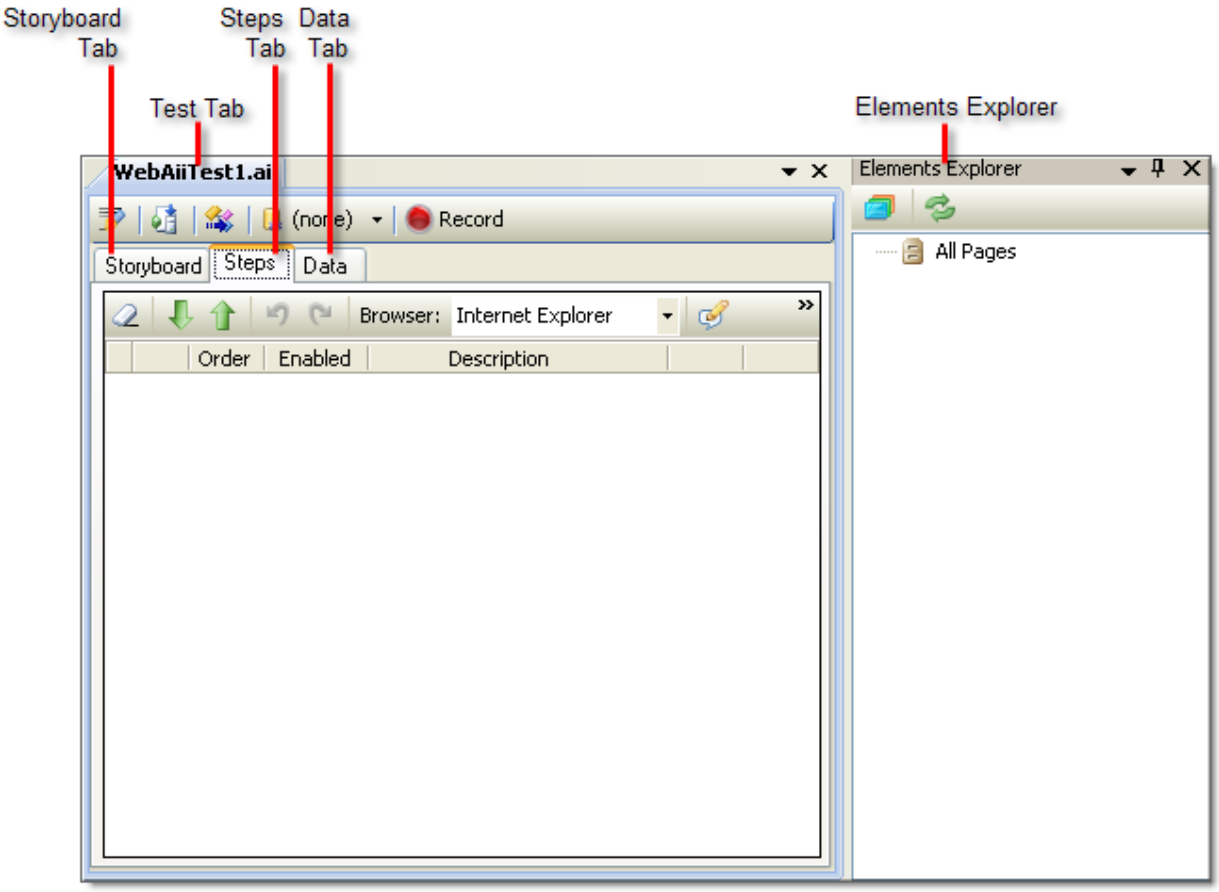
### 3.2.2 Create a New WebAii Test

1) From the Solution Explorer, right-click the test project and select **Add > New Test...** from the context menu. This will display the "Add New Test" dialog.

2) In the "Add New Test" dialog, select the "WebAii Test" template, provide a meaningful Test Name and click **OK** to create the test.

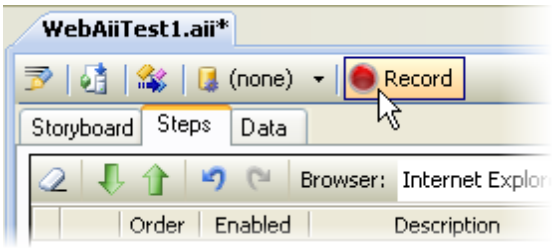


3) When the WebUI Test Studio test is created, several new areas become available: a Test Tab that contains a Storyboard Tab, Steps Tab and Data Tab, and a Elements Explorer that outlines named elements that are specific to your test. We will be talking in detail about all of these in the upcoming "Visual Studio Integration" chapter.



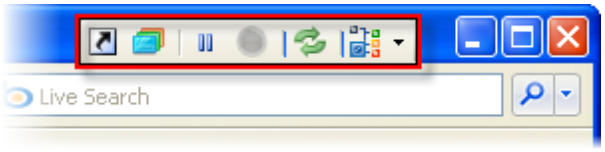
### 3.2.3 Record a WebAii Test

1) Locate the Record button and click it. This will display the Recording Surface.

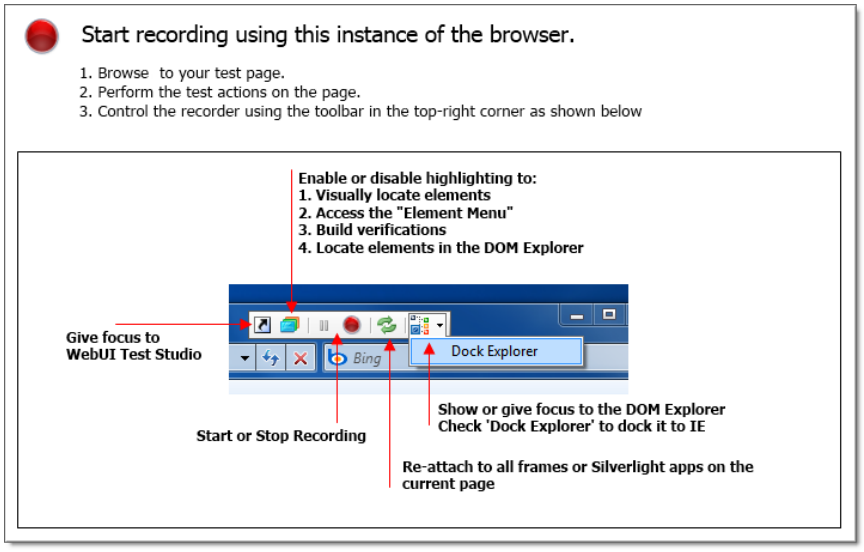



***Notes***

It may take a moment to initially invoke the browser. The WebUI Test Studio toolbar will be anchored to the title bar area of the browser.

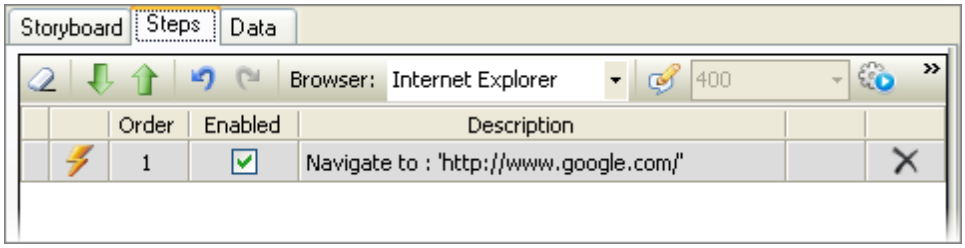



...and the browser will display instructions on how to use the Recording Surface:

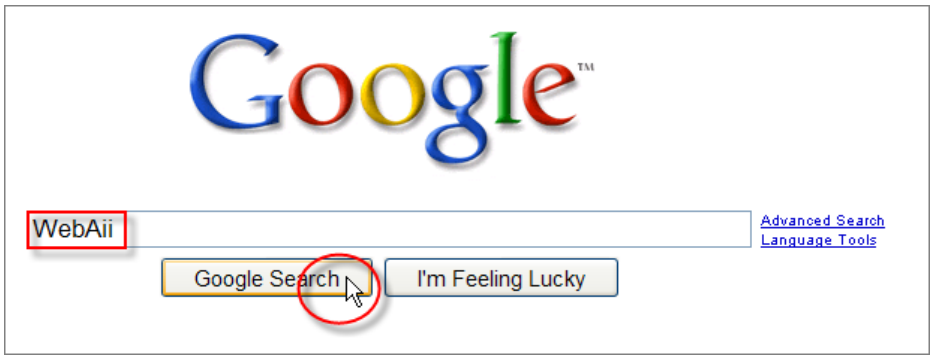



2) In the the Recording Surface, enter "www.google.com" to the browser address bar and then press the **Enter** key. This will load the "Google" web page.

3) Notice that a "Navigate to..." test step has been added in the Steps Tab.



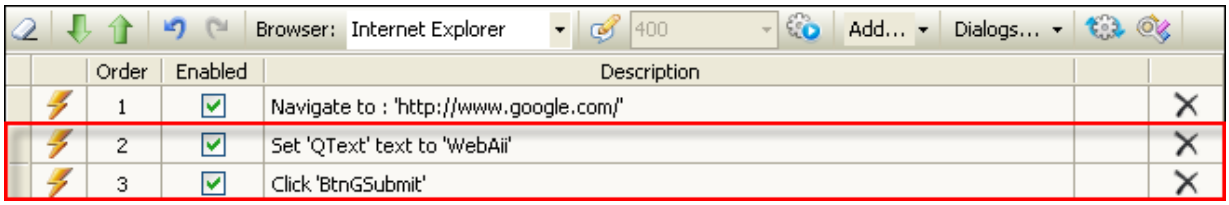

4) Enter "WebAii" to the Google web page search edit box and click the "Google Search" button.



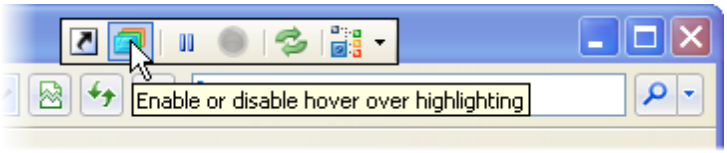

***Notes***

If the Google page displays a drop down list of search possibilities, you can click the search button that appears there, but object names and step descriptions will not exactly match those described below.

5) Notice that two new lines, the entry and the button click, have been added as test steps to the Steps Tab.




6) Locate the **Highlighting button** above the Recording Surface and click it.



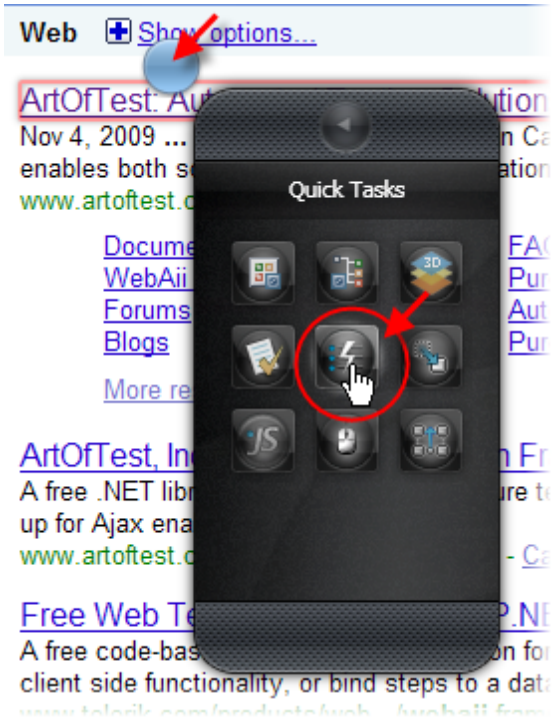
7) Now, elements will be highlighted as the mouse passes over in the Recording Surface. The screenshot below shows the mouse hovering above the "Google" logo image. After waiting a moment, a small circle called a "Nub" displays above the element.

***Notes***

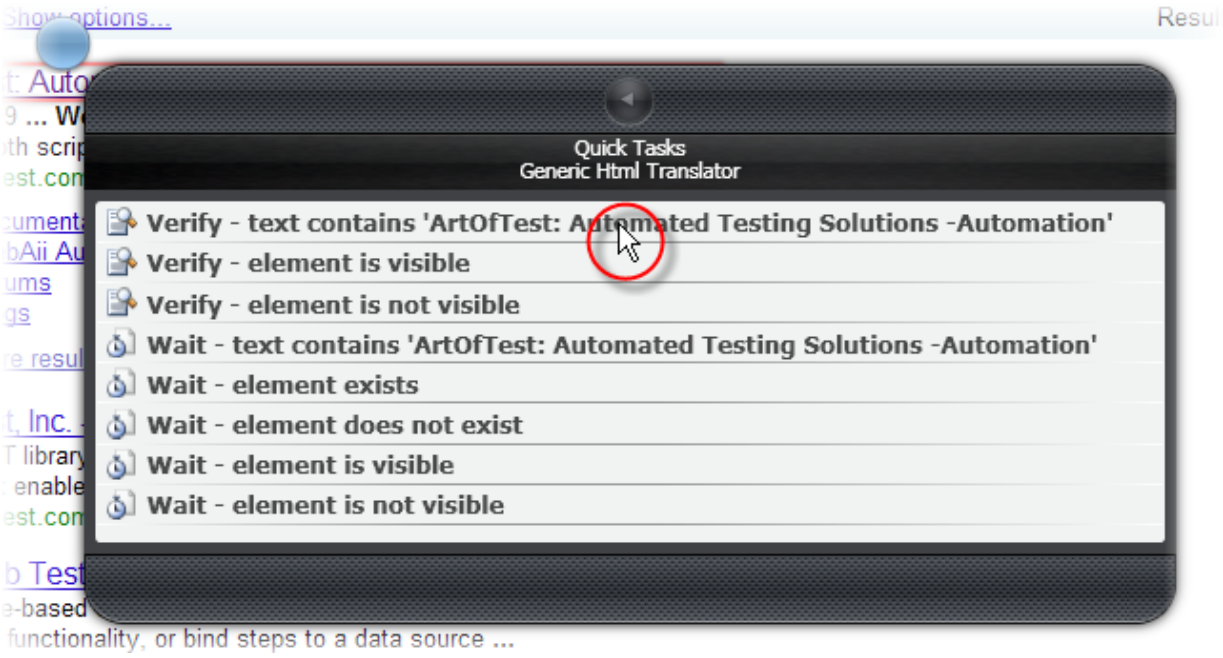
The Nub displays a menu of tasks that can be performed on the web page element. This interface will be described in detail in the upcoming Visual Studio Integration chapter.

8) Hover the mouse over the first item in the Google search. Click the **Nub** to display the Elements Menu. Move the mouse over the buttons until you locate the Quick Tasks button. Click the **Quick Tasks button**.

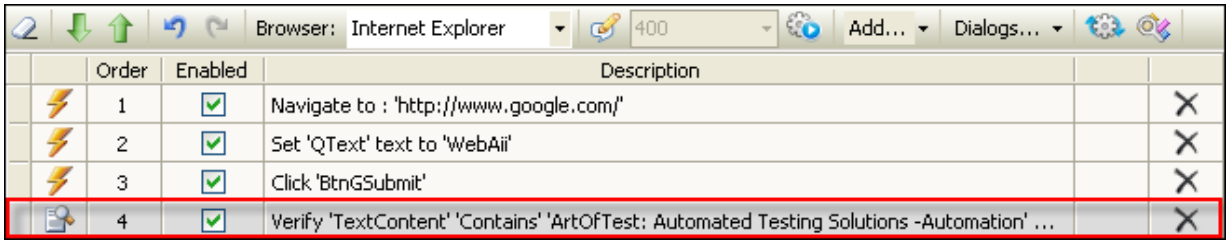
**Note:** The results that Google returns may change over time, but will typically contain the word we're counting on: "WebAii".



9) Clicking the Quick Tasks button produces a context sensitive list of useful possibilities that we can perform against the highlighted element. Double click the first item in the list. This will close the Elements Menu and add a test step.

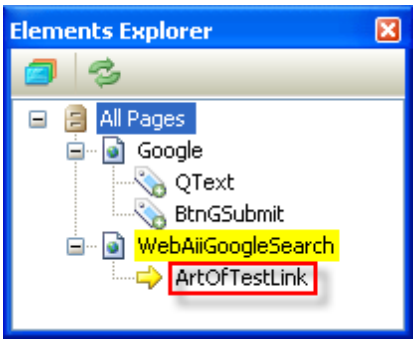



10)Notice that the new test step performs a verification that the text of the highlighted element contains a specific string...

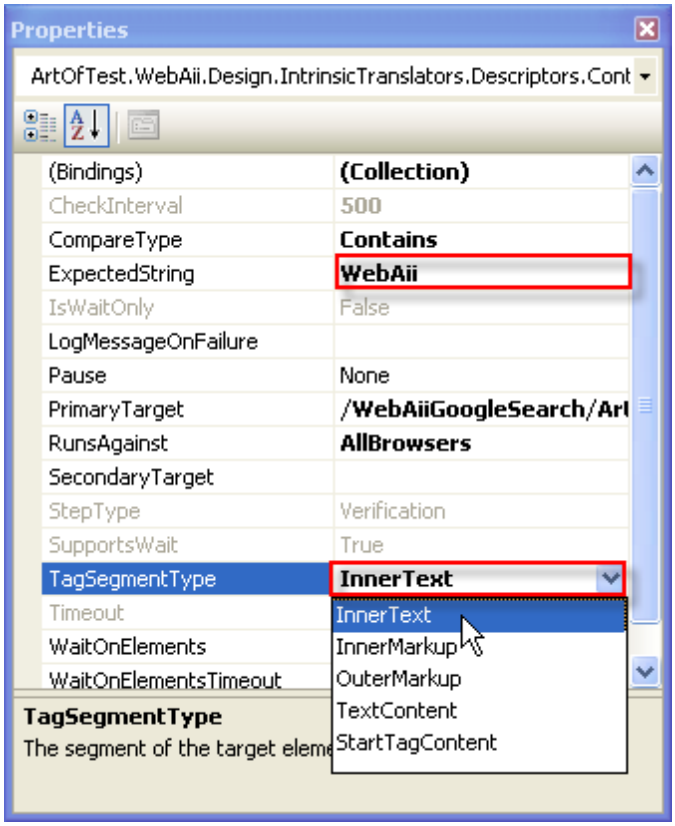


### *Notes*

Also notice that an "ArtOfTestLink" element used by the test step is added to the Elements Explorer.




11)With the "Verify..." step still selected in the Elements Explorer, in the Properties pane, locate the **ExpectedString** property and change its value to "WebAii". Locate the **TagSegmentType** property and select "InnerText" from the drop down list.

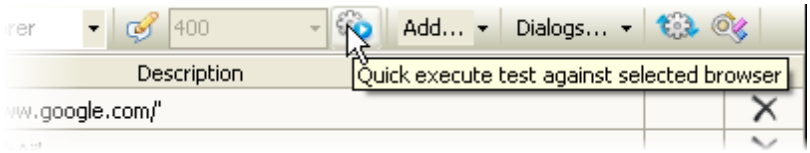


***Notes***

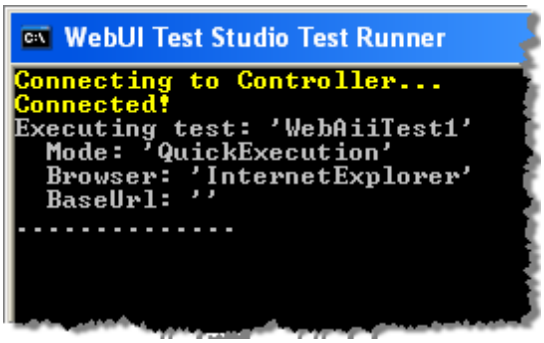
WebUI Test Studio lets you verify properties of an element by looking at different parts of the element, e.g. the "InnerMarkup" and comparing that information against a particular string. You can compare with simple criteria such as "contains" or "doesn't contain". You can also use Regular Expressions for more detailed and rigorous examinations. We will look at verifications in more detail in upcoming chapters "Visual Studio Integration" and "Verification Engine".

### 3.2.4 Run a WebAii Test

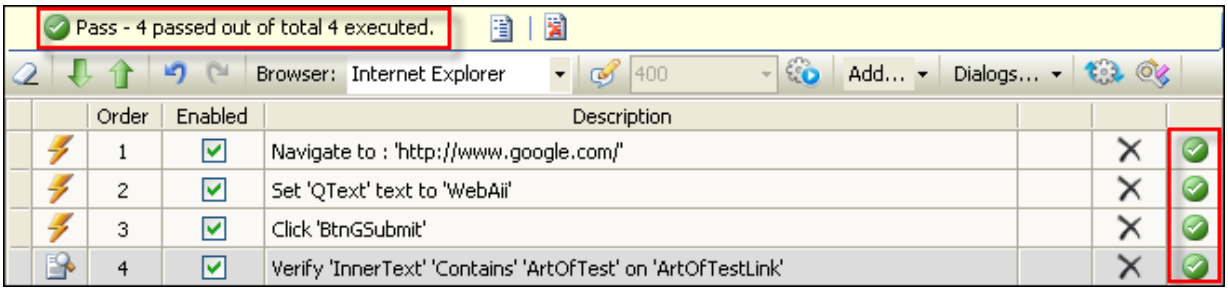
1) Locate the **Quick Execute button** in the Steps Tab and click it. This will run the WebUI Test Studio test with all the test steps you have defined so far.



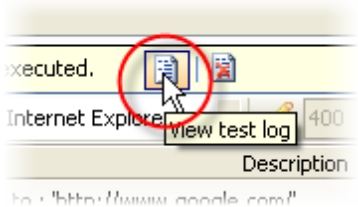

2) A console window outputs the results as the test is run. The actions you defined in the test will also take place in the browser, but at a very quick pace. When the test completes, the browser window will close.



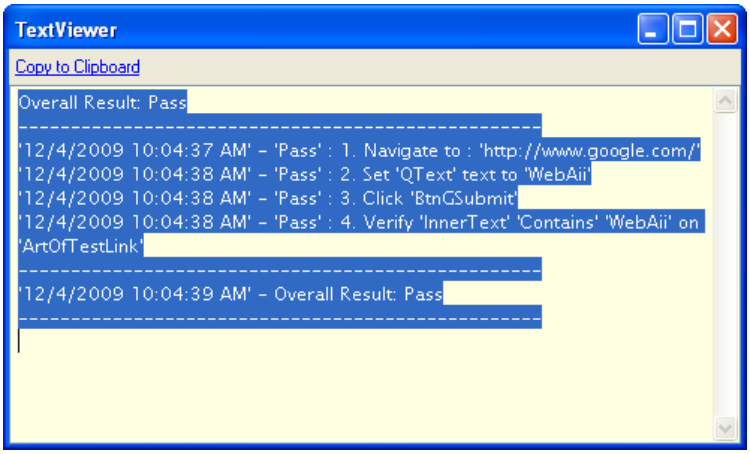

3) The Steps Tab will display a test summary. Each test step will show a green icon to indicate success.




4) In the Steps Tab toolbar, locate the View Test Log button and click it.



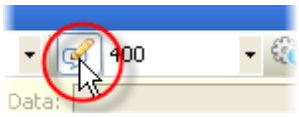
5) This will display a simple text viewer that will allow you to view or copy the test results to the clipboard.



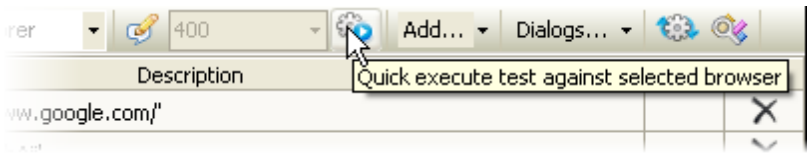

***Notes***

This is just a quick summary look at the test results. If you need to do more with the test results or automate based on the output, you can use MSTest to run the test. See the MSTest chapter for more information.

6) The test may have run quickly without allowing you to easily see what was happening in the browser. We can turn on "Annotations" to slow down the action and automatically display notes as the test runs. Click the Enable Annotations to enable annotations.




7) Click the **Quick Execute button** to rerun the test.

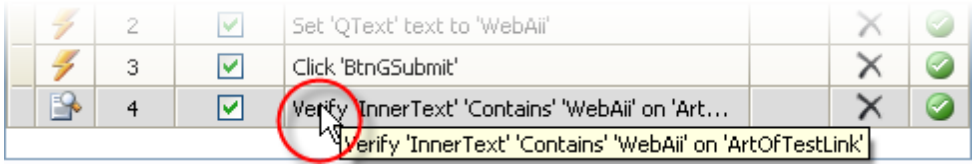


8) As the test runs the element being tested will be highlighted and an annotation will display.

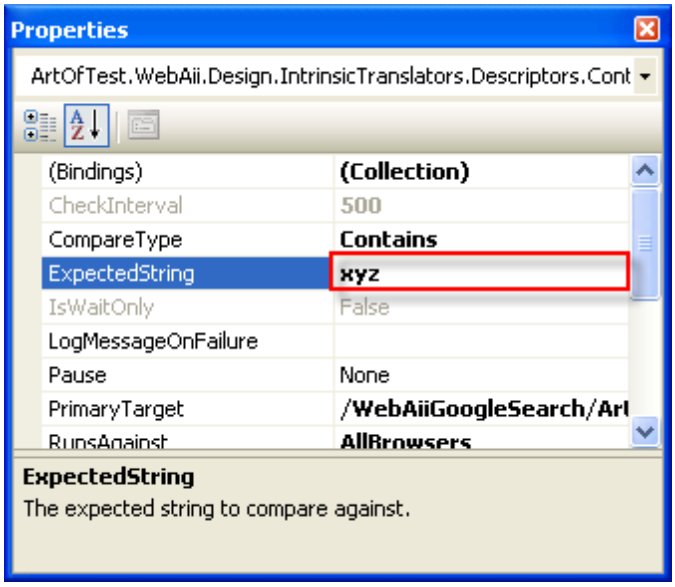
### 3.2.5 Modify a WebAii Test

Lets go back to the verification we performed against an element's inner text and change it so the test fails.

1) In the Steps Tab, select the verification step.

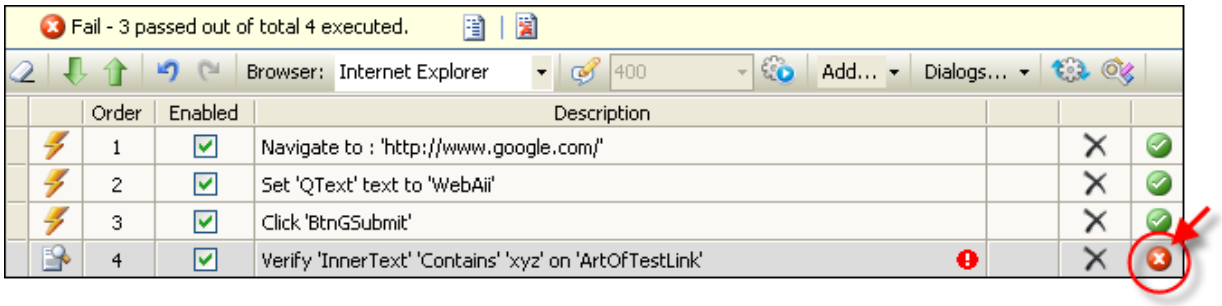



2) In the Properties pane, change the **ExpectedString** to "xyz".

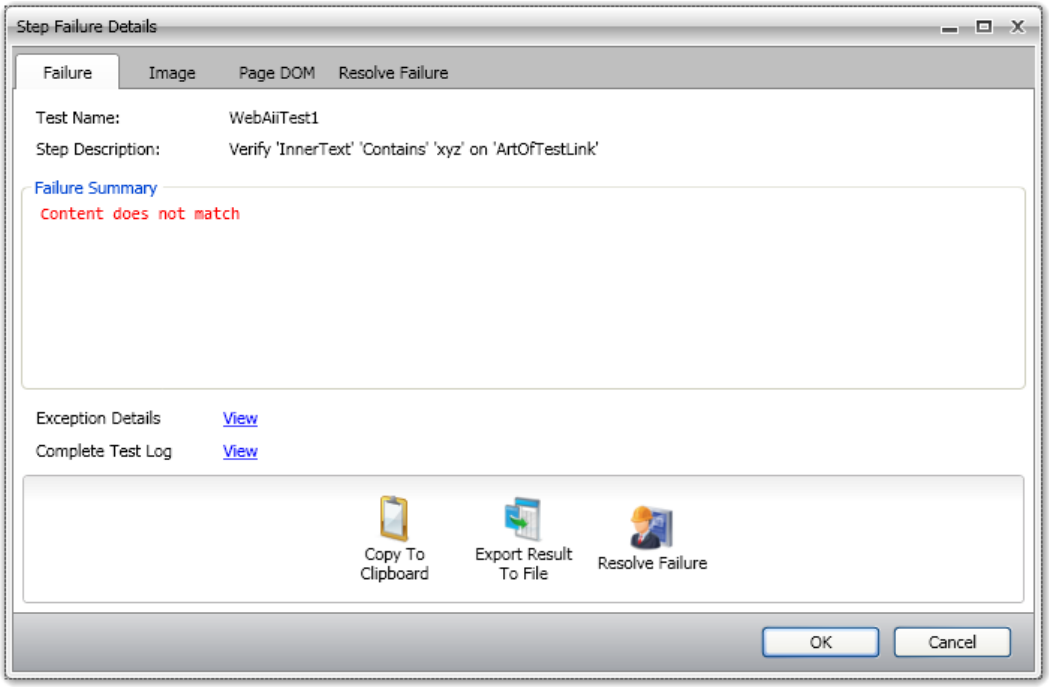



3) Click the **Quick Execute button** to rerun the test.

4) The Steps Tab summary indicates that only three of the four tests passed and the step that failed displays an "X" icon. Double-click the icon to display detail about the error.



5) The "Step Failure Details" breaks down the exact cause for the test failing. Here we can see that the expected value for the **ExpectedString** property was "xyz", but the actual value was "ArtOfTest:..". This screen also allows you view the exception details, the complete test log and to resolve the failure right on the spot. See the Visual Studio Integration for more information on the Step Failure Details dialog.




## 3.3 Wrap Up

In this chapter you learned how to create a new Visual Studio test project and a WebUI Test Studio test. You learned how to interactively record a test, how to run the test and how to look at the results. You ran the test at full speed and also learned how to enable "Annotations" in order to watch the test at a slower speed. You learned how to modify the test to see both failing and succeeding test steps and you examined the information provided for failing steps.

# Part IV

Visual Studio Integration

# 4 Visual Studio Integration

## 4.1 Objectives

This chapter explains how WebUI Test Studio is integrated with Visual Studio. The chapter starts with a tour of the panels and toolbars in Visual Studio that make up the WebUI Test Studio environment, then shows how to use the Storyboard Tab to organize and navigate test steps, the basics of building simple data driven tests and how to interact with the Visual Studio testing mechanism. You will learn how to create tests interactively using the Recording Surface, the Common Tasks Menu and the Elements Menu.

You will work with the Elements Explorer to organize elements used in test steps, learn how the Elements Explorer interacts with the Properties pane and fine tune how elements are located and recognized. You will use the Steps Tab to manage test steps and add to add unique types of test steps such as screen captures, delays and annotations. You will also learn how to reuse existing tests.

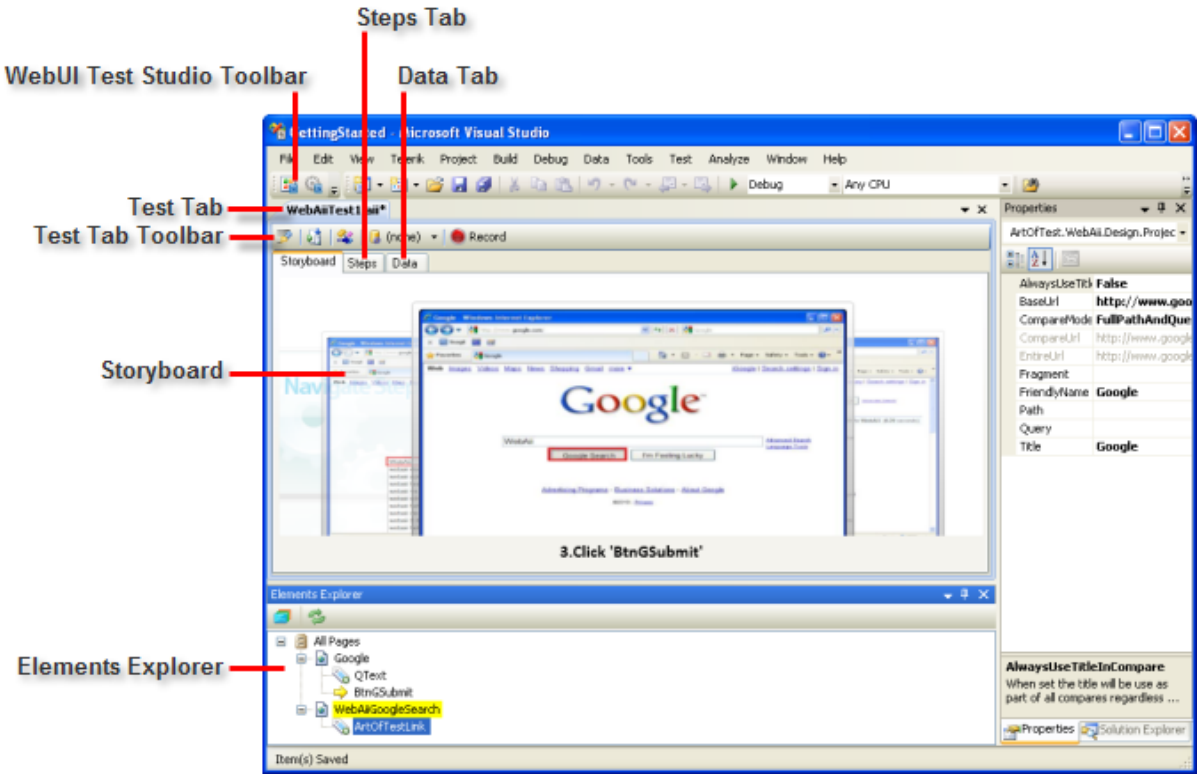
You will use the DOM Explorer to look at all elements in a page. In particular, you will learn how to search for elements using simple comparisons and more complex searches using comparison operators, Regex and XPath.

**Find the projects for this chapter at...**

\Courseware\Projects\<CS|VB>\Visual Studio Integration\VisualStudioIntegration.sln

## 4.2 Tour of the Environment

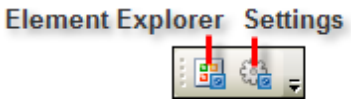
The WebUI Test Studio environment is contained within multiple Visual Studio windows. The panes may be docked in different locations in your particular Visual Studio configuration.




- **WebUI Test Studio Toolbar:** has tools to display the Elements Explorer pane and the Settings dialog.
- **Test Tab**: This tab represents a single WebAii test and contains tabs for the Storyboard Tab, Steps Tab and Data Tab. The toolbar at the top of this area allows you to perform actions against the test as a whole, such as converting the test to an MSTest or starting test recording.
- **Elements Explorer**: Unlike the DOM Explorer, the Elements Explorer shows only specific elements relevant to your test. You can right-click an element to select actions from the context menu. These actions can be performed against the element or children of an element. Elements are added to the Elements Explorer from multiple origins including the Recording Surface and the DOM Explorer. Elements are also added automatically when they are used in test steps.

## 4.3 WebUI Test Studio Toolbar

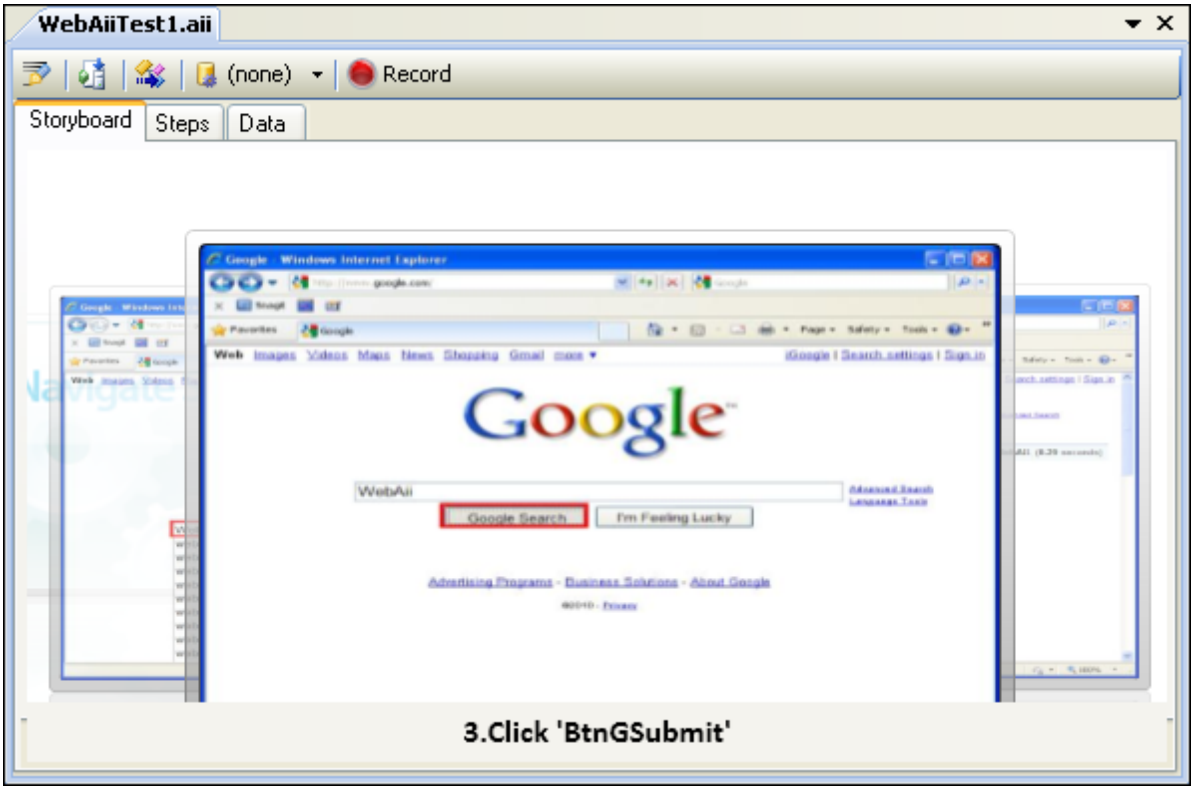
The WebUI Test Studio Toolbar is a top level Visual Studio toolbar that have actions that work against the product in Visual Studio as a whole. If the toolbar is not visible, enable it through the Visual Studio menu **View > Toolbars > WebUI Test Studio** option. The screenshot below shows the toolbar and the available tools.



## 4.4 Test Tab

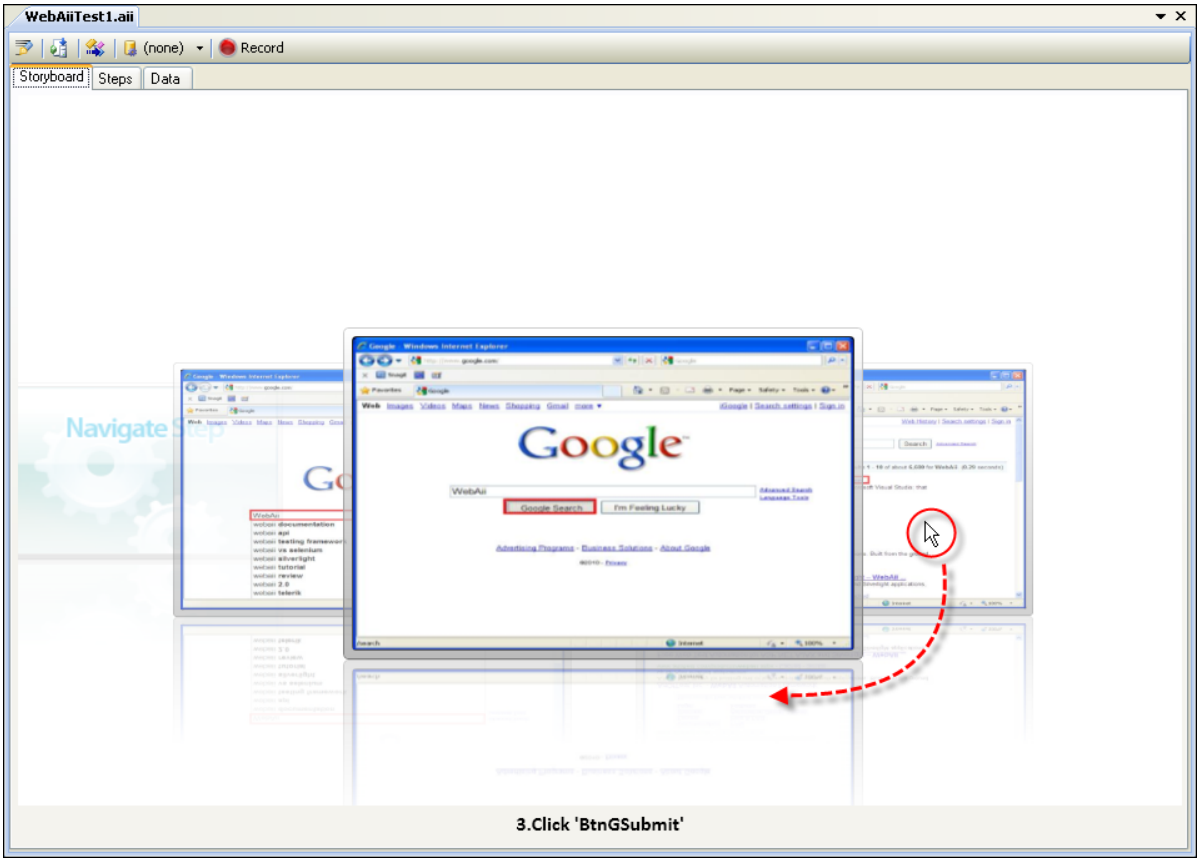
The Test Tab represents a single WebAii test and contains Storyboard Tab, Steps Tab, Data Tab and a toolbar. Use this panel to:

- Organize and navigate test steps.
- Build Data Driven tests.
- Start recording a test interactively.
- Add code to a test.
- Interact with the built-in testing features of Visual Studio.



### 4.4.1 Storyboard Tab

The Storyboard Tab is a three-dimensional, visual representation of test steps. WebUI Test Studio automatically takes screenshots where appropriate and highlights the element of interest for each screenshot. You can click background images to bring the image "up front". The screenshot below shows step #3, "Click 'BtnGSubmit'" as the current step, and where the right-most background is being clicked to make that step the current test step.

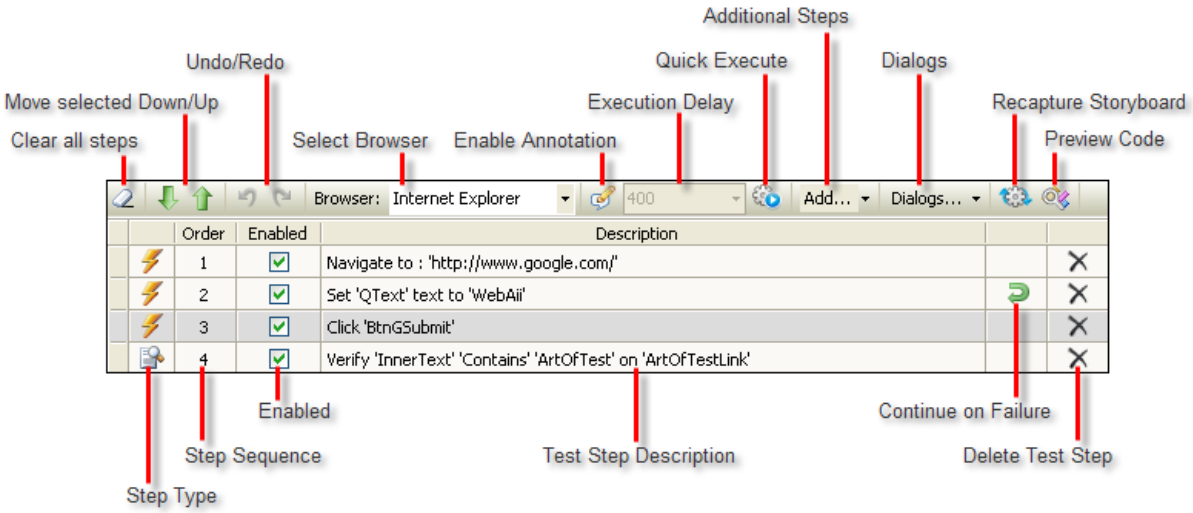




The Storyboard Tab is synchronized with the selected item of the Steps Tab so that items clicked in the Storyboard Tab will be highlighted in the Steps Tab list of test steps.

## 4.4.2 Steps Tab

The Steps Tab contains the individual list of test steps, controls test step order and the speed that the steps are executed. You can also use the Steps Tab to add delays, annotations, dialog handling and other specialized test steps.

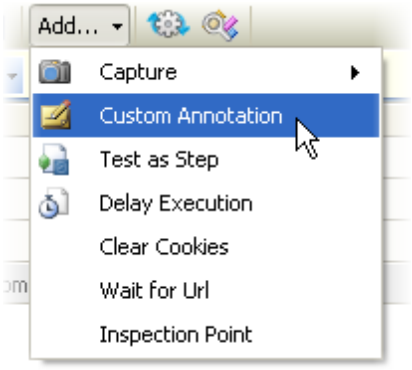



### Toolbar

Starting with the buttons across the top of the Steps Tab toolbar:

- **Clear all Steps** removes all test steps in one shot.
- **Move Selected Down/Up** moves selected test steps down or up in the list.
- **Undo/Redo** rewinds previous actions in the Steps Tab.
- **Select Browser** presents a drop down list of browsers (e.g. Internet Explorer, Firefox, Safari). The browser selected here will be used when the Quick Execute button is pressed.
- **Enable Annotation** causes the test to run slower, with pauses specified in milliseconds by Execution Delay. A highlighted annotation will appear in the browser as the test executes describing each step.
- **Quick Execute** runs the test in the selected browser.

- **Additional Steps** allows you to add specialized test steps that don't originate from interaction with the browser. Here are some of the items you can add and the unique properties that can be set for each:




**Screen capture: CaptureType** can be Desktop or Browser. **FileNamePrefix** is "Snapshot" by default. Important note!: To make screen capture work you cannot run the test using the "Quick Execute button". Instead, run the test from the Visual Studio Test menu. You can use the Visual Studio menu **Test > Windows > Test View**, right-click your WebAii Test and select **Run Selection** from the context menu.

**Custom Annotation**: Annotations are notes that display right in the browser as the test executes. You can use Custom Annotation to communicate with whoever is reviewing the tests. If you wanted to point out some information to the developer, e.g. "This test step fails intermittently", then you could add that message as an annotation. **AnnotationText** is the text displayed on the screen. **DisplayLocation** determines where the annotation displays relative to the element such as TopCenter or TopLeftCorner. **DisplayTime** is the number of milliseconds that the annotation displays.

**Test as Step** lets you run a another WebAii Test as a single step. **TestPath** is the path to the WebAii test.

**Delay Execution** lets you set the **WaitTime** property to a number of milliseconds.

**Clear Cookies** clears ***all*** cookies from the active browser unconditionally. This is useful when you want to start a clean test without saved information (e.g. user id), or saved state information ("logged in", last visit date, preferences, etc).

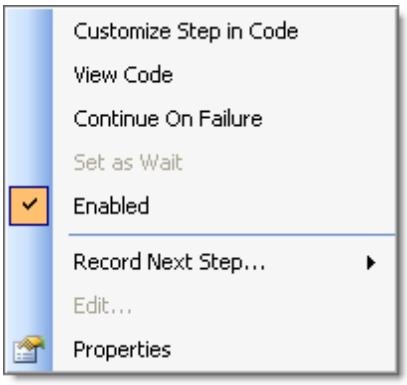
**Wait for Url** suspends the test until a particular Url is loaded into the browser address bar. "Wait for Url" is particularly useful when you have a redirection and need to wait for the final Url to be loaded.

**Inspection Point** pauses the test and displays the DOM Explorer.

- **Recapture Storyboard** runs your test and stores new screenshots of each test step to the storyboard.
- **Preview Code** shows what your test will look like in code-behind form.

### Test Steps Grid

The actions you can take on test steps are to enable/disable a step or delete a step. The other columns are informational and show the type of test step as an icon, a sequential number, a description of the test step and a "Continue on Failure" icon. Each test step also has a context menu with further actions that can be taken. Note that some context menu items will show up only for certain test step types. For example, the Load Page... item displays for Navigation test steps.




- **Customize Step in Code** creates a new test method in the code-behind. You'll also notice that the description and icon change to indicate the step is coded. Once you have converted to code, you cannot convert back. See the WebAii Framework chapter for more information


- **View Code** simply navigates you to the code-behind for the test.
- **Continue On Failure** allows the test to carry on even if the step fails.
- **Set as Wait** can be used to convert a Verification step type to a Wait step type. Instead of passing or failing based on a comparison, we're waiting for the comparison to be true before proceeding. Converting to a Wait enables the Timeout and CheckInterval properties. You can toggle back and forth between the test step as Verification and Wait.
- **Enabled** allows you to temporarily turn off a test step and is the same option that can be set in the Enabled checkbox within the test steps grid.
- **Record Next Step...** allows you to start recording following the step selected in the grid or optionally, to start recording after the last step.
- **Edit...** is enabled for Verification steps and displays the Sentence Verification Builder. See the "Verification Engine" chapter for more information.
- **Properties** simply navigates over to the Properties pane so you can change the properties for the selected test step.

#### 4.4.2.1 Walk Through

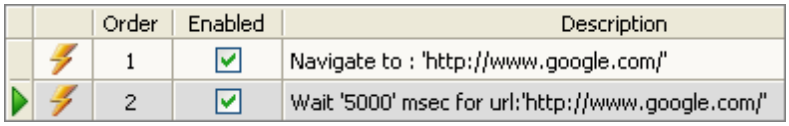
This walk through will help familiarize you with the Steps Tab and give you a chance to use some of the features discussed so far.

## Adding New Test Steps

1) Start with the "Getting Started" Walk Through project or a copy.

2) In the Steps Tab, select the first test step (that navigates to www.google.com). Right-click the step and select **Record Next Step > After Selected Step** from the context menu.

3) From the **Add...** menu select **Wait for Url**. This will add a test step that waits for the www.google.com url. The first two steps should now look like the screenshot below. Set the Url to www.google.com



| | Order | Enabled | Description |
|---|---|---|---|
| | 1 | ☑ | Navigate to : 'http://www.google.com/' |
| ▶ | 2 | ☑ | Wait '5000' msec for url:'http://www.google.com/' |

***Notes***

Once the Wait step is added you can tweak the **WaitTimeout** (in milliseconds), the **Url** and the **IsPartial** properties. When IsPartial is true, the Url can be appended with additional path information and still match. For example, Url could be "http:\\www.xyz.com" and both http:\\www.xyz.com\test and http:\\www.xyz.com\version2 would match.

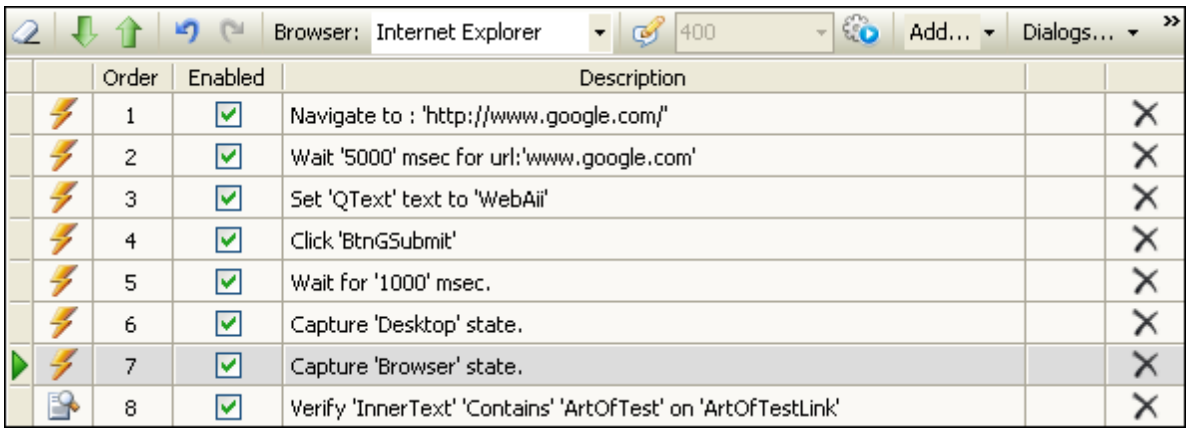
4) In the Steps Tab, select the step that clicks the "Search" button. Right-click the step and select **Record Next Step > After Selected Step** from the context menu.

5) From the **Add...** menu select **Delay Execution**. Use the Properties pane to change the **WaitTime** property to "1000".

6) From the **Add...** menu select **Capture > Desktop**. Use the Properties pane to change the **FileNamePrefix** property to "GoogleTest".

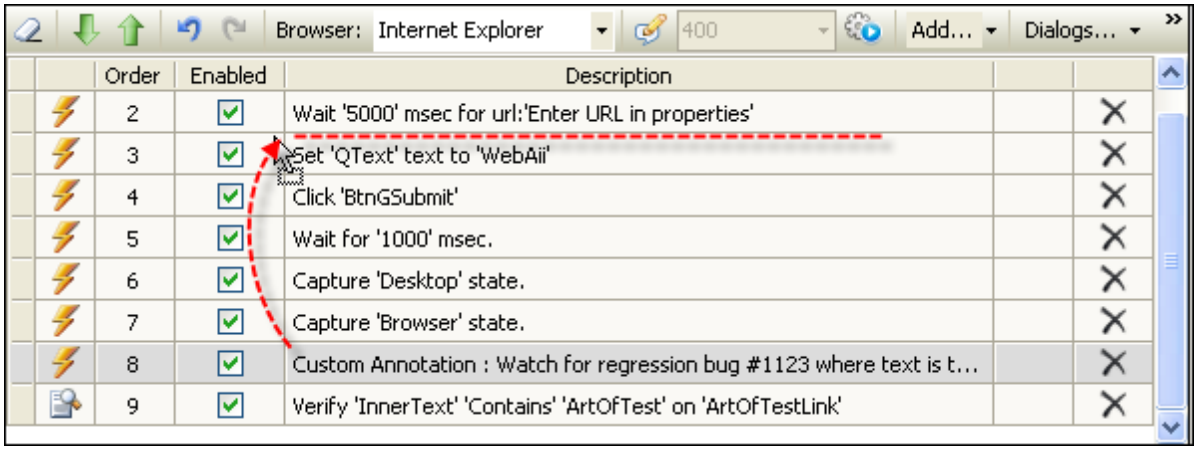
7) From the **Add...** menu select **Capture > Browser**. Use the Properties pane to change the **FileNamePrefix** property to "GoogleTest".

The affected steps in the Steps Tab should now look like the screenshot below:




8) From the **Add...** menu select **Custom Annotation**. Use the Properties pane to change the **AnnotationText** property to "Watch for regression bug #1123 where text is truncated". Set the **DisplayLocation** to "AbsoluteCenter" and the **DisplayTime** to "2000". Now drag the test step to a point just before "Set 'QText' text to 'WebAii'".

*This step simulates communication you might have as a tester with a developer, using the annotation to let the developer know of a specific condition in the test to look out for.*



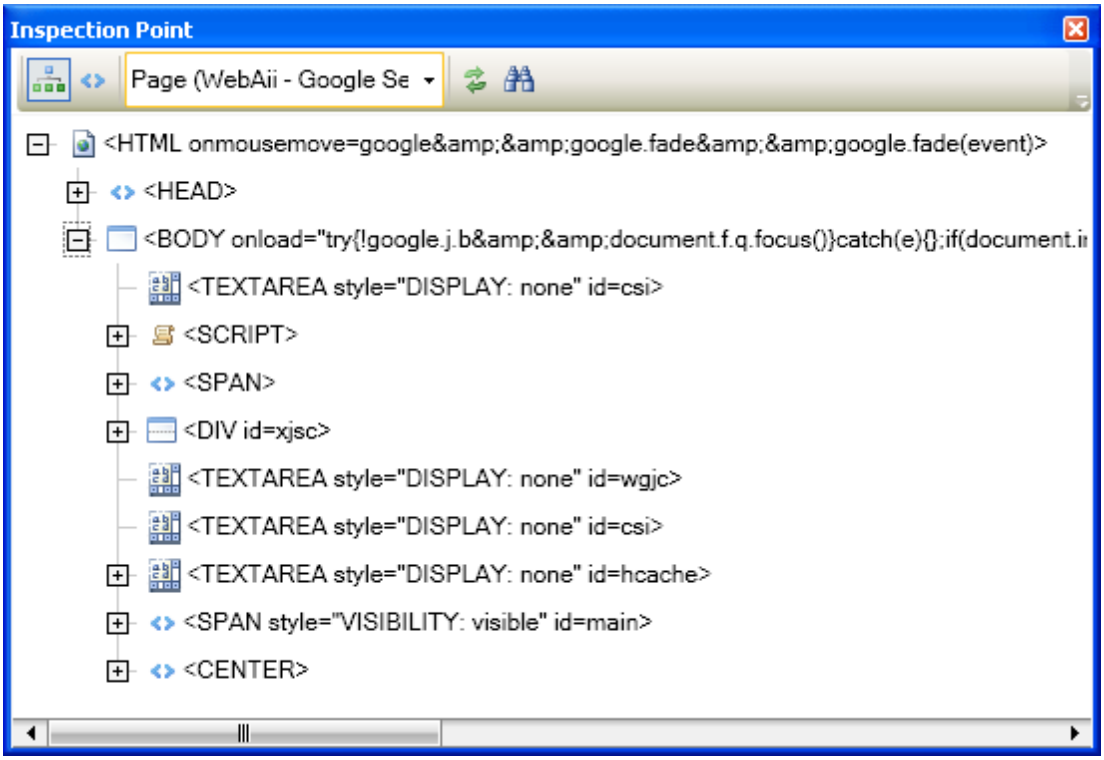

9) From the **Add...** menu select **Inspection Point**. Make this the last step in the test.

## Running the Test

1) In the Steps Tab tool bar, select Internet Explorer as the browser type.

2) Click the **Enable Annotations** button.

3) Click the **Quick Execute** button to run the test. All test steps should pass. Watch out for the Inspection Point step at the end that displays the Inspection Point dialog that shows a tree view of the page in DOM (Document Object Model) form. Close the Inspection Point dialog to let the test complete.



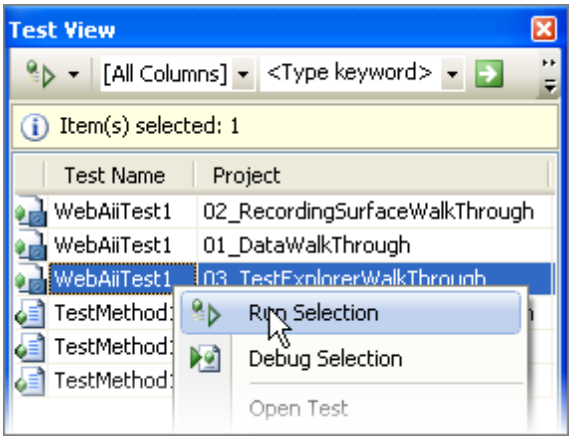

4) Change the browser type to "Safari" and rerun the test using the Quick Execute button. All test steps should pass. Note that you need to have the Safari browser installed to perform this step.

5) Change the browser type to "Firefox" and rerun the test using the Quick Execute button. All test steps should pass. Note that you need to have the Firefox browser installed to perform this step.

So, where are the screen captures kept? The screen capture steps will run, but the log will show "Image not captured to disk. CreateLogFile is set to 'false'." To save your screen captures to disk, see the next section that shows how to execute your test in MSTest.

## Running in MSTest

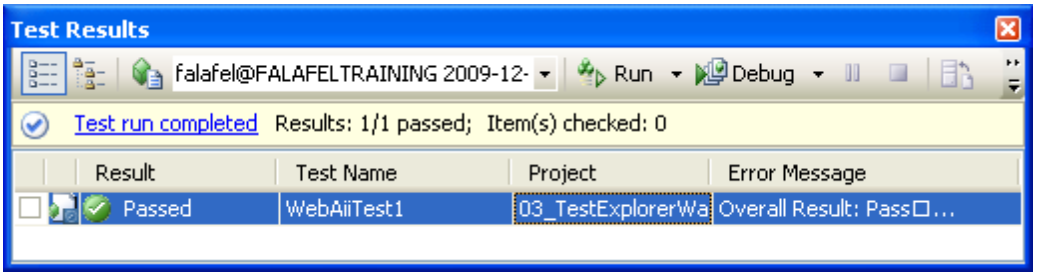
Some features, notably screen capture, need to be run from MSTest to work. Also be aware that annotations will only show when initiated from the Quick Execute button. To run from MSTest, first select the Visual Studio menu **Test > Windows > Test View**. Locate the test in the Test View panel, right-click and select the **Run Selection** option from the context menu.



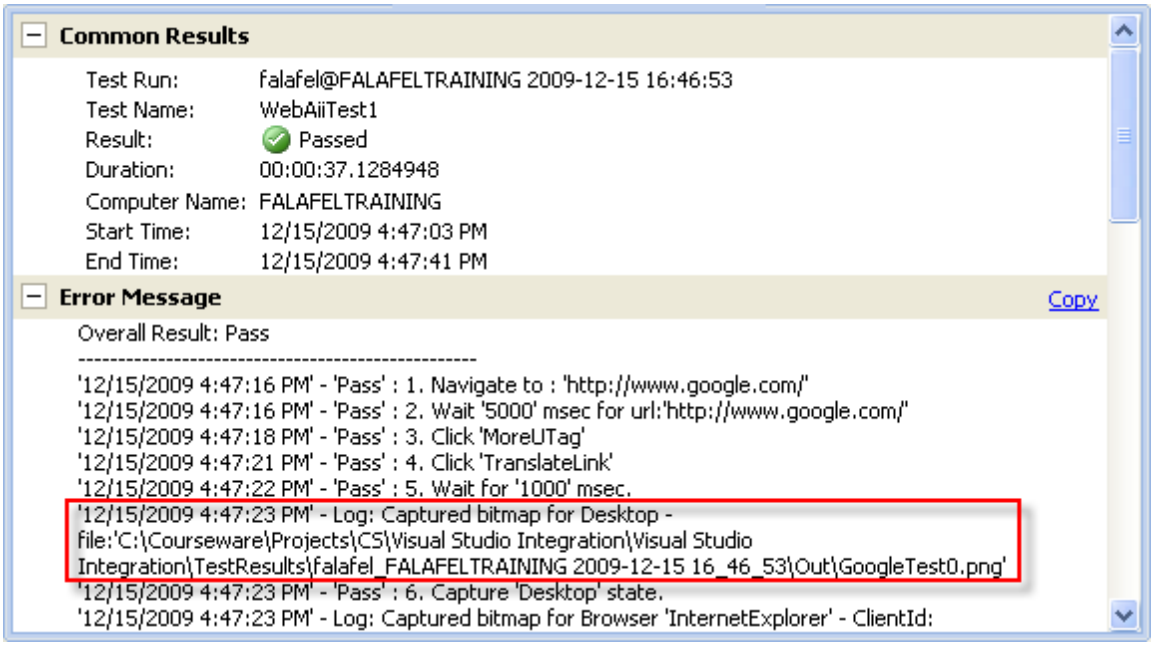

The Test Results panel will display in Visual Studio and show the current status of the test. Be aware that annotations will *not* display when running from MSTest.

## Viewing Test Results

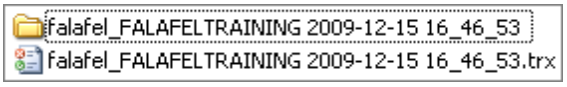
When the test completes, a completion message link will display and the result of the test will display next to the test. Clicking the link will display a short summary. Double-clicking the test detail will display the log of the test.



The log shows a path to the persisted test results where you can find the screen captures for the test.



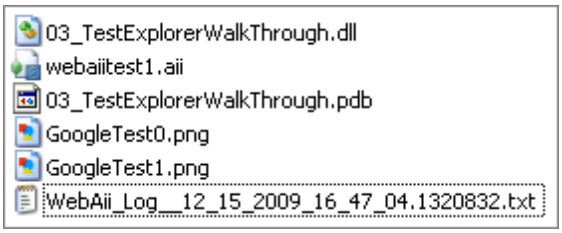

A "\TestResults" folder will have been created in your project directory. It will contain a "trx" test results file and a matching folder that contains the screenshots. Both items will use the name of the computer and append a date and time stamp.



The "trx" file is simply an XML file something like the example shown in the screenshot below.


```
<?xml version="1.0" encoding="UTF-8" ?>
- <TestRun id="74b1e56e-cad3-444d-8bde-98e39d5a4983" name="falafel@FALAFELTRAINING 2009-12-
    15 16:46:53" runUser="FALAFELTRAINING\falafel"
    xmlns="http://microsoft.com/schemas/VisualStudio/TeamTest/2006">
  + <TestRunConfiguration name="Local Test Run" id="d07ad3da-d63f-4ece-80d1-e190af36ce06">
  - <ResultSummary outcome="Completed">
      <Counters total="1" executed="1" passed="1" error="0" failed="0" timeout="0" aborted="0"
        inconclusive="0" passedButRunAborted="0" notRunnable="0" notExecuted="0" disconnected="0"
        warning="0" completed="0" inProgress="0" pending="0" />
    </ResultSummary>
    <Times creation="2009-12-15T16:46:53.3165312-08:00" queuing="2009-12-15T16:47:00.3466400-
      08:00" start="2009-12-15T16:47:03.4410896-08:00" finish="2009-12-15T16:47:41.5358672-
      08:00" />
  + <TestDefinitions>
  + <TestLists>
  + <TestEntries>
  - <Results>
    - <WebAiiTestResult executionId="47e362a6-ebcd-4739-8ad7-e18936ab32c1" testId="d2c2d766-
        0983-49ce-b8ae-1e0001a99c22" testName="WebAiiTest1" computerName="FALAFELTRAINING"
        duration="00:00:37.1284948" startTime="2009-12-15T16:47:03.6513920-08:00"
        endTime="2009-12-15T16:47:41.1753488-08:00" testType="c4241296-3c6e-46b1-bf92-
        e0bd3e0eb017" outcome="Passed" testListId="8c84fa94-04c1-424b-9868-57a2d4851a1d"
        xmlns="http://artoftest.com/schemas/WebAiiDesignCanvas/1.0.0">
      - <Output xmlns="http://microsoft.com/schemas/VisualStudio/TeamTest/2006">
        + <ErrorInfo>
        </Output>
      </WebAiiTestResult>
    </Results>
  </TestRun>
```


The matching folder holds an "\Out" directory with the output from the text, including an "aii" XML file of the test itself and the screen captures from the test execution. Notice the two "png" screen capture files have been named according to the FileNamePrefix property that we set earlier to "GoogleTest".



#### 4.4.2.2 Test Case Reuse Walk Through

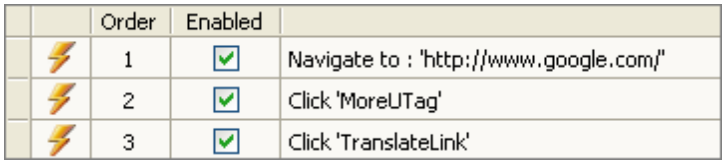
A frequently asked question in forums and webinars is "how do I reuse some set of test steps?". Without the ability to reuse a test in multiple locations, any changes to a test must be maintained in every location where it occurs. For example, if you have a login page that gets used in ten different tests, and the login page changes to require more password characters, you certainly don't want to change your password tests in ten different locations. This maintenance is time consuming, labor intensive and error prone.

WebUI Test Studio allows you to reuse an existing test as if it were a single step. This allows you to modularize tests so that they can be maintained in one place and plugged in where needed. To illustrate, we can add a test to the previous walk through that navigates to the Google Translate page. The new test will have several steps, but will be invoked as a single step.

### Create the New Test

1) In the Visual Studio Solution Explorer, right-click the project and select **Add > New Test...** from the context menu.
2) In the Add New Test dialog, select the WebAii test type. Name the test "GoogleTranslate.aii" and click the **OK** button to create the test and close the dialog.
3) In the GoogleTranslate.aii Test Tab, click the **Record button**. This will bring up the Recording Surface.
4) Enter "http://www.google.com" in the browser address bar.
5) In the Recording Surface browser, click the Google "more" menu link. Select "Translate" from the menu.

   The "GoogleTranslate" test should now contain the three steps shown in the screenshot below.



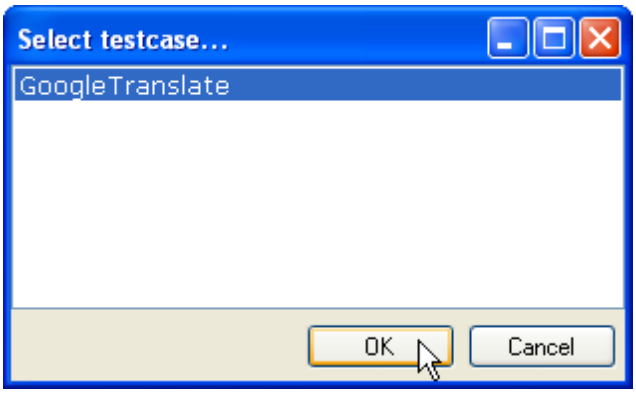

***Notes***

Note that the naming of the links may vary depending on the current state of the Google site and the part of the "More" link you click on. In this example we click on the "More" text, not the drop down arrow.

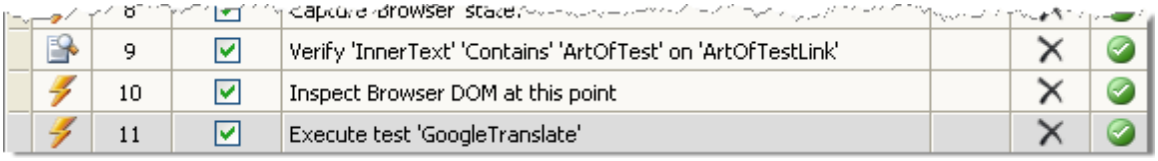
### Call the New Test

1) In the Steps Tab, locate your original test (the copy of the "Getting Started" test) and double-click to open it.

2) From the Steps Tab toolbar, select **Add...** > **Test as Step**.This will display the "Select Testcase" dialog. Select "GoogleTranslate" from the list and click the **OK** button to create the step and close the dialog.

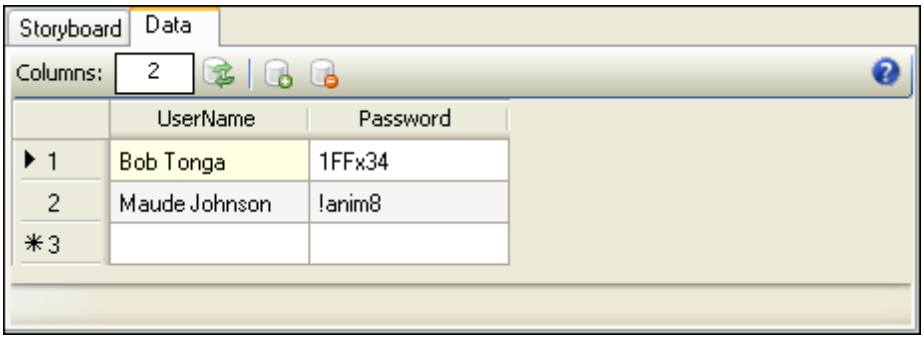



3) Now when you run your test, the "GoogleTranslate" test is invoked as a single step.




### 4.4.3 Data Tab

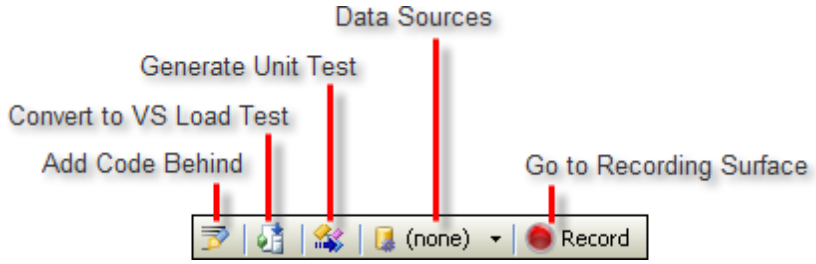
The Data Tab tab allows you to build simple, ad-hoc, data-driven tests. For example, you may have a test with a login screen and want to feed the test several user names and passwords, but you're not really interested in building and connecting to a database. With the Data Tab you can define "user name" and "password" columns in a table, add several sample rows and immediately run your test. The screenshot below shows a sample table with "UserName" and "Password" columns.



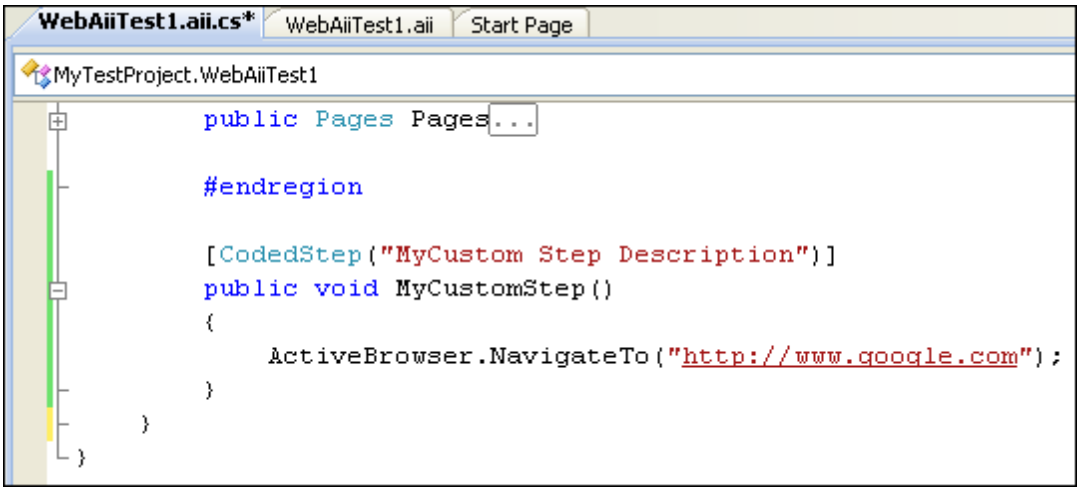

See the "Data Driven Testing" chapter for information on more complex data driven scenarios.

## 4.5 Test Tab Toolbar

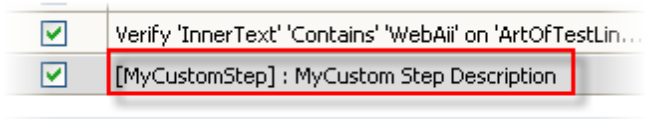
Perform high-level tasks with the Test Tab Toolbar including adding your own test methods as code, converting your test steps into a Visual Studio load tests, converting your test into unit tests, defining data sources and navigating to the Recording Surface.




- **Add Code Behind**: This button navigates to "code-behind", i.e. either Visual Basic or C# code that run as as part of a WebAii Test. You don't need to use code-behind to create and run tests, but code-behind allows special behaviors and fine-tune control over your test steps. WebUI Test Studio lets you mix-and-match test steps created in Test Tab with coded steps so you can gradually add code at your own pace. The screenshot below shows a test step created in code called "MyCustomStep".

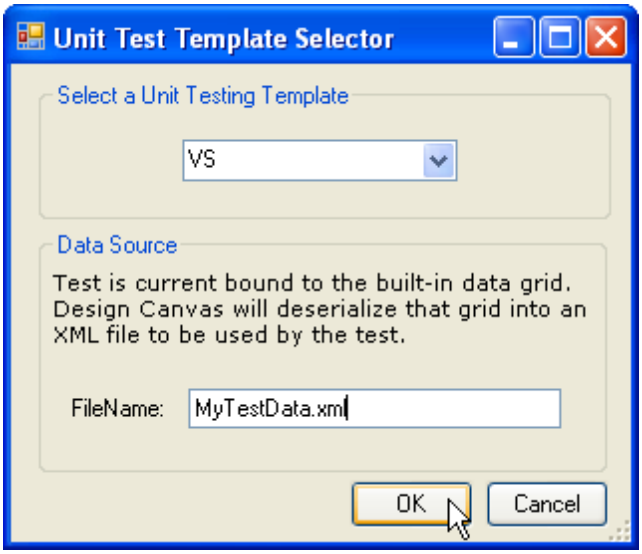



Code-behind steps automatically show up in the Test Tab. The screenshot below shows "MyCustomStep" has been added as the last test step in the Test Tab. See the chapters "WebAii Framework" and "MSTest" for more detailed information.



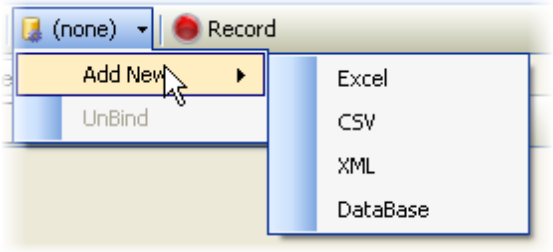

- **Convert to VS Load Test**: "Load" tests help measure how well a web site performs when stressed with a given amount of traffic. The "Convert to VS Load Test" button generates a Visual Studio load test based on your WebAii functional test created in the Test Tab. A file is created automatically with the name in the format of: "<my test name>.Load.WebTest". You can configure the load test to run any number of times and to simulate different browsers and types of networks. See the "Load Testing" chapter for more information.

- **Generate Unit Test**: "Unit" tests verify specific behavior as opposed to overall business logic. In other words, unit tests tell us that we're "doing things right", while functional/business logic tests tell us that we're "doing the right things". There are a number of popular frameworks for managing tests such as "NUnit" and "JUnit". The "Generate Unit Test" button converts your WebAii test into a unit test. When you click the button, the "Unit Test Template Selector" dialog appears and lets you choose a "Unit Testing Template", i.e. testing framework. If you have defined some rows of data in the Data Tab tab, the data is converted to XML (Extended Markup Language) and placed in a file name you specify for use by the selected testing framework.




See the "Unit Testing" chapter for more information.

- **Data Sources**: This drop down allows you to create new data sources to Excel, CSV (Comma delimited files), XML and external databases, select existing data sources and to UnBind the test from any data. See the Data Driven Testing chapter for more information.



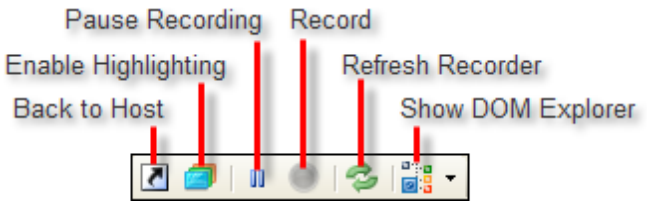

- **Go to Recording Surface**: This button navigates to the Recording Surface.

## 4.6 Recording Surface

Much of the time you spend with WebUI Test Studio will be in the Recording Surface. The Recording Surface browser provides the ability to record all your actions against a web page. The Recording Surface also lets you identify specific elements in the page and to handle many common dialogs that might pop up.

### 4.6.1 Toolbar

The Recording Surface toolbar controls your interaction with the browser page and has tools to start and pause the recording of test steps, refresh the recorder, display a DOM explorer and return back to the host (Visual Studio).




The parts of the toolbar are:

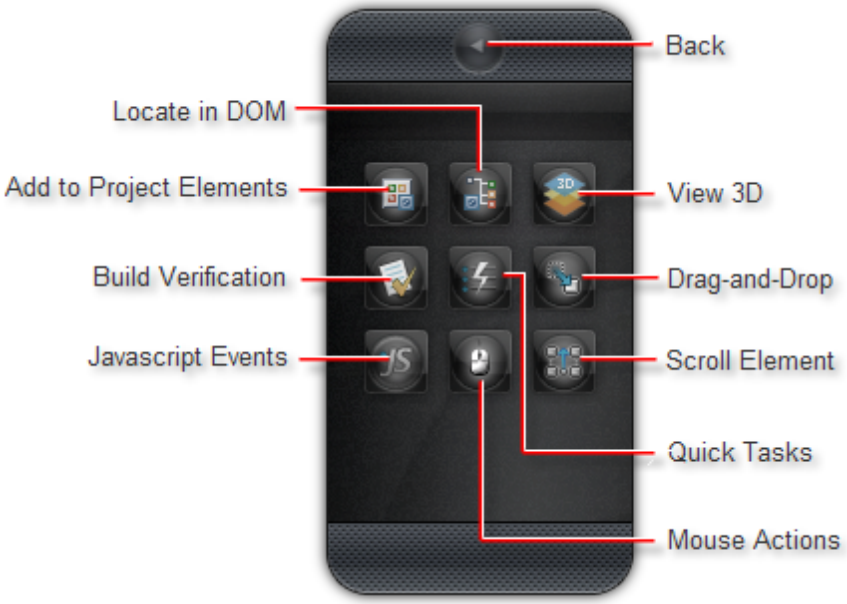
- **Back to Host**: Activates Visual Studio.
- **Enable Highlighting**: When pressed, elements in the page are highlighted as the mouse passes over. Highlighting allows you to add elements to the Elements Explorer and to display the Elements Menu for additional actions.
- **Pause** Recording: Use this button to temporarily disable recording.
- **Record**: When you click this button, actions din the Recording Surface are added as test steps in the Steps Tab.
- **Refresh Recorder**: Refreshes the DOM.
- **Show DOM Explorer**: Displays a tree view of the DOM (Document Object Model) showing all the elements in the page.

### 4.6.2 Elements Menu

When the mouse pauses over highlighted element in Recording Surface, a "Nub" appears. This rich element menu makes it easy to work with the recording surface. The Elements Menu is extensible and will be open to user applications that can incorporated into the menu.

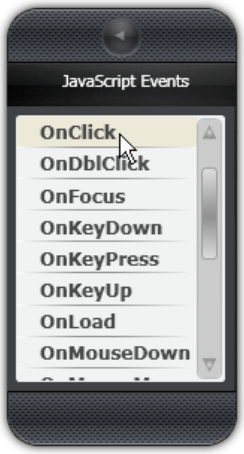
Clicking the Nub displays the Elements Menu.


The Elements Menu provides quick access to relevant functions right in the page you are testing.

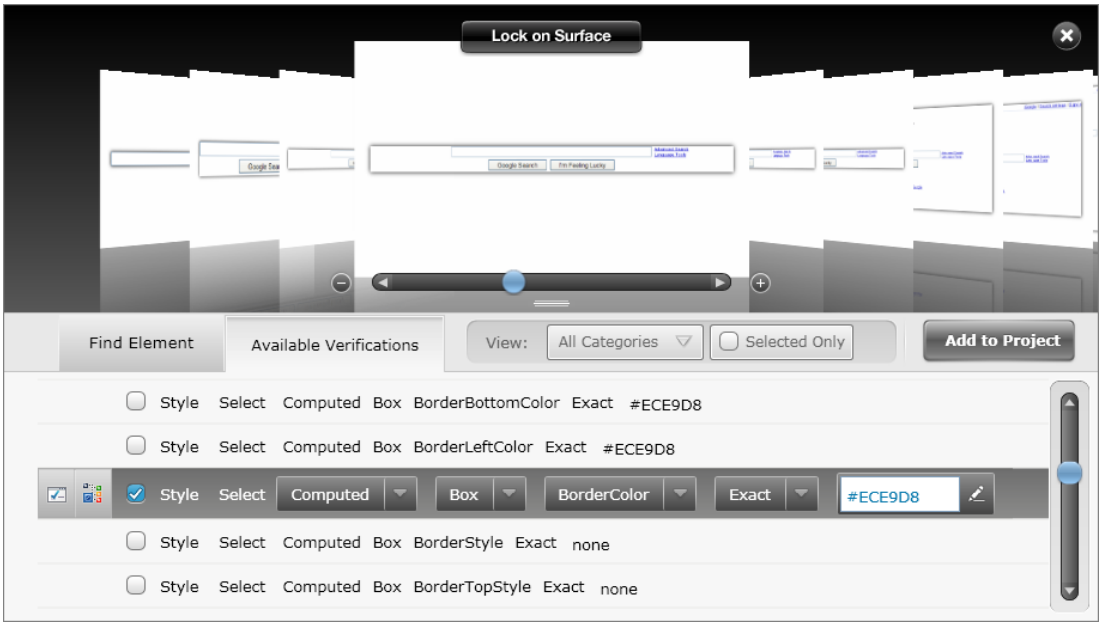



- **Locate in DOM** navigates to DOM Explorer and selects the corresponding element.
- **Add to Project Element** adds the highlighted element to the Elements Explorer.
- **Build Verification** navigates to the Sentence Verification Builder where you can interactively build verification criteria based on the elements Content, Style, Attributes or visibility. See the "Verification Engine" chapter for details.

- **Javascript Events** can be invoked against the highlighted element and supports OnBlur, OnChange, OnClick, OnDblClick, OnFocus, OnKeyDown, OnKeyPress, OnKeyUp, OnLoad, OnMouseDown, OnMouseMove, OnMouseOut, OnMouseOver, OnMouseUp, OnReset, OnSelect, OnSubmit and OnUnload.

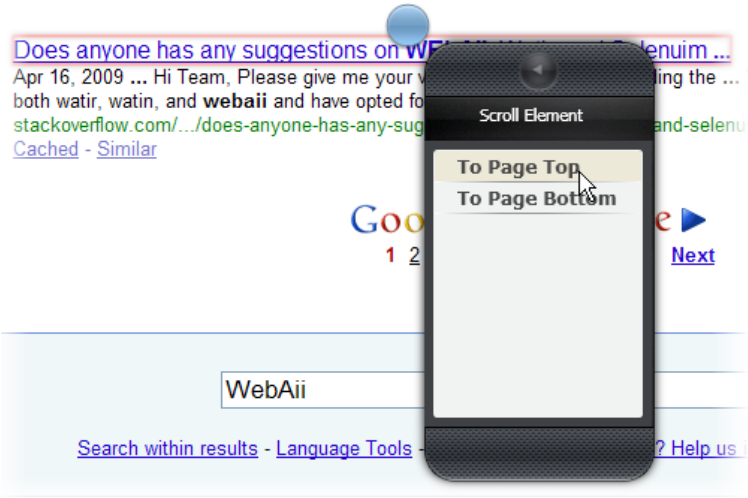



- **View 3D** provides an alternate view of all the elements in the DOM. The top portion displays elements in a 3D representation and allows you to "flip" through the elements as with a stack of cards, either by using the mouse to click on background elements, using the mouse wheel or using the slider. "View" controls allow you to filter the elements. The lower part of the screen has tabs that list the elements and allow you to select and build verifications. Buttons on screen let you "Lock on Surface", i.e. navigate to the Recording Surface with the corresponding element highlighted. The "Add to Project" button adds all verifications you have checked as test steps, all at one time. See the section "3D Viewer" in the "Verification Engine" chapter for more information.




- **Drag-and-Drop** allows you to interactively setup a drag and drop operation. You can drag an element onto another element or anywhere in the window. You can also configure the position of the dragged element and the drop target. See the "Drag and Drop" chapter for more information.

- **Scroll Element** scrolls the highlighted element to the top or the bottom of the page.




- **Quick Tasks** presents a context sensitive list of tasks that can be performed against the highlighted element. The screenshot below shows a Verify and a Wait task that is appropriate used with a RadControls for AJAX input control. Clicking the Quick Task creates a test step that can be manipulated in the Steps Tab and Properties pane.

- **Mouse Actions** can be invoked, as if the user was directly using the mouse to click or hover the highlighted element. This option mimics a click on the button and is browser based.

### 4.6.3 Common Tasks Menu

The Common Tasks Menu allows you to associate an element in the Recording Surface to a common task. You can add the element to the Elements Explorer, the 3D Viewer or the DOM Explorer.


To get at common tasks quickly, drag the Nub to the left side of the screen.


The Common Tasks Menu will pop out. Drop the element onto an icon to initiate a task you want performed against that element. The screenshot shows the Google logo image element being dragged to the Elements Explorer icon.

## 4.6.4 Walk Through

This walk through will help familiarize you with the Recording Surface, the Elements Menu and the Common Tasks Menu. During the walk through you will add test steps using the Recording Surface.

### Create a New WebAii Test

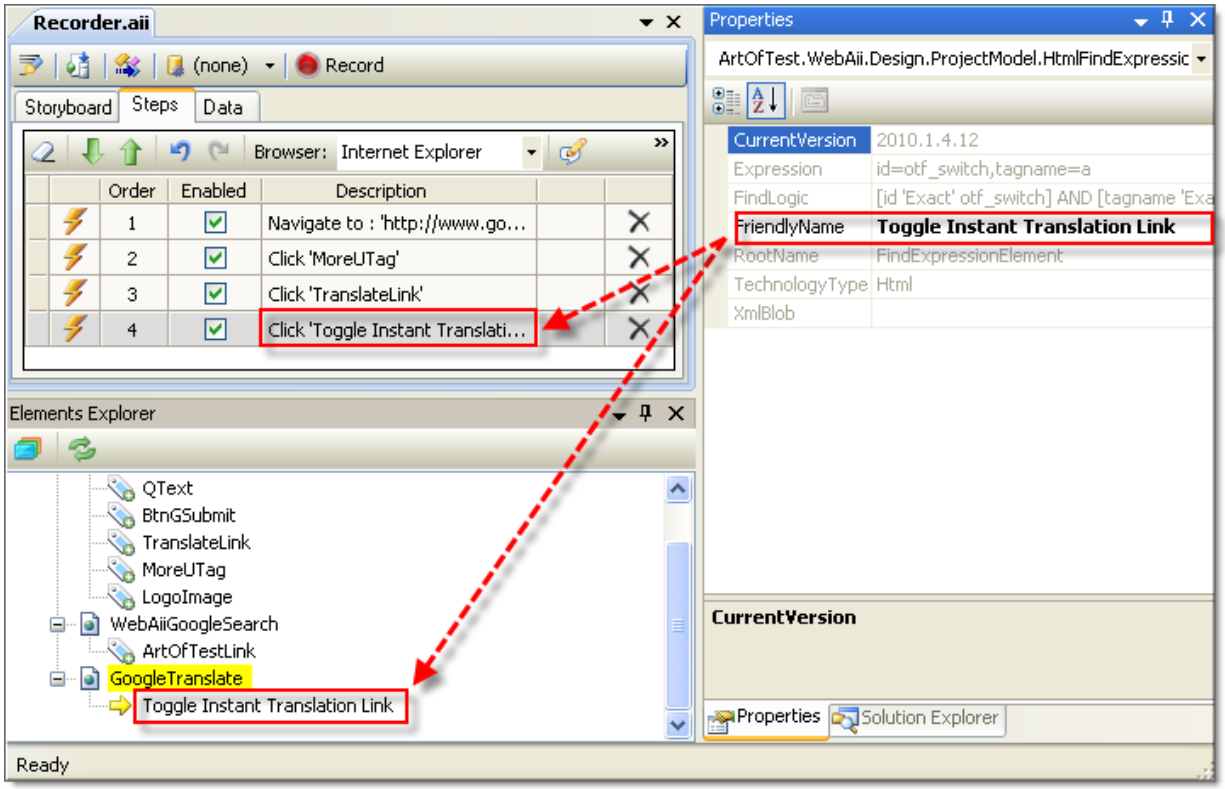
1) Open an existing Test Project or create a new Test Project (See the "Getting Started" walk through for directions on creating a new Test Project).
2) From the Solution Explorer, right-click the test project and select **Add > New Test...** from the context menu. This will display the "Add New Test" dialog.
3) In the "Add New Test" dialog, select the "WebAii Test" template, provide a meaningful Test Name and click **OK** to create the test.

### Use the Recording Surface to Create Test Steps

1) Click the Record button from the Test Tab toolbar. This step will navigate to the Recording Surface, with recording turned on.
2) In the browser address bar, enter "www.google.com".
3) Click the Go to Url button or press **Enter**. This step will display the Google home page.
4) In the Google home page, click the "More" link.


5) Click the "Translate" link from the menu. This step will navigate the browser to Google Translate page.
6) Click the link that toggles instant translation.

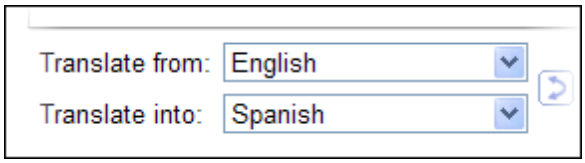
7) During the last step where the 'toggle instant translation' link was clicked, the link element was added automatically to the Elements Explorer. Locate the element in the Elements Explorer and select it. In the Properties pane, change the **FriendlyName** property to "Toggle Instant Translation Link". The new FriendlyName should reflect in both the Elements Explorer and any test steps where its used in the Steps Tab. The screenshot below shows the Properties pane, Elements Explorer and Steps Tab where the "Toggle Instant Translation Link" is shown.




8) In the Google Translate text box, enter the text "Hello world" (this is case sensitive, so enter this exactly to get a consistent translation).

9) Select "English" in the "Translate From" drop down list.

10) Select "Spanish" in the "Translate To" drop down list.




11) Click the Highlighting button from the Recording Surface toolbar.

12) Pass the mouse over the "Translate From" drop down list and wait for the Nub to display under the mouse. Click to display the Elements Menu.

13) Click the Elements Menu **Build Verification**  icon. This will display the Sentence Verification Builder.

14)In the "Available Verifications" area, locate the **Content** button and click it.

15)Change the comparison to "Contains" using the drop down list. Click the "pencil" button to allow editing, change the value to "English", then click the "pencil" button a second time to save your changes.


16)Click the **OK** button to create the verification test step and close the Sentence Verification Builder.

17)Pass the mouse over the "Translate Into" drop down list and wait for the Nub to display under the mouse. Click to display the Elements Menu.

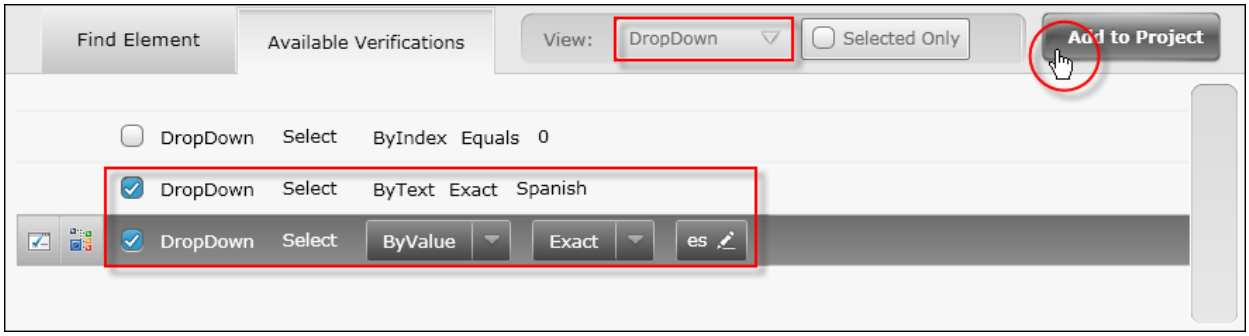
18)Click the Elements Menu **View 3D**  icon.

19)Click the **Available Verifications** tab.

20)From the View group, click the categories drop down list and select the "DropDown" item. This step will filter the list so that only the "DropDown" related verifications will show.

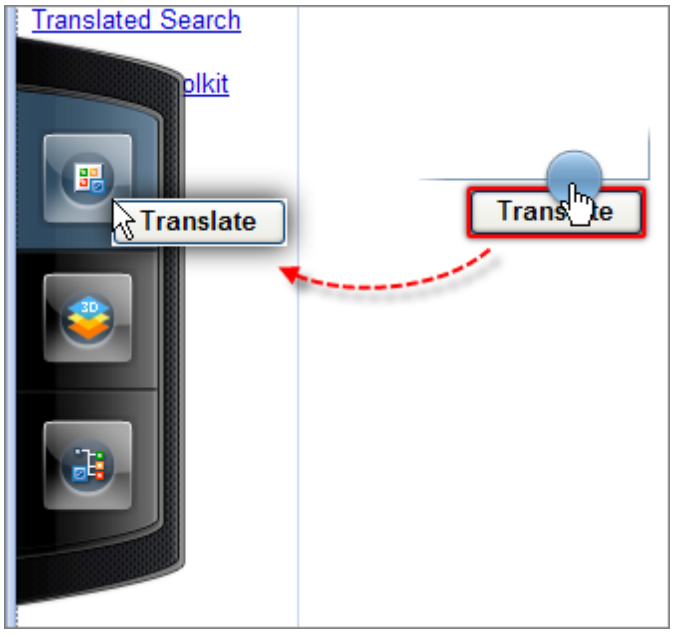
21)Click the "ByText Contains Spanish" and ByValue Contains es" check boxes. Change the comparison criteria from "Contains" to "Exact" using the drop down list for both verifications.

22)Click the **Add to Projects** button to add the verifications test steps.



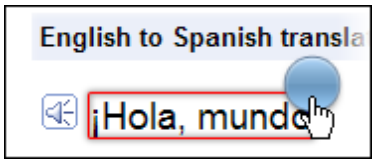

23)Click the **Close**  button located in the upper right of the 3D Viewer.

24)Pass the mouse over the "Translate" button and wait for the Nub to display under the mouse. Drag the Nub over to the left side of the screen. The Common Tasks Menu will appear. Drop the Nub onto the "Add to Element Explorer" icon. This step will add the "Translate" button to the Elements Explorer.

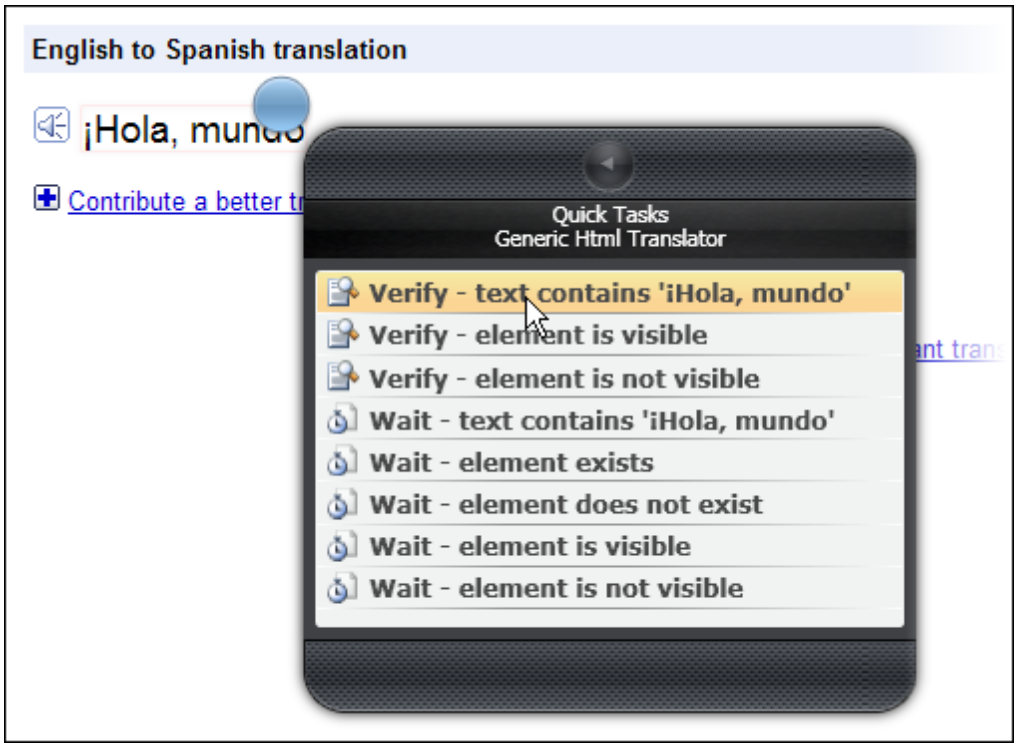



25)Click the "Translate" button.

26)Pass the mouse over the translated text and wait for the Nub to display under the mouse.

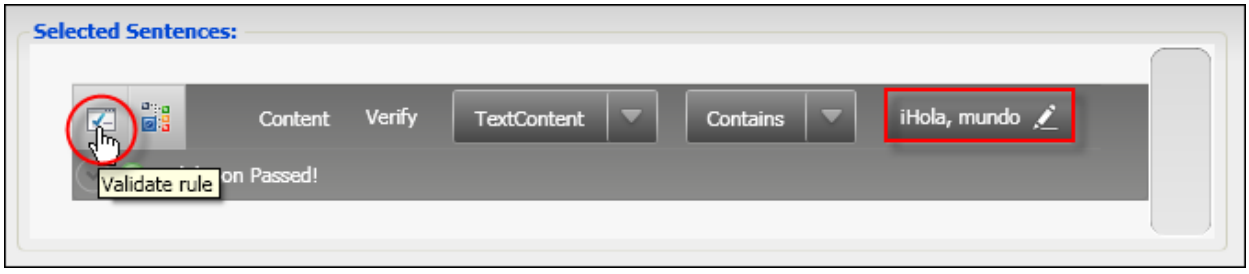


27)Select Quick Tasks  from the Elements Menu. Select the "Verify - text contains" task from the list and double click to create the test step and close the dialog.



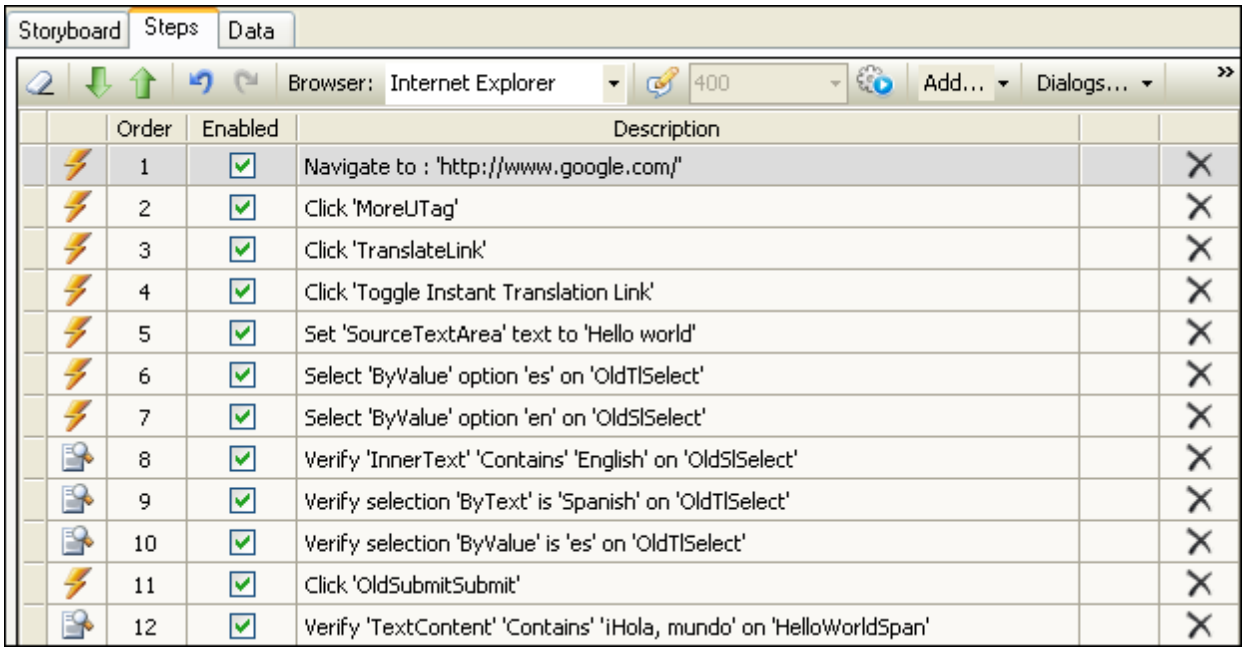

28)In the Steps Tab, double-click the last test step (this should be the "Verify - text contains" step just added from Quick Tasks. Double-click the test step. This will display the Sentence Verification Builder.

29)Change the comparison operator to "Exact" using the dropdown list. Click the "Validate Rule" button. The "Validation Passed!" message should display.




30)Click **OK** to change the validation and close the Sentence Verification Builder.

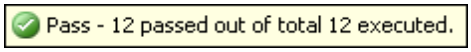
31)The steps in the Steps Tab should now look something like the list in the screenshot below.




## Run the Test

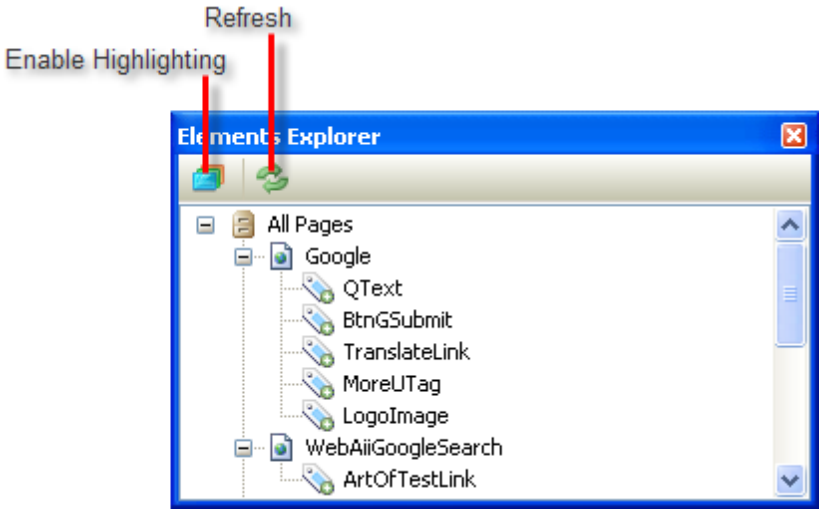
1) Click the Quick Execute button to run the test.

2) The test steps should run through to completion and all test steps should show as passed:



## 4.7 Elements Explorer

The Elements Explorer is similar to the DOM Explorer in that it displays a tree of elements, but the Elements Explorer only contains elements you want to use in your tests. Also, the elements in the tree view have properties that are more specific to testing. Although elements may be used in several tests and test steps, each element is shown only once in the Elements Explorer.



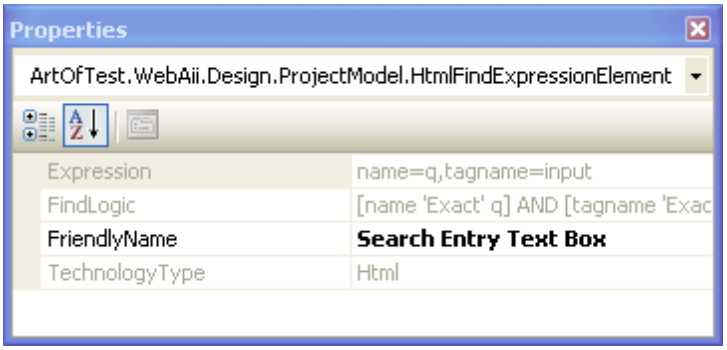

### 4.7.1 Toolbar

The toolbar options are:

- **Enable Highlighting**: When this button is pressed, elements selected in the tree view are highlighted in the Recording Surface.
- **Refresh**: Reloads the tree view.

### 4.7.2 Properties pane

As elements are selected, the Properties pane contents change to reflect the current element. The screenshot below shows the Google search text box element. Notice that the FriendlyName property, originally the cryptic "QText", is now "Search Entry Text Box". Friendly names are automatically reflected in Steps Tab test step descriptions. Also notice the FindLogic property is used for locating and identifying the element while the Expression property is a representation of the FindLogic. See the upcoming "Find Expression Builder" section for information on fine-tuning the find logic.



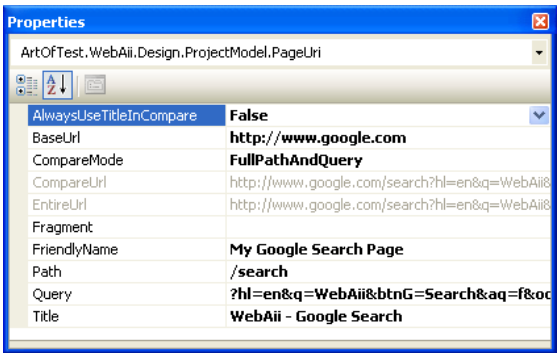

**Tip!**

Use friendly names to make the Elements Explorer tree view and test step descriptions more readable. You can also use friendly names to make the names in the code-behind conform to your organization's naming conventions.

**Gotcha!**

Friendly names also affect the naming conventions for objects in the code-behind. Try to set your FriendlyName properties straight away as the current version of the product does not automatically rename objects in the code-behind to match.

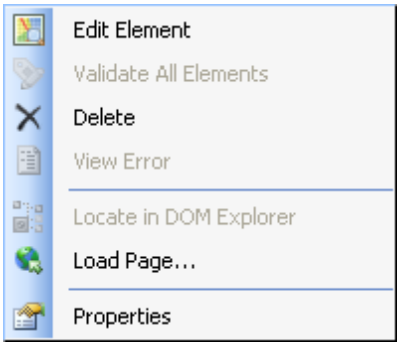
If we click on one of the pages in the Elements Explorer, the properties displayed are used to locate and identify the page.




- **AlwaysUseTitleInCompare**: Title is used regardless of CompareMode.
- **BaseUrl**: Url without full path or query string.
- **CompareMode**: The method used to match the URL. Possible values: BaseUrl, FullPath, FullPathAndQuery, FullPathAndQueryNoFragment, RelativePathOnly, RelativePathAndQuery, RelativePathQueryNoFragment, Title.
- **Fragment**: From the "#" to the end of the Url.
- **Path**: Located between the base url and the query string.
- **Query**: Query string, i.e. follows the "?".
- **Title**: The Title element located in the head tag of the page.

### 4.7.3 Context Menu

The screenshot below shows the context menu for items in the Elements Explorer tree view.




- **Edit Element** invokes the Find Expression Builder so that you can detail exactly how the element should be located.
- **Validate All Elements** is enabled when using the context menu against a page with elements that have verification test steps. A green check will appear next to all elements of the page that pass the verification and a red X icon next to any failures. If validation fails, you can click the **View Error** context menu item to read the detail.
- The **Delete** item is available for any element that isn't already involved in a test step.

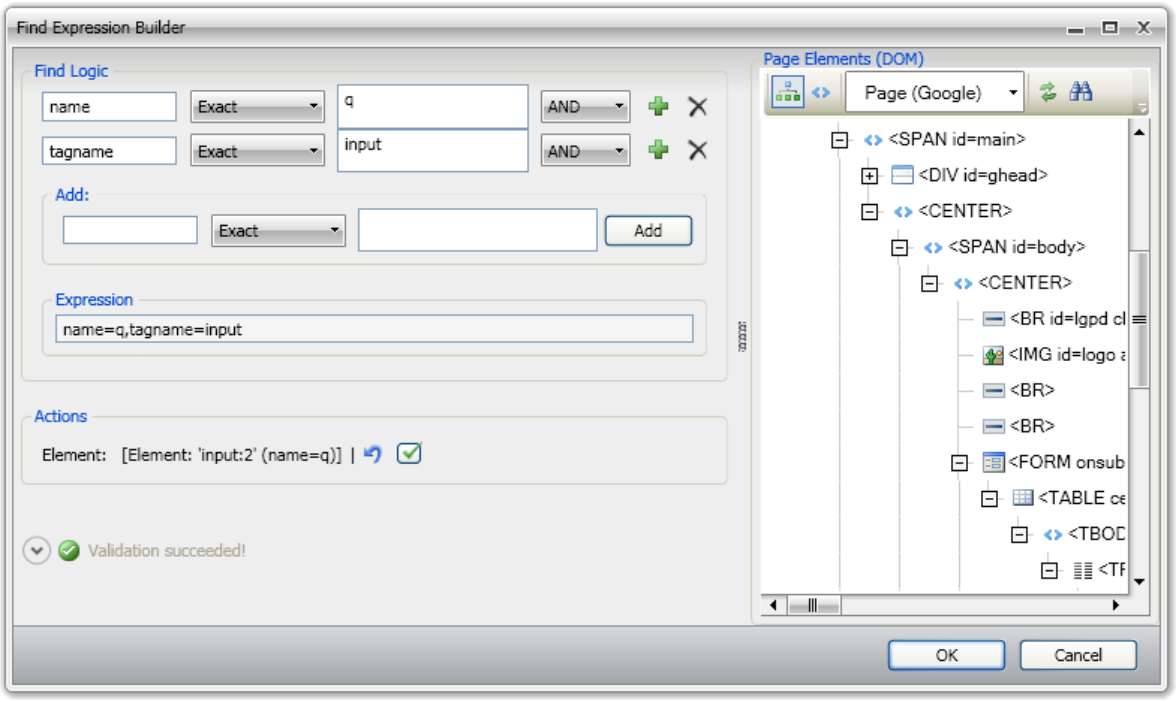
**Gotcha!**

Currently, if the test step has been converted to code-behind, you will be able to delete the item. The deletion can take place even though your code-behind may no longer compile.

- **Locate in DOM Explorer** navigates to the DOM Explorer and selects the item that matches the current item in the Elements Explorer. It will also highlight the corresponding item in the Recording Surface.
- **Load Page...** loads or re-loads the page that contains the element.
- **Properties** navigates to the Properties pane and displays properties for the selected element.

### 4.7.4 Find Expression Builder

The Find Expression Builder allows you to interactively build the comparison expression used to locate and identify an element. The screenshot below shows the Find Expression Builder populated with default values for the Google search text box element.



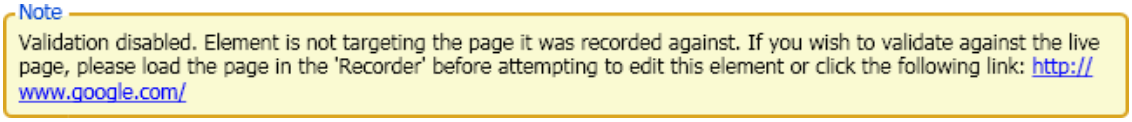

If we look at the element in the DOM explorer we can see that the actual HTML tag might look something like this abbreviated example:

```
<input name="q" .../>
```

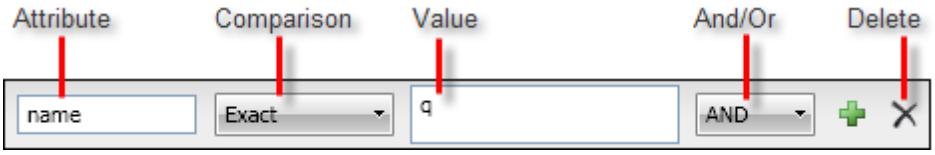
The expression in the "Find Logic" portion of the builder is telling us that the element must have a "name" attribute exactly equal to "q" and it must be an "input" tag. You can click the Validate button to make sure that the find logic will work.

**Tip!**

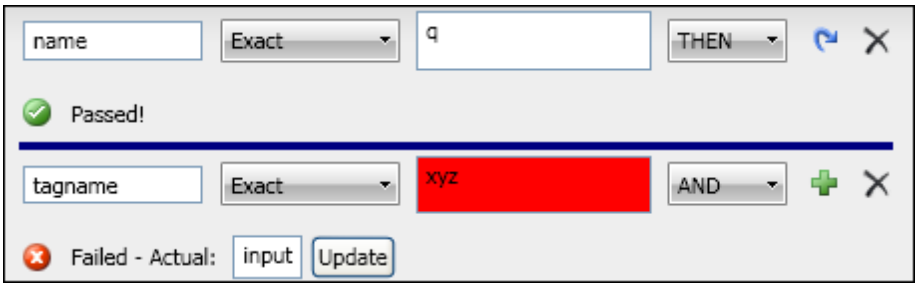
You must have the page loaded to validate. When you first load the Find Expression Builder, a note displays that has a link to the page. Click the link to load page.




Each line of criteria for the Find Logic window has a Delete button to remove the line and a Attribute and Value comparison. The And/Then drop down establishes a relationship between the criteria and any criteria following. By default, "And" will be used and all verification passes or fails as a single entity.




If you select "Then", this criteria is verified separately from any following verification. The screenshot below shows two lines of criteria separated by a "Then" where the first verification passes, but the second fails. The icon following the And/Then is an read-only indicator and has no effect when pressed.

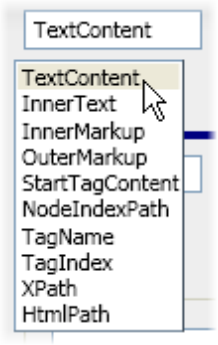



***Notes***

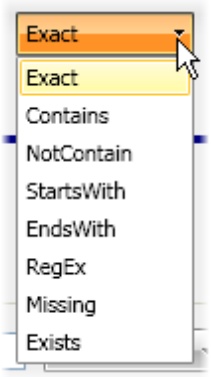
Also notice that the Failed message includes a suggestion to fix the problem. The Find Expression Builder already knows that we're comparing against "tagname" and that the tagname for the element is "input". Clicking the "Update" button will change the invalid value "xyz" to "input". Then, if you click the Validate button, the validation will pass.

### Building a Find Expression

You can click the Attribute textbox to get a list of possible attribute names for the element. The list includes the usual possibilities such as "InnerText" or "TagName". You can also build the find criteria by comparing to a node or HTML path. You can even use powerful XPath expressions to build the expression.



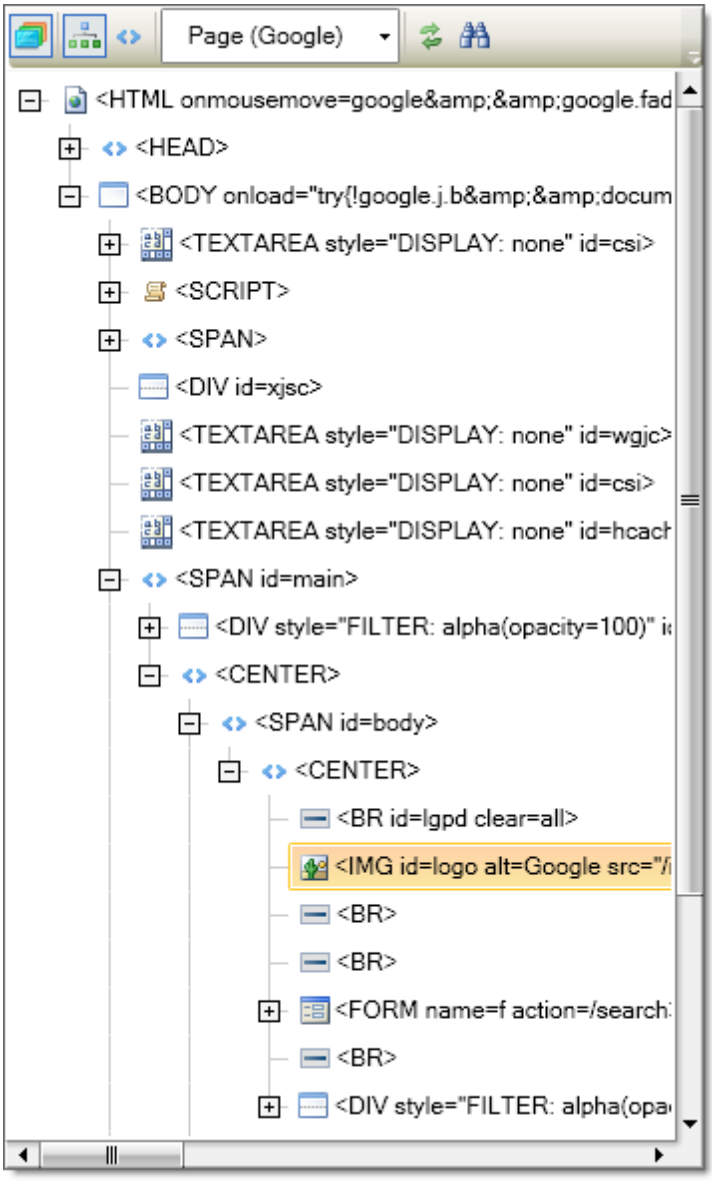

The comparison drop down list defaults to "Exact" and also includes Contains, NotContain, StartsWith, EndsWith, RegEx, Missing and Exists.



## 4.8 DOM Explorer

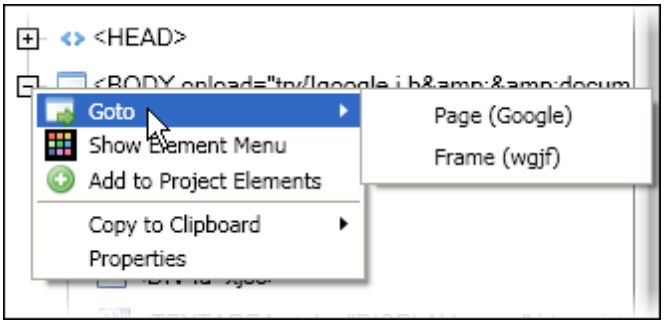
The DOM (Document Object Model) is a language neutral and platform independent abstraction that allows the content, structure and style of HTML pages to be updated dynamically. The DOM Explorer displays the DOM in a tree format where you see all elements of a page at one time. The DOM Explorer is helpful when the page is complex enough that simply using the mouse with the highlighter may not be enough to find the element you're looking for.

The DOM Explorer typically shows the HTML page as the parent element, with HEAD and BODY element forming the next level. The elements we're testing are usually within the BODY element.



### Context Menu

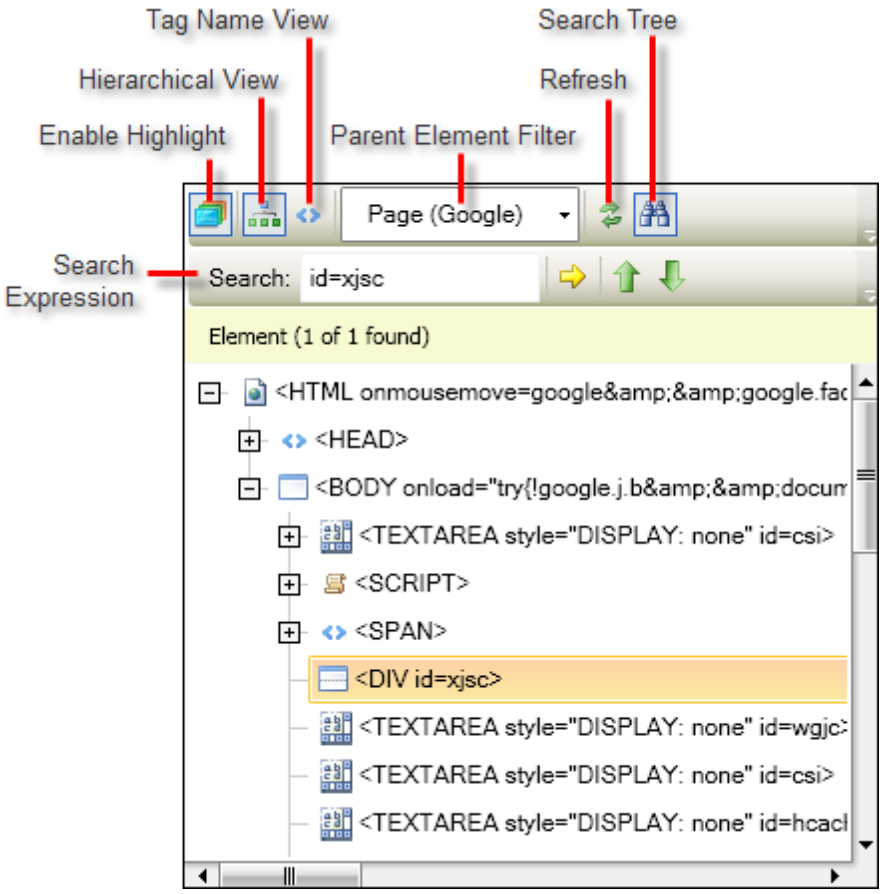
When right-clicking any of the elements, the context menu lets you to perform several operations against a DOM element. The **Goto** menu item lets you navigate to a parent element. The screenshot below shows the Goto submenu has entries for the entire page and for an IFrame element within that page.




The **Show Element Menu** item brings up the Recording Surface with the element highlighted and the Elements Menu showing. The **Add to Project Elements** option adds the selected element to the Elements Explorer. The Elements Explorer in turn allows you to name an element and use test specific properties. The **Copy to Clipboard** item lets you copy the DOM element as HTML, either copying the **Tag Only** or **Tag and Children**.

## Toolbar

The DOM Explorer toolbar allows you to display the DOM outline hierarchically or show in tag order. If you press the Enable Highlight button, elements are highlighted in the Recording Surface to match selections in the DOM Explorer tree view. Clicking the Refresh button re-reads the DOM and reloads the tree view. The Goto drop down list has entries for the entire page and for IFrame elements within the page. The Show Find Toolstrip button displays a second toolbar that performs searches over the DOM tree.

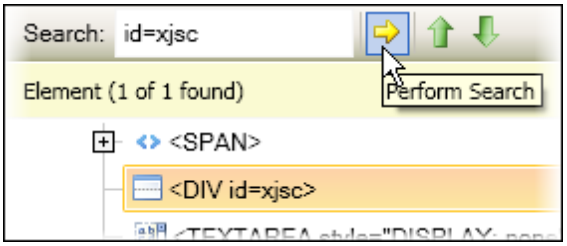



You can enter a Find Expression and click the Evaluate expression button. The Search Results drop down will display the number of elements found or an "Invalid Expression" error message. If elements are found, the first element is selected. Use the Previous/Next Result buttons to navigate between the found elements.

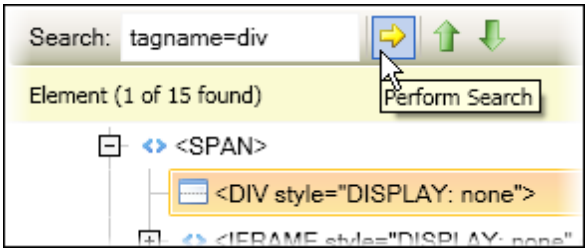
### Searching for Elements

A simple text search may not have enough horsepower to locate elements located deep in a large or complex DOM. For that reason, the DOM Explorer search tool has rich element identification capabilities that range from simple "find an element called 'myElement'" to complex criteria expressed using XPath and Regular Expression searches.

You can use simple find expressions that test an element against some value as shown in the screenshot below. Here we're looking for an element that has an "id" attribute of "pmolnk".



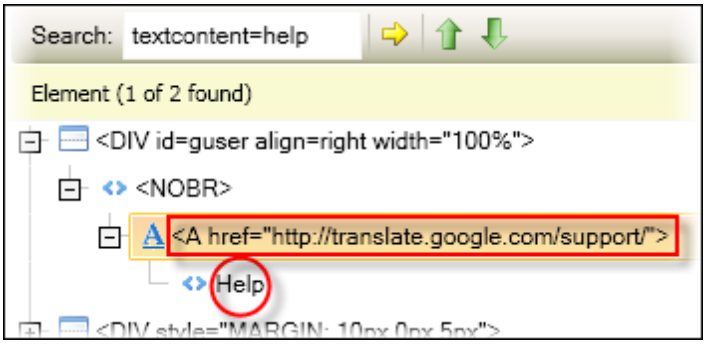

On the left side of the expression you can use any valid attribute name (i.e. "id", "div", "name", etc), or any of the following: **TextContent** (the element has text that matches), **InnerText**, **InnerMarkup**, **OuterMarkup**, **StartTagContent**, **NodeIndexPath**, **TagName**, **TagIndex** (zero based) or **XPath**. In the screenshot below we're looking for any tag named "div" and seventeen results are returned.



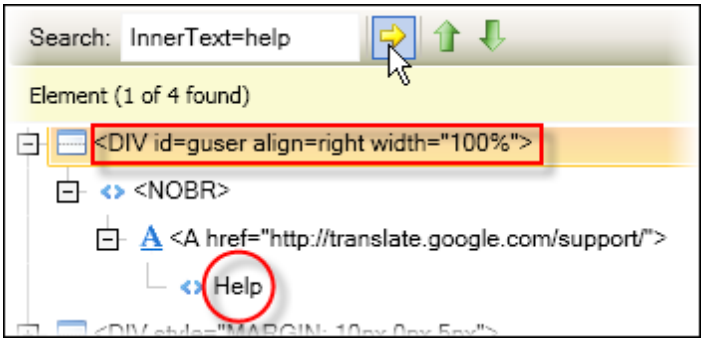

***Notes***

In Silverlight applications you can use the Silverlight specific search terms **AutomationId, TextContent, XamlTag,** and **Name**.

**TextContent** simply returns an element that has certain text within it. The screenshot below shows a search for the text content "help" where a link element (the "<a>" tag) that contains "Help" is returned.



**InnerText** looks for text content inside some set of element tags. This type of search returns all elements that contain the text content. The example in the screenshot below looks for the text "help" and returns all the nested elements that ultimately contain the matching inner text.




**From the Forums...**

**Question**: What is the difference between TextContent and InnerText?

**Answer**: InnerText is the combined text for a given node and everything below it. TextContent is only the text at the same level as the node. For example:

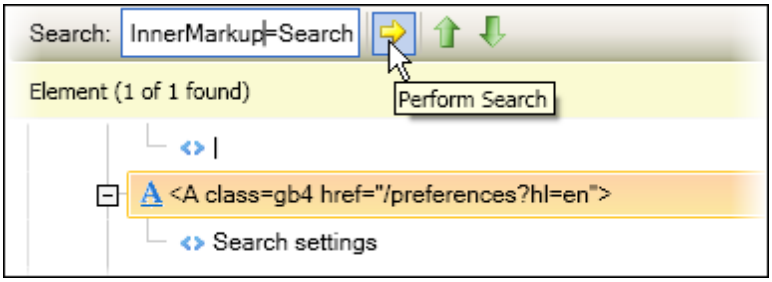
```
<div id="div1">
  Text1<div id="div2">
    Text2</div>
```

In this example, the TextContent of div1 is "Text1" while the InnerText for div1 is "Text1Text2". Likewise, InnerMarkup and OuterMarkup are "recursive" and look at all the nodes contained within a given node.

**InnerMarkup** returns an element with specific HTML markup inside it. Here is an example search that looks for a link tag that contains "Search settings".

```
InnerMarkup=Search settings
```

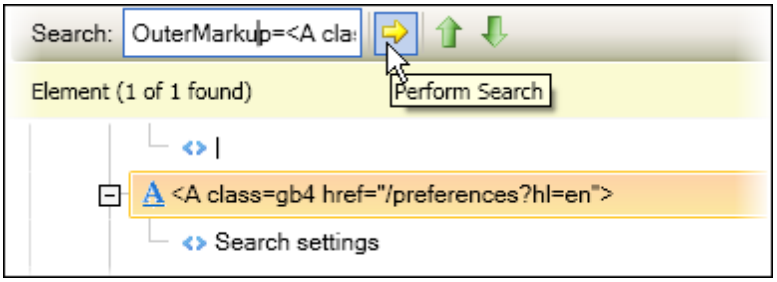
The search returns the element that contains the inner markup "Search settings". In this case the returned element a link "<A>" tag shown in the screenshot below.



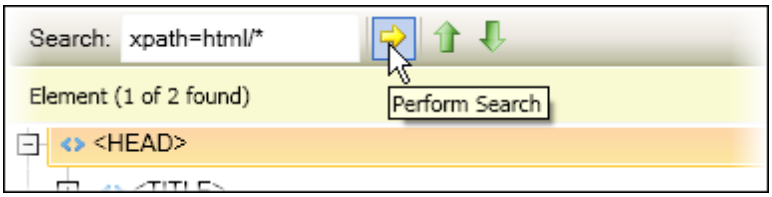

**OuterMarkup** looks for specific HTML markup *inclusive* of the element itself. If we change the left hand side of the previous search to "OuterMarkup" and include the entire tag:

```
OuterMarkup=<A class=gb4 href="/preferences?hl=en">Search settings</A>
```

...this search returns the same link element.




The DOM Explorer search tool recognizes XPath (XML Path Language) expressions, a syntax for selecting elements in an XML document. The screenshot below shows an XPath expression that selects child elements of the "HTML" element. The search returns two elements, the "HEAD" element for the page and the IFrame.




***Notes***

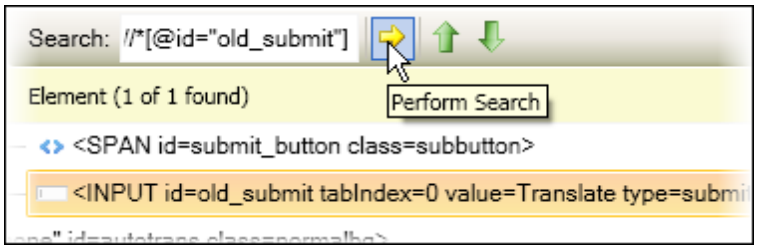
Find more XPath examples at http://msdn.microsoft.com/en-us/library/ms256086.aspx.

**Tip!**

XPath syntax can be daunting, so here's a shortcut: Install the Firebug debugger in the Firefox browser. Press the "Inspect Element" button  , then click the element that you need to create an XPath for. In the Firebug HTML tab, right-click the element and select "Copy XPath" from the context menu. You can use this path on the right hand side of the search. For example, using this method against the "Translate" button, the copied XPath on the clipboard is:

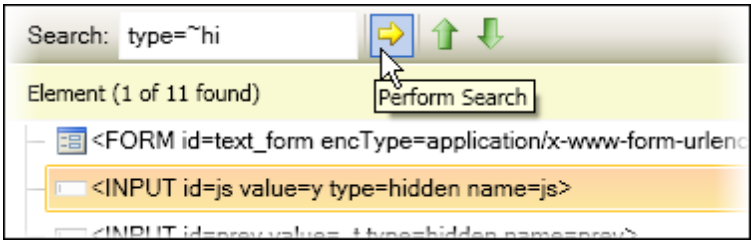
```
//*[@id="old_submit"]
```

The screenshot below shows the DOM Explorer search with the new XPath. A single element, the "Translate" button, has been located and selected in the DOM Explorer.

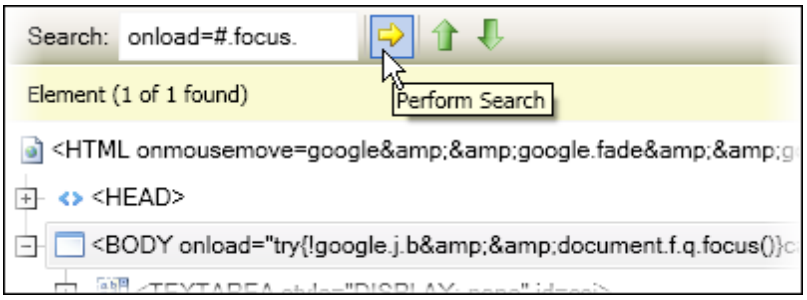



Firebug can be found at http://getfirebug.com/.

You're not stuck with only an "=" operator to make comparisons. You can use "~" (contains), "**!**" (does not contain), "**^**" (starts with), "**?**" (ends with) and "**#**" (regular expression). The screenshot below shows a search looking for elements where the "type" attribute starts with "hi". Eleven elements are returned where the type is "hidden".




**Regular expressions** ("regex" for short) are sequences of text characters used to describe a search pattern. Regular expressions are somewhat akin to "wildcard" characters, i.e. "*" or "?", but much more flexible and powerful. Regular expressions start with the "#" character. Regular expressions are a big topic all on their own, but here is a quick example showing a regular expression in the DOM Explorer search tool shown in the screenshot below. The example searches for an element with the "onload" attribute containing the word "focus".




**From the Forums...**

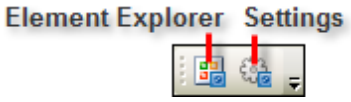
**Question**: I want to make sure my tests are easy to maintain. What are the best practices for finding elements?

**Answer**: Using a tag index in your test code results in test code that isn't very robust. A small change to the page being tested can easily break your code because the tag index number may change. Generally the best way to find a control is by its name or ID. When the control I want doesn't have a name or ID then I resort to finding the closest control that has a name or ID and navigate from it to the control that I want. Other testers like to use the XPath to the control. Another method is to use test regions (see the Test Regions chapter for more information).

# 4.9 User Settings

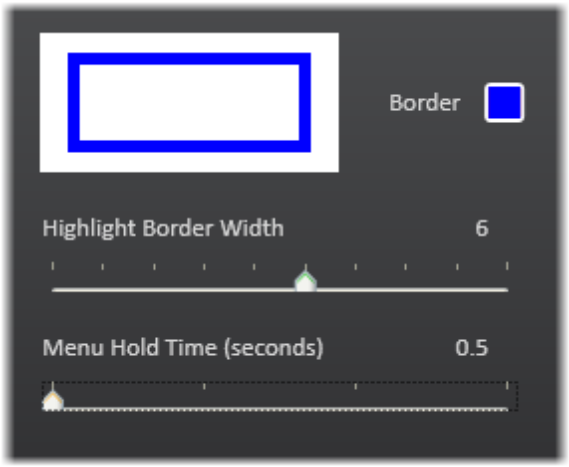
## 4.9.1 Overview

WebUI Test Studio user settings can be accessed from the WebUI Test Studio toolbar and from the Recording Surface. The screenshot shows the Settings button in the WebUI Test Studio toolbar.




## 4.9.2 Automation Overlay Surface

The "Automation Overlay Surface" defines the highlight border color, border width and the amount of time before the Nub displays. Click the Border button to display a color selector and use the sliders to adjust Highlight Border Width and Menu Hold Time. The screenshot below shows a blue border with a six pixel width and a Menu Hold Time of .5 seconds before the Nub displays. Delaying the Nub display can be helpful if you're trying to hover the mouse and the Nub is popping up too quickly.



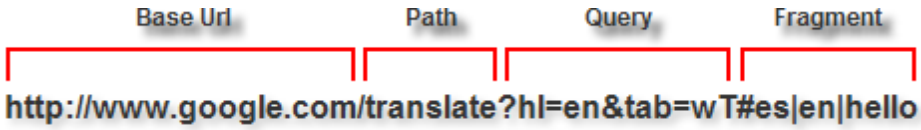

The highlight for these settings looks something like the screenshot below.

### 4.9.3 Recording Options

"Recording Options" configures how recording will take place. Some of the options are only needed for very specialized circumstances. Other options such as "Clear Url History" and "Enable Storyboard" configure the Recording Surface environment.

- **Base Url**: The base url is pre-pended to urls in your test steps. If our base url is "www.google.com" and we first navigate to Google, then click the "News" link, the steps in the Steps Tab will look like this screenshot where the first step navigates to "/", i.e. "www.google.com" and the second to "NewsLink", i.e. "www.google.com/Newslink".


- **Code Base Class**: This option allows you to create and use a specialized test class. For example, you could create a class that knows how to log to your corporations database. The code-behind for a WebAii Test uses "BaseWebAiiTest" by default. BaseWebAiiTest is an object that knows how to execute test steps, find elements on a page, perform actions against all browser types (i.e. click, scroll, etc) and log test results. BaseWebAiiTest also keeps track of the active browser and the test data.
- **Silverlight Connect Timeout**: The amount of time in milliseconds to wait for WebUI Test Studio to connect to a Silverlight application.
- **Elements Page Compare Mode**: This setting determines the **CompareMode** property to use when adding a page to the Elements Explorer. CompareMode determines when a page is active. CompareMode can be checked against the page's Title or one of multiple settings that look at various parts of a Url. The screenshot below shows the parts of a url. The base url is the part of the url before the first "/". The Path is the part of the url after the base and before the "Query" portion. The query is the portion after the "?" and before "#". The "#" marks the beginning of the "Fragment" portion of the url, if present.

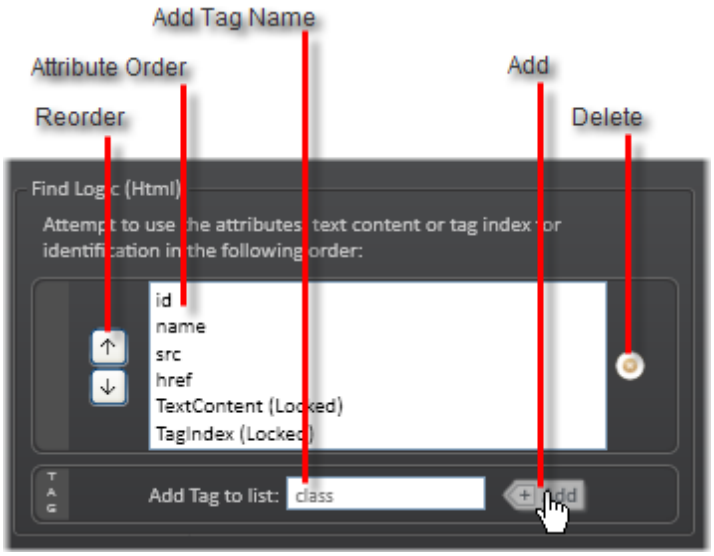



  The possible CompareMode values are BaseUrl, FullPath, FullPathAndQuery, FullPathAndQueryNoFragment, RelativePathOnly, RelativePathAndQuery, RelativePathQueryNoFragment and Title. Comparisons may ignore part of the url, e.g. "FullPath" ignores the query and fragment portions of the url.

- **Default DropDown Record Option**: This setting determines the **SelectDropDownType** property value will be used when recording selections in a drop down control. Possible values are ByText, ByValue and ByIndex.
- **Enable Storyboard**: By default, screenshots are automatically added to the Storyboard Tab. When this option is unchecked, a placeholder image is used. Uncheck the option when you want to conserve memory and disk space. The Scale slider adjusts the size of the Storyboard Tab between "10%" and "100%".
- **Verbose Mode**: If enabled, exceptions are written to the output window.
- **Simulate Real Clicks/Real Typing:** By default, clicks and typing are recorded as sending clicks or text directly to the element. Setting these options will cause clicks and typing to be simulated by sending Windows events.

### 4.9.4 Identification Logic

As elements are added to the Elements Explorer, WebUI Test Studio uses an intelligent element identification scheme to auto generate find expressions. When an element is first about to be added to the Elements Explorer, WebUI Test Studio tries to use the first item in the list, e.g. "id". Using this criteria, if the item is unique for the entire page, the element is added, a find expression is created and WebUI Test Studio stops evaluating. You can add new tags to the list, reorder them or delete them (except for TextContent and TagIndex which are locked from deletion).



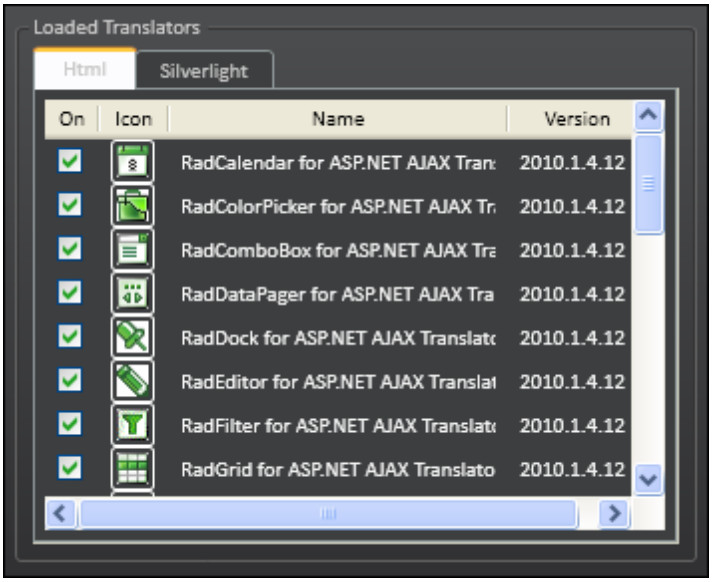

**From the Forums...**

**Question**: On the Web application we are automating, we have one page that has four Tables on it. When we record, we record the Tables Menu to add, delete; move; etc. to each table - this all seems to record correctly. During Playback it performs all the tests (adding, deleting, moving, etc.) on just the first table (all tests from all four tables). The same is happening with Verification. Looking at the recording WebUI Test Studio seems to have called all the four tables the same ID.

**Answer**: You can start by studying the HTML code generated by your web server and sent to the browser window. If all 4 tables are given the same ID by the web server, then that's what we're going to record against. You may need to modify your code on the server in order to give each table a unique ID. Another option you can try is to modify the Identification Logic of WebUI Test Studio. Open up the User Settings, go to the Identification Logic tab, then modify it such that it works properly for your web page. This may take some trial and error as this is a very advanced feature.

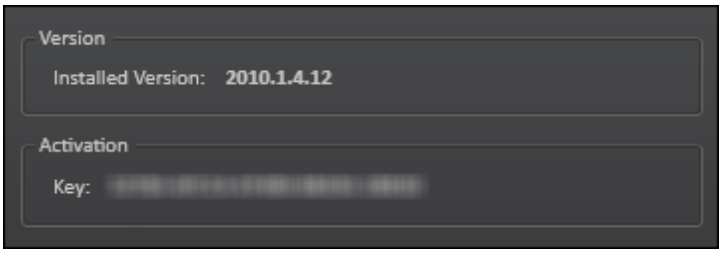
### 4.9.5 Translators

Translators are extensions to Visual Studio that open up an element to work with WebUI Test Studio, to allow interaction with the Elements Menu and to expose a rich set of verification tasks. The product ships with basic translators for HTML and Silverlight, and translators built specifically for each AJAX and Silverlight RadControl. WebUI Test Studio was built with extensibility in mind, so as additional controls become available, new translators can be plugged in. Telerik is committed to maintaining translators in step with RadControl changes, so you can expect the translators to always be up-to-date. The settings on this page list loaded translators and allow you to disable translators.




### 4.9.6 Installation

The installation page simply shows you the version of the product that's installed and the activation key.

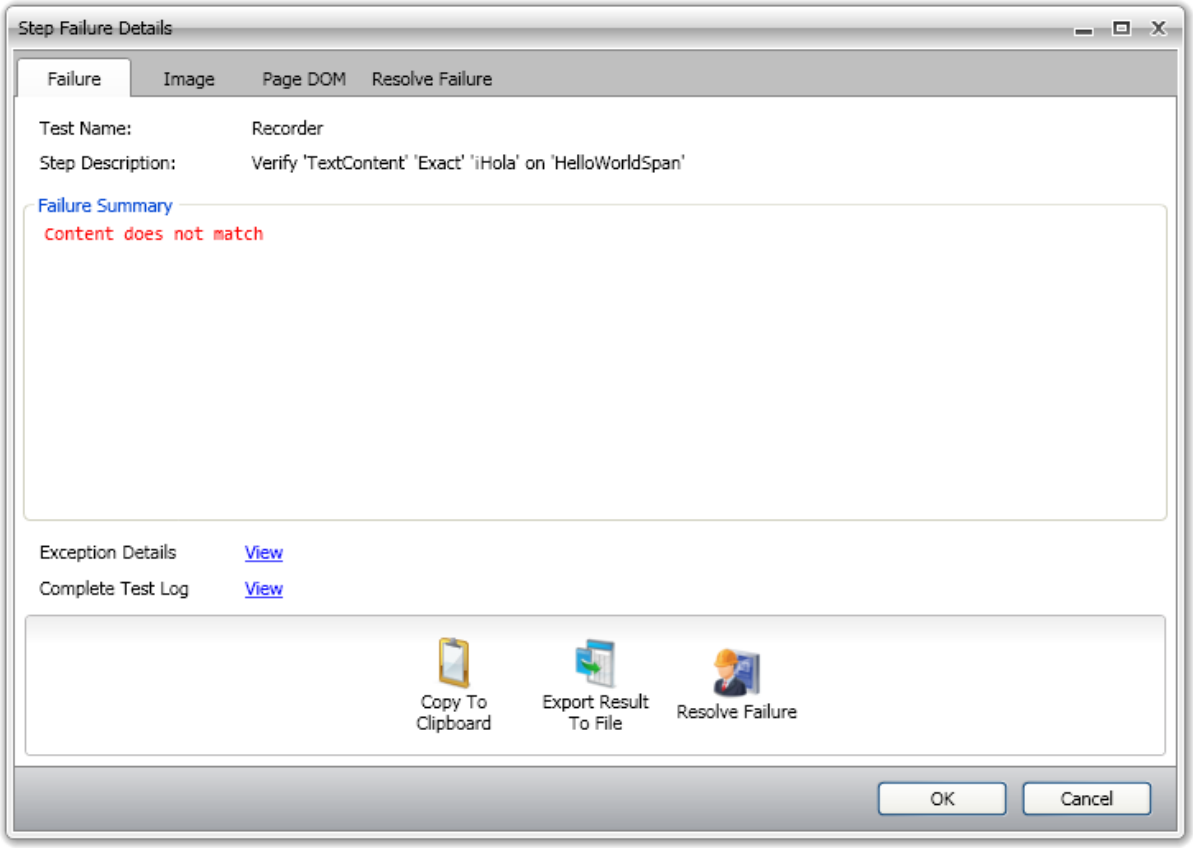


## 4.10 Step Failure Details Dialog

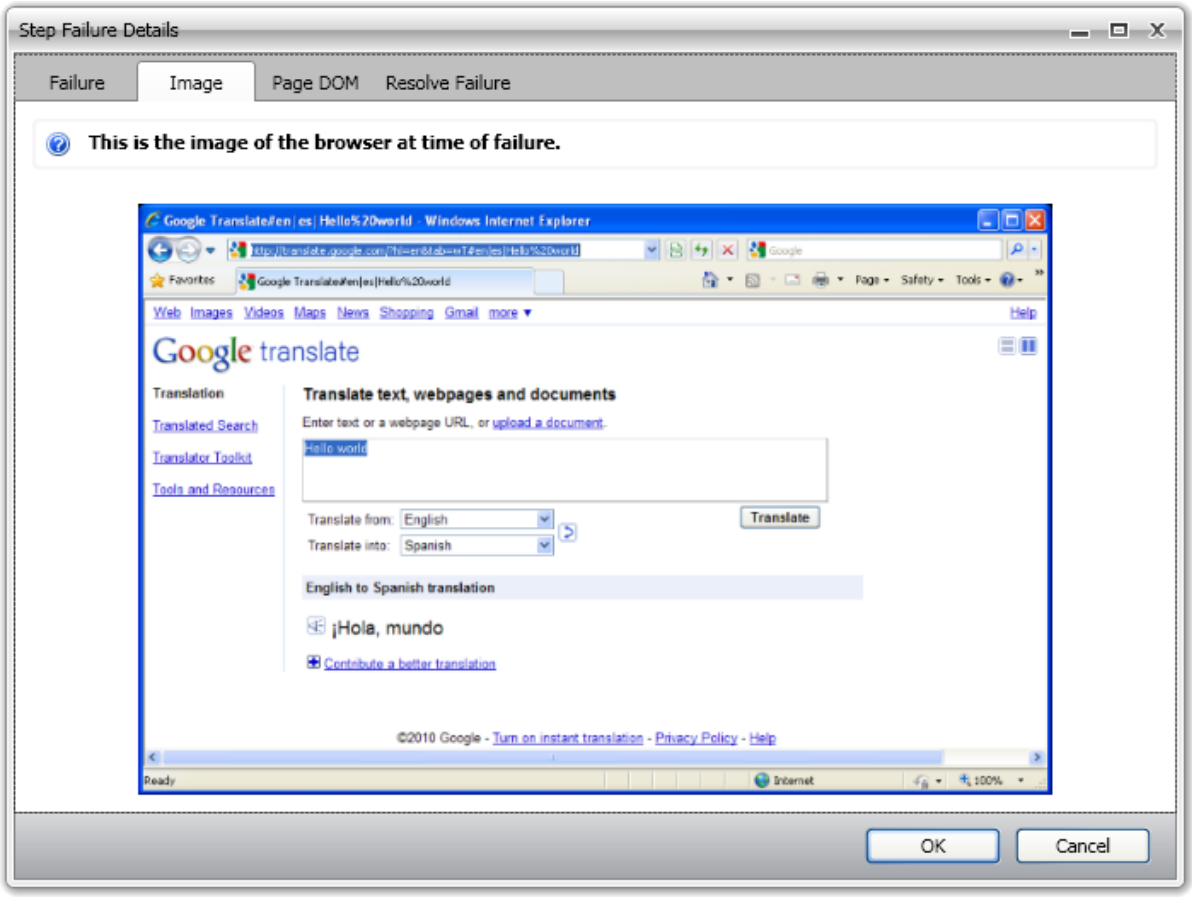
The Step Failure Details dialog collects all the information related to a single failed test step, including, failure details, screenshots, and a snapshot of the DOM, are all in this one window.

The **Failure** tab displays the test name, the test step description and a summary of what caused the step to fail. The example test step below is verifying that the TextContent exactly matches the string "iHola". The **Exception Details** View link under the summary lists the log for just the failed test step while the **Complete Test Log** View link displays the entire log (also accessible from the Steps Tab View Test Log button).

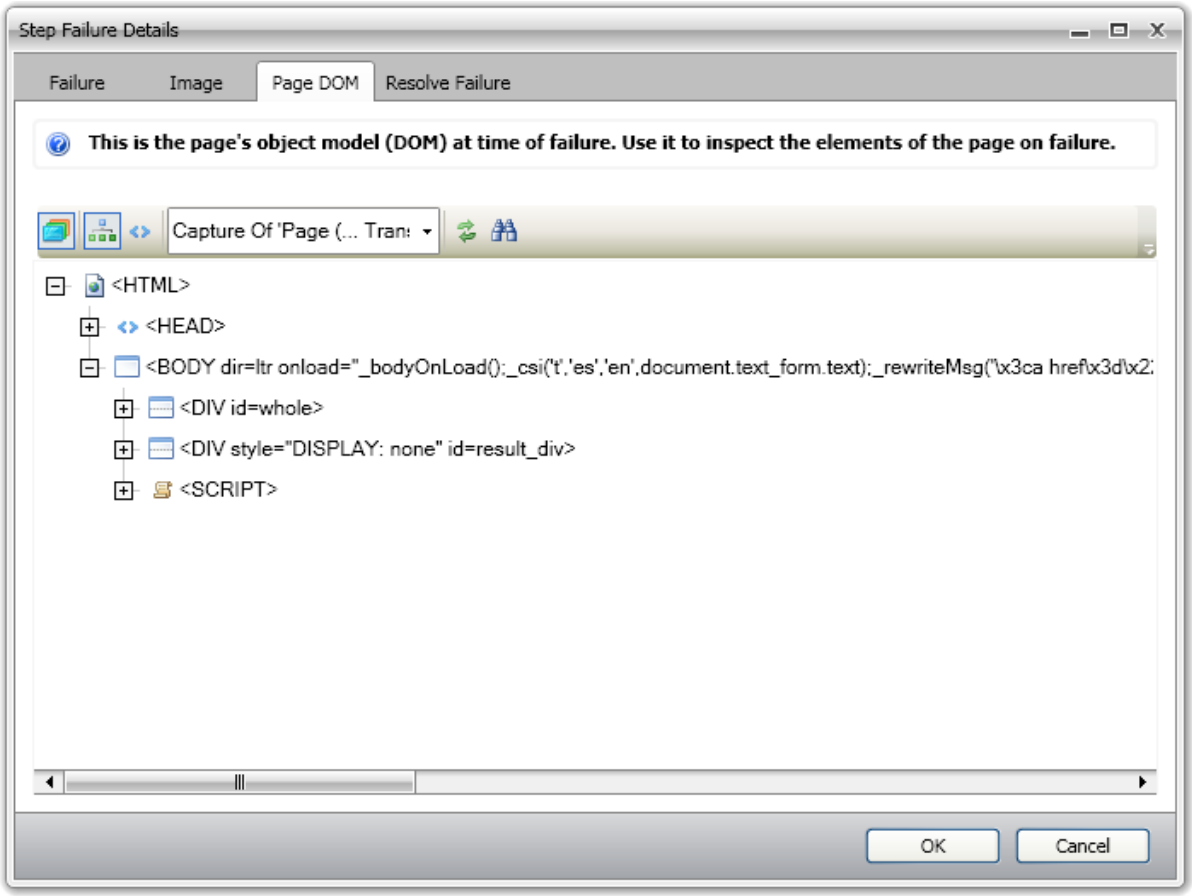
The **Copy to Clipboard** icon copies the exception details for the failed step to the clipboard where you can easily paste them to some other application. **Export Result To File** saves a zipped file containing a text file with the failed step log, the complete test log, a screenshot of the web page when the failure occurred and a snapshot of the DOM tree in XML form. The **Resolve Failure** icon navigates to the Resolve Failure tab.



The **Image** tab displays an screenshot of the browser taken when the failure occurred.

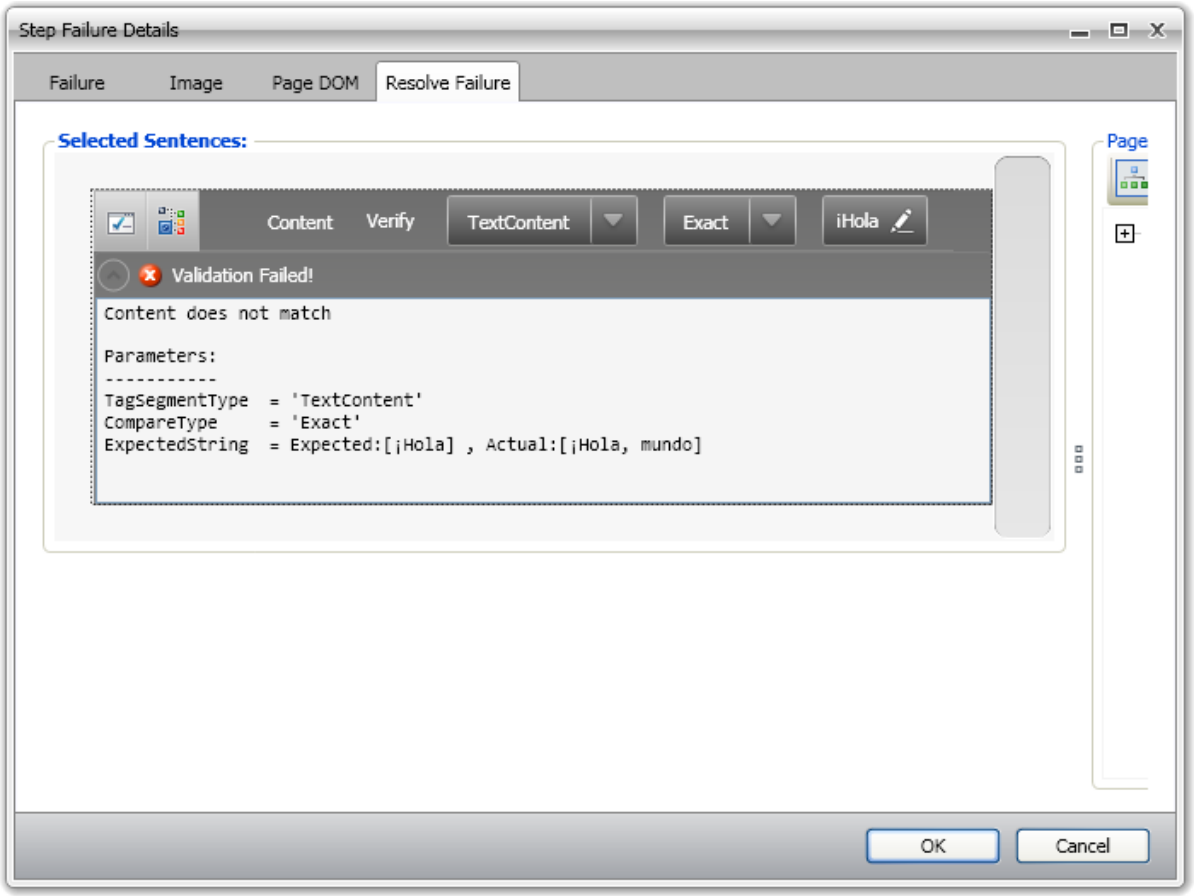


The **Page DOM** tab displays a tree view of the page's object model state at the time of failure. The DOM explorer here has the same controls and search utilities as explained in the previous DOM Explorer topic.



The **Resolve Failure** tab of the dialog provides the opportunity to identify and correct the issue that caused the failure. For validation steps, for example, the Sentence Verification Builder allows you to reload the page, make changes to the verification sentence and re-run the verification until the verification passes.

The example below shows that a validation failed because the TextContent was supposed to be "¡Hola", exactly, but when the validation button is pressed, the actual value turns out to be "¡Hola, mundo".




By changing the value to "¡Hola, mundo", the validation passes.

## 4.11 Wrap Up

This chapter explained how WebUI Test Studio is integrated with Visual Studio. The chapter started with a tour of the panels and toolbars in Visual Studio that make up the WebUI Test Studio environment. You learned how to use the Storyboard Tab to organize and navigate your test steps, the basics of building simple data driven tests and how to interact with the Visual Studio testing mechanism. You used the Recording Surface, the Common Tasks Menu and the Elements Menu to create test steps.

You worked with the Elements Explorer to organize elements used in test steps, learned how the Elements Explorer interacts with the Properties pane and fine tuned how elements are located and recognized. You used the Steps Tab to manage test steps. You also used Steps Tab to add unique types of test steps such as screen captures, delays and annotations. You also learned how to reuse existing tests.

You used the DOM explorer to look at all elements in a page. In particular, you learned how to search for elements using simple comparisons and also learned about more complex searches using comparison operators, Regex and XPath.

# Part V

Verification Engine

# 5 Verification Engine

## 5.1 Objectives

In this chapter you will learn what a verification is and how verifications are applied using WebUI Test Studio. You will learn how WebUI Test Studio implements verification using "sentences", how these sentences are structured and how sentences are accessed using WebUI Test Studio. You will learn the types of verification available for basic HTML and how the consistent interface provided by WebUI Test Studio allows Silverlight and AJAX elements to be verified using the same mechanism. You will use both the Sentence Verification Builder and the 3D Viewer to create verifications. Finally, you will create a suite of tests against a login page that exercises several different verifications.

**Find the projects for this chapter at...**

\Courseware\Projects\<CS|VB>\Verification\Verification.sln

## 5.2 Overview

Automated testing actually consists of two parts. The first part, automation, is the ability of WebUI Test Studio to manipulate the browser automatically without the tester having to intervene. Once you've automated the interaction you still have to test something, i.e. measure and record that some occurrence happened or do not happen as expected.

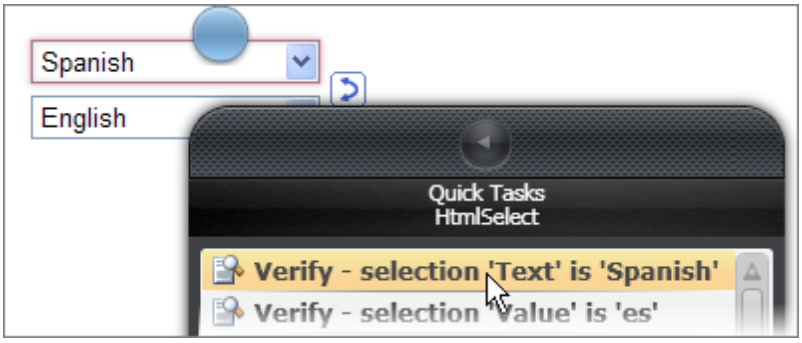
That's where verification come in. WebUI Test Studio verification allows you to measure against multiple criteria at one time and to build these measurements in an interactive manner without code. With verifications you detect if elements are in a particular state (e.g. visible, exist) and that attribute and properties compare with specific values. WebUI Test Studio can verify content, attributes, styles, visibility, drop down selections, checkboxes, radio buttons, tables and Silverlight property values. WebUI Test Studio implements verification through "sentences" that compare a portion of an element to a value, e.g. "textbox content is equal to 'order 8599'" or "image path contains 'http://www.falafel.com'".

"Translators", i.e. extensions that open up the internals of a control to WebUI Test Studio, allow for rich verifications against the exposed portions of the translated control. See the Translators chapter for more information.

## 5.3 Verification Access

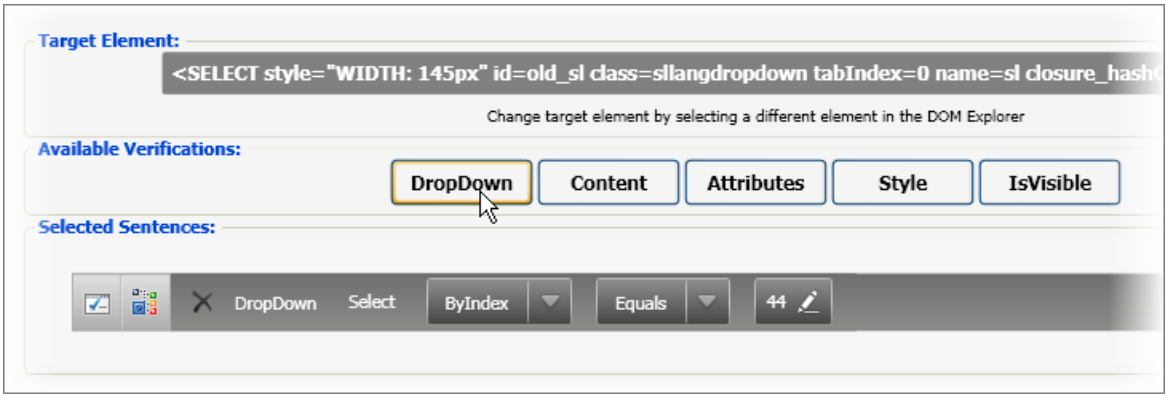
You can reach sentence verification through a number of avenues:

- You can reach sentence verification builder through the Elements Menu Quick Tasks button. The screenshot below shows a "Verify" quick task selected for a highlighted drop down list.




A single verification is created for you automatically where the verification type is inferred from the task and is read-only. This route doesn't provide options for adding more verifications or changing the type. The screenshot below shows a single "DropDown" verification. The verification checks that the selected item text in the drop down contains "Spanish".

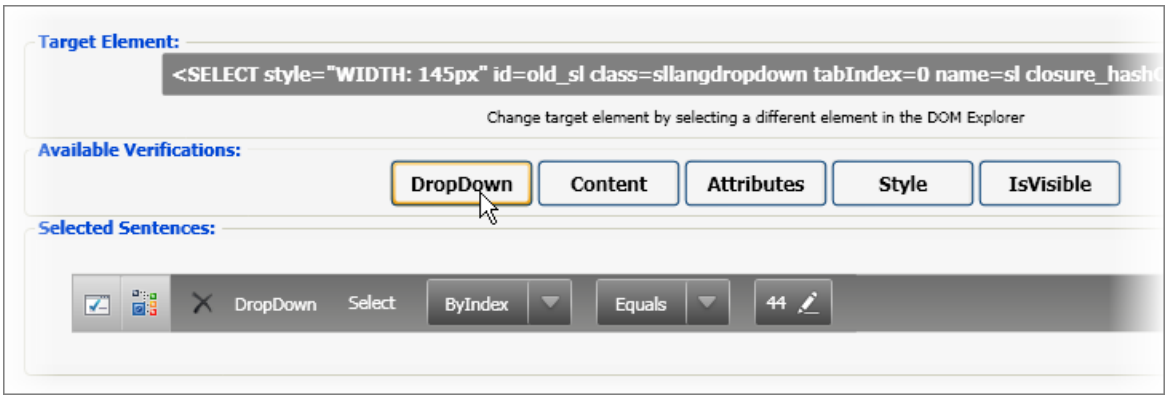

- Through the Elements Menu Sentence Verification Builder you can access all available verification types and create multiple verifications for a single element. The element can be changed on-the-fly by selecting a new target item in the DOM Explorer. The screenshot below shows a "DropDown" type verification added where the selection index must equal the value "44".



- The 3D Viewer takes verifications a step further by letting you generate multiple pre-built verifications all at one time. The screenshot below shows two "DropDown" verifications checking against both index and text. Because the 3D Viewer loads an element and everything that contains it, you can switch between elements and add verifications to the project as you go.

## 5.4 Sentence Verification Builder

The Sentence Verification builder has three main sections: the **Target Element** shows the complete element in HTML or XAML (Extended Application Markup Language used for Silverlight). The **Available Verifications** section is populated with buttons for each type of verification that can be applied to a given element. More verification types may be available depending on the presence of "Translators" (see the "Translators" chapter for more information). Click the buttons to add verification sentences to the **Selected Sentences** area.




The **Selected Sentences** area allows you to interactively build a verification rule, validate it against a live document or to invoke the DOM Explorer with the target element selected. Sentences have a general "key - comparison - value" structure. The "key" is some aspect of the markup that we want to compare a value against.

Once you have added a sentence using one of the Available Verifications buttons, use the drop down lists to select a key and comparison, then use the Edit "Pencil" icon to open the value for editing. After editing the value, click the Edit button a second time to close it. Then click the Validate Rule button to verify that the rule is satisfied.


You can create or modify several verifications at one time. When your sentences are complete, click the **OK** button to create the verifications in the Steps Tab or **Cancel** to abandon your edits.

## 5.5 Sentence Structure

The structure of the "sentence" changes according to verification type. The general pattern is a "key" followed by a comparison and then a value. For example, a Content verification has the following parts: Content Type - Comparison - Value. The screenshot shows a verification of a text box where the *InnerText Content* must *Compare* exactly with the *Value* "Spanish" to satisfy the verification rule.


Here are the general structures for some of the basic verification types used against HTML elements.

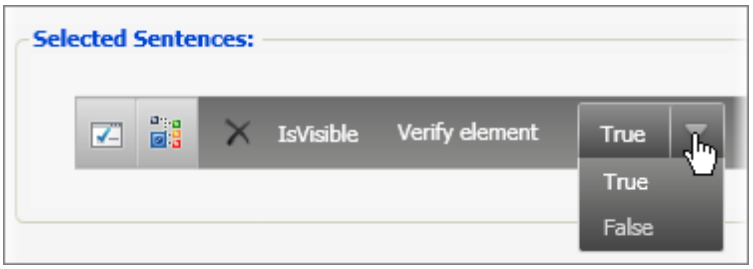
| Verification Type | Structure |
|---|---|
| Content | Content Type-Compare-Value |
| Attribute | Attribute Name-Compare-Value |
| IsVisible | Value |
| Style | Inline/Computed-Category-Attribute-Compare-Value |
| DropDown | Attribute Name-Compare-Value |

Notice that in some cases there may be additional "keys" that come before the comparison. For example, Style actually has three different pieces: Inline/Computed, a style category and the attribute name. For simple Boolean true/false verifications, such as "IsVisible", only the value part of the sentence is used.

## 5.6 Verification Types

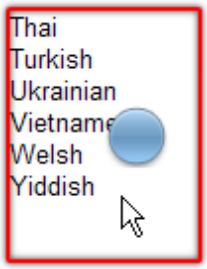
### 5.6.1 IsVisible

The most basic verification you can perform is against an element's visibility. WebUI Test Studio determines visibility by following the CSS chain and analyzing "visibility" and "display" attributes for an element. Despite the underlying complexity, all you need to do is set the verification value "True" to test if the element is visible or "False" to check that the element is not visible.



## 5.6.2 Content

Content verifications test some portion of element content against a string of characters. Consider a table cell tag "<td>" that has content shown in the screenshot below. What comparisons would successfully match?




The full HTML for the table cell is shown below. It has a starting "<td>" tag that contains a style attribute followed by several bits of text content separated with break "<br>" tags and ending up with a closing tag "</td>".

```
<td style="white-space:nowrap">Thai<br>Turkish<br>Ukrainian<br>Vietnamese<br>Welsh<br>Yiddish
</td>
```

### Content Type

The portion of the element that is matched against is determined by a content type. The **InnerText** content type exactly matches "ThaiTurkishUkrainianVietnameseWelshYiddish", i.e. the text contained by the element without markup.


**InnerMarkup** matches text content + markup located inside the table cell element.


**OuterMarkup** matches the entire table cell element, including the start tag, content and closing tag.


**StartTagContent** matches only the start tag, not the content between the start and end tags, and also excludes the ending tag.


**TextContent**, like InnerText content matches "ThaiTurkishUkrainianVietnameseWelshYiddish". So what's the difference between TextContent and InnerText?

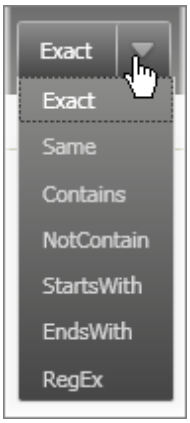

Similar to the searches discussed in the Visual Studio Integration chapter on the DOM Explorer, TextContent only looks at the content of the immediate element while InnerText is "recursive" and looks at all the text content in elements contained by the current element. Let's look at the HTML table that contains the "<td>" cell we've been working with:


In this case, TextContent is an exact match to *"Languages available for translation:",* i.e. the content of the table, but not of any of the cells.


...while InnerText looks at the table element content and all the cells element content within the table:

> *"Languages available for translation: AfrikaansAlbanianArabicBelarusianBulgarianCatalanChineseCroatianCzechDanishDutchEnglishEstonianFilipinoFinnishFrenchGalicianGermanGreekHebrewHindiHungarianIcelandicIndonesianIrishItalianJapaneseKoreanLatvianLithuanianMacedonianMalayMalteseNorwegianPersianPolishPortugueseRomanianRussianSerbianSlovakSlovenianSpanishSwahiliSwedishThaiTurkishUkrainianVietnameseWelshYiddish".*

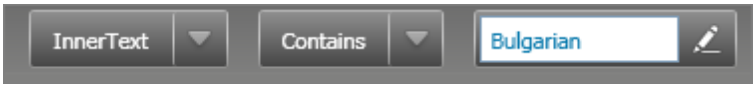
### Comparison




Content is equated against a value using a comparison. The possible comparisons depend on the content type. For strings of alpha numeric characters, the possible comparisons are shown in this screenshot.

**Exact** performs a strict match against the value and is case sensitive. **Same** also performs a precise match but the value can be upper or lower case. Consider the previous table example where the content is *"Languages available for translation:"*. If we change the content to "languages available for TRANSLATION:" where the "L" in "languages" is lower case and "TRANSLATION" is all uppercase, the Exact comparison will fail and the Same comparison will pass.

You will often need partial matches against an element value. For these, use the **Contains**, **NotContain**, **StartsWith** and **EndsWith** comparisons. The screenshot below shows a comparison of InnerText that contains the word "Bulgarian".




Regular expressions ("regex" for short) are concise sequences of text characters used to describe a search pattern. Regular expressions are somewhat akin to "wildcard" characters, i.e. "*" or "?", but much more flexible and powerful. Complex matches can be performed using the **RegEx** comparison. The example below matches InnerText containing either "Estonian" or "English". This is a relatively simple expression by RegEx standards, so you may want to look up a good regular expression reference online, particularly if you have a number of complex comparisons that require this potent syntax.



Numeric comparisons have their own set of operators as shown in this screenshot.

### Value

The **Value** portion of the content verification is simply text entered to the edit box. Click the edit "Pencil" button to open the edit for modification and click the same button a second time to close it.

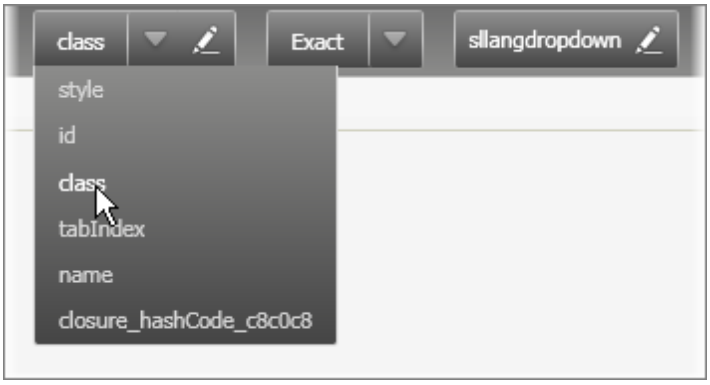
**Gotcha!**

After editing a value, be sure to close the edit box.


If you click the comparison button without closing, you will be comparing against the initial value, not the modified text that's currently in the edit box.

## 5.6.3 Attribute

The Attribute verification allows a great deal of flexibility against any attribute in an element. The attribute name drop down lists all the attributes in the element. When you select an attribute, the Value will automatically populate with appropriate matching text.




The Comparison and Value portions of the sentence work identically to the Content verification.

### 5.6.4 Style

Style verification has a relatively complex sentence structure. The first part is style type that can be "Computed" (follows the CSS chain to get the active style setting) or "Inline" (uses only styles applied directly to the element). The next drop down is a category, such as "Display", "Font" or "Text". The category is used to filter the style attributes that populate the next drop down list to the right. The last two drop down lists are the comparison and value. The screenshot below shows a style verification where the category is "Display", the style attribute is "Top" and the comparison is Exact against a value of "758px".


The drop down values were populated automatically against the HTML element shown below.

```
<img alt="Falafel Blog" src="images/blogs_home.gif" style="border-top-style: solid;
  border-right-style: solid; border-bottom-style: solid; border-left-style: solid; border-top-width: 0px;
  border-right-width: 0px; border-bottom-width: 0px; border-left-width: 0px; position: absolute;
  top: 758px; left: 25px; " />
```

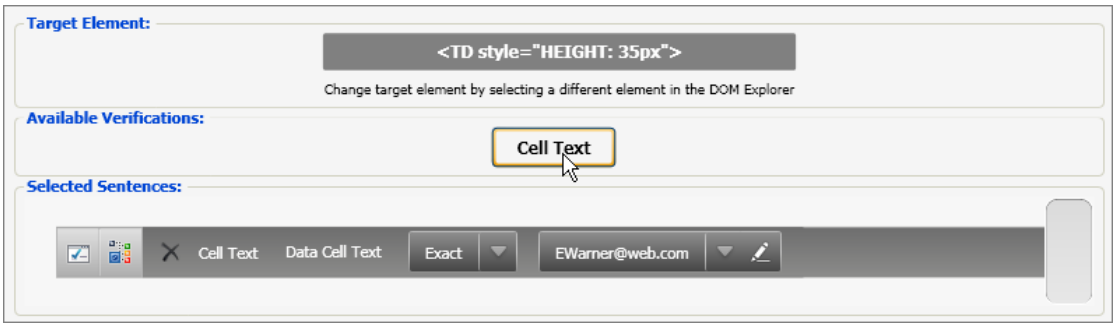
### 5.6.5 DropDown

The DropDown verification has built-in attribute types "ByIndex", "ByValue" and "ByText".

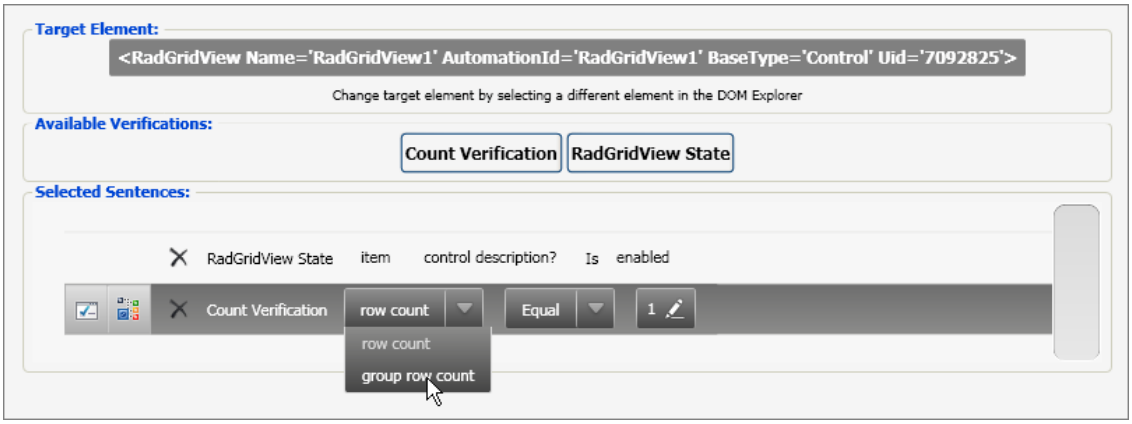
### 5.6.6 AJAX and Silverlight

AJAX and Silverlight up to now have been difficult to test against, but WebUI Test Studio puts AJAX and Silverlight testing on an "equal footing" with standard HTML by way of the consistent Sentence Verification Builder interface. Sentence Verification Builder is a consistent mechanism that works the same way no matter if you are working with plain HTML, AJAX or Silverlight elements.

In later chapters when we test AJAX and Silverlight controls and work with "Translators", i.e. extensions that open up RadControls for more complete examination, the number of verifications expands. For example, the screenshot below shows a verification against an ASP.NET AJAX grid. Using the built-in translator we're able to drill right down to the grid cell and test the Cell Text.




WebUI Test Studio also allows us to peer into the world of Silverlight. Instead of looking at the HTML DOM, we're able to test against Silverlight elements in a XAML (Extended Application Markup Language) document. The screenshot below shows a verification against a RadControls for Silverlight RadGridView. Notice that the markup for the "Target Element" section is XAML, not HTML markup.



# 5.7 Verification Types Walk Through

## 5.7.1 Test Project Setup

1) From the Visual Studio choose **File > New > Project...** This will display the "New Project" dialog.

2) Define a new test project:

   a) In the "New Project" dialog, select the "Test" project type for your language (C# or VB).

   b) Select the "Test Project" template.

   c) Enter a descriptive name for the test project, a location path and a Solution name.

   d) Click **OK** to create the new test project.

3) From the Solution Explorer, right-click the test project and select **Add > New Test...** from the context menu. This will display the "Add New Test" dialog.

4) In the "Add New Test" dialog, select the "WebAii Test" template, provide a meaningful Test Name and click **OK** to create the test.

### 5.7.2 Create Verifications

1) Locate the Record button and click it. This will display the Recording Surface.

2) In the Recording Surface, enter "http://www.google.com/translate" to the browser address bar and then click the Go to Url button. This will load the Google translation web page.

3) Press the Highlighting button to enable it.

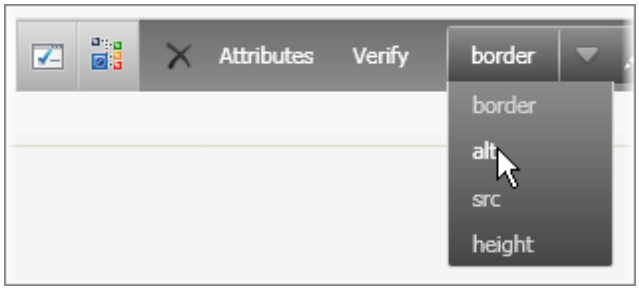
4) Move the mouse over the "Google Translate" image to highlight it. Wait for the Nub to appear.


5) Click the Nub to display the Elements Menu.

6) Click the Build Verification icon 

7) Click the **Attributes** button from the Available Verifications area. This will add an Attributes verification sentence to the Selected Sentences area.

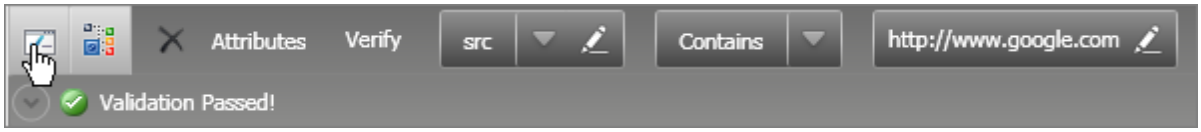
8) In the new sentence, select the "alt" attribute from the drop down list. The comparison and value should be automatically set to "Exact" and "Google", respectively.




9) Click the Validate Rule button. The message should read "Validation Passed".


10)Click the **Attributes** button from the Available Verifications area to add a second verification sentence.

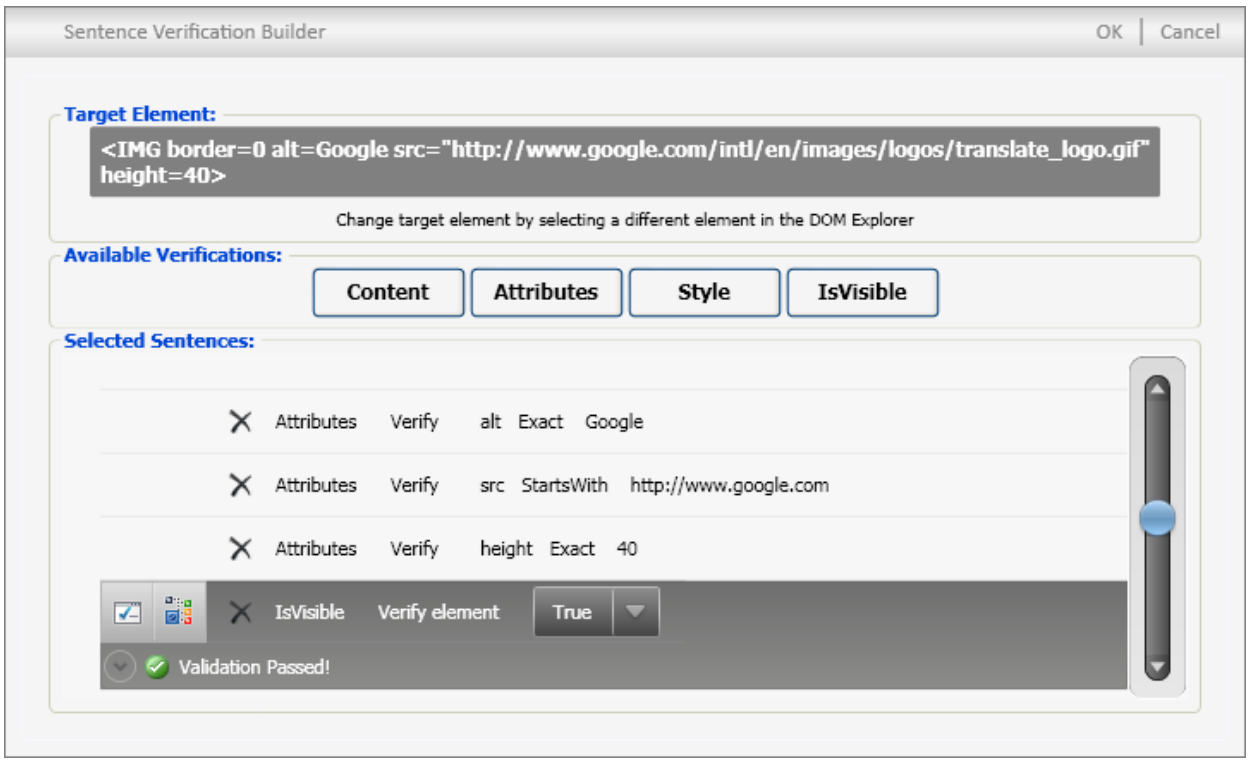
11)In the new sentence, select the "src" attribute from the drop down list, set the comparison to "Contains" and the value to "http://www.google.com". Click the Validate Rule button. The message should read "Validation Passed".



12)Click the **Attributes** button from the Available Verifications area to add a third verification sentence. In the new sentence, select the "height" attribute from the drop down list, set the comparison to "Exact" and the value to "40". Click the Validate Rule button. The message should read "Validation Passed".

13)Click the **IsVisible** button from the Available Verifications area and leave the default settings. Click the Validate Rule button. The message should read "Validation Passed".

The Sentence Verification Builder should now look like the screenshot below:



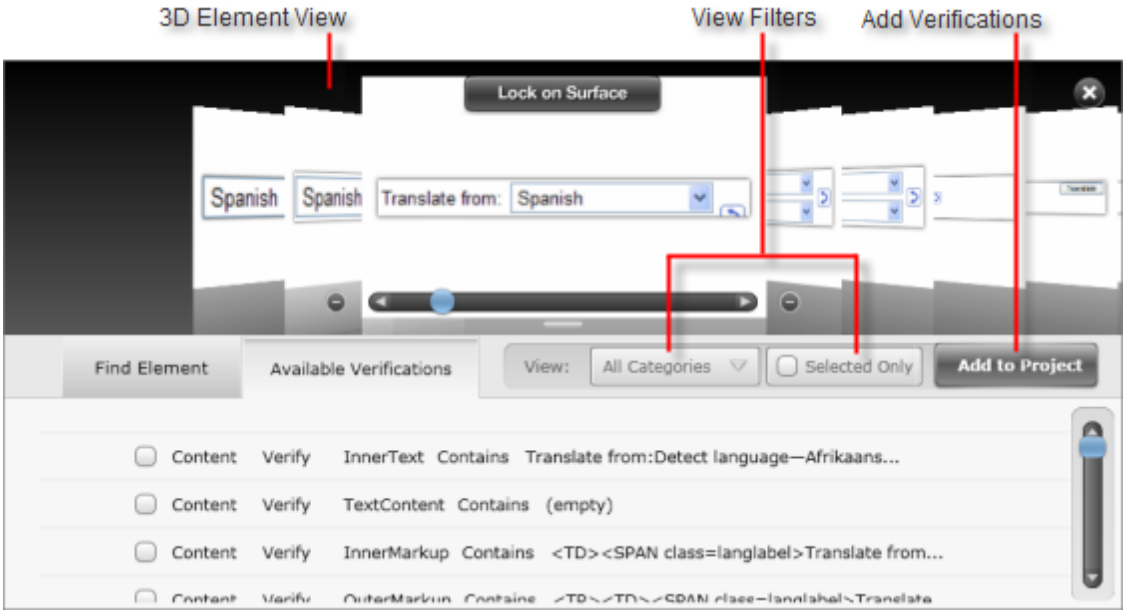

14)Click the **OK** button to create test steps for all four verification rules. The steps in the Steps Tab should look like the screenshot below:

| | Order | Enabled | Description | | |
|---|---|---|---|---|---|
| | 1 | ☑ | Navigate to : 'http://translate.google.com/' | | ✕ |
| | 2 | ☑ | Verify attribute 'alt' has 'Contains' value of 'Google' on 'GoogleImage' | | ✕ |
| | 3 | ☑ | Verify attribute 'src' has 'StartsWith' value of 'http://www.google.com'... | | ✕ |
| | 4 | ☑ | Verify attribute 'height' has 'Exact' value of '40' on 'GoogleImage' | | ✕ |
| | 5 | ☑ | Verify element 'GoogleImage' 'is' visible. | | ✕ |

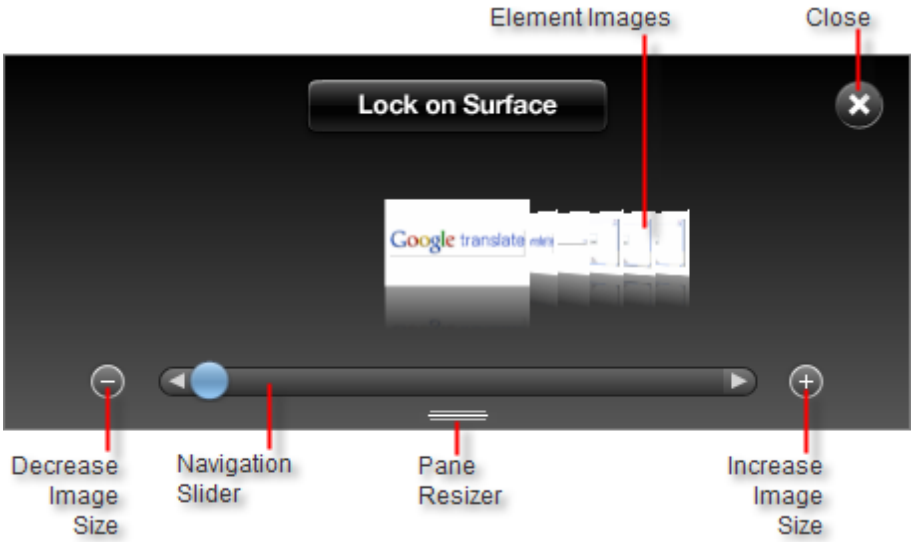
1) Click the Quick Execute button to run the test. All test steps should pass.

## 5.8 3D Viewer

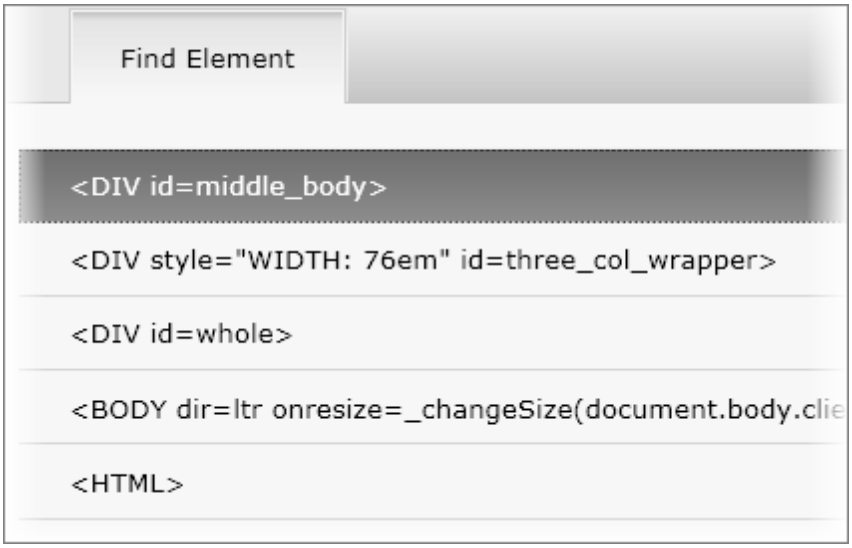
The 3D Viewer is an innovation that saves time by handling verifications for multiple elements all at one time. It consists of a top area showing a 3D view of elements, a Find Element tab, an Available Verifications tab and a set of buttons to filter verifications and to add selected verifications to the project.



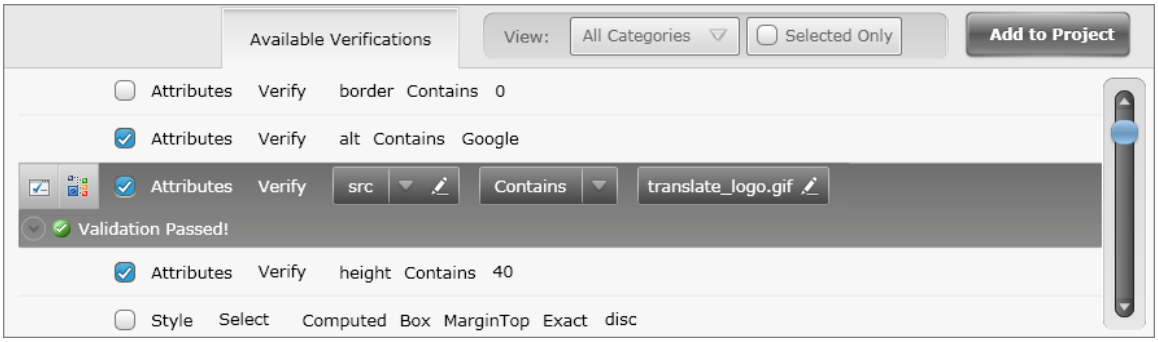

The top of the screen shows elements as three dimensional panels. To navigate, click any of the panels in the 3D view to select and bring the panel to the front, or drag the navigation slider or select an element in the Find Element tab list. The selected, front-most item corresponds to the items loaded into the Find Element and the Available Verifications tabs. Click the Lock on Surface button to navigate back to the Recording Surface with the element highlighted and the Elements Menu already showing.




The Find Element tab lists all elements from the target element you first used to invoke the Elements Menu, right up to the root of the tree (the HTML element in this case). As you select items in the Find Element tab, the corresponding 3D panel at the top of the screen rotates to the front.




The heart of the 3D Viewer is the Available Verifications tab that lists all possible verifications against a selected element. You can limit the number of verifications by using the View Filters to show verifications for a particular category or only checked verifications. Click any of the verification sentences to make the selected row the active sentence. Click the checkbox on the left of any verification sentence to select it for adding to the project. Edit any one sentence in the same manner as explained in the preceding "Sentence Verification Builder" section, i.e. by selecting from the drop down lists and editing the Value. When all your verification sentences have been selected and edited, click the Add to Project button. This step will create test steps in the Steps Tab for each verification sentence.



# 5.9 3D Viewer Walk Through

## 5.9.1 Test Project Setup

1) From the Visual Studio choose **File > New > Project...** This will display the "New Project" dialog.

2) Define a new test project:

   a) In the "New Project" dialog, select the "Test" project type for your language (C# or VB).

   b) Select the "Test Project" template.

   c) Enter a descriptive name for the test project, a location path and a Solution name.

   d) Click **OK** to create the new test project.

3) From the Solution Explorer, right-click the test project and select **Add > New Test...** from the context menu. This will display the "Add New Test" dialog.

4) In the "Add New Test" dialog, select the "WebAii Test" template, provide a meaningful Test Name and click **OK** to create the test.

### 5.9.2 Use 3D Viewer to Create Verifications

1) Locate the Record button and click it. This will display the Recording Surface.

2) In the Recording Surface, enter "http://www.google.com/translate" to the browser address bar and then click the Go to Url button. This will load the "Google" translation web page.

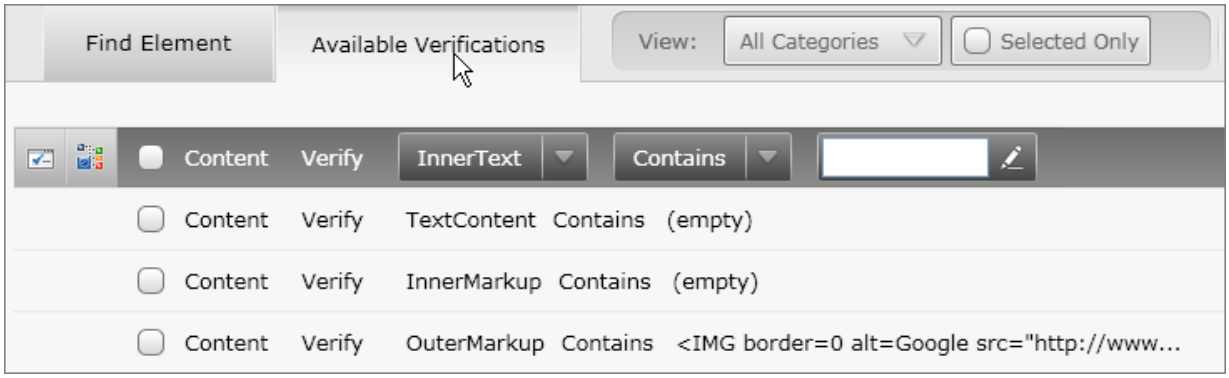
3) Press the Highlighting button to enable it.

4) Move the mouse over the "Google Translate" image to highlight it. Wait for the Nub to appear.

5) Click the Nub to display the Elements Menu.

6) Click the 3D Viewer option  .

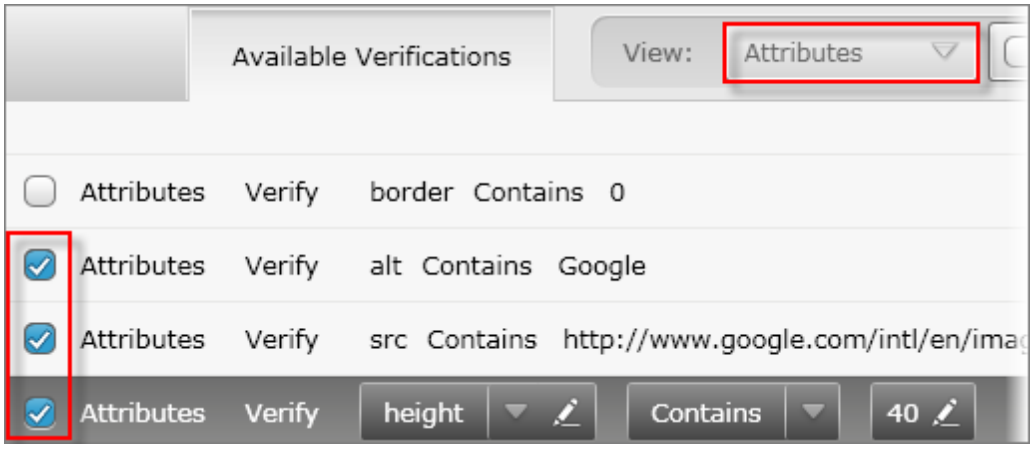
7) Click the Available Verifications tab. This will display all verifications for the "Google Translate" image element.



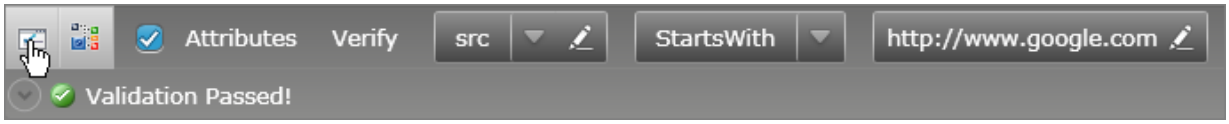

8) Drop down the categories filter list and select "Attributes".

9) In this step we want to verify that the "alt" (alternate tag for browsers that don't support images) is "Google", that the path to the image points to a Url at Google.com and that the height of the image is 40 pixels.

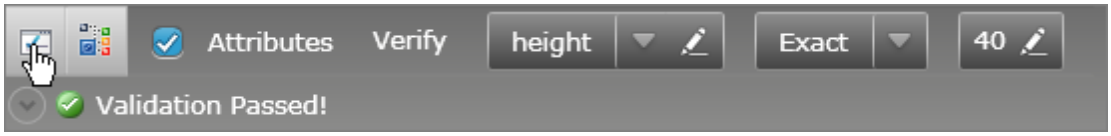
Select checkboxes for "alt Contains Google", "src Contains http://www.google.com...", and "height Contains 40".



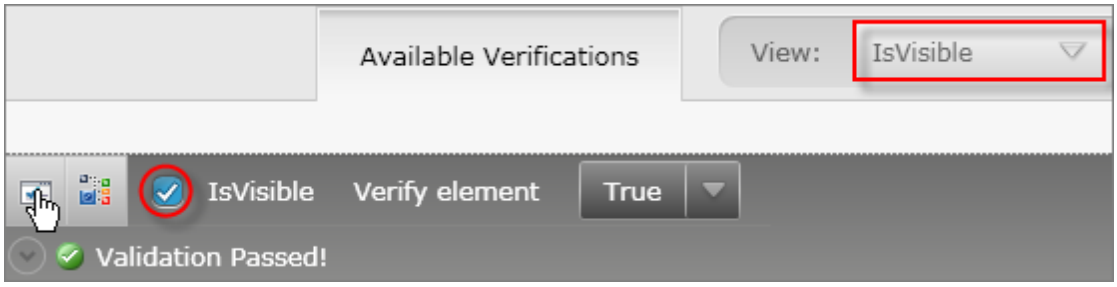

10)Select the "src Contains http://www.google.com..." sentence. Change the comparison to "StartsWith" and the value to "http://www.google.com". Click the Validate Rule button. A message should display below the sentence that reads "Validation Passed".



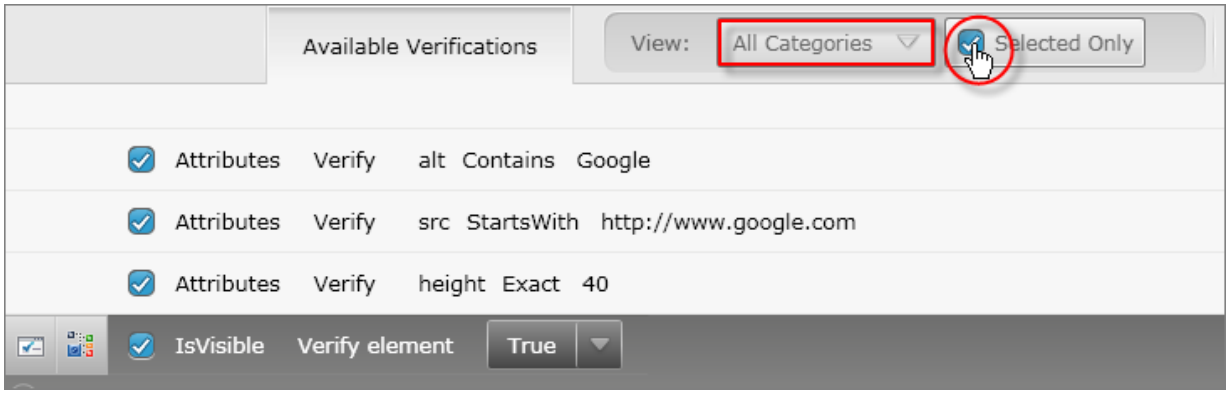

11)Select the "height Contains 40" sentence. Change the comparison to "Exact". Click the Validate Rule button. A message should display below the sentence that reads "Validation Passed".




12)From the View filter select "IsVisible" from the drop down list. In the verification sentence, select the "IsVisible True" sentence check box. Click the Validate Rule button. A message should display below the sentence that reads "Validation Passed".



13)From the View filter, select "All Categories". Then check the **Selected Only** check box. This will display all the verifications we intend to add to the project.




14)Click the **Add to Project** button.

15)Click the Close, "X" icon to exit the 3D Viewer.

16)The test steps in the Steps Tab should look like the steps shown in the screenshot below:

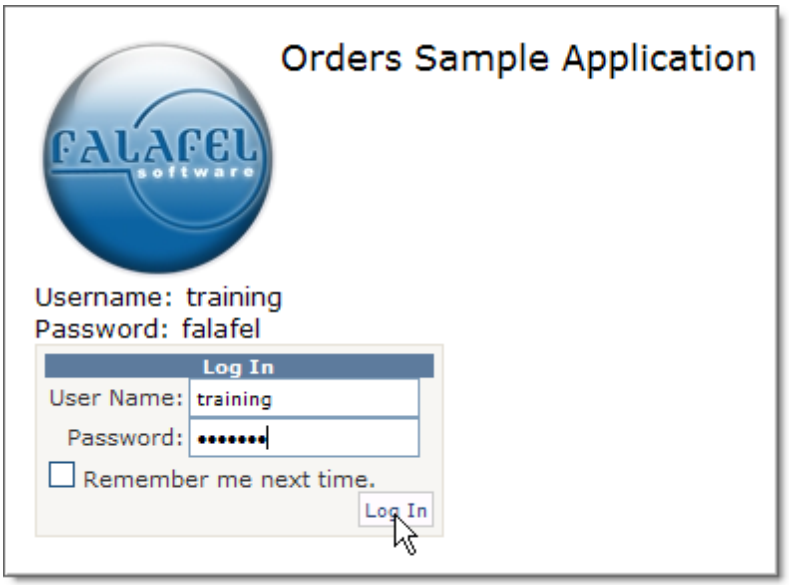
| | Order | Enabled | Description | | |
|---|---|---|---|---|---|
| | 1 | ☑ | Navigate to : 'http://translate.google.com/' | | ✕ |
| | 2 | ☑ | Verify attribute 'alt' has 'Contains' value of 'Google' on 'GoogleImage' | | ✕ |
| | 3 | ☑ | Verify attribute 'src' has 'StartsWith' value of 'http://www.google.com'... | | ✕ |
| | 4 | ☑ | Verify attribute 'height' has 'Exact' value of '40' on 'GoogleImage' | | ✕ |
| | 5 | ☑ | Verify element 'GoogleImage' 'is' visible. | | ✕ |

17)Click the Quick Execute button to run the test. All test steps should pass.

## 5.10 Verification Walk Through

In practice, verification usually involves additional planning and steps. For example, to automate a suite of tests to verify that a login page is working correctly involves not only the actual logging in and verifying that the login was successful, but also that we check the correct behavior for failed logins, blank passwords, incorrect passwords and so on. Running all of these tests requires that we return to some known state before launching the next test. For example, our login scenario will go nicely through the first login attempt, but unless we log back out, the second login attempt may not perform as expected.

We will be working with a login page and simple "Orders" page to test various paths from the login. For this walk through we will not be testing the "Remember me" feature as this would require some coding steps to test a cookie that's created when the option is checked. See the WebAii Framework chapter for more information about testing cookies in code.




Depending on the rigor of your testing standards, you may want to test each feature in various combinations with the other features. For our purposes we will create the following tests, each with their own unique verifications:

1. Successful login.
2. Incorrect user name.
3. Incorrect password.
4. Blank user name
5. Blank password

### 5.10.1 Test Project Setup

1) From the Visual Studio choose **File > New > Project...** This will display the "New Project" dialog.

2) Define a new test project:

a) In the "New Project" dialog, select the "Test" project type for your language (C# or VB).

b) Select the "Test Project" template.

c) Enter a descriptive name for the test project, a location path and a Solution name.

d) Click **OK** to create the new test project.

## 5.10.2 Successful Login Test

### Automate Test Steps

1) From the Solution Explorer, right-click the test project and select **Add > New Test...** from the context menu. This will display the "Add New Test" dialog.
2) In the "Add New Test" dialog, select the "WebAii Test" template, name the test "LoginSuccess.aii" and click **OK** to create the test.
3) Locate the **Record button** and click it. This will display the Recording Surface.
4) In the Recording Surface, enter "http://training.falafel.com/orders/" to the browser address bar and then click the **Go to Url button**. This will load the login page.
5) In the User Name edit box enter "training". Note that the user name and password are printed right on the page for your reference.
6) In the Password edit box enter "falafel".
7) Click the Login button.

The Orders Sample Application screen should now be displayed in the browser.


The test steps so far should match the list in the screenshot below:

| Description |
| --- |
| Navigate to : 'http://training.falafel.com/orders/' |
| Set 'ContentPlaceHolder1LoginView1Login1UserNameText' text t... |
| Set 'ContentPlaceHolder1LoginView1Login1PasswordPassword' t... |
| Click 'ContentPlaceHolder1LoginView1Login1LoginButtonSubmit' |

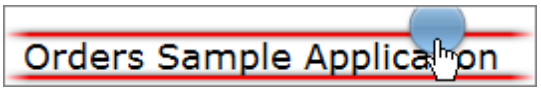
### *Notes*

You may have extra steps due to navigating in the page or extra keystrokes. You can delete these steps from the Steps Tab.

## Add Verifications

1) Press the Highlighting button to enable it.

2) Pass the mouse over the "Order Sample Application"




3) Click the Nub to display the Elements Menu, then click the Build Verification icon  .

### Gotcha!

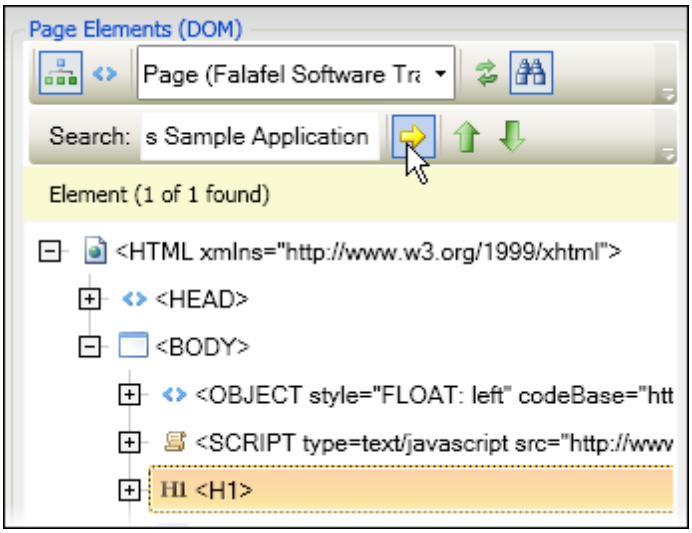
Notice that the target element is an "object", not something that would contain text content such as a DIV. What's happened here?


The element we expected to select was actually covered by a Flash object. The blue logo image to the left of the text uses Flash to animate a slight ripple effect on its surface when the mouse passes over. This is a perfect time to use the DOM Explorer to find elements hidden by other elements. See the next step that demonstrates using the DOM Explorer search capability to locate the "Orders Sample Application" element. Then we will use the "Lock on Surface" feature to highlight the element an popup the Elements Menu all in one move.

4) In the Recording Surface, navigate to the DOM Explorer.

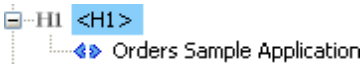
5) In the DOM Explorer, click the search button  .

6) In the "Find" edit box enter the expression "TextContent=Orders Sample Application". Click the Search button. The search should find one element, a heading tag: "<H1>".




***Notes***

If you open up the H1 tag you'll find that it contains our text:

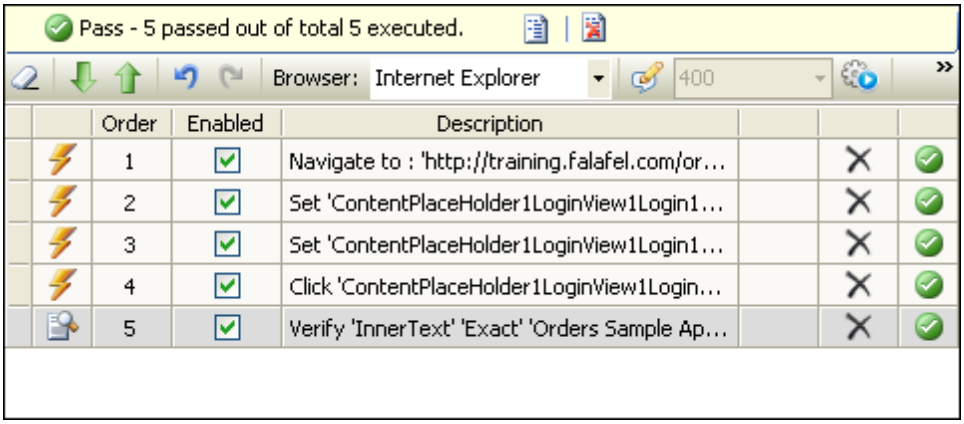



7) Click the **Content** button from the Available Verifications area.

8) The verification sentence should have "InnerText - Exact - Orders Sample Application" as shown in the screenshot below. Click the Validate Rule button . The completion message should read "Validation Passed!". Click the **OK** button to add the verification as a test step.


9) In Visual Studio, save and build the application.

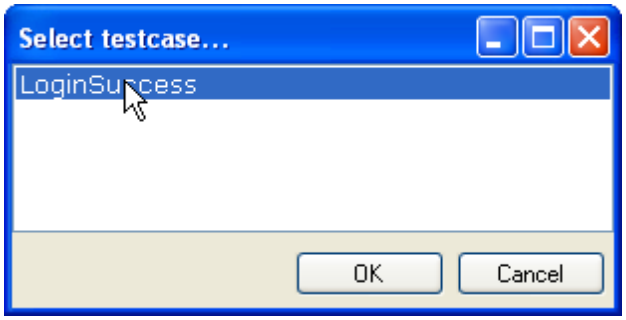
10)Click the **Quick Execute button** to run the test. All test steps should pass.



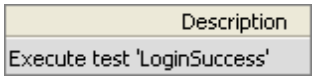
### 5.10.3 Build Master Test

Building a single successful test by itself isn't enough. You need a mechanism to add more related tests, e. g. "login fails". The Steps Tab allows you to call other tests by adding a "Test as Step". In this next part of the walk through we will create a new "master" test and call the successful login test from it. We will call the successful login test a second time to verify that we have a repeatable test. As we add other tests, we can run the entire suite each time to see how the tests work when run in a series.

1) From the Solution Explorer, right-click the test project and select **Add > New Test...** from the context menu. This will display the "Add New Test" dialog.
2) In the "Add New Test" dialog, select the "WebAii Test" template, name the test "Login.aii" and click **OK** to create the test.
3) In the Steps Tab, click **Add... > Test as Step**. This will display the "Select testcase..." dialog.
4) Select the "LoginSuccess" test case and click the **OK** button to add that test as a step.



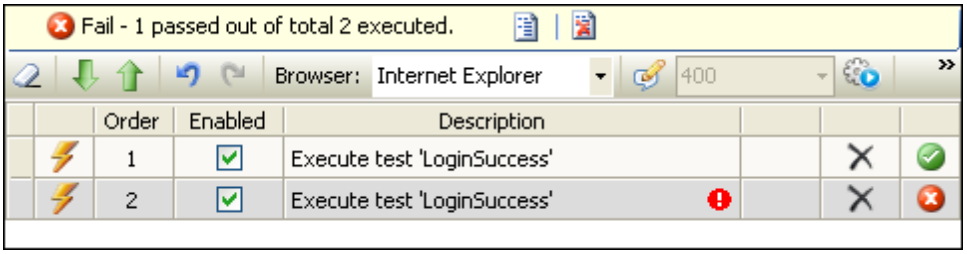

"Execute test 'LoginSuccess' should show up in the Steps Tab as a test step, as shown in the screenshot below.




5) In the Steps Tab, click **Add... > Test as Step** a second time. Select the "LoginSuccess" test case again and click the **OK** button to add that test as the second test step.
6) In Visual Studio, save the project.
7) Click the **Quick Execute button** to run the test.

### Gotcha!

The second iteration of the test fails unexpectedly. What happened here?




We can take a look at the log file by clicking the Steps Tab View Test Log button . In the log, shown below, the failure occurred when the "User Name" edit box entry couldn't be found.

```
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
'12/23/2009 11:47:25 AM' - Executing test 'LoginSuccess' as a step.
--------------------------------------------------
'12/23/2009 11:47:28 AM' - 'Pass' : 1. Navigate to : 'http://training.falafel.com/orders/'
'12/23/2009 11:47:30 AM' - 'Pass' : 2. Set 'ContentPlaceHolder1LoginView1Login1UserNameText' text to 'tra
'12/23/2009 11:47:31 AM' - 'Pass' : 3. Set 'ContentPlaceHolder1LoginView1Login1PasswordPassword' text
'12/23/2009 11:47:34 AM' - 'Pass' : 4. Click 'ContentPlaceHolder1LoginView1Login1LoginButtonSubmit'
'12/23/2009 11:47:34 AM' - 'Pass' : 5. Verify 'InnerText' 'Exact' 'Orders Sample Application' on 'OrdersSample
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
'12/23/2009 11:47:34 AM' - 'Pass' : 1. Execute test 'LoginSuccess'
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
'12/23/2009 11:47:34 AM' - Executing test 'LoginSuccess' as a step.
--------------------------------------------------
'12/23/2009 11:47:57 AM' - 'Pass' : 1. Navigate to : 'http://training.falafel.com/orders/'
'12/23/2009 11:48:03 AM' - 'Fail' : 2. Set 'ContentPlaceHolder1LoginView1Login1UserNameText' text to 't
--------------------------------------------------
Failure Information:
~~~~~~~~~~~~~~~~
Unable to locate element. Details: Attempting to find [Html] element using
Find logic
```

One of the first steps in troubleshooting is to look at the steps preceding an error. What happened just before we tried to set the user name? The step just prior to the failure was a navigation to the orders page. The navigation does not guarantee that a particular page is loaded, just that navigation was attempted. Another debugging measure you can take is to Enable Annotation so you can see the action in slow motion and get a better idea of what happened. With annotation enabled you may be able to see that the second time "Navigate to http://training.falafel.com/orders" runs, the browser is already at the orders page, not the expected login page.

To fix the problem and allow the test to complete successfully, we simply need to logout at the end of the test, so the browser is in the exact same state it was when the test was first run.

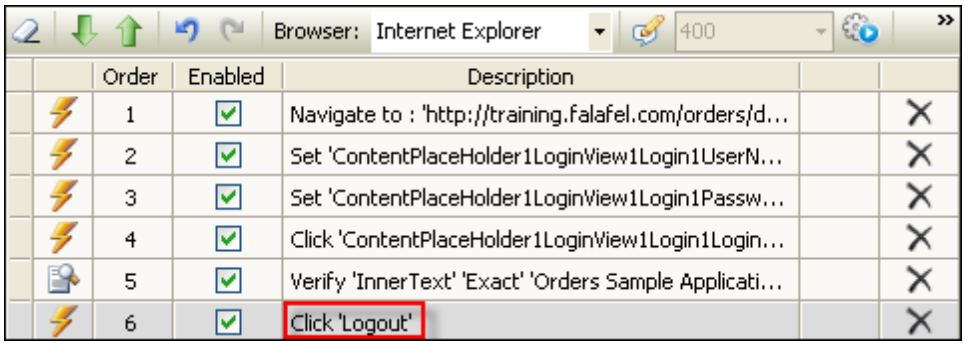
8) Using Visual Studio Solution Explorer, double click "LoginSuccess.aii" to open it.

9) In the Elements Explorer, right click "OrdersSampleH1Tag" and select **Load Page...** from the context menu.

10)Using the Recording Surface toolbar, make sure that Recording is turned on.

11)Click the "Logout" link.

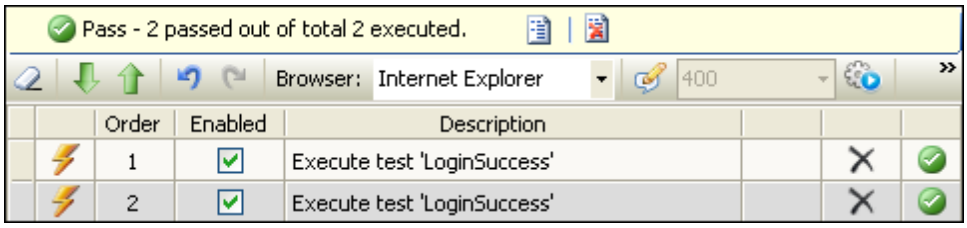
12)In the Elements Explorer, select the "ContentPlaceHolder1LoginView1LoginStatus1Link", then in the Properties pane, change the **FriendlyName** property to "Logout". The test steps should now look something like the screenshot below.




13)In Visual Studio, save and build the project.

14)Using Visual Studio Solution Explorer, double click "Login.aii" to open it.

15)Click the **Quick Execute button** to run the test. Both test steps should pass.



## 5.10.4 Incorrect User Name Test

Now that our test can be repeated, i.e. the state of the browser returns to its original state each time the test is run, we can add other tests and verify that the entire suite works as a whole.

### Automate Test Steps

1) From the Solution Explorer, right-click the test project and select **Add > New Test...** from the context menu. This will display the "Add New Test" dialog.

2) In the "Add New Test" dialog, select the "WebAii Test" template, name the test "IncorrectUserName.aii" and click **OK** to create the test.

3) Locate the **Record button** and click it. This will display the Recording Surface.

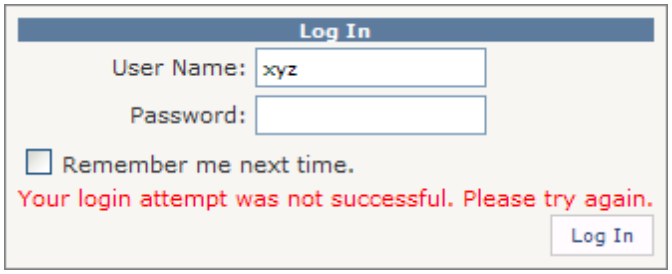
4) In the Recording Surface, enter "http://training.falafel.com/orders/" to the browser address bar and then click the **Go to Url button**. This will load the login page.

5) In the User Name edit box enter "xyz".

6) In the Password edit box enter "falafel".

7) Click the Login button.


8) An error message will display in red type.




### Add Verifications

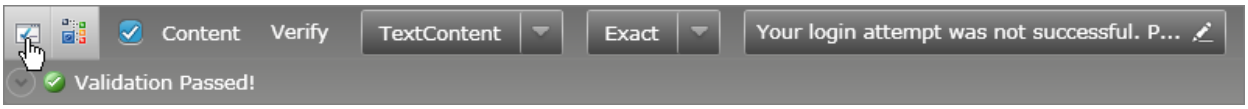
1) Press the Highlighting button to enable it.

2) Pass the mouse over the error message to highlight it and wait for the Nub to appear.

3) Select the View 3D icon  from the Elements Menu.

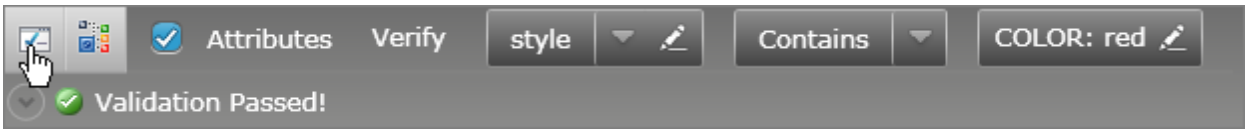
4) Select the 3D Viewer "Available Verifications" tab.

5) Select the checkbox for the "TextContent - Contains - "Your login attempt was not successful..." verification rule. Change the comparison to "Exact". Click the Validate rule button. A message should display "Validation Passed!".




6) Drop down the View filter list and select "Attributes".

7) Select the checkbox for the "style - contains - "COLOR:red" verification rule. Click the Validate rule button. A message should display "Validation Passed!".

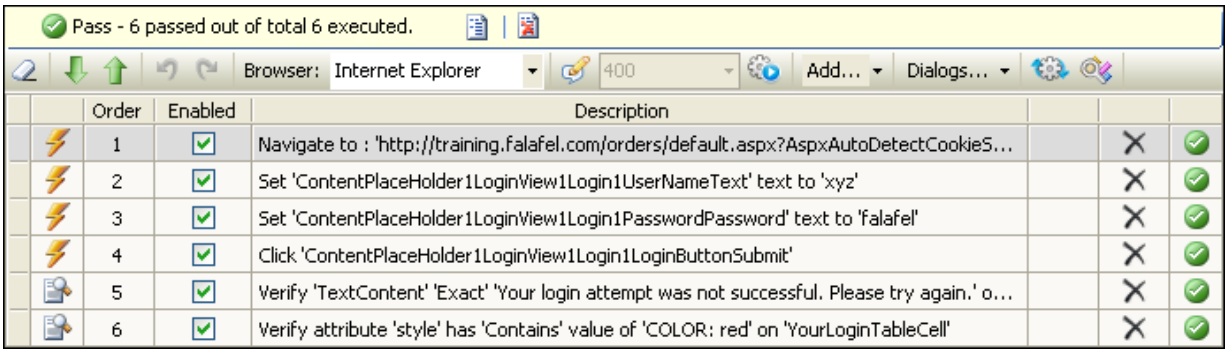



8) Click the **Add To Project** button.

9) Click the close "X" button.

10) In Visual Studio, save and build the application.

11) Click the Quick Execute button to run the test. All test steps should pass.

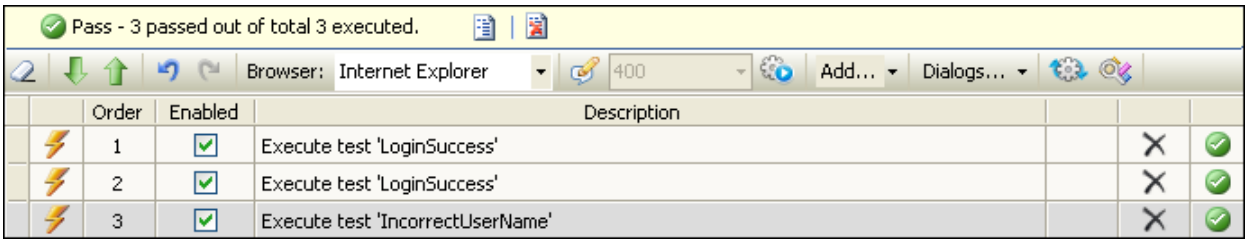



12) In the Visual Studio Solution Explorer, double click "login.aii" to open it.

13) Select the Steps Tab **Add... > Test as Step**. In the Select Testcase Dialog, select the "IncorrectUserName" test and click **OK** to create the test step.

14) In Visual Studio, save and build the application.

15) Click the Quick Execute button to run the test. All test steps should pass.



## 5.10.5 Incorrect Password Test

The next test will be nearly identical to the "Incorrect User Name" test, except that we will test the password. You can try building this test yourself and check your results against this section of the tutorial.

### Automate Test Steps

1) From the Solution Explorer, right-click the test project and select **Add > New Test...** from the context menu. This will display the "Add New Test" dialog.

2) In the "Add New Test" dialog, select the "WebAii Test" template, name the test "IncorrectPassword.aii" and click **OK** to create the test.

3) Locate the **Record button** and click it. This will display the Recording Surface.

4) In the Recording Surface, enter "http://training.falafel.com/orders/" to the browser address bar and then click the **Go to Url button**. This will load the login page.

5) In the User Name edit box enter "training".

6) In the Password edit box enter "xyz".

7) Click the Login button.

8) An error message will display in red type.


### Add Verifications

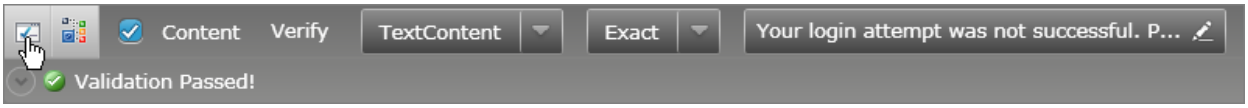
1) Press the Highlighting button to enable it.

2) Pass the mouse over the error message to highlight it and wait for the Nub to appear.

3) Select the View 3D icon  from the Elements Menu.

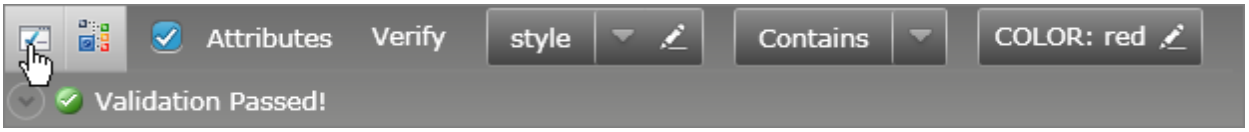
4) Select the 3D Viewer "Available Verifications" tab.

5) Select the checkbox for the "TextContent - Contains - "Your login attempt was not successful..." verification rule. Change the comparison to "Exact". Click the Validate rule button. A message should display "Validation Passed!".



6) Drop down the View filter list and select "Attributes".

7) Select the checkbox for the "style - contains - "COLOR:red" verification rule. Click the Validate rule button. A message should display "Validation Passed!".

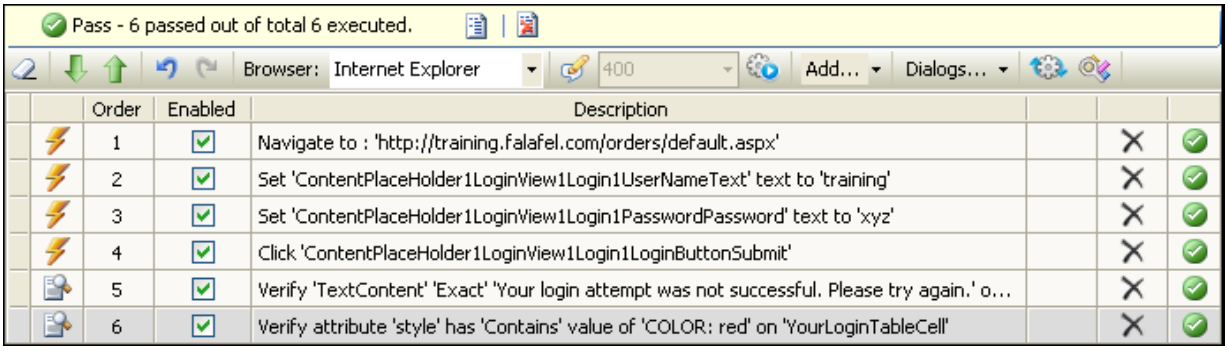



8) Click the **Add To Project** button.

9) Click the close "X" button.

10) In Visual Studio, save and build the application.

11) Click the Quick Execute button to run the test. All test steps should pass.

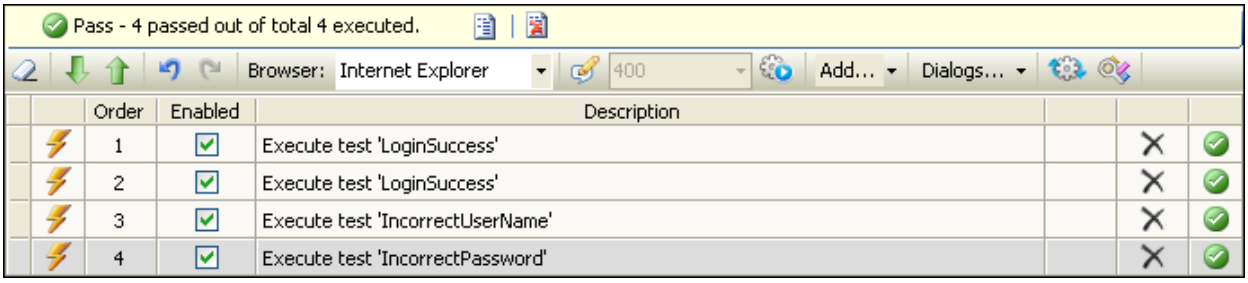



12) In the Visual Studio Solution Explorer, double click "login.aii" to open it.

13) Select the Steps Tab **Add... > Test as Step**. In the Select Testcase Dialog, select the "IncorrectPassword" test and click **OK** to create the test step.

14) In Visual Studio, save and build the application.

15) Click the Quick Execute button to run the test. All test steps should pass.



## 5.10.6 Empty User Name Test

The next test verifies that the login displays an error if the user name is left empty.

### Automate Test Steps

1) From the Solution Explorer, right-click the test project and select **Add > New Test...** from the context menu. This will display the "Add New Test" dialog.

2) In the "Add New Test" dialog, select the "WebAii Test" template, name the test "EmptyUserName.aii" and click **OK** to create the test.

3) Locate the **Record button** and click it. This will display the Recording Surface.

4) In the Recording Surface, enter "http://training.falafel.com/orders/" to the browser address bar and then click the **Go to Url button**. This will load the login page.

5) In the Password edit box enter "falafel" (Leave the User Name edit box empty).

6) Click the Login button.

7) A red asterisk will display next to the User Name edit box.


### Add Verifications

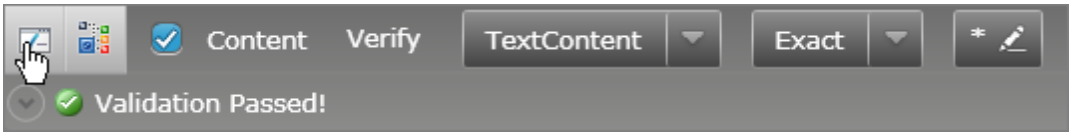
1) Press the Highlighting button to enable it.

2) Pass the mouse over the asterisk to highlight it and wait for the Nub to appear.

3) Select the View 3D icon  from the Elements Menu.

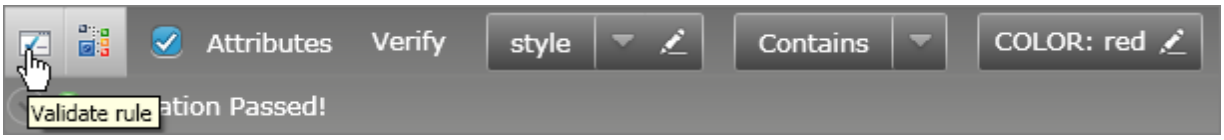
4) Select the 3D Viewer "Available Verifications" tab.

5) Select the checkbox for the "TextContent - Contains - "*" verification rule. Change the comparison to "Exact". Click the Validate rule button. A message should display "Validation Passed!".




6) Drop down the View filter list and select "Attributes".

7) Select the checkbox for the "style - contains" verification rule. Change the value portion of the verification sentence to "COLOR: red". Click the Validate rule button. A message should display "Validation Passed!".

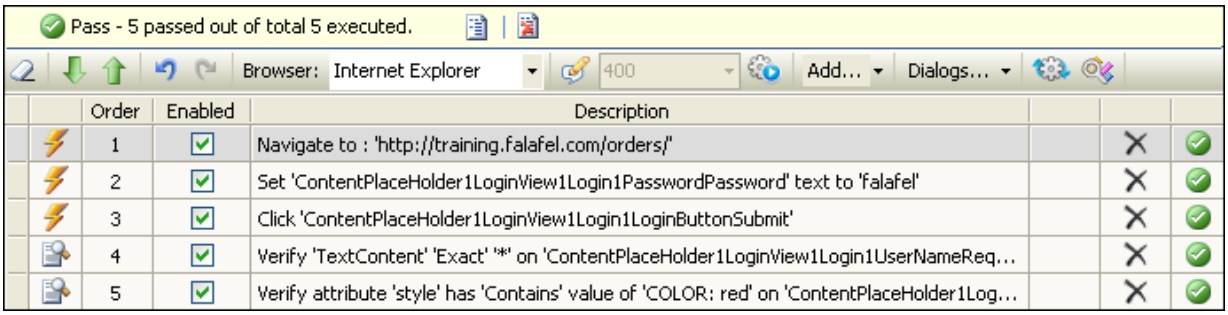



8) Click the **Add To Project** button.

9) Click the close "X" button.

10) In Visual Studio, save and build the application.

11) Click the Quick Execute button to run the test. All test steps should pass.

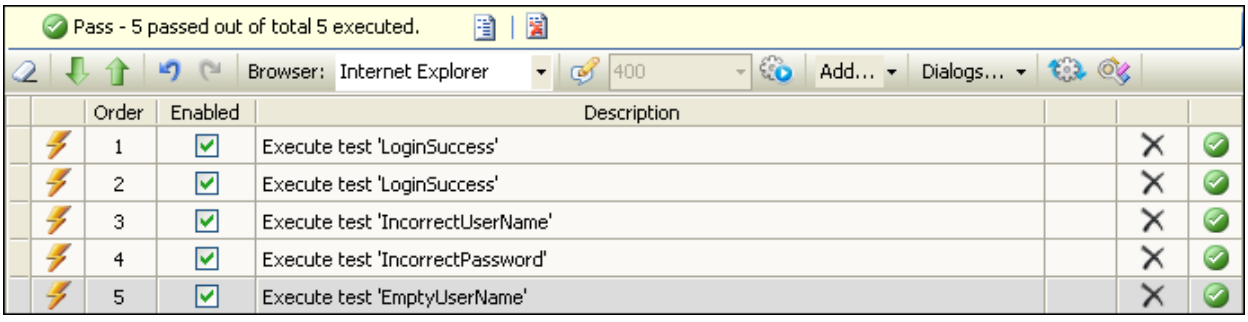



12) In the Visual Studio Solution Explorer, double click "login.aii" to open it.

13) Select the Steps Tab **Add... > Test as Step**. In the Select Testcase Dialog, select the " EmptyUserName" test and click **OK** to create the test step.

14) In Visual Studio, save and build the application.

15) Click the Quick Execute button to run the test. All test steps should pass.



## 5.10.7 Empty Password Test

The next test verifies that the login displays an error if the password is left empty. The test is essentially the same as the previous test. You can try building this test yourself and check your results against this section of the tutorial.

### Automate Test Steps

1) From the Solution Explorer, right-click the test project and select **Add > New Test...** from the context menu. This will display the "Add New Test" dialog.

2) In the "Add New Test" dialog, select the "WebAii Test" template, name the test "EmptyPassword.aii" and click **OK** to create the test.

3) Locate the **Record button** and click it. This will display the Recording Surface.

4) In the Recording Surface, enter "http://training.falafel.com/orders/" to the browser address bar and then click the **Go to Url button**. This will load the login page.

5) In the User Name edit box enter "training" (Leave the Password edit box empty).

6) Click the Login button.

7) A red asterisk will display next to the Password edit box.

### Add Verifications

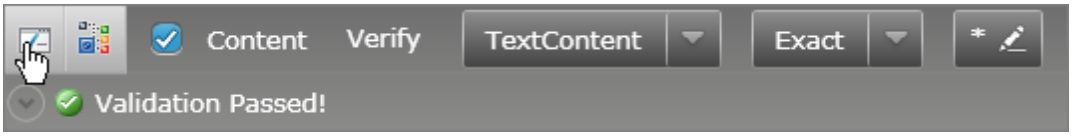
1) Press the Highlighting button to enable it.

2) Pass the mouse over the asterisk to highlight it and wait for the Nub to appear.

3) Select the View 3D icon  from the Elements Menu.

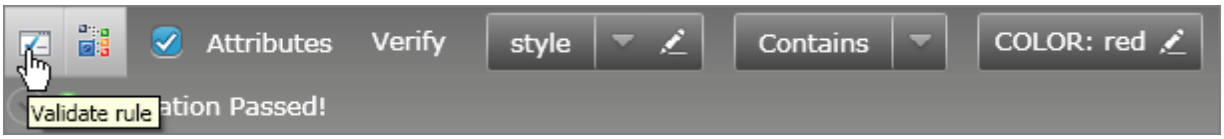
4) Select the 3D Viewer "Available Verifications" tab.

5) Select the checkbox for the "TextContent - Contains - "*" verification rule. Change the comparison to "Exact". Click the Validate rule button. A message should display "Validation Passed!".




6) Drop down the View filter list and select "Attributes".

7) Select the checkbox for the "style - contains" verification rule. Change the value portion of the verification sentence to "COLOR: red". Click the Validate rule button. A message should display "Validation Passed!".

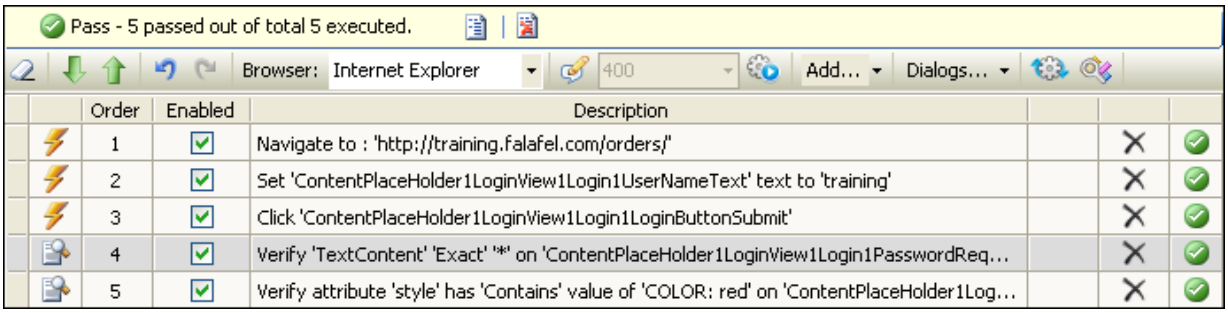



8) Click the **Add To Project** button.

9) Click the close "X" button.

10)In Visual Studio, save and build the application.

11)Click the Quick Execute button to run the test. All test steps should pass.

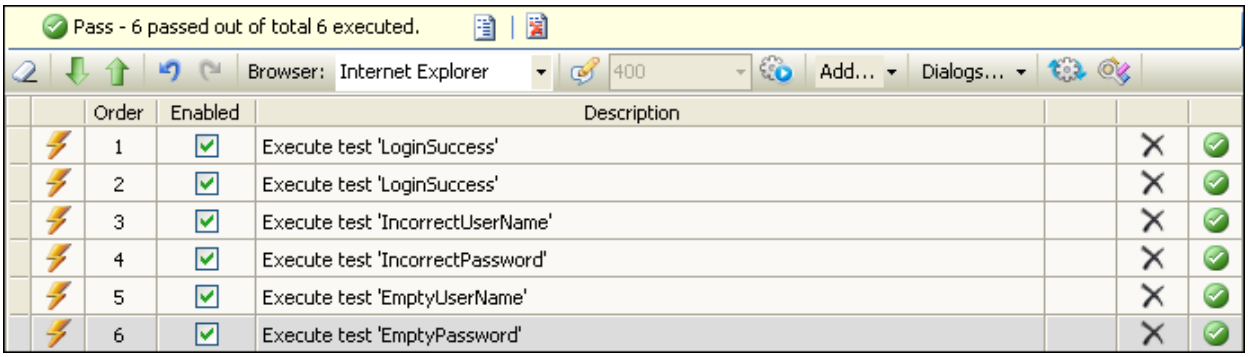



12)In the Visual Studio Solution Explorer, double click "login.aii" to open it.

13)Select the Steps Tab **Add... > Test as Step**. In the Select Testcase Dialog, select the "EmptyPassword " test and click **OK** to create the test step.

14)In Visual Studio, save and build the application.

15)Click the Quick Execute button to run the test. All test steps should pass.




## 5.11 Wrap Up

In this chapter you learned what a verification is and how verifications are applied using WebUI Test Studio. You learned how WebUI Test Studio implements verification using "sentences", how these sentences are structured and how sentences are accessed using WebUI Test Studio. You learned the types of verification available for basic HTML and how the consistent interface provided by WebUI Test Studio allows Silverlight and AJAX elements to be verified using the same mechanism. You used both the Sentence Verification Builder and the 3D Viewer to create verifications. Finally, you created a suite of tests against a login page that exercised several different verification types.

# Part VI

Translators

# 6 Translators

## 6.1 Objectives

In this chapter you will learn about the WebUI Test Studio extensibility model and how "translators" are used to surface deep information about controls. You will see how information from the "intrinsic", generic translators differ from the translators built specifically for RadControls. You will learn where the translator binaries and documentation are kept, and work with the supplied sample projects.

**Find the projects for this chapter at...**

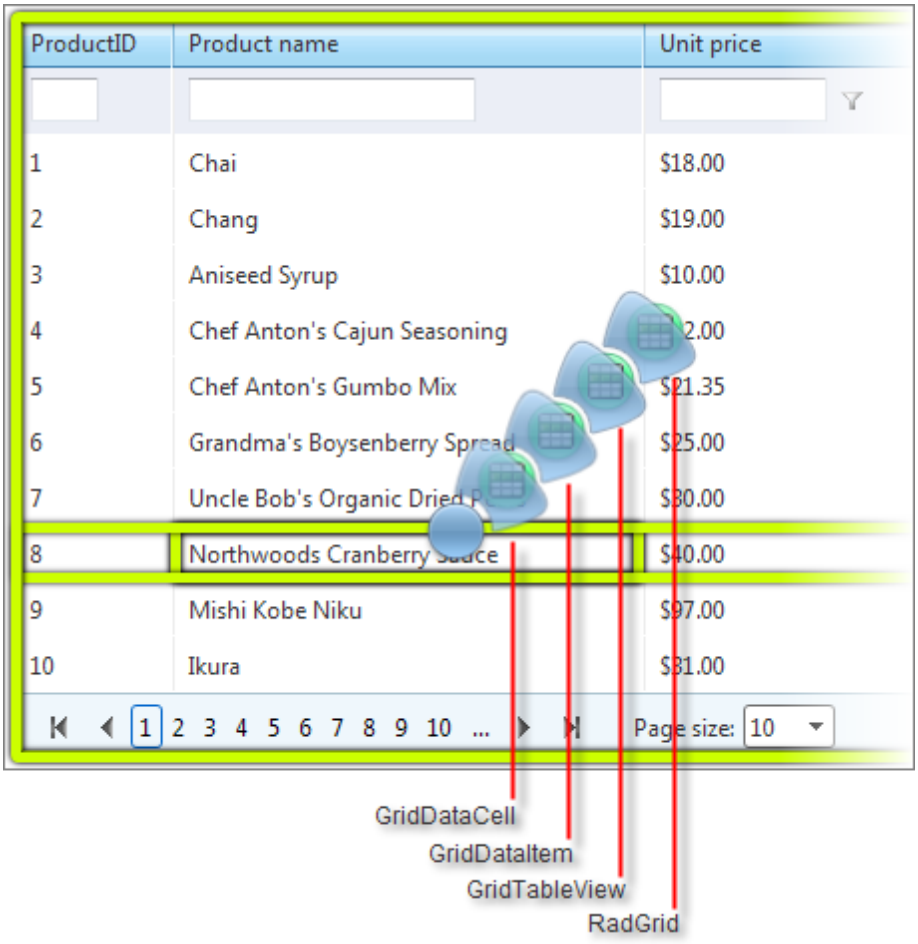
<install path>\Samples\RadControls Translators\ASP.NET AJAX\SampleTests.sln

## 6.2 Overview

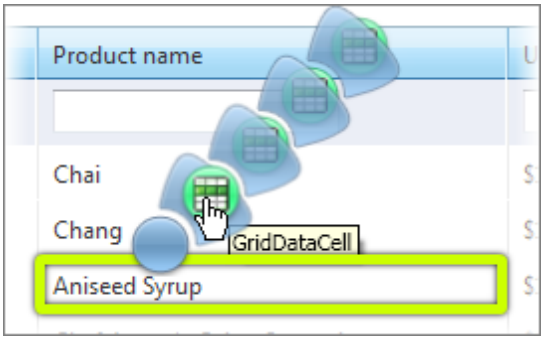
The party that best understands the internals of a component is the party that built it. The WebUI Test Studio extensibility model allows 3rd party web component vendors to encapsulate deep knowledge of component internals to share with their customers.

Translators are extensions to Visual Studio that open up an element to work with WebUI Test Studio. A translator describes the actions of an element that can be automated and verifications that can be performed. Translators allow interaction with the WebUI Test Studio user interface including the Elements Menu, Elements Explorer and Steps Tab. WebUI Test Studio ships with basic translators for HTML and Silverlight, and translators built specifically for AJAXand Silverlight RadControls. WebUI Test Studio was built with extensibility in mind, so as additional controls become available, new translators can be plugged in. Telerik is committed to maintaining translators in step with RadControl changes, so you can expect the translators to always be up-to-date.

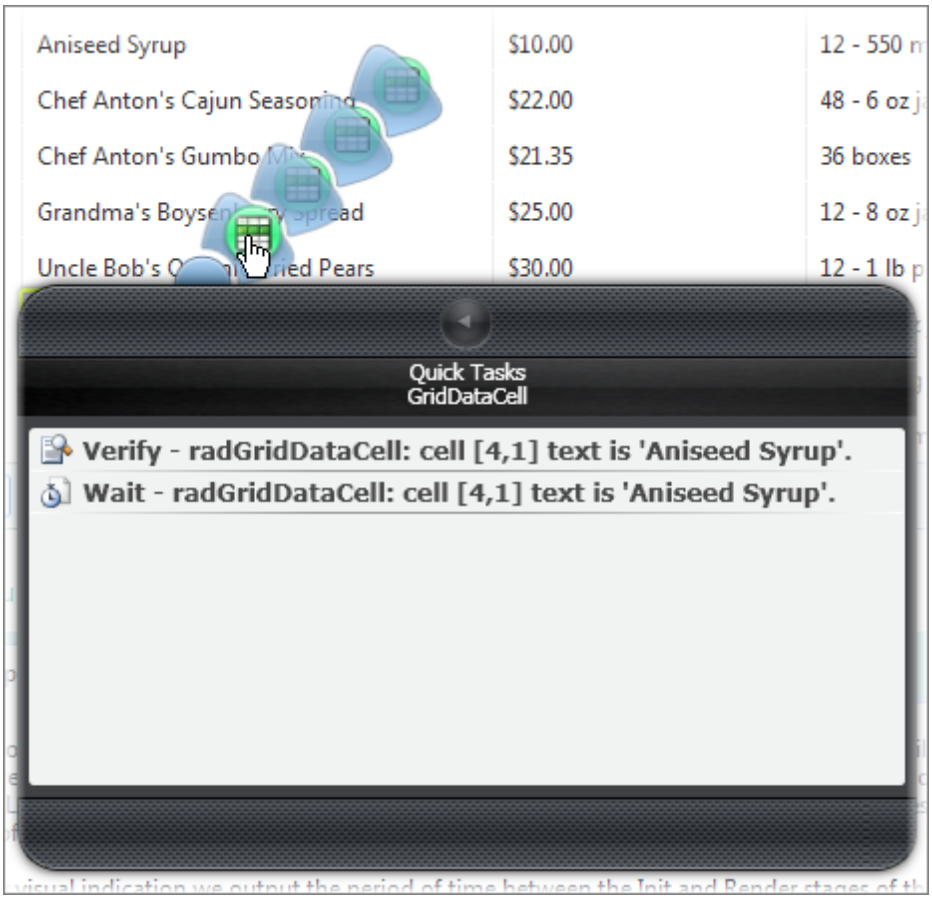
As your mouse hovers over elements in the Recording Surface, the Nub will fan out to indicate progressively more specific translators. The screenshot below shows the translators for a RadGrid cell. The Recording Surface shows enhanced highlighting in the form of green borders around a "translated" element that indicate how elements are contained within each other. In the screenshot of the RadGrid below we see a the enhanced highlighting and fan of translator nubs where the innermost leaf is a GridDataCell and the outermost leaf is a RadGrid.



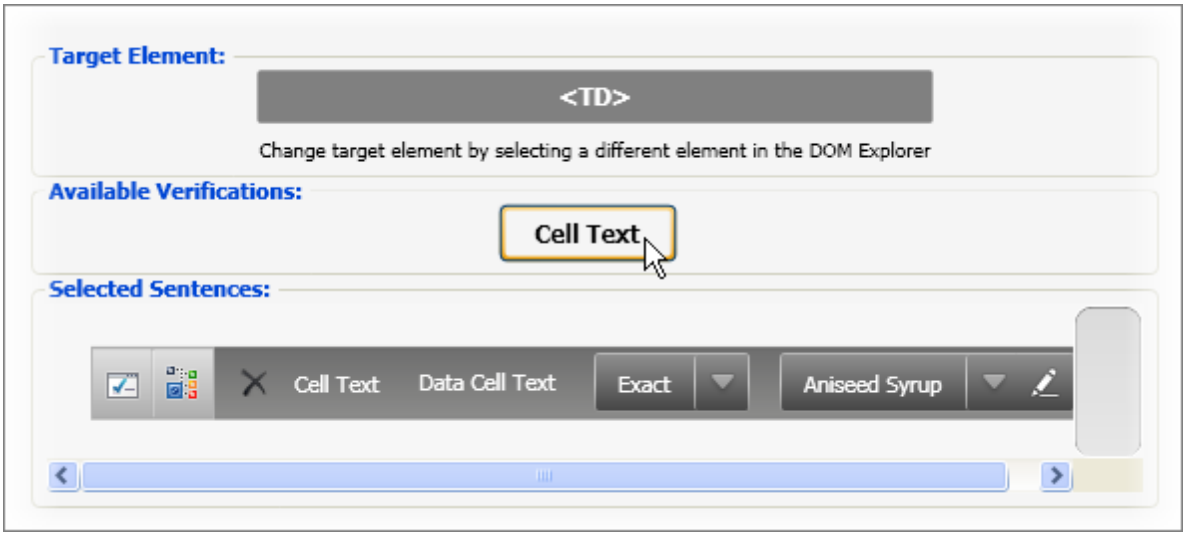
As the mouse passes over translated elements, tooltip text will popup showing the identity of a specific element and the green highlight will show where it places in terms of containership.




When you click on one of the nub leaves, the Elements Menu, Quick Tasks button displays tasks for the specific leaf. The screenshot below shows verification and wait tasks for a particular grid cell. Without the translator you couldn't get to this level of detail easily.

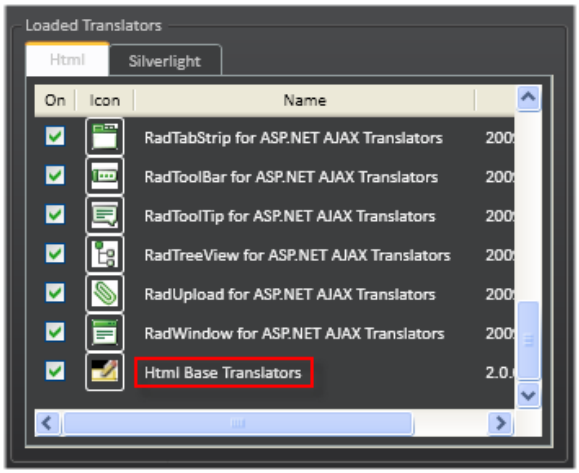


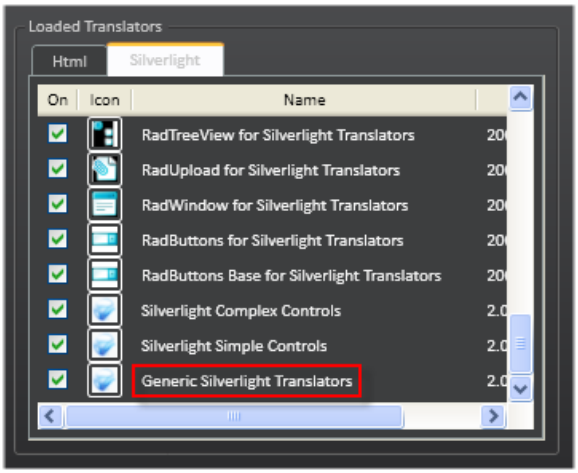

If you click the Elements Menu, Build Verification button, you can create a verification sentence using criteria supplied by the translator. In the screenshot below, the translator makes the Cell Text of a grid available.




Your WebUI Test Studio installation directory has sub-directories with:

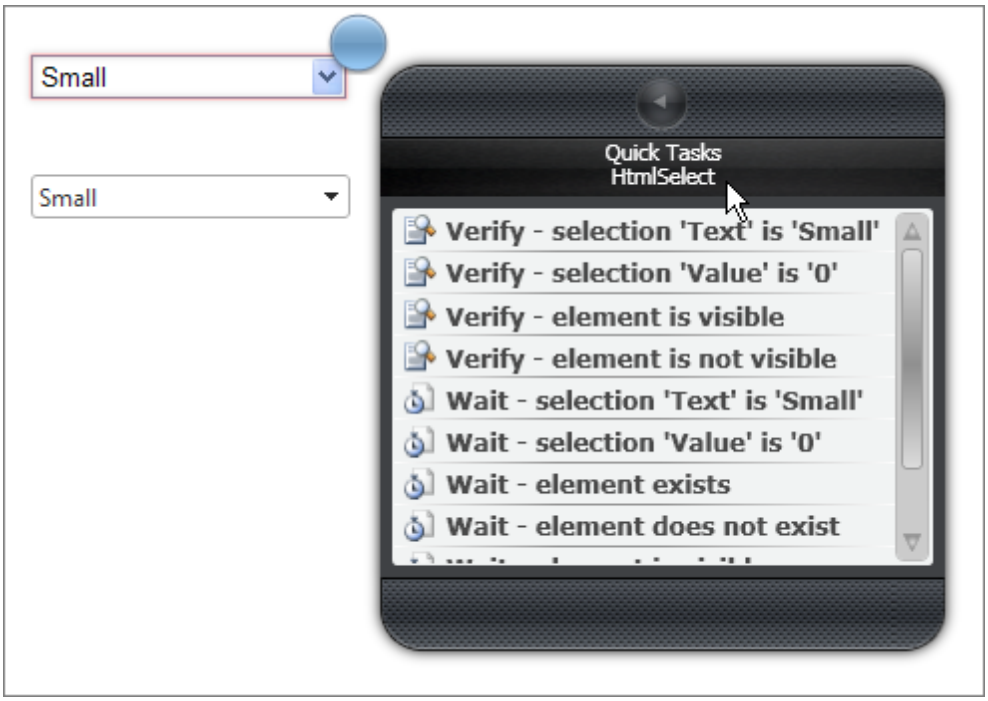
- Documentation for both HTML and Silverlight translators. The documentation lists each of the translators along with the actions it can automate, the verification sentences it can build and any Quick Tasks that it can implement.
- Sample test solutions for Visual Studio with tests that exercise each type of RadControl against a "demos" web site.
- The binary files that contain translators. These are "assemblies" with the file extension ".dll". Translators defined in these files show up in the "Loaded Translators" tab of the User Settings. Notice that both the HTML and Silverlight translators have a "base" or "generic" group of intrinsic translators that are used whenever a more specific translator is not available.



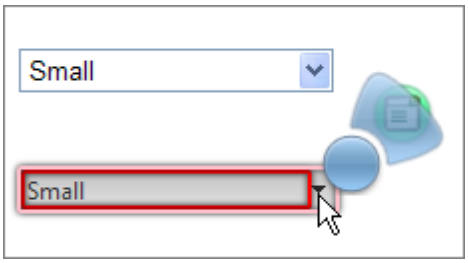



## 6.3 Standard vs Translated Comparison

Depending on the complexity of the control, translators can surface volumes of detail about a control's inner workings. To get a feel for the differences between generic translation using the intrinsic translators and translators used for specific controls, lets compare a DropDownList control (standard ASP.NET) with a RadComboBox (RadControls for MS AJAX ). When you display the Elements Menu over the DropDownList, the "intrinsic" HTML translator kicks in and recognizes a standard "<Select>" tag. Looking at the Quick Tasks we can see that the HTML translator knows about the Text, Value, existence and visibility of the element.




In contrast, the translator for RadComboBox recognizes a text box portion of the control, a drop down arrow, and the RadComboBox as a whole.




The RadComboBox is actually a relatively complex control in the browser made up of a "<DIV>", a "<TABLE>" and a number of special CSS styles, yet the translator doesn't bury you in detail you can't use. The screenshot below shows that we can find the text, selected index, item count and the "drop down" status. We can also show the Elements Menu for the text box or drop down arrow portion of the control.


## 6.4 Walk Through

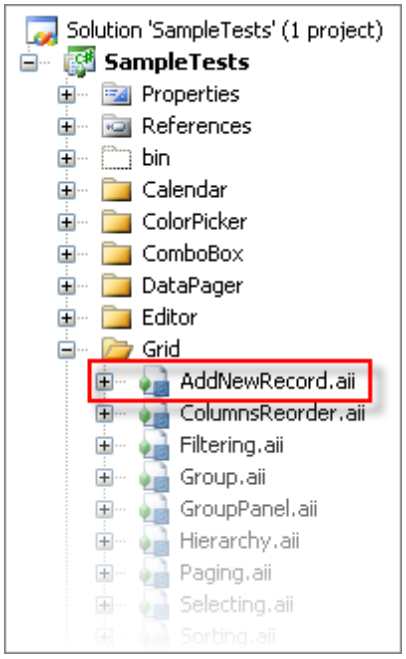
In this walk through, you will open the ASP.NET AJAX samples solution and run one of the pre-built tests against a RadGrid. Along the way we will open up the Telerik demo site for ASP.NET AJAX RadControls and look at some of the RadGrid translators.

1) In Visual Studio menu click **File > Open Project/Solution...** The Open Project dialog will display. In the Open Project dialog, navigate to the WebUI Test Studio installation directory, then to the directory / RadControls Translators/ASP.NET AJAX/SampleTests.sln

***Notes***

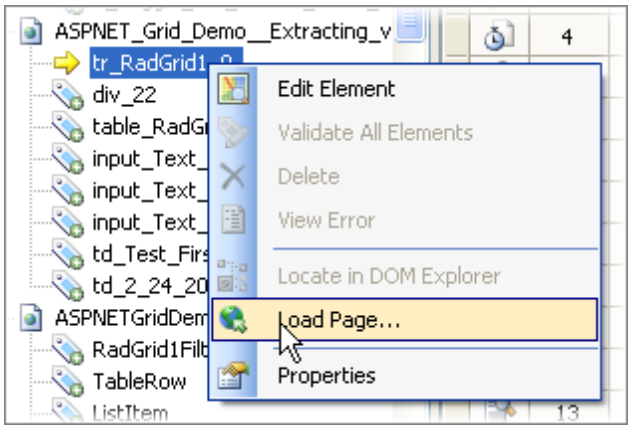
You can also find all of the sample programs from the Start menu. There you can find folders for both ASP.NET AJAX and Silverlight sample tests. Both folders will contain a "Translators" solution that focuses on the WebUI Test Studio tests built in the Recording Surface.

2) In the Solution Explorer, open the Grid directory, then double-click the "AddNewRecord.aii" test to open it.



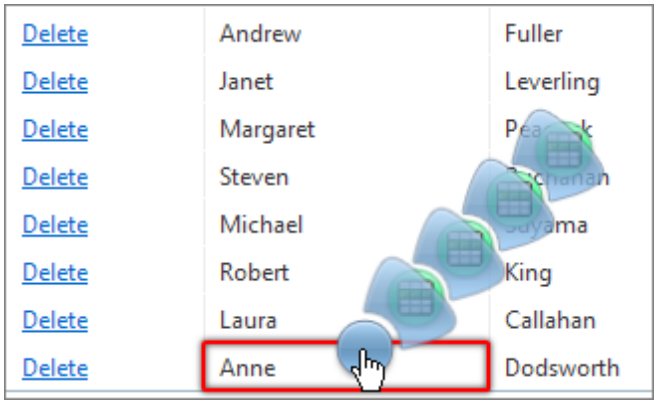

3) In the Steps Tab, click the second test step that reads "RadGridDataItem: item '0' is in edit mode'".

4) In the Elements Explorer, find the currently selected element with the yellow arrow next to it. Right click the element and select **Load Page...** from the context menu.



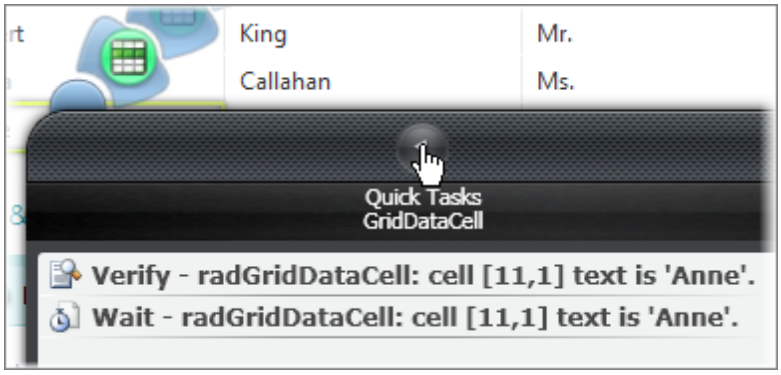

5) Click the Highlighting button to enable highlighting in the Recording Surface.

6) Hover the mouse over the "First Name" column of the last visible row in the grid.



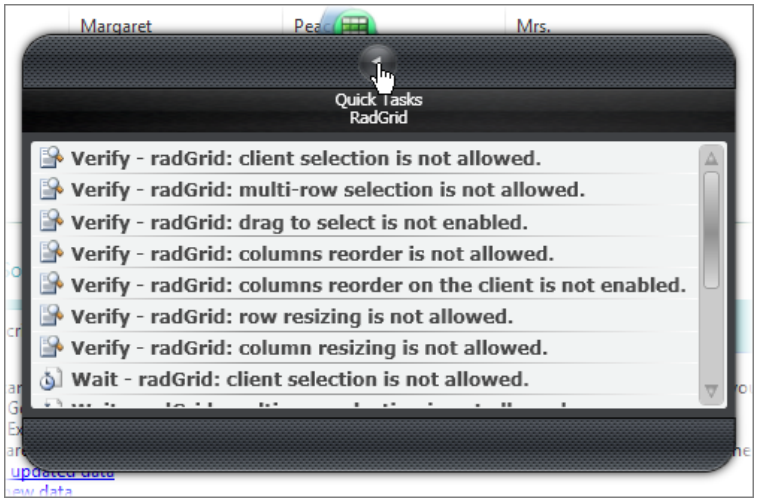

7) Move the mouse over each of the fanned out translators to view the tool tip for each. Notice the enhanced highlighting as the mouse passes over each element.

8) Click the innermost leaf "GridDataCell" to invoke the Elements Menu. Click the Quick Tasks button. Quick Tasks should show a Verify and a Wait task for the GridDataCell element. Click the "Back" button  until the Elements Menu closes.




9) Again, hover the mouse over the "First Name" column of the last visible row in the grid.

10)Click the outermost leaf for the RadGridView. Click the Quick Tasks button to see the available verifications. The screenshot below shows verifications for properties that enable various features such as client selection or row resizing. Click the "Back" button  until the Elements Menu closes.




11)In the Steps Tab, review the test steps.

The test steps will navigate to the RadControls for ASP.NET AJAX Grid demos, check that the first row is in edit mode, cancel the edit mode, then wait for the row to exit the edit mode. Then the test will create a new row, populate the new row and verify the data exists in the last row cells.

| Description |
|---|
| Navigate to : '/grid/examples/dataediting/editmodes/defaultcs.aspx' |
| RadGridDataItem: item '0' is 'in edit mode'. |
| RadGridDataItem: 'InForms' mode item '0' 'Cancel' action. |
| RadGridDataItem (Wait for): item '0' is not 'in edit mode'. |
| RadGridTable: 'Data' item count 'Equals' '9'. |
| RadGrid: command item action -> 'AddNew' |
| Set 'input_Text_RadGrid1' text to 'test first name' |
| Set 'input_Text_RadGrid1' text to 'testl □last name' |
| Set 'input_Text_RadGrid1_1' text to '2/24/2009' |
| RadGridDataItem: 'InForms' mode item '1' 'Insert' action. |
| RadGridTable (Wait for): table is not in insert mode. |
| RadGridTable (Wait for): 'Data' item count 'Equals' '10'. |
| RadGridDataCell: cell [11,1] text is 'test first name'. |
| RadGridDataCell: cell [11,4] text is '2/24/2009'. |

12)Press the Quick Execute button to run the test. All test steps should pass.

## 6.5 Wrap Up

In this chapter you learned about the WebUI Test Studio extensibility model and how "translators" are used to surface deep information about controls. You saw how information from the generic translators differed from specific translators built specifically for RadControls. You learned where the translator binaries and documentation are kept, and you worked with the supplied sample projects.

# Part VII

Testing AJAX Applications

# 7 Testing AJAX Applications

## 7.1 Objectives

In this chapter we will talk about the evolution of web applications, the challenges these new evolutionary changes present to testing and how WebUI Test Studio addresses each of these issues. In particular, you will learn how to overcome timing issues that occur in AJAX applications. You will test against sample controls from the Microsoft asp.net AJAX web site and RadControls for ASP.NET AJAX.

**Find the projects for this chapter at…**

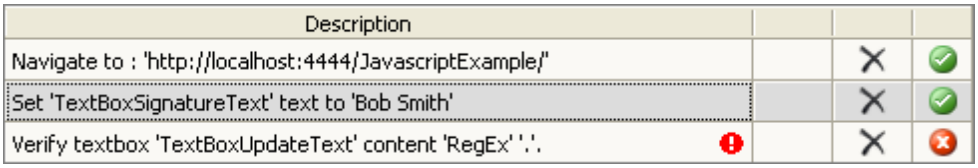
\Courseware\Projects\<CS|VB>\TestingAjax\TestingAjax.sln

## 7.2 JavaScript

Prior to the advent of JavaScript in 1995 by Netscape, the browser experience was completely server based. That is, you pressed a button or clicked a link, the page was sent to the server and a new batch of HTML was constructed and sent back to the browser. The entire page was refreshed each time. With JavaScript you can have an action take place instantly without refreshing the page. This presents a whole new set of paths that need to be checked to get full testing coverage.

Let's take an example where we have two text boxes. The first text box takes a user name and the second is automatically updated with a unique ID. We will want to test that after the first text box is filled, the second text box is not blank. The screenshot below shows two text boxes exhibiting correct behavior for this scenario.


By default, WebUI Test Studio wants to directly assign the textbox a value without using JavaScript or simulating actual typing. This is a best practice for most situations; robust and unlikely to be disturbed by changes in the environment. But in our example scenario, the JavaScript event is never triggered and the second text box is left empty. The screenshot of the Steps Tab below shows a failing test step where a regular expression is verifying that the second text box is non-blank.




Fortunately, WebUI Test Studio can trigger specific JavaScript events as test steps. Use the Elements Menu, JavaScript Events button  to invoke available events. This JavaScript example happens to have an OnKeyPress event hooked up to the first text box.

With the OnKeyPress event invocation test step, the text for the second text box is updated properly and the test runs successfully.


## 7.3 Introducing AJAX

AJAX stands for **A**synchronous **J**avascript **A**nd **X**ML and is a melding of browser, "client side" functionality with traditional server communication. JavaScript is capable of making calls to the server and updating selected portions of the page. From the testing perspective, AJAX may add a pause while information is retrieved from the server. AJAX is also "asynchronous" where not all parts of the page are necessarily updated at one time. For example, if you have prices from multiple locations, these prices can be displayed as they are received, and in any order. Web application testing has to take into account that any portion of a page can be updated at any time, without a total page refresh.

One of the principal ways AJAX can be handled in WebUI Test Studio is by waiting for a particular element to reach some state, e.g. "text content = '1234'".

**From the Forums...**

**Question**: Am I right in thinking that the test is not waiting for the AJAX call, specifically, to end, it's just waiting for the element to change to what it expects?

**Answer**: Since the problem with AJAX is that you must wait some unknown time for the operation to complete, you can easily introduce reliable testing by simply configuring the test to "wait" before validating. This method, while simple, is very useful, effective and reusable for many scenarios.

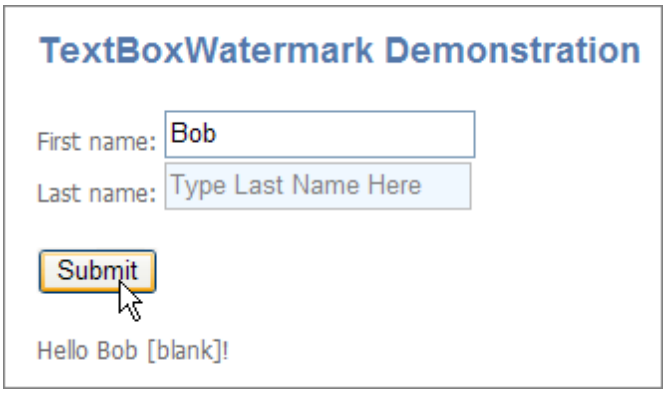
## 7.4 ASP.NET AJAX

ASP.NET AJAX is a Microsoft framework for AJAX web development. Here's the marketing blurb:

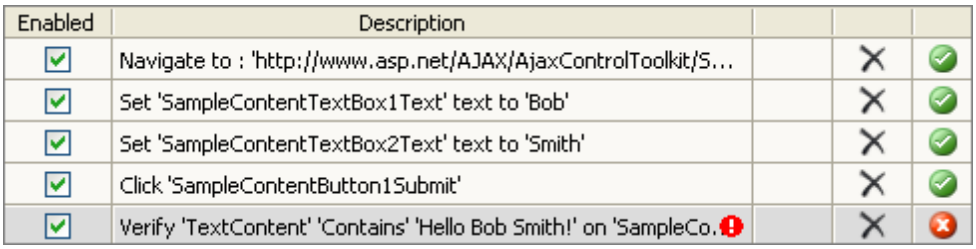
"ASP.NET AJAX is the free Microsoft AJAX framework for building highly interactive and responsive web applications that work across all popular browsers...NET AJAX enables developers to choose their preferred method of AJAX development, whether it is server-side programming, client-side programming, or a combination of both."

ASP.NET AJAX comes with a set of components that make AJAX applications easier to develop, including a ScriptManager that supplies the JavaScript used to enable AJAX functionality, an UpdatePanel that allows AJAX updates to selected portions of a page and a number of AJAX enabled controls that use the ASP.NET AJAX framework. From a testing perspective, ASP.NET AJAX can be thought of as similar to manually programmed AJAX.

You can find more information about ASP.NET AJAX at the official site, http://www.asp.net. The toolkit samples at http://www.asp.net/ajax/ajaxcontroltoolkit/Samples/ are particularly useful when learning to test AJAX scenarios. You can encounter some of the common AJAX related challenges by using the "Text Box Watermark" demonstration project. A text box "watermark" shows as gray prompt text, e.g. "Type Last Name Here" that displays when a text box is empty. The demonstration page takes a first and last name. When the "Submit" button is pressed, a label is updated from the server using AJAX.




The typical smoke test here is to simply fill in the first and last name, click the "Submit" button and verify that the label ends up with the expected text. If we set up just those steps in the Steps Tab, the last test step fails. Why? And just as important, how do we find out what is wrong with the construction of the test?



### *Notes*

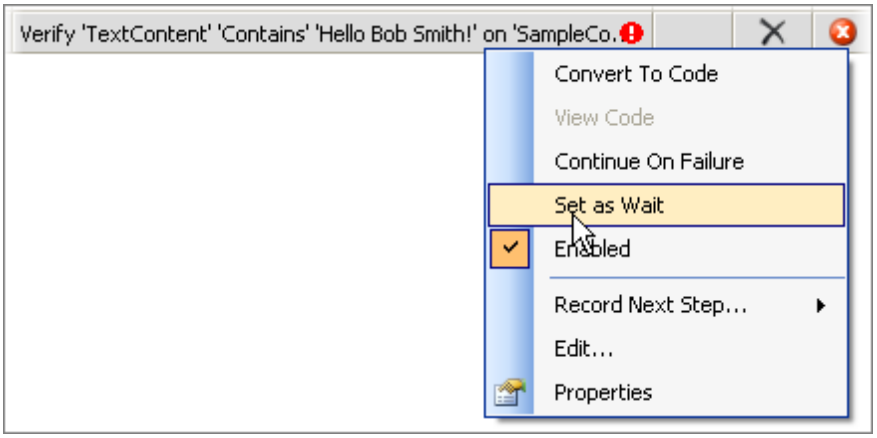
WebUI Test Studio is a tool that measures behavior of a software product under certain conditions. As you become proficient with WebUI Test Studio you can turn your attention from "what's wrong with the construction of my test?" to "what's wrong with the product I'm testing?"

There's no magic involved. Working with smaller examples, such as the www.asp.net sample projects, you can learn *why* test steps succeed or fail. You can also begin to develop a "base line" of test steps that are repeatable and always return the results you expect. Then, when a test step fails, you will know that your test is valid and that some change in the product or environment has caused the product being tested to fail.

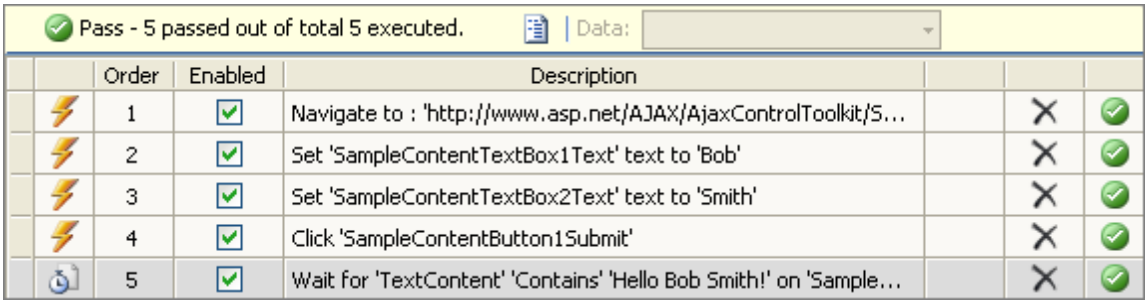
By turning on annotations in the Steps Tab, we can slow down the action a bit. When the "Submit" button is clicked, the text is blanked out and the label still shows "Hello [blank] [blank]!". We don't have visibility to what is happening when the first and last name is typed in. There could be JavaScript events firing or even processing on the server happening in the background. We can turn on the **SimulateRealTyping** property for the two test steps that set the first and last name text content. Now when we run the test, the label contains the expected text.

What if we turn off annotations and run the test at full speed? The last step fails again. How do we troubleshoot this situation? We know that the test runs successfully when slowed down, and that it fails when speeding up. This appears to be a timing issue and we know that AJAX can take extra time waiting for the server to respond. The verification test step fails because the text content of the label is checked *before* the server has responded. What we need is a way to perform the verification *after* the server has responded.

WebUI Test Studio allows you to change any verification step to a "wait" step. You can add a wait step from the Elements Menu Quick Tasks or you can right-click the verification test step and select **Set as Wait** from the context menu.

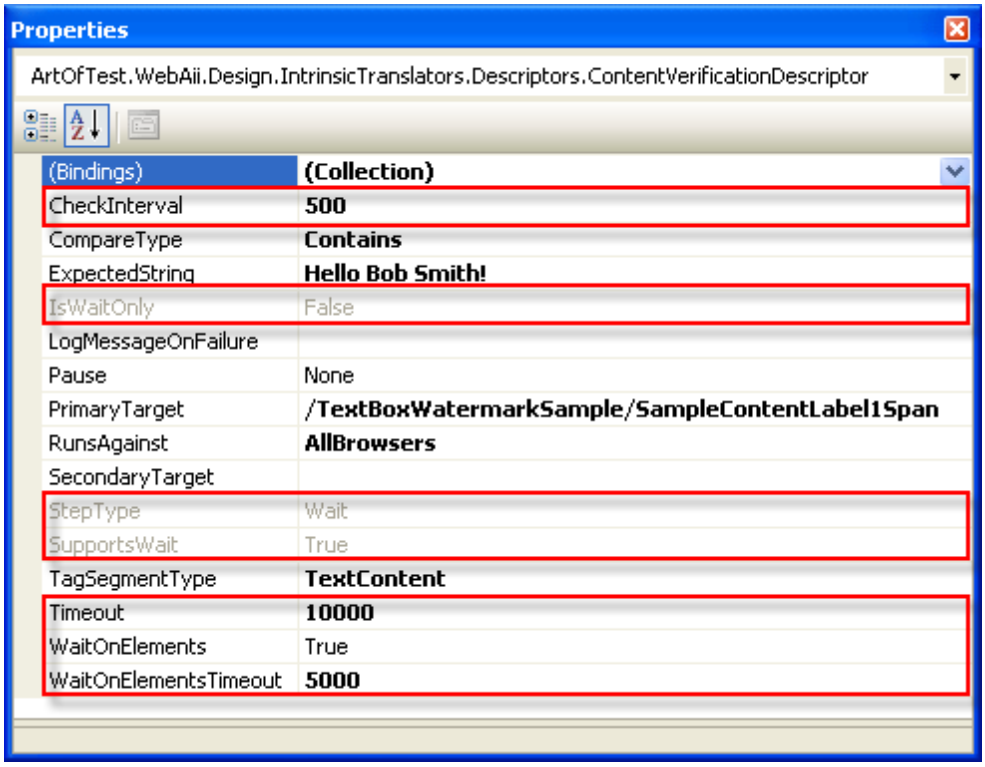



Now when we run the test, all test steps pass. The key portions of the test that we changed to work with AJAX: 1) Simulated real key strokes to invoke underlying JavaScript events and 2) Waited for the AJAX to return a response from the server.




The verification step, when acting as a wait, has a few properties to know about:

- **CheckInterval** is the number of milliseconds between evaluations of the verification.
- **Timeout** is the number of milliseconds before the test step will fail when used as a wait.
- **WaitOnElements** indicates that the test step should wait **WaitOnElementsTimeout** milliseconds for step elements to exist before executing the test step.
- **SupportsWait, IsWaitOnly** and **StepType** are read-only properties that indicate this test step can be used as a wait, that the step can *only* be used as a wait and that this particular test step *is* a wait step.




## 7.4.1 Walk Through

In this walk through you will construct a simple test of the ASP.NET AJAX Toolkit "ValidatorCallout" control. The demo contains AJAX functionality, so we can expect to see the same timing issue as shown in the preceding text box "watermark" example. This walk through will show the failing test, then change the verification steps to work as "wait" steps.

#### 7.4.1.1 Project Setup

1) From the Visual Studio choose **File > New > Project...** This will display the "New Project" dialog.

2) Define a new test project:

   a) In the "New Project" dialog, select the "Test" project type for your language (C# or VB).

   b) Select the "Test Project" template.

   c) Enter a descriptive name for the test project, a location path and a Solution name.

   d) Click **OK** to create the new test project.

3) From the Solution Explorer, right-click the test project and select **Add > New Test...** from the context menu. This will display the "Add New Test" dialog.

4) In the "Add New Test" dialog, select the "WebAii Test" template, provide a meaningful Test Name and click **OK** to create the test.

#### 7.4.1.2 Add Test Steps

1) Locate the Record button and click it. This will display the Recording Surface.

2) In the Recording Surface, enter "http://www.asp.net/AJAX/AjaxControlToolkit/Samples/ValidatorCallout/ValidatorCallout.aspx" to the browser address bar and then click the Go to Url button. This will load the "ValidatorCalllout" demonstration web page.

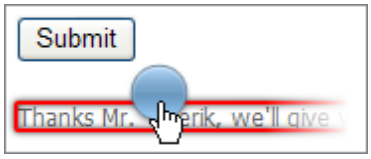
3) Enter the content "Mr. Telerik" into the "Name" text box.

4) Enter the content "(555)123-1234" into the "Phone Number" text box.

5) Click the "Submit" button.

6) Press the Highlighting button to enable it.

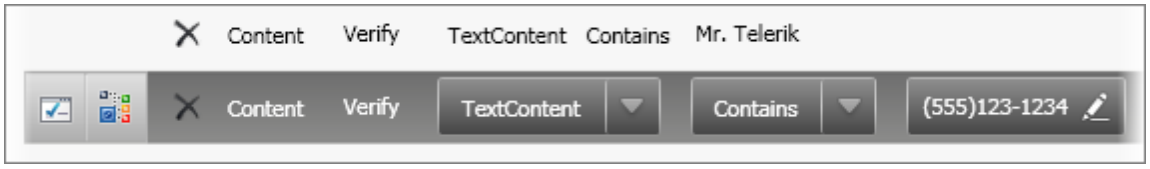
7) Move the mouse over the confirmation label, just below the "Submit" button, to highlight it. Wait for the Nub to appear.




8) Click the Nub to display the Elements Menu.

9) Click the Build Verification icon

10) In the Sentence Verification Builder, create two Content verifications. In the first, TextContent should Contain "Mr. Telerik". In the second, TextContent should Contain "(555)123-1234". Click **OK** to close the dialog and create the verification steps.



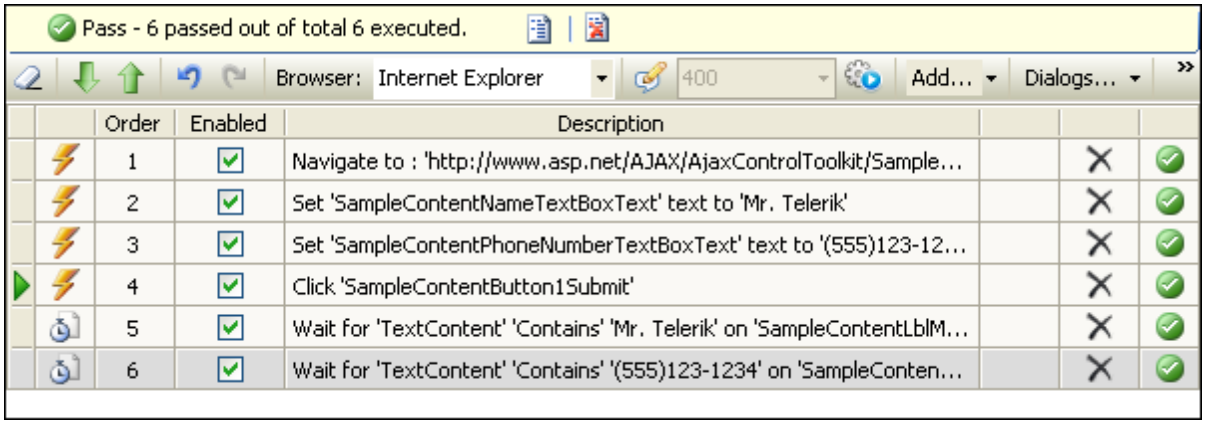

11) In the Steps Tab, click the Quick Execute button to run the test. *The first "TextContent" verification may fail (depending on the speed that the test was run at) because the verification takes place before the label is updated from the server. The last step is canceled because the step just before it failed.*

| Description | | | |
|---|---|---|---|
| Navigate to : 'http://www.asp.net/AJAX/AjaxControlToolkit/Samples/ValidatorC... | | ✕ | ✓ |
| Set 'SampleContentNameTextBoxText' text to 'Mr. Telerik' | | ✕ | ✓ |
| Set 'SampleContentPhoneNumberTextBoxText' text to '(555)123-1234' | | ✕ | ✓ |
| Click 'SampleContentButton1Submit' | | ✕ | ✓ |
| Verify 'TextContent' 'Contains' 'Mr. Telerik' on 'SampleContentLblMessageSpan' | | ✕ | ✗ |
| Verify 'TextContent' 'Contains' '(555)123-1234' on 'SampleContentLblMessageSp... | | ✕ | ⚠ |

12) Convert the last two test steps to "wait" steps. Right click each step and select "Set as Wait" from the context menu.

13)In Visual Studio, save the project.

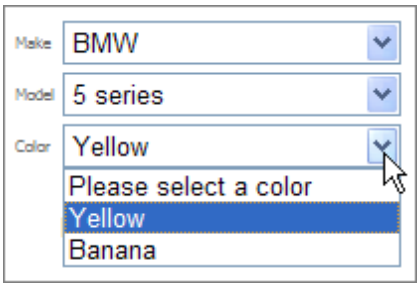
14)In the Steps Tab, click the Quick Execute button to run the test. Now all test steps will complete successfully.



#### 7.4.1.3 Intermittent Timing Problems

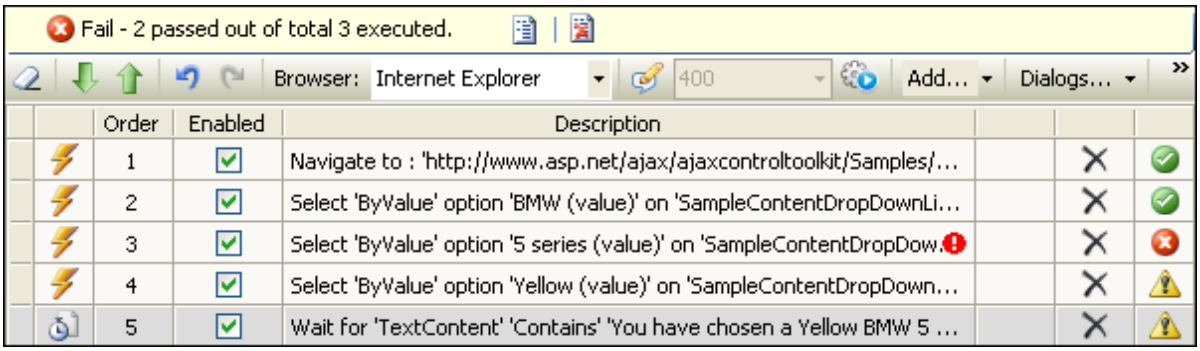
When AJAX enabled components interact on a page you may experience intermittent timing issues where sometimes the server returns quickly enough to satisfy a test and at other times fails. We can see this in action using the "CascadingDropDown" AJAX toolkit demonstration project.

"CascadingDropDown enables a common scenario in which the contents of one list depends on the selection of another list ... All the logic about the contents of the set of DropDownList controls lives on the server in a web service."




When the "Make" of the car is selected from the top-most list, the "Model" list is populated from the server and enabled. As items are selected in each drop down list, the next list in line is populated and enabled. When all three lists have selections, one last trip to the server creates a confirmation message, e.g. "You have chosen a Yellow BMW 5 series. Nice car!"

Any one of these steps can fail if the trip to the server takes too long. Multiple runs of the test show different lines flagged as an error, or in some cases, no error at all. The screenshot below shows the third test step for this particular test run happened to fail.




The logged error message for this test run indicates that a drop down value wasn't found. The step was expecting a value of "5 series", but instead found nothing.

> "Unexpected error thrown while setting the dropdown ---> System. ArgumentOutOfRangeException: Specified argument was out of the range of valid values. Parameter name: DropDown was unable to find the item '5 series (value)' in the dropdown requested..."

In this walk through we will create test steps "defensively" and assume a slow trip to the server that will not complete in time for the following test step. We will check that a given option already exists in the list before trying to pick it.

1) Define a new test project:

a) In the "New Project" dialog, select the "Test" project type for your language (C# or VB).

b) Select the "Test Project" template.

c) Enter a descriptive name for the test project, a location path and a Solution name.

d) Click **OK** to create the new test project.

2) From the Solution Explorer, right-click the test project and select **Add > New Test...** from the context menu. This will display the "Add New Test" dialog.

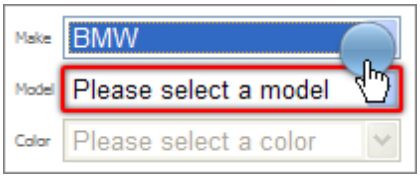
3) In the "Add New Test" dialog, select the "WebAii Test" template, provide a meaningful Test Name and click **OK** to create the test.

4) Locate the Record button and click it. This will display the Recording Surface.

5) In the Recording Surface, enter "http://www.asp.net/AJAX/AjaxControlToolkit/Samples/CascadingDropDown/CascadingDropDown.aspx" to the browser address bar and then click the Go to Url button. This will load the "CascadingDropDown" demonstration web page.

6) Select "BMW" from the "Make" drop down list.

7) Hover the mouse above the "Model" drop down list until the Elements Menu appears.




8) Select Build Verification from the Elements Menu. Click the **Content** button from the "Available Verifications" section. Configure the verification sentence so that "OuterMarkup - Contains - "<OPTION value="5 series (value)">5 series</OPTION>". Click **OK** to create the verification step.The screenshot below shows the completed verification sentence.


**Tip!**

Note that when you edit the value portion of the sentence, the entire markup including all the OPTION tags will exist. You can simply cut away everything but the option you want to verify. If the option contains the attribute "selected", be sure to remove it. When the drop down list is first loaded and enabled, selection is on the "Please select a model" option.

### *Notes*

This step ensures that the "5 series" option will exist in the drop down list before we try to pick it. Later in this chapter we will use a RadControls drop down list. The special purpose translator provides additional information, such as the number of items in the list. The special purpose translator can answer the question "has my drop down list been populated" more directly and easily. When you don't have a special purpose translator, you can still use the intrinsic translator to examine an element.

9) Select Build Verification from the Elements Menu. Click the **Content** button from the "Available Verifications" section. Configure the verification sentence so that

10) Right-click the verification step and select "Set as Wait" from the context menu.

11) Select "5 Series" from the "Model" drop down list.

12) Hover the mouse above the "Color" drop down list until the Elements Menu appears.

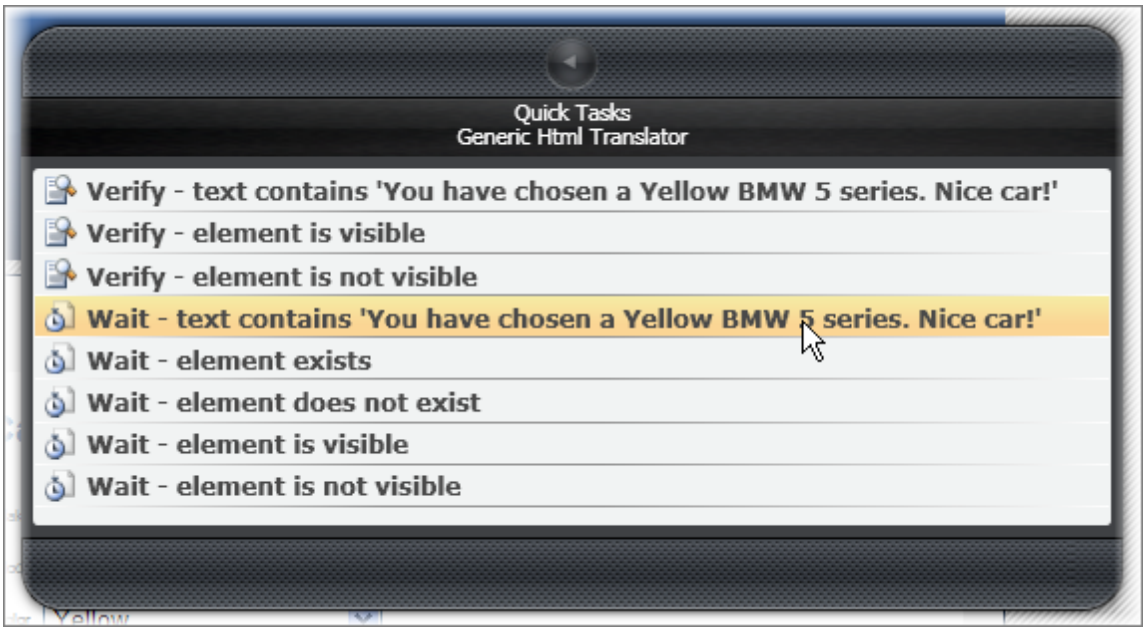
13) Select Build Verification from the Elements Menu. Click the **Content** button from the "Available Verifications" section. Configure the verification sentence so that "OuterMarkup - Contains - "<OPTION value="Yellow (value)">Yellow</OPTION>". Click **OK** to create the verification step.


14) Right-click the verification step and select "Set as Wait" from the context menu.

15) Select "Yellow" from the "Color" drop down list.

16) Now we need to find the "You have chosen a Yellow BMW 5 series. Nice car!" confirmation label. This label is located deep within a collection of DIV, TABLE and SPAN tags, making it hard to locate with the highlighter. Instead, go to the DOM Explorer button. Click the search button and enter the search "TextContent=~Yellow BMW 5". This will locate the SPAN tag that contains the confirmation message. Right-click the SPAN tag and select "Show Elements Menu" from the context menu.

17)Click the Elements Menu, Quick Tasks button. Double-click the "Wait - text contains 'You have chosen a Yellow BMW 5 series. Nice car!'".




18)Click the Steps Tab, Quick Execute button to run the test. All test steps should pass.

| Description | | | |
|---|---|---|---|
| Navigate to : 'http://www.asp.net/AJAX/AjaxControlToolkit/Samples/CascadingDropDown/CascadingDropDown.aspx' | | ✕ | ✓ |
| Select 'ByValue' option 'BMW (value)' on 'SampleContentDropDownList1Select' | | ✕ | ✓ |
| Wait for 'OuterMarkup' 'Contains' '<OPTION value="5 series (value)">5 series</OPTION>' on 'SampleContentDropDownList2Select' | | ✕ | ✓ |
| Select 'ByValue' option '5 series (value)' on 'SampleContentDropDownList2Select' | | ✕ | ✓ |
| Wait for 'OuterMarkup' 'Contains' '<OPTION value="Yellow (value)">Yellow</OPTION>' on 'SampleContentDropDownList3Select' | | ✕ | ✓ |
| Select 'ByValue' option 'Yellow (value)' on 'SampleContentDropDownList3Select' | | ✕ | ✓ |
| Wait for 'TextContent' 'Contains' 'You have chosen a Yellow BMW 5 series. Nice car!' on 'SampleContentLabel1Span' | | ✕ | ✓ |

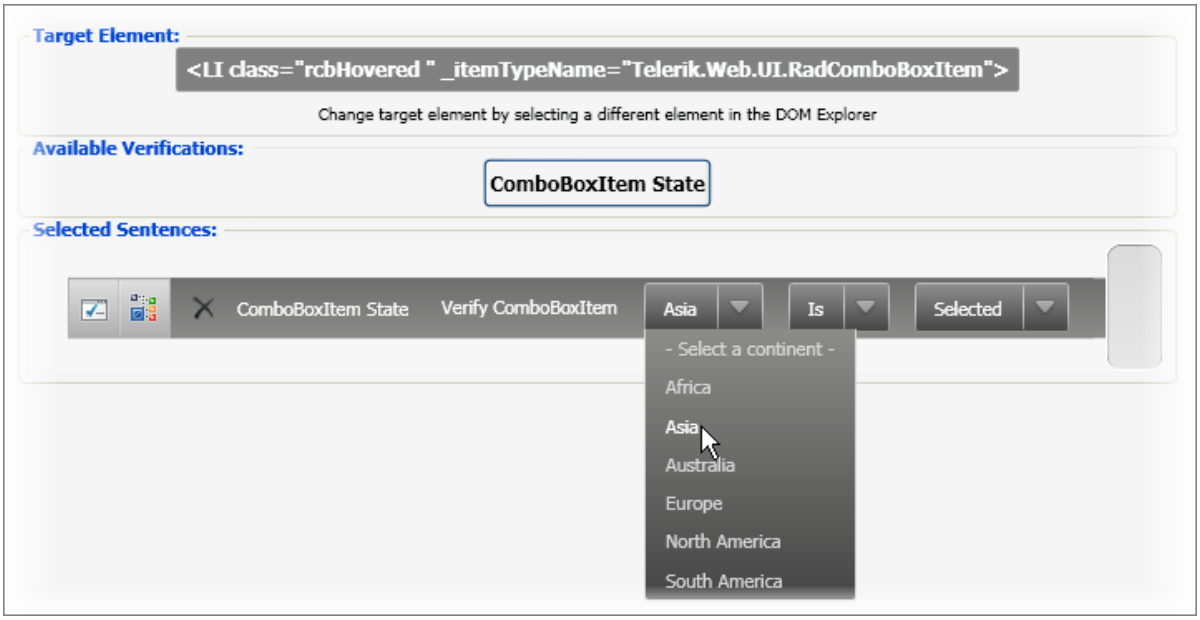
## 7.5 RadControls for ASP.NET AJAX

RadControls for ASP.NET AJAX are built on top of the ASP.NET AJAX framework. They include components for handling partial updates of pages easily and with fine-grained control. RadControls for ASP.NET AJAX also includes a full suite of productivity enhancing, skinnable controls, e.g. grid, tree, etc. Testing these controls may involve previously mentioned techniques for handling JavaScript and AJAX. What makes testing RadControls for ASP.NET AJAX different from any other control are the translators provided by Telerik. Translators provide a greater degree of specificity but at a reduced cost in terms research and expertise on the part of the tester.

In the ASP.NET AJAX example, we looked at the outer markup of a "<SELECT>" (i.e., a drop down list) to see if options had been loaded. If we found an option we could assume the list was loaded from the server. The translator for the RadComboBox lets you simply check the count of items. The screenshot below shows the Quick Tasks for the RadComboBox where you can find the text, the index of the currently selected item, the item count and the drop down state of the combo box (i.e. is the drop down open or not).

Each of the items in the drop down list also has its set of verifications and Quick Tasks. The screenshot below shows the quick tasks for the third item in the drop down.


The State verification for a RadComboBox item has all the possible values in the drop down and also lets you test if a particular item is Selected, Visible, Enabled or Highlighted.




Along with this extra information provided from the translators, you still have access to the HTML elements that make up the RadComboBox as rendered in the browser.

## 7.5.1 Walk Through

In this walk through we will exercise some of the major RadControls for ASP.NET AJAX using the Telerik demos page. First we will use a set of RadComboBox controls set in a cascading series similar to the previous ASP.NET AJAX example. You can contrast this approach with the earlier version that used only intrinsic translators.


### 7.5.1.1 Project Setup

1) From the Visual Studio choose **File > New > Project...** This will display the "New Project" dialog.

2) Define a new test project:

a) In the "New Project" dialog, select the "Test" project type for your language (C# or VB).

b) Select the "Test Project" template.

c) Enter a descriptive name for the test project, a location path and a Solution name.

d) Click **OK** to create the new test project.

### 7.5.1.2 Testing RadComboBox

1) From the Solution Explorer, right-click the test project and select **Add > New Test...** from the context menu. This will display the "Add New Test" dialog.

2) In the "Add New Test" dialog, select the "WebAii Test" template, provide a meaningful Test Name and click **OK** to create the test.

3) Locate the Record button and click it. This will display the Recording Surface.

4) In the Recording Surface, enter "http://demos.telerik.com/aspnet-ajax/combobox/examples/functionality/multiplecomboboxes/defaultcs.aspx" to the browser address bar and then click the Go to Url button. This will load the "Related Combo Boxes" demonstration web page.

5) Select "Europe" from the "Continent" drop down list.

6) Press the Highlighting button to enable it.

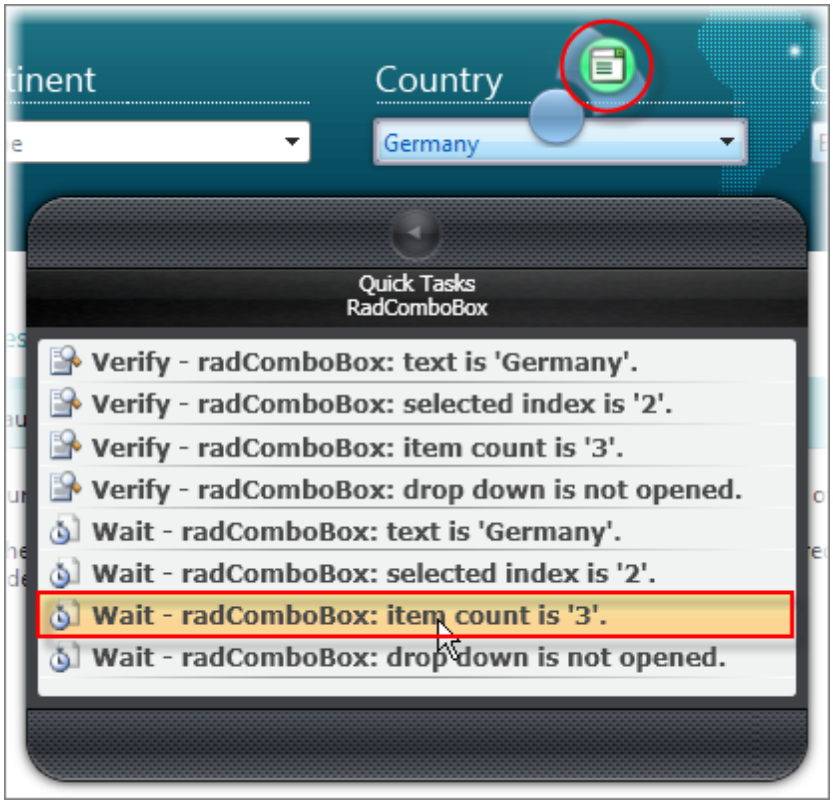
7) Check that the "Country" drop down list item count is greater than zero:

a) Move the mouse over the "Country" drop down list. Wait for the Nub to appear and fan out.

b) Click the RadComboBox leaf to display the Elements Menu (see the screenshot below).

c) Click the Quick Tasks button to display Quick Tasks for RadComboBox.

d) Double-click the "Wait - radComboBox: item count is..." item to add the wait step.



e) Select the new "wait" test step in the Steps Tab. In the Properties pane, change the **CompareType** property to "GreaterThan" and the **ItemCount** to "0".

The test steps so far should look like the screenshot below:


8) Select "Germany" from the "Country" drop down list.

9) Check that the "City" drop down list item count is greater than zero:

a) Move the mouse over the "City" drop down list. Wait for the Nub to appear and fan out.

b) Click the RadComboBox leaf to display the Elements Menu.

c) Click the Quick Tasks button to display Quick Tasks for RadComboBox.

d) Double-click the "Wait - radComboBox: item count is..." item to add the wait step.

e) Select the new "wait" test step in the Steps Tab and in the Properties pane, change the **CompareType** property to "GreaterThan" and the **ItemCount** to "0".

10) Select "Frankfurt" from the "City" drop down list.

11) Verify the text contents of all three combo boxes.

a) Move the mouse over the "Continent" drop down list and wait for the Nub to appear, then click the RadComboBox leaf.

b) Click the Quick Tasks button and double-click "Verify - radComboBox: text is 'Europe'" to add the item as a verification test step.

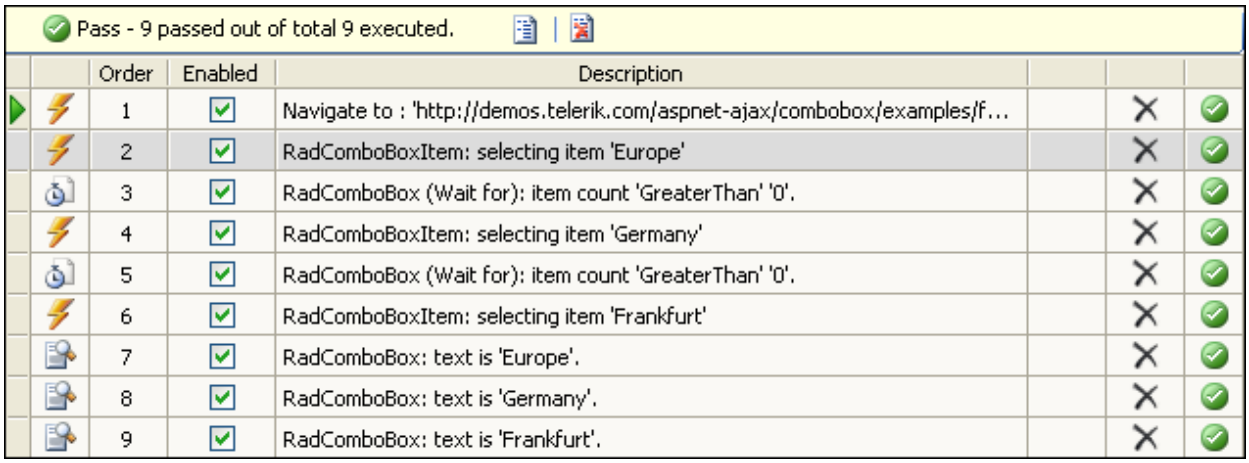
c) Move the mouse over the "Country" drop down list and wait for the Nub to appear, then click the RadComboBox leaf.

d) Click the Quick Tasks button and double-click "Verify - radComboBox: text is 'Germany" to add the item as a verification test step.

e) Move the mouse over the "City" drop down list and wait for the Nub to appear, then click the RadComboBox leaf.

f) Click the Quick Tasks button and double-click "Verify - radComboBox: text is 'Frankfurt" to add the item as a verification test step.

12)Click the Quick Execute button to run the test. All test steps should pass.




| | Order | Enabled | Description |
|---|---|---|---|
| ▶ | 1 | ☑ | Navigate to : 'http://demos.telerik.com/aspnet-ajax/combobox/examples/f... |
| | 2 | ☑ | RadComboBoxItem: selecting item 'Europe' |
| | 3 | ☑ | RadComboBox (Wait for): item count 'GreaterThan' '0'. |
| | 4 | ☑ | RadComboBoxItem: selecting item 'Germany' |
| | 5 | ☑ | RadComboBox (Wait for): item count 'GreaterThan' '0'. |
| | 6 | ☑ | RadComboBoxItem: selecting item 'Frankfurt' |
| | 7 | ☑ | RadComboBox: text is 'Europe'. |
| | 8 | ☑ | RadComboBox: text is 'Germany'. |
| | 9 | ☑ | RadComboBox: text is 'Frankfurt'. |

**Tip!**

In this example, JavaScript code automatically opens the drop down list after the list is loaded. Selecting an item in the drop down list does just that (selects the item), but no more. To close the combo box, you can open it, start recording and close it. The translator for the RadComboBox running in the Recording Surface will pick this up as a "Drop Down Action - Close". You can place the test step just after the item selection.

| |
|---|
| RadComboBox (Wait for): item count 'GreaterThan' '0'. |
| RadComboBoxItem: selecting item 'Germany' |
| RadComboBox: drop down action -> 'Close' |

## 7.6 Testing RadGrid

In this example you will add a row of data to a RadGrid, verify that the expected values ended up in the correct cells and delete the row.

1) From the Solution Explorer, right-click the test project and select **Add > New Test...** from the context menu. This will display the "Add New Test" dialog.
2) In the "Add New Test" dialog, select the "WebAii Test" template, provide a meaningful Test Name and click **OK** to create the test.
3) Locate the Record button and click it. This will display the Recording Surface.
4) In the Recording Surface, enter "http://demos.telerik.com/aspnet-ajax/grid/examples/client/insertupdatedelete/defaultcs.aspx" to the browser address bar and then click the Go to Url button. This will load the "Grid / Client Side Insert/Update/Delete" demonstration web page.
5) Click the "Add new employee" tab.
6) In the "Last Name" edit box enter "Smith".
7) In the "First Name" edit box enter "Bob".
8) In the "Title" edit box enter "Sales Representative".
9) Select "Mr." from the "Title of courtesy" drop down list.
10) In the "Birth date" enter "12/2/1965".

***Notes***

Optionally, you could click the "Fast navigation" or "Year" button, but that will add a number of steps as you navigate to the correct year. The additional steps in the Steps Tab will look something like the screenshot below.

| |
|---|
| RadPicker: calendar popup button click |
| RadCalendar: Navigate -> Open FastNavigation popup |
| CalendarFastNavigation: Fast Navigate -> 1995 - 2004 |
| CalendarFastNavigation: Fast Navigate -> 1985 - 1994 |
| CalendarFastNavigation: Fast Navigate -> 1975 - 1984 |
| CalendarFastNavigation: Fast Navigate -> 1965 - 1974 |
| CalendarFastNavigation: Fast Navigate -> Select Year: 1965 |
| CalendarFastNavigation: Fast Navigate -> Select Month: Dec |
| CalendarFastNavigation: Fast Navigate -> Ok clicked |

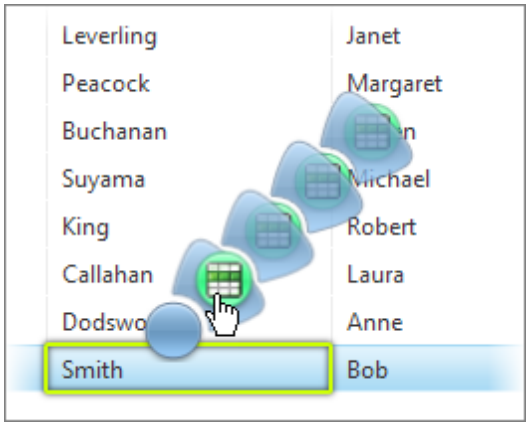
11) In the "Notes" edit box enter "Formerly worked for WidgetCo in wholesale division.".

12)Click the "Add" button. At this point the test steps in Steps Tab should look like the screenshot below.

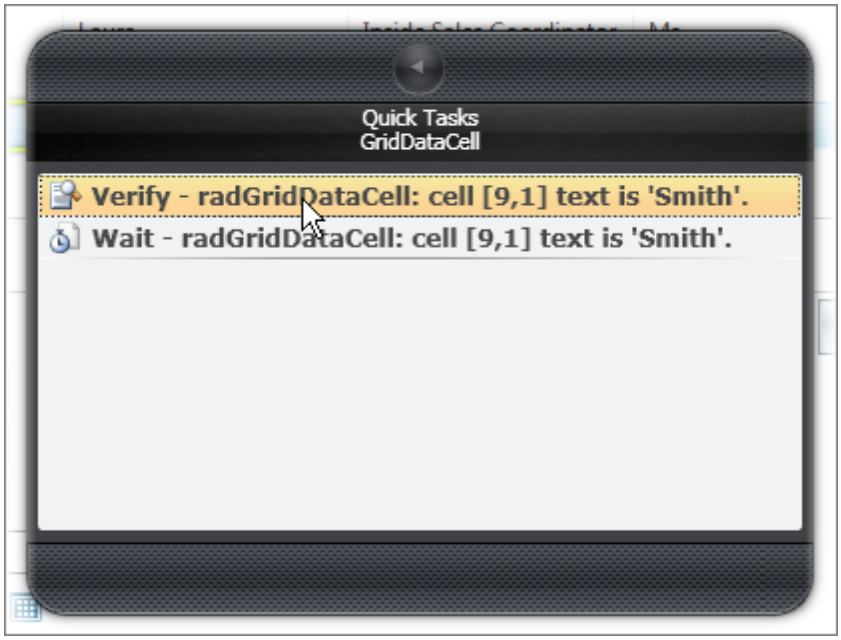
| Description |
| --- |
| Navigate to : 'http://demos.telerik.com/aspnet-ajax/grid/examples/client/insertu... |
| Click 'AddNewSpan' |
| RadInput: value 'Smith' entered. |
| RadInput: value 'Bob' entered. |
| RadInput: value 'Sales Representative' entered. |
| RadComboBoxItem: selecting item 'Mr.' |
| RadInput: value '12/2/1965' entered. |
| RadEditor: Set text: 'Formerly worked for ...'. |
| Click 'SaveChangesSubmit' |

13)Press the Highlighting button to enable it.

14)Locate the grid row that holds the newly added "Bob Smith" record. Hover the mouse over the cell for the "LastName" column. When the Elements Menu appears, select the "GridDataCell" leaf.



15)Double-click the entry titled "Verify - radGridDataCell: cell[9,1] text is 'Smith'.".




16)Hover the mouse over the cell for the "FirstName" column of the grid. When the Elements Menu appears, select the "GridDataCell" leaf. Double-click the entry titled "Verify - radGridDataCell: cell[9,2] text is 'Bob'.".

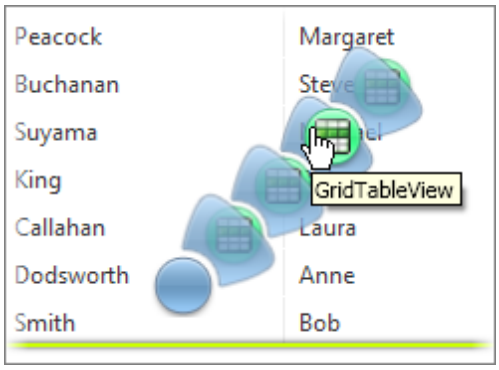
17)Hover the mouse over the cell for the "Title" column of the grid. When the Elements Menu appears, select the "GridDataCell" leaf. Double-click the entry titled "Verify - radGridDataCell: cell[9,3] text is 'Sales Representative'.".

18)Hover the mouse over the cell for the "TitleOfCourtesy" column of the grid. When the Elements Menu appears, select the "GridDataCell" leaf. Double-click the entry titled "Verify - radGridDataCell: cell[9,4] text is 'Mr.'.".

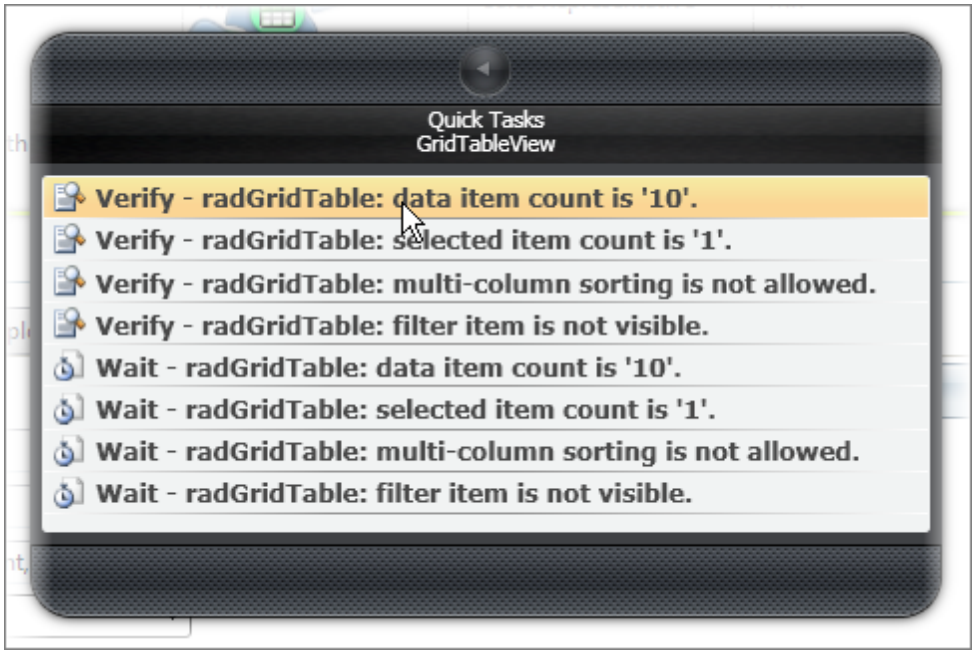
19)Hover the mouse over the cell for the "BirthDate" column of the grid. When the Elements Menu appears, select the "GridDataCell" leaf. Double-click the entry titled "Verify - radGridDataCell: cell[9,5] text is '12/02/1965'.".

20).Press the **TAB** key to exit the "BirthDate" field. This workaround needs to be done to register the entry made in the "BirthDate" field. If you miss this step, the birth date doesn't show up in the row that is added.

21)Hover the mouse over any cell in the grid. When the Elements Menu appears, select the second from the outermost leaf of the Nub. The hint shows that this leaf is the displayed by the "GridTableView" translator.




22) Double-click the Quick Tasks entry titled "Verify - radGridTable: data item count is '10'.".

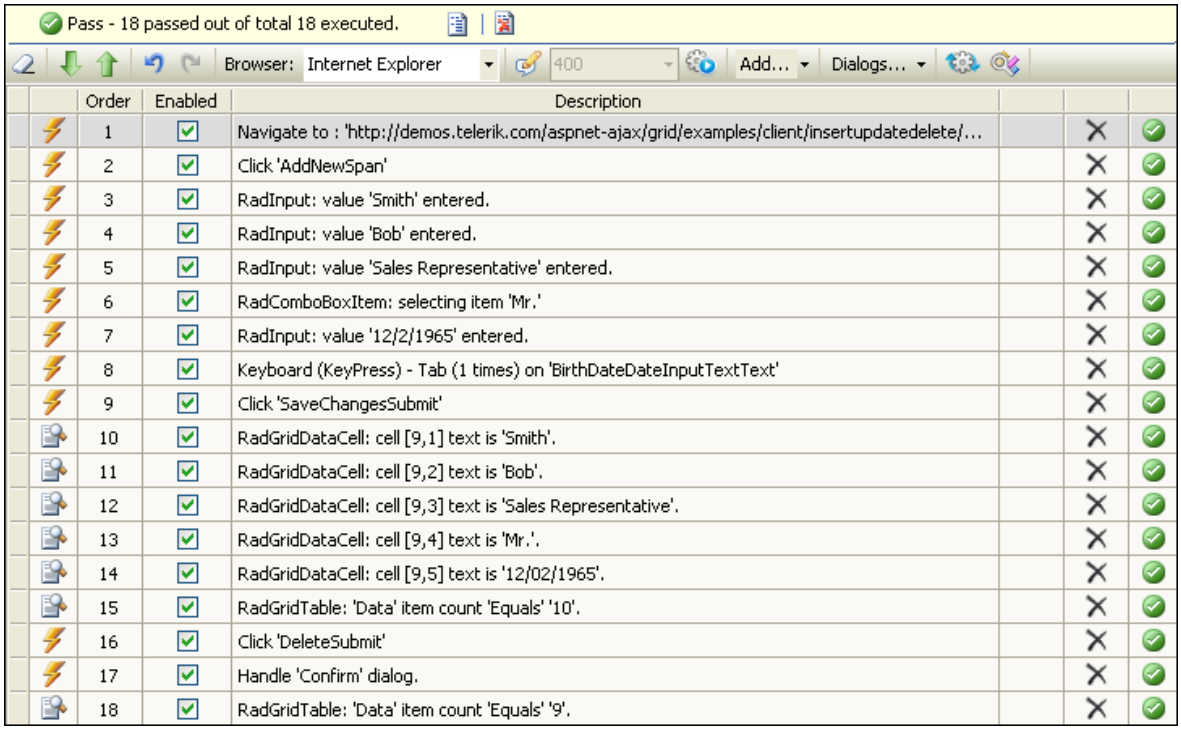



23)Click the "Delete" button located below the grid in the "Edit employee" tab.

24)In the confirmation dialog that appears, click the **OK** button.

25)Hover the mouse over any cell in the grid. When the Elements Menu appears, select the second from the outermost leaf of the Nub. The hint shows that the leaf is displayed by the "GridTableView" translator. Double-click the Quick Tasks entry titled "Verify - radGridTable: data item count is '9'.".

The test steps at this point should look like the screenshot below:




26)Click the Quick Execute button to run the test. All test steps should pass.

### Gotcha!

There are a few possible issues you may run into while performing this exercise:

- The birth date as entered is "12/2/1965". When the row is added to the grid, the cell displays "12/02/1965". You need to verify the date exactly as formatted in the grid.
- The step can fail because the a cell doesn't contain the right data, but also because it can't be found. To see how the element is located, use the Elements Explorer, right-click and select the Edit option from the context menu.
- Make sure that the browser is maximized. When the space is reduced, all input may end up in the same control, e.g. "Last: SmithBobSales...".

## 7.7 Wrap Up

In this chapter we talked about the evolution of web applications, the challenges these new evolutionary changes present to testing and how WebUI Test Studio addresses each of these issues. In particular, you learned how to overcome timing issues that occur in AJAX applications. You tested sample controls from the Microsoft asp.net AJAX web site and RadControls for ASP.NET AJAX.

# Part VII

Drag and Drop

# 8 Drag and Drop

## 8.1 Objectives

In this chapter you will learn how to automate drag & drop operations, both by simply recording the drag & drop and by using the enhanced user interface of the Drag & Drop option in the Elements Menu. You will learn basic terminology and some of the key properties used in drag & drop operations. You will learn how to automate the drag of an element to an arbitrary point in the browser window and also learn how to drag elements onto other elements. You will learn how to control offsets to precisely set the location of the dragged element and the drop target.

**Find the projects for this chapter at...**

\Courseware\Projects\<CS|VB>\DragAndDrop\DragAndDrop.sln

## 8.2 Overview

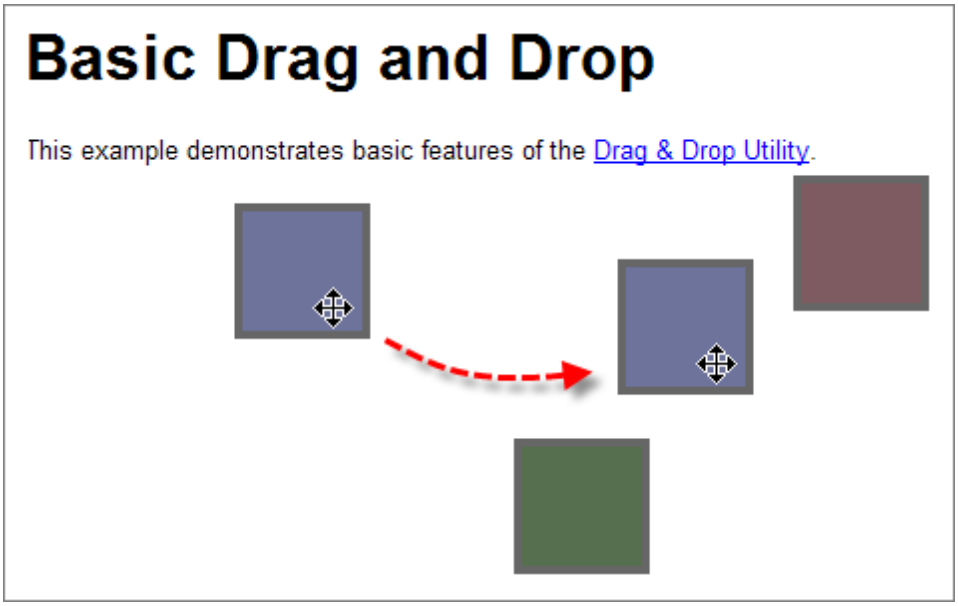
You can automate the drag & drop of any element in a web page including simple HTML elements, RadControls for ASP.NET AJAX and Silverlight elements. Automating drag & drop operations has been abstracted by WebUI Test Studio so that the general automation process is the same no matter if we're dragging an HTML element or a Silverlight element.

There are two ways to automate a drag & drop operation. The first is simply to turn on recording in the Recording Surface and drag elements directly with the mouse. The second is to use the Elements Menu Drag & Drop option. This second method provides enhanced visual tools that assist in placing dragged elements in precise locations. The result of both methods is to create a drag & drop action as a test step in the Steps Tab.

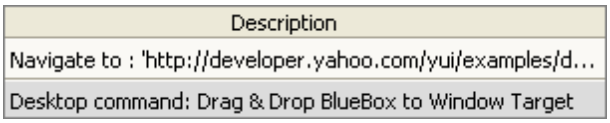
The destination of the drag & drop operation is called the "drop target" and can be the entire browser window or another element. The drag & drop mechanism is flexible enough to handle drop targets that are resized or that move multiple times. WebUI Test Studio allows you to fine tune the exact placement of the element in relation to the drop target.

## 8.3 Drag & Drop Basics

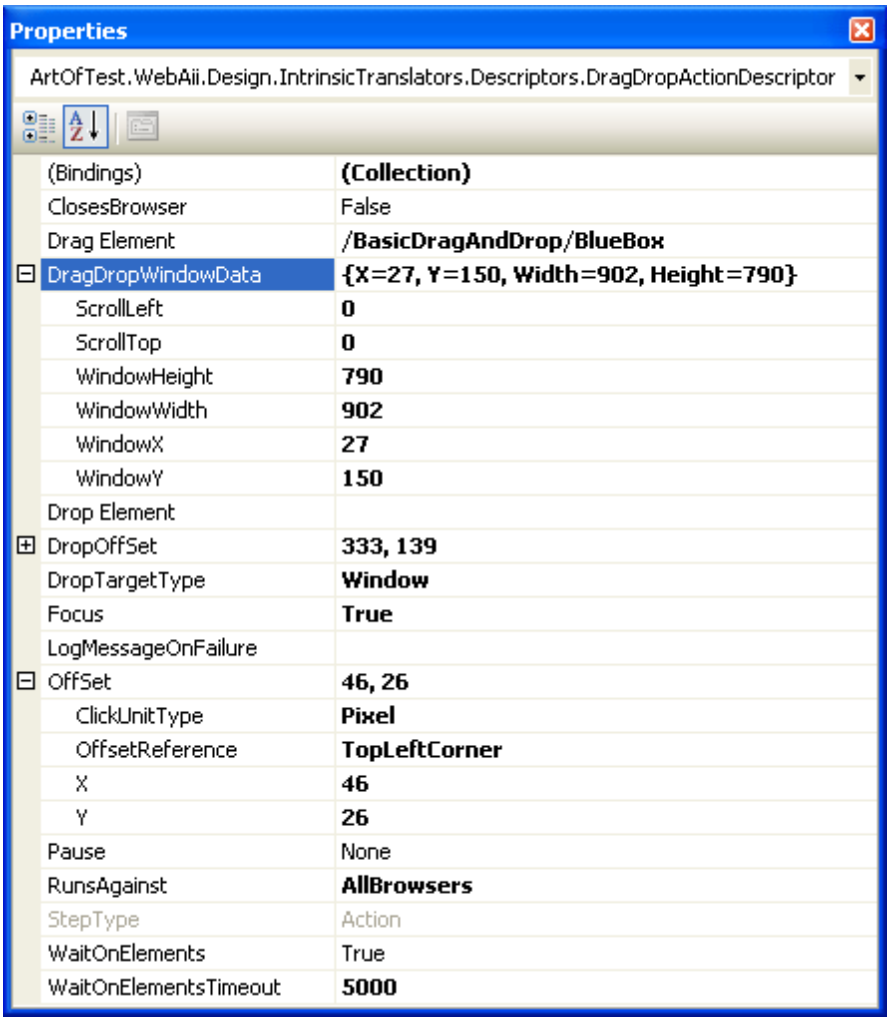
Let's try dragging a simple element to some location in the page. The example page has three HTML "<DIV>" elements colored green, blue and maroon. Recording is turned on in the Recording Surface and the blue element is dragged and dropped onto a new location in the page.




A new "Desktop command" action is created and added to the Steps Tab list of test steps as shown in the screenshot below.



The properties for this drag & drop test step control the behavior and location of the drop.




- The **Drag Element** property is the key to the element being dragged, i.e. the name of the element as listed in the Elements Explorer. In the screenshot, the Drag Element is named "BlueBox" and lives on a page called "BasicDragAndDrop".
- The **Offset** property controls where the element is being dragged from. This is the location of the mouse cursor when the user presses the left mouse button to begin the drag. In this example, the Offset 46,26 pixels from the top left corner of the element. The **ClickUnitType** setting can be "Pixel" or "Percentage". The offset is calculated in relation to the OffsetReference which may be set to any of the corners of the element (e.g. "BottomRightCorner"), the center line of the element (e.g. "LeftCenter", "BottomCenter") or to the "AbsoluteCenter" of the element.
- **DropTargetType** is a key property that can be "Window" or "Element" and determines if the element is dragged to some location in the browser window or to a location on another element.
- **DropElement** is the key to the element being dropped onto. In the screenshot, the DropTargetType is "Window", so the Drop element is not specified. **DropOffset** serves the same purpose as the Offset property, but allows you to fine tune the exact location of the drop.

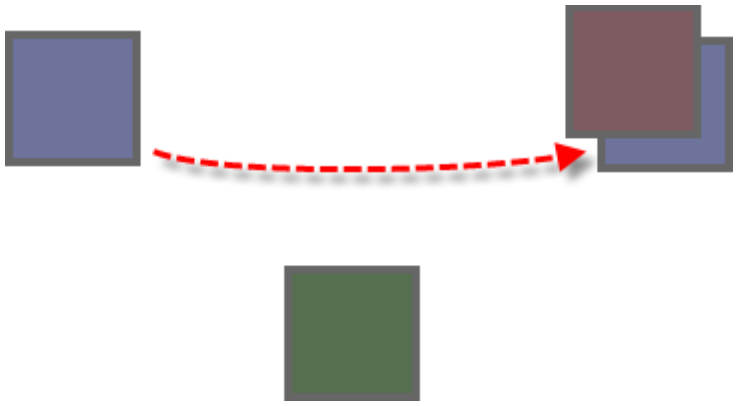
**Tip!**

The ClickUnitType property "Percentage" setting is handy when the element may be resized to some unknown dimensions. For example, if the ClickUnitType is "Percentage" and the X & Y location is 50, 50, the element will be dropped at the center point.

- If the **Focus** property value is "True", then elements are scrolled into view before performing actions on them. If the DropTargetType property value is "Window", then the browser window will be resized as well.
- **DragDropWindowData** is used when the entire window is considered the drop target. The settings in this property determine the location and dimensions of the window. If DropTargetType is "Window", and Focus is "True", then the browser window is resized to the dimensions specified in this property.

## 8.4 Dragging to an Element

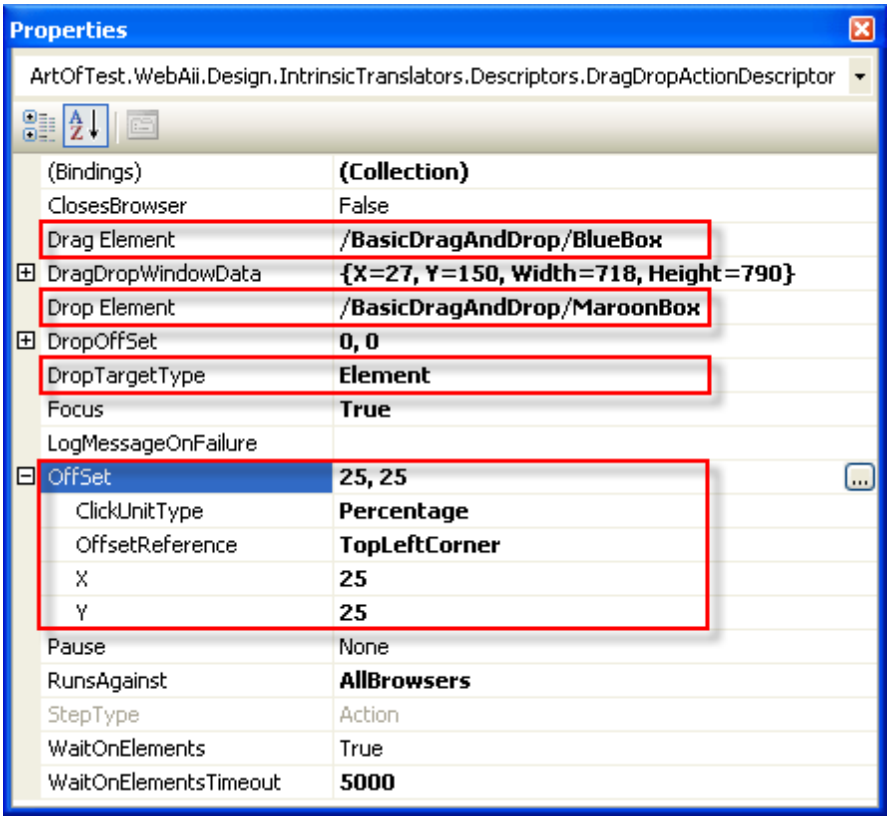
For the sake of comparison, here's another drag & drop automation sample where the "Blue" element is being dragged and the "Maroon" element is the drop target.



***Notes***

Even though we're dropping elements "on to" other elements, the dragged element may appear underneath the drop target, as shown in the screenshot above where the drop target "maroon box" partially covers the dragged "blue box" element.

The properties of this drag operation in the screenshot below show the Drag Element is the "BlueBox", the Drop Element is the "MaroonBox", the DropTargetType is "Element", and finally the Offset of the dragged element is 25% from the top left corner.

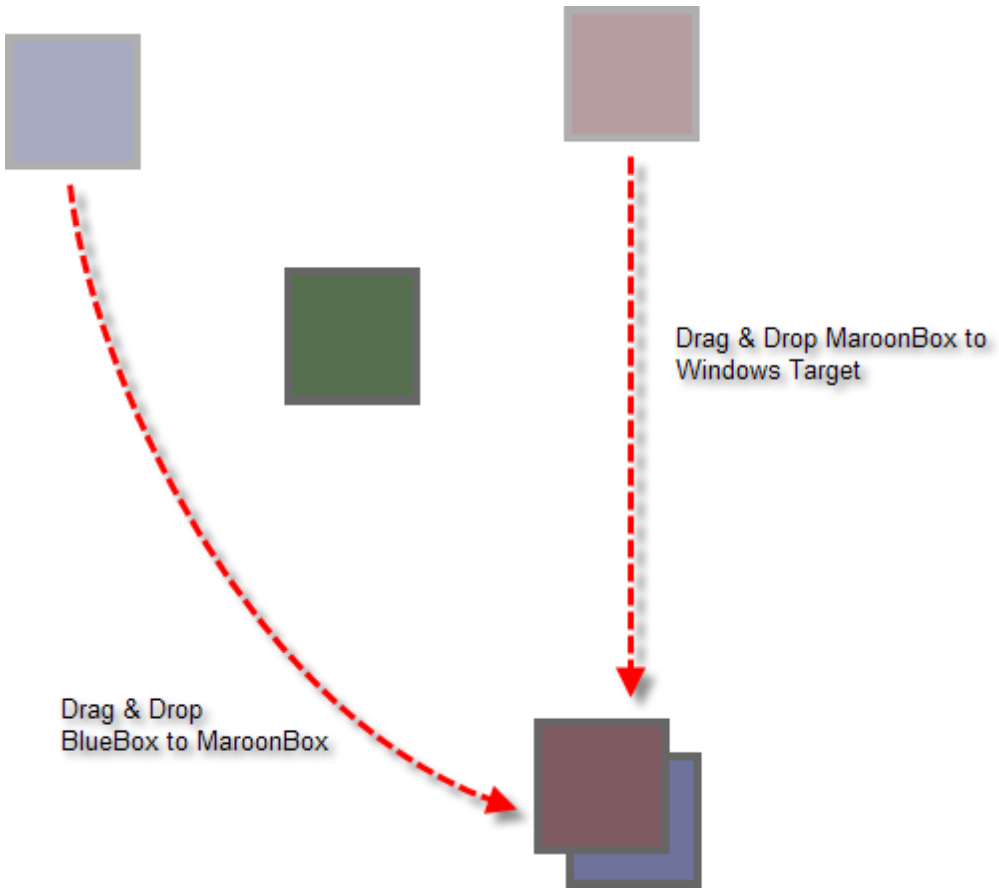


## 8.5 Hitting a Moving Target

One of the nice drag & drop features in WebUI Test Studio is the ability to drag to an element that may have moved by the time the drag takes place. The drag target is an *element*, not a fixed location on the web page, so if the element gets moved, you can still drop to it. You can test this yourself by adding a step that drags the target element to a new location just before it becomes a drop target. Starting from the previous example, we can drag the "MaroonBox" to a new location, just before we try to drop the "BlueBox" element on it. The test steps appear in the screenshot below.

| Description |
| --- |
| Navigate to : 'http://developer.yahoo.com/yui/examples/dragdro... |
| Desktop command: Drag & Drop MaroonBox to Window Target |
| Desktop command: Drag & Drop BlueBox to MaroonBox |

The "Drag & Drop MaroonBox to Windows Target" step drags the maroon box to a point below its initial position. The next step uses the exact same definition as the previous example where the blue box element is dragged to the maroon box. Nothing has changed in this step, yet the blue box "follows" the maroon box to its new position.



## 8.6 Using the Elements Menu

The Drag & Drop feature of the Elements Menu displays an enhanced user interface that allows you to visually pinpoint the exact offset locations for both the dragged element and drop target. The enhanced UI is somewhat like a "Wizard" dialog that asks a series of questions. To illustrate how the enhanced UI is used, we will again drag the "BlueBox" onto the "MaroonBox" element.

1) From the Visual Studio choose **File > New > Project...** This will display the "New Project" dialog.

2) Define a new test project:

   a) In the "New Project" dialog, select the "Test" project type for your language (C# or VB).

   b) Select the "Test Project" template.

   c) Enter a descriptive name for the test project, a location path and a Solution name.

   d) Click **OK** to create the new test project.

3) From the Solution Explorer, right-click the test project and select **Add > New Test...** from the context menu. This will display the "Add New Test" dialog.

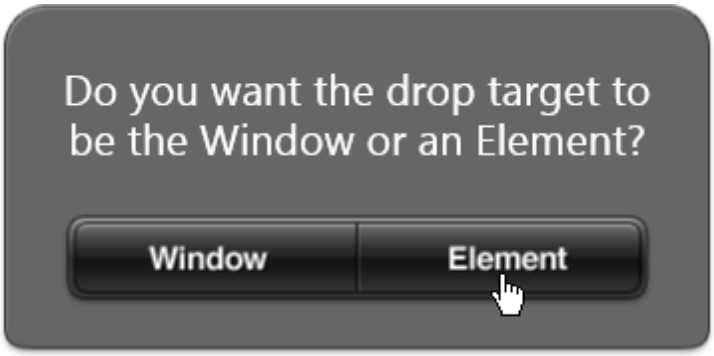
   a) In the "Add New Test" dialog, select the "WebAii Test" template, name the test "Basics.aii" and click **OK** to create the test.

   b) Locate the Record button and click it. This will display the Recording Surface.

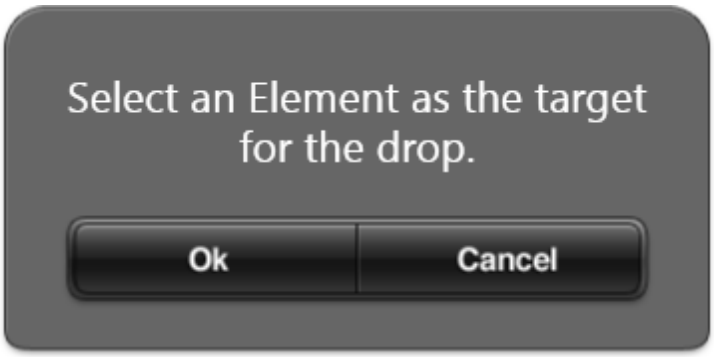
   c) In the Recording Surface, enter "http://developer.yahoo.com/yui/examples/dragdrop/dd-basic_clean.html" to the browser address bar and then click the Go to Url button.

4) To use the enhanced UI, click the Highlighting button  , hover the mouse above the element to be dragged until the Nub appears. Click the Nub, then click the Drag & Drop button.

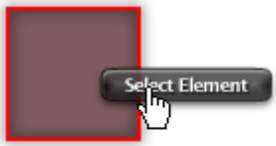
5) A prompt appears where you set the drop target be the entire window or a specific element.



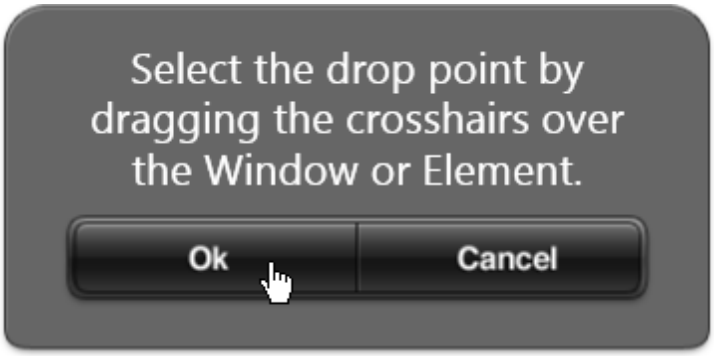

6) The next prompt asks you to select a drop target element. Click **OK** to continue.



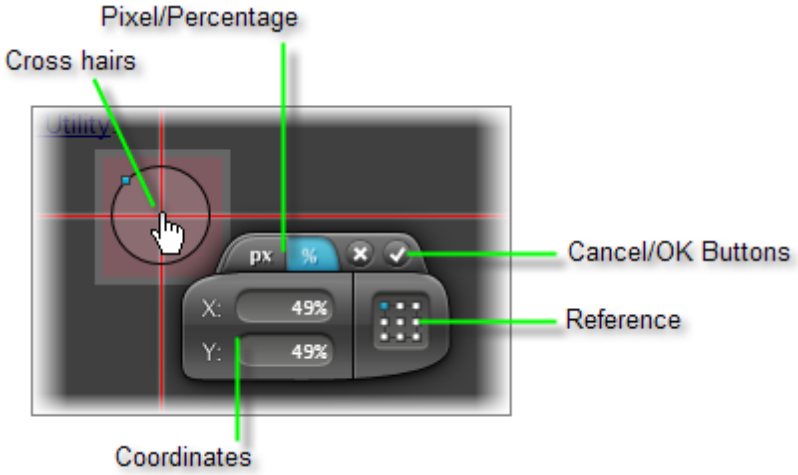

7) Hover the mouse over the maroon element and click the "Select Element" button.



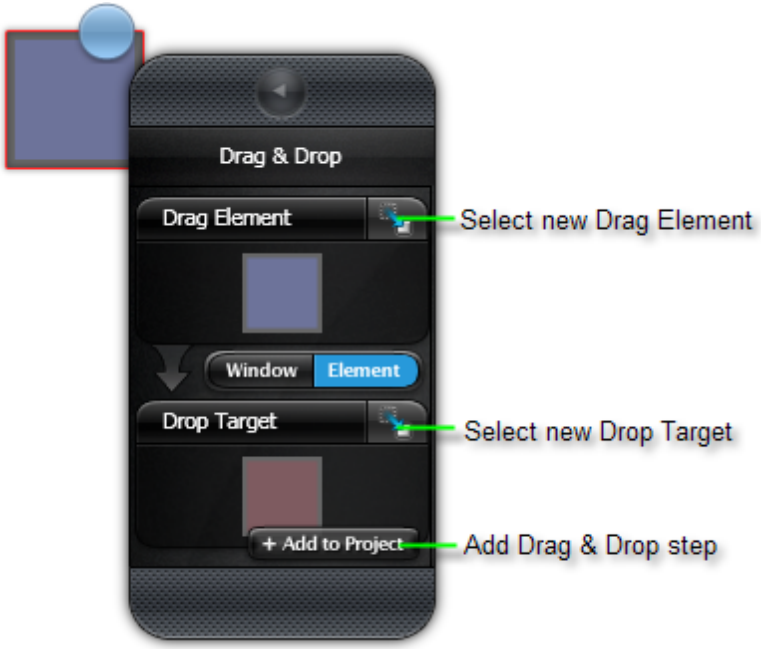

8) The next prompt asks you to select a drop point. Click **OK** to continue.



9) Position the crosshairs near the center of the element, click the "%" percentage button, leave the "Reference" button in the upper left hand corner and finally, click the **OK** ("check mark") button.



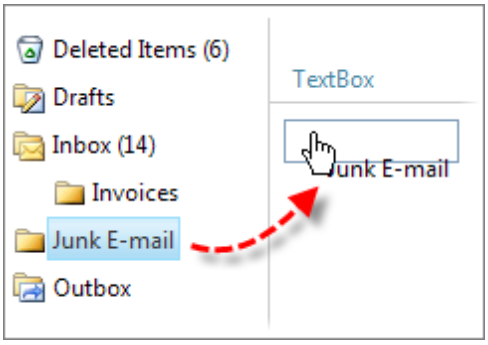

10)A summary of the Drag & Drop operation will appear in the Elements Menu. Click the "Add to Project" button to include the new drag & drop action as a test step.



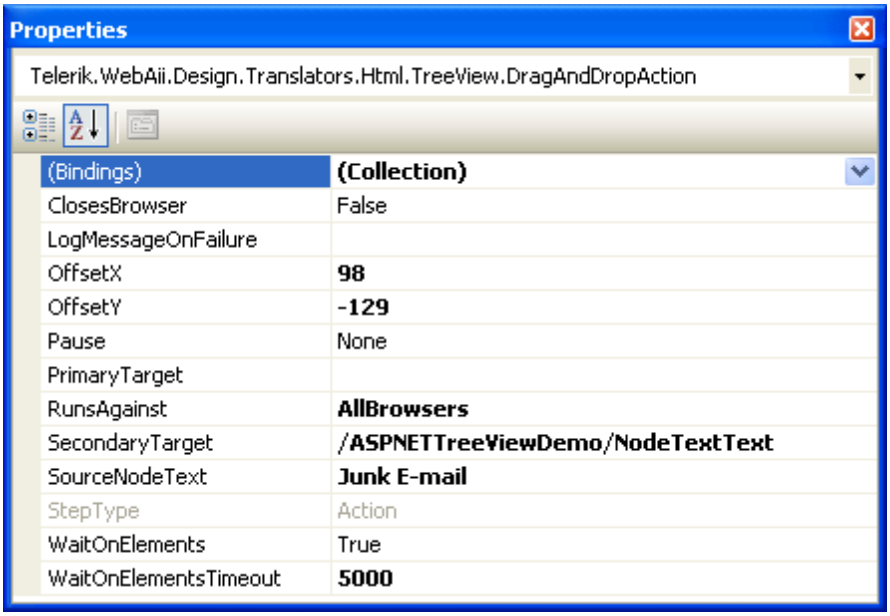

11)Run the test. The blue element should drag over to the maroon element. All test steps should pass.

## 8.7 Translators

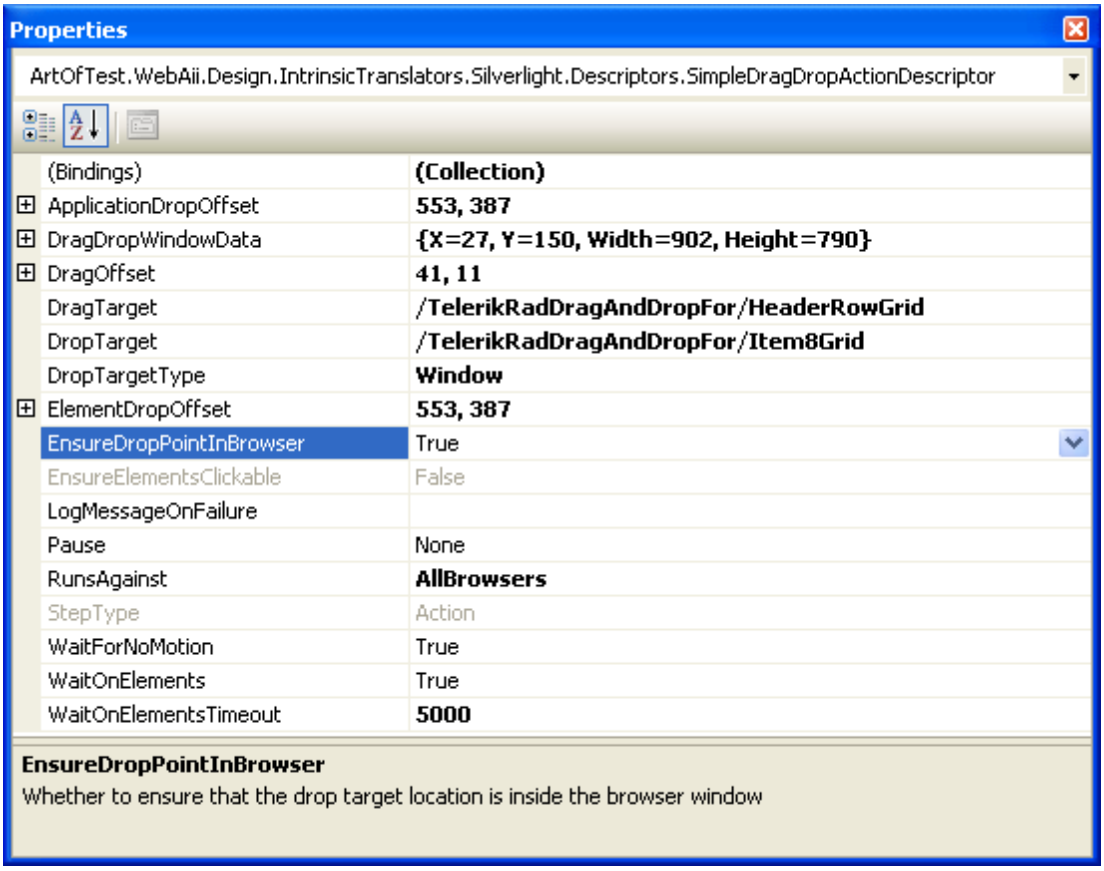
The translators for RadControls for ASP.NET AJAX or Silverlight allow more sophisticated drag & drop operations based on the translators internal knowledge of the elements being manipulated. In the example below, an AJAX enabled RadTreeView node is dragged to a TextBox.




A TreeView DragAndDropAction action is created automatically with properties that contain the identity of the tree view node being dragged and the location of the drop target. The drop target can be an offset in relation to the source node using the **OffsetX** and **OffsetY** properties or can be another element by using the **SecondaryTarget** property.




Here's another example where a RadControls for Silverlight RadTreeView node was dragged to a data grid. The properties look familiar except for the **EnsureDropPointInBrowser** and **ApplicationDropOffset** properties. The ApplicationDropOffset is similar to the other offset properties we've seen so far except that the offset, in this case, is relative to the Silverlight application.




## 8.8 Wrap Up

In this chapter you learned how to automate drag & drop operations, both by simply recording the drag & drop and by using the Drag & Drop option in the Elements Menu. You learned basic terminology and some of the key properties used in drag & drop operations. You learned how to automate the drag of an element to an arbitrary point in the browser window and also learned how to drag elements onto other elements. You learned how to control offsets that set the location of the dragged element and the drop target.

# Part


Testing Silverlight Applications

# 9 Testing Silverlight Applications

## 9.1 Objectives

In this chapter we will talk briefly about what Silverlight is, the unique issues that arise when testing Silverlight applications and how WebUI Test Studio addresses Silverlight specific situations. You will review differences to the Visual Studio environment when testing a Silverlight application, paying special attention to the 3D Viewer, DOM Explorer and Elements Explorer.

You will perform walk through exercises that test cascading combo boxes and RadGridView. You will also test an entry form that performs validation and has several Silverlight controls including standard TextBlock, RadSlider, RadCalendar and RadMaskedTextBox.

**Find the projects for this chapter at…**

\Courseware\Projects\<CS|VB>\TestingSilverlight\TestingSilverlight.sln

## 9.2 Overview

Microsoft created Silverlight to support the building of rich media applications. Silverlight is a "plug in", object embedded to a standard web page that runs right in the browser. Silverlight applications present unique testing issues, e.g. the Silverlight elements are not readily accessible, the user interface can be asynchronously updated and elements are likely to be animated.

WebUI Test Studio is the first scriptless record and playback solution for Silverlight. With WebUI Test Studio you can build a single test case that interacts with both HTML and Silverlight elements, even on the same page. WebUI Test Studio allows you to test applications that have heavy interaction between HTML, AJAX and Silverlight, for example when an HTML element triggers an event in the Silverlight application. With WebUI Test Studio you can automate end-to-end scenarios, verify results and re-test against multiple browsers (IE/Firefox/Safari).

WebUI Test Studio features a consistent user interface that makes testing HTML and Silverlight elements substantially similar. Once the tester is familiar with the WebUI Test Studio user interface, learning to test Silverlight elements is a short learning curve. Likewise, the programming interface (see the "WebAii Framework" chapter) doesn't require you to learn an entirely new automation framework. Silverlight tests can be added incrementally to match Silverlight "islands" as they are added to existing applications.

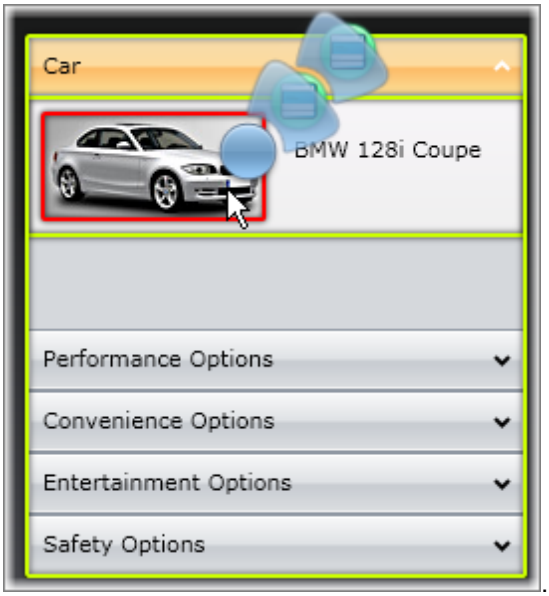
### Silverlight Basics

Although you don't have to be thoroughly familiar with the inner workings of Silverlight to use WebUI Test Studio against a Silverlight application, you should be aware of a few fundamental terms and concepts.

Silverlight user interfaces are defined using **XAML** (rhymes with "Camel" and stands for "Extended Application Markup Language"). The XAML for a new, empty Silverlight application looks something like the example below. Again, you may not have occasion to use XAML directly, so this example is only to familiarize you with what the basic building blocks look like.


```
<UserControl x:Class="SilverlightApplication2.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d" d:DesignWidth="640" d:DesignHeight="480">
  <Grid x:Name="LayoutRoot">

  </Grid>
</UserControl>
```

When the XAML is rendered in the browser, the Silverlight elements form a conceptual "**visual tree**". Each object in the visual tree may contain other objects. For example, a panel may contain a button and the button may in turn contain a text box. Fortunately, the Recording Surface allows you to navigate the visual tree with the mouse, using advanced highlighting cues to guide your way. The screenshot below shows an Image element contained by a RadPanelBarItem that in turn is contained by a RadPanelBar. The highlighting shows the relationship of the elements in the visual tree.



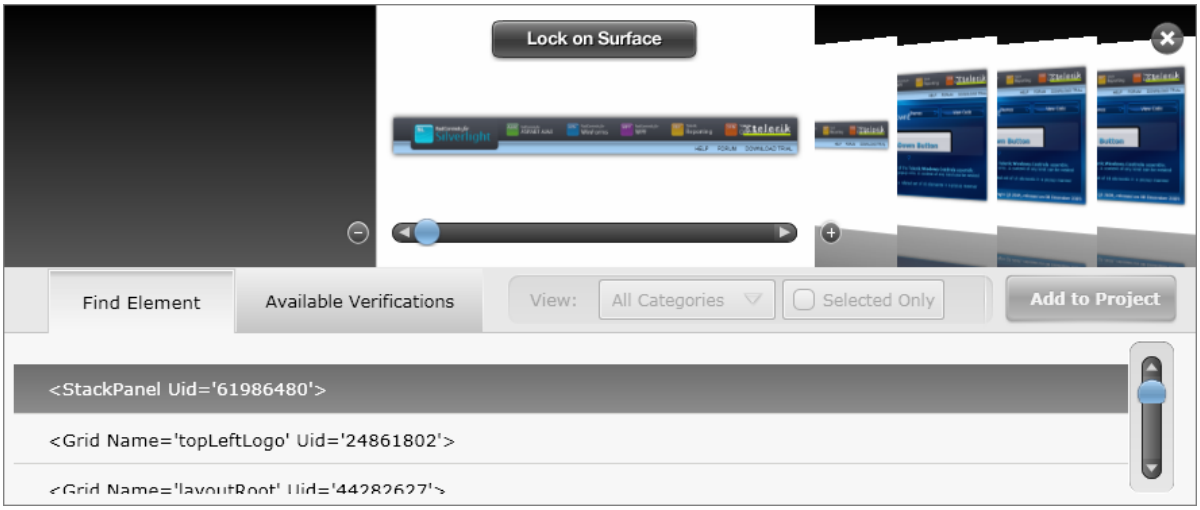

## How Does WebUI Test Studio Address Silverlight Testing Issues?

- **Identification:** Real Silverlight applications may use "control templates" and data binding to produce quite complex visual trees. The visual tree can contain elements that are not easily searchable by name because the name is not known ahead of time or the element names may be duplicated. WebUI Test Studio allows elements to be identified and located by other criteria or combination of criteria, e.g. by text, partial text or element type.
- **Synchronization**: Testing Silverlight, like testing AJAX applications, requires synchronizing with events that can occur at any time. But Silverlight throws in a new twist: elements may be moving when you want to perform some action against it. When you perform an action, not only does WebUI Test Studio need to wait for an element to be exist and be visible, but Silverlight elements may also fly-in, fly-out, expand, collapse or animate. WebUI Test Studio provides a robust run time mechanism that performs several checks against an element's state. You can wait for an element to exist in the Visual Tree or be removed from the Visual Tree. You can wait for an element to be visible or not. For elements that are animated and may still be moving at any time, you can wait for the element to stop moving. All of these checks happen automatically by default.
- **Reveal Control Internals**: The generic Silverlight translator provides common property information for any Silverlight element while control specific translators surface additional information about RadControls for Silverlight elements.

## 9.3 Visual Studio Integration

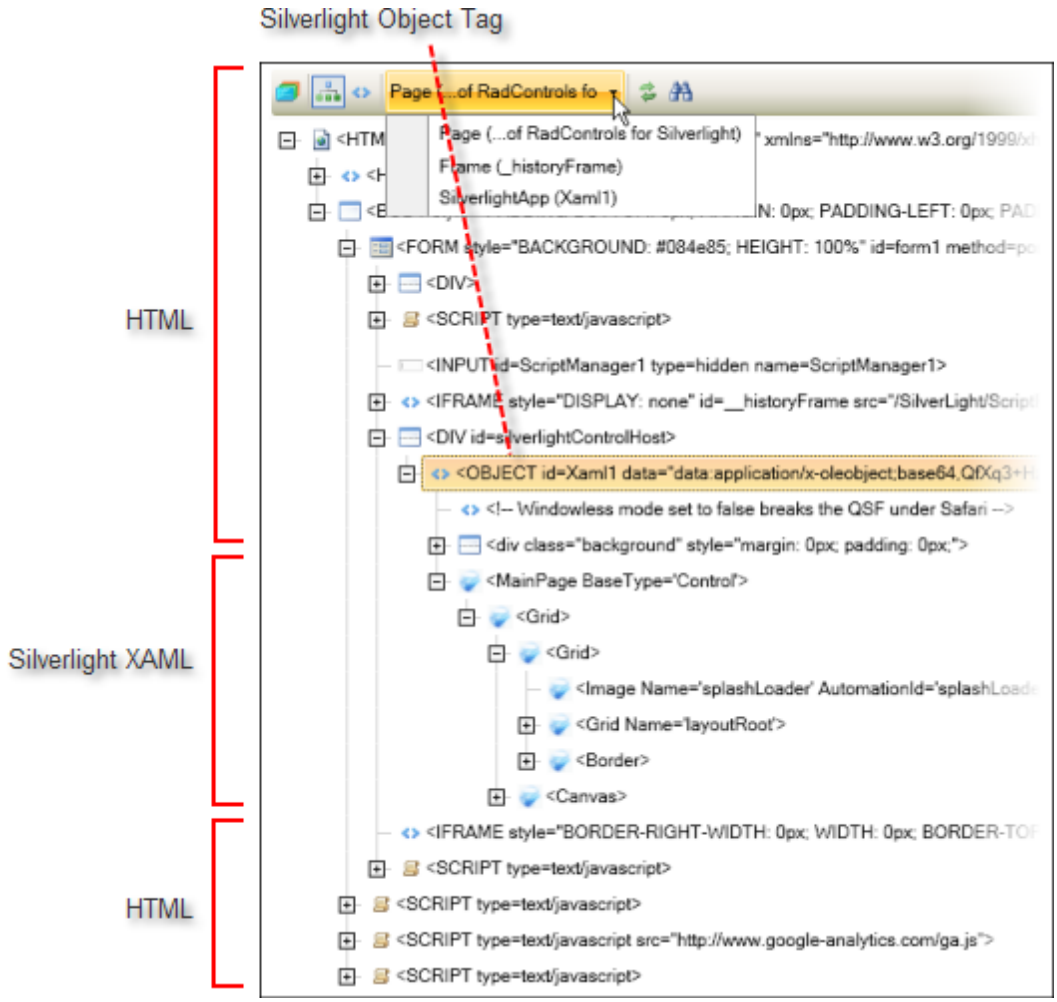
WebUI Test Studio provides seamless integration with Visual Studio that is consistent between standard HTML based applications and Silverlight applications. If you are working with an HTML or AJAX application, then navigate to a Silverlight application embedded on the page, all the WebUI Test Studio tools in Visual Studio work with few or no changes.

For example, the 3D Viewer graphical representation of elements show the highlighted element up to the Silverlight Page element. How the tester finds elements and works with the 3D Viewer has the same usage pattern, regardless of the type of element. The screenshot below shows the 3D Viewer where a Silverlight "StackPanel" element has been highlighted. Elements containing the StackPanel are displayed to the right side in the graphical representation. The "Find Element" tab also lists the StackPanel and elements that contain the StackPanel. "Available Verifications" list the possible verification sentences for any selected element. These functions are identical when to their HTML counterparts.

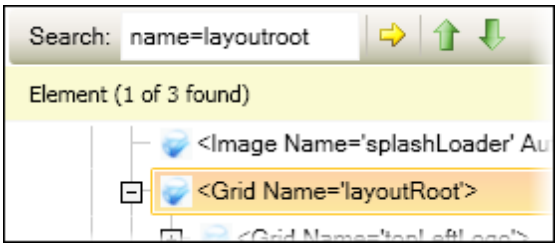



### DOM Explorer

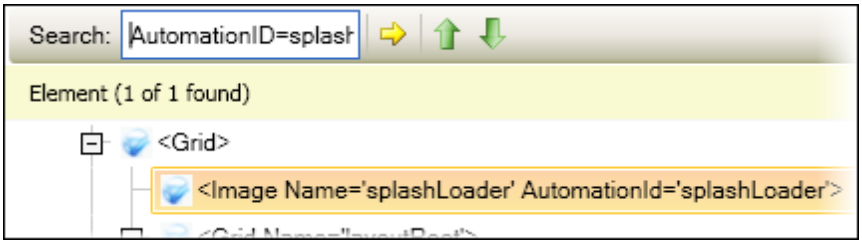
The DOM Explorer is unique in that it shows the relationship between the HTML page and a Silverlight application. In a Silverlight application, a web page contains a special "<object>" tag that "hosts" the Silverlight application. The Silverlight application itself is defined in a special XML dialect called "XAML" (Extended Application Markup Language). The DOM Explorer lets you navigate the web page elements and the Silverlight application in a seamless manner that doesn't require you to know the internal structure of the web page or the Silverlight XAML. The screenshot below shows a typical Silverlight application in the DOM Explorer. The Silverlight logo displays next to each Silverlight element.



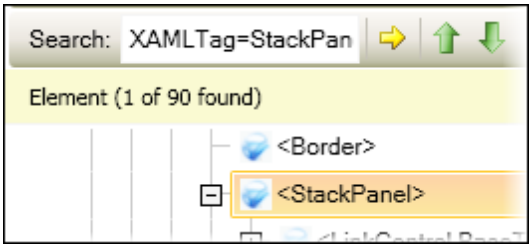

If you right-click an element in the DOM Explorer, the context menu will allow you to "Goto" the Page, Frame, or Silverlight application. To find Silverlight elements with the DOM Explorer search tool use the Silverlight specific search terms **AutomationId**, **TextContent**, **XamlTag** and **Name**. For example, to find an element with the name of "layoutRoot", use the search criteria "Name=layoutRoot".



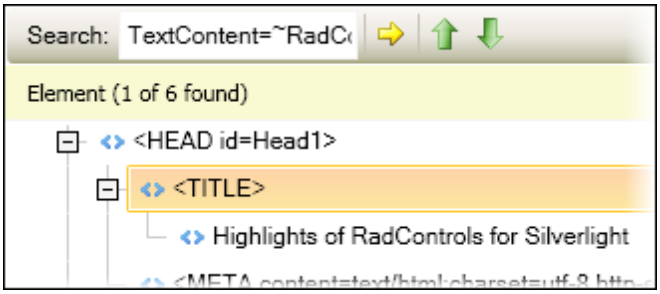

AutomationId is a property used to identify an element that can be automated. You can search for elements with specific AutomationId property values. The example in the screenshot below shows a Silverlight image element with an AutomationId of "splashLoader". The search string is "AutomationID=splashloader".




You can also look for specific types of XAML tags. The screenshot below shows that 90 elements were found in the example application where the tag was "<StackPanel>". The search string is "XAMLTag=StackPanel".

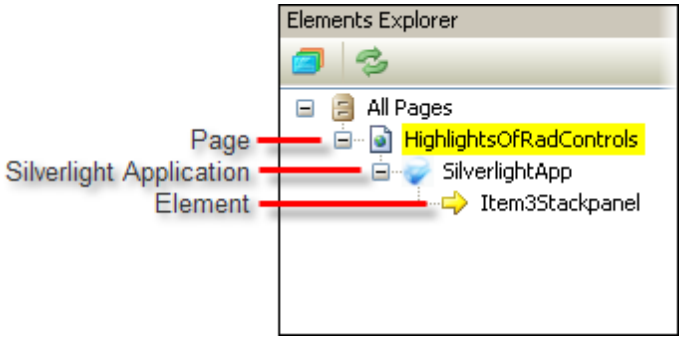



Silverlight elements with particular text can be found using TextContent. The screenshot below shows that 5 elements were found where the TextContent contains ("~") the text "RadControls". The search string is "TextContent=~RadControls".

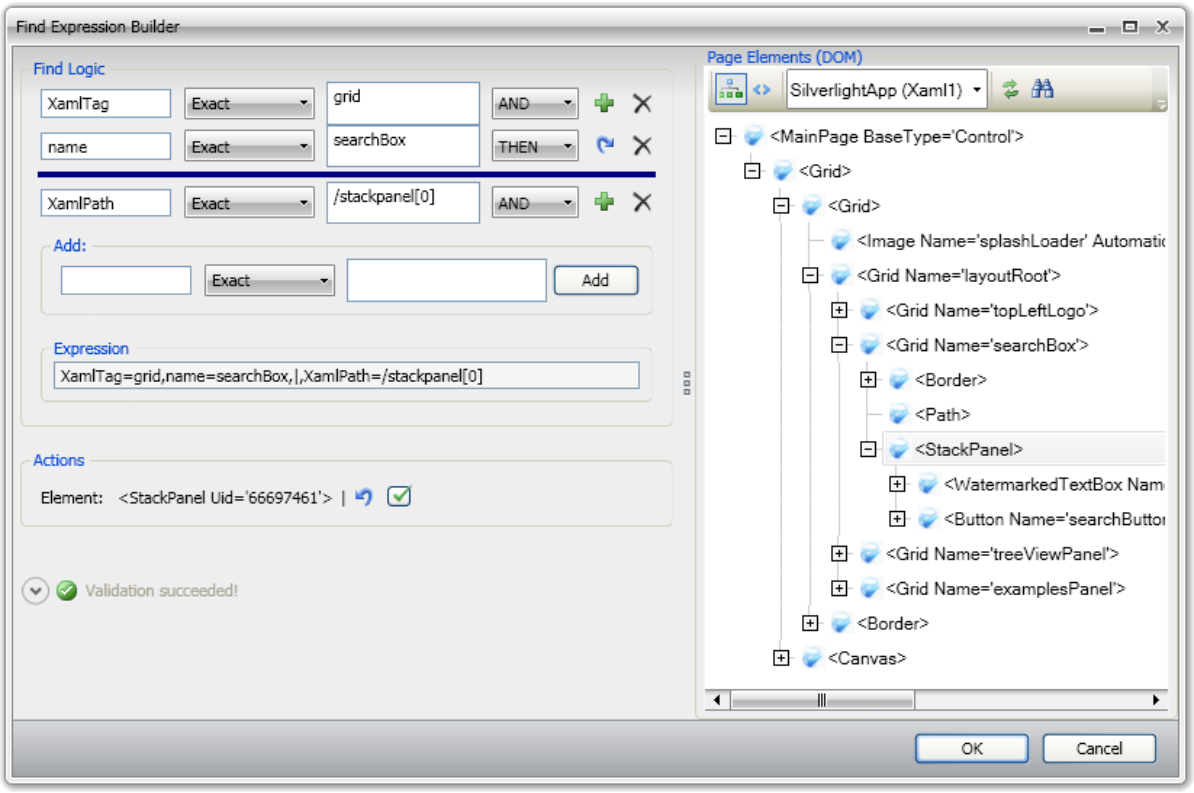


## Elements Explorer

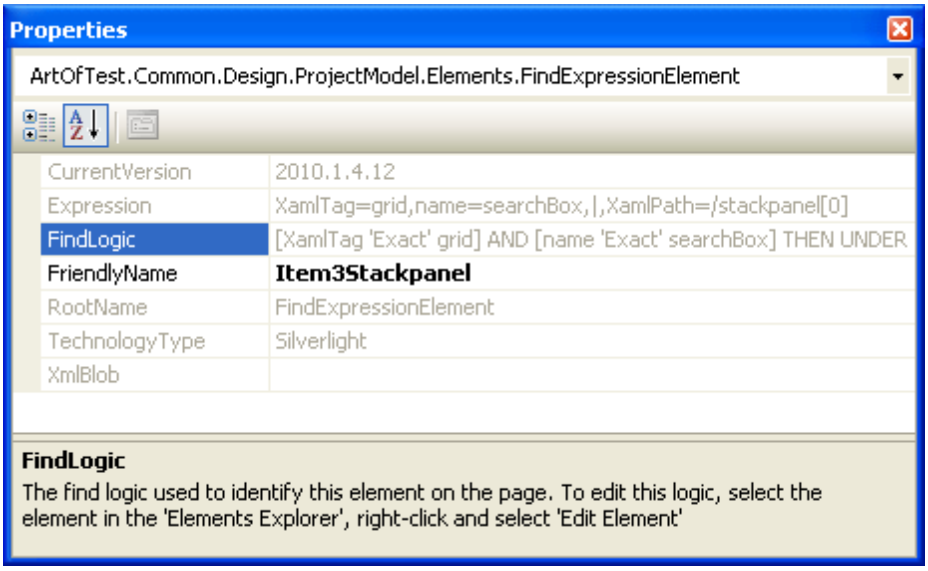
The Elements Explorer identifies and names elements to support centralized maintenance. Silverlight elements are located beneath the page and the Silverlight application object.




Right-clicking a Silverlight element in the Elements Explorer displays the context menu where you can choose to edit the element in the Find Expression Builder. Again, this is the same behavior that occurs when working with HTML elements. Notice in the screenshot below that the "Find Logic" uses Silverlight specific criteria "XamlTag" and "XamlPath" to uniquely identify the element. The "Find Logic" first locates a Silverlight "Grid" element with the name "searchBox", then locates the first "StackPanel" element within the grid.

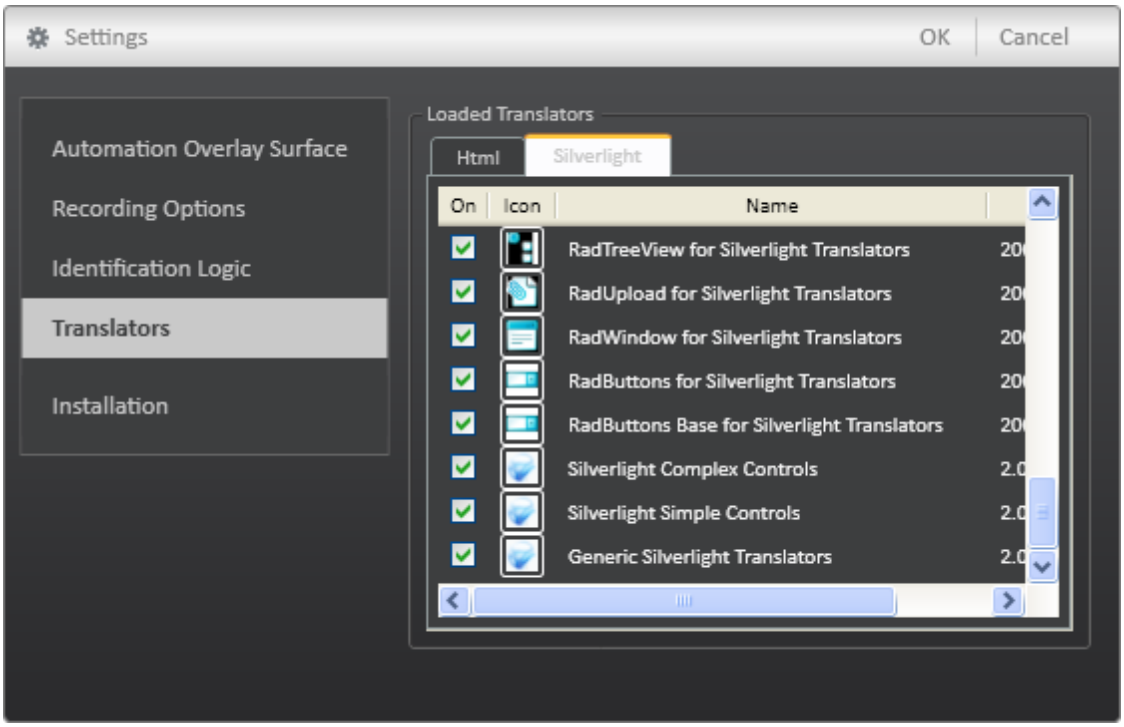


When a Silverlight element is selected in the Elements Explorer, the Properties pane displays a set of properties similar to the those for a HTML element. The differences are that the TechnologyType property is "Silverlight", not "HTML" and the find logic uses some of the Silverlight specific expressions.

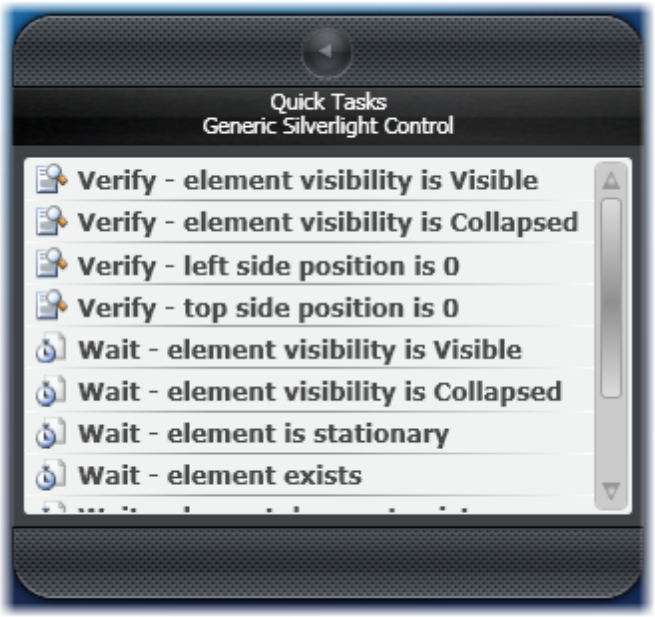


### RadControls for Silverlight Translators

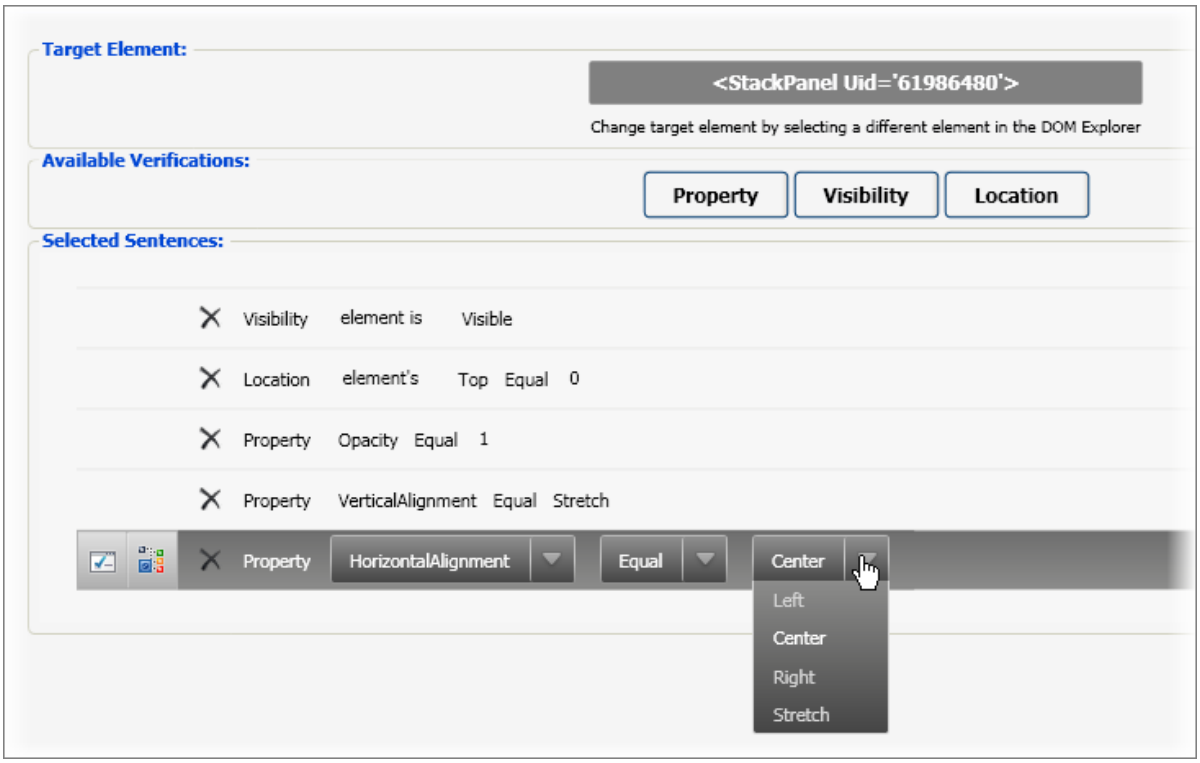
The Translators tab of the Settings dialog shows translators used to provide deep information about elements in Silverlight applications. The screenshot below shows the "Generic Silverlight Translators", the translators for groups of controls (e.g. "Silverlight Simple Controls") up to the more specific controls for individual RadControls (e.g. "RadGridView").




The Generic Silverlight translator allows you to work with the common set of properties available to all Silverlight elements. The properties that will be used most often are listed in the Quick Tasks list, as shown in the screenshot below.

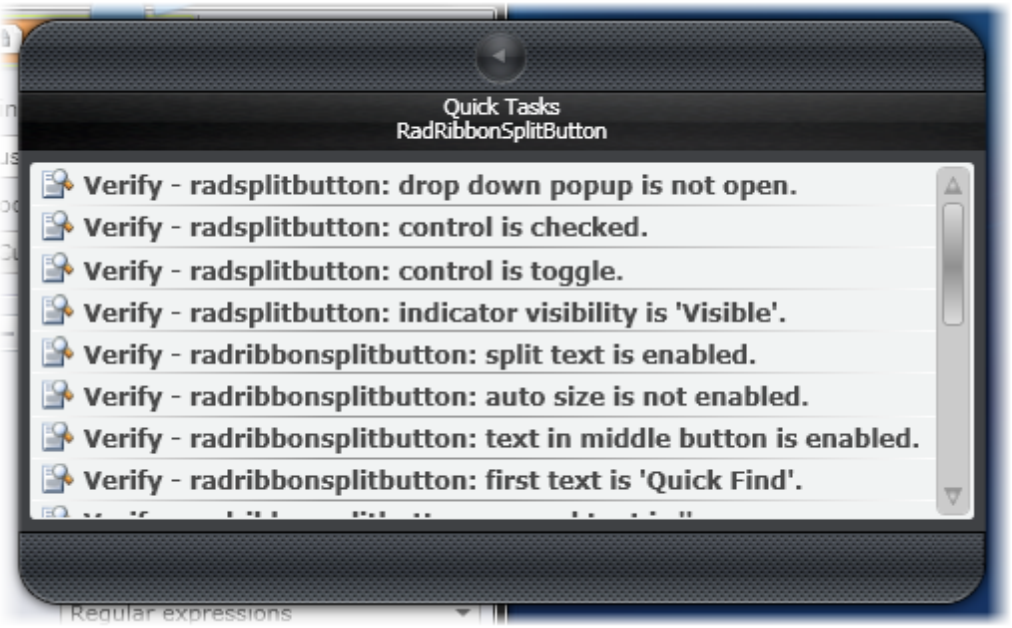



The complete set of properties can be found in the Sentence Verification Builder.The Visibility can be either "Visible" or "Collapsed". You can verify if a Silverlight element is visible or collapsed. You can verify a Silverlight element's location relative to the element's Top, Bottom, Left or Right edge. If the element has an irregular shape, the rectangle will be sized sufficient to contain the entire shape. The "Property" verification button shown in the screenshot below is a catch-all for any properties not in the other available verifications.



RadControls specific translators show up as "leaves" in the Elements Menu. The screenshot below shows the Elements Menu for a RadButton control.


These translators augment the generic translator by providing tasks specific to a given control. For example, the tasks for a "RadRibbonSplitButton" are shown in screenshot below. Because the translator has knowledge about the internals of the RadRibbonSplitButton, it knows that the button has a drop down, a "checked" state, a "split text" property and so on.



## 9.4 Cascading Combo Boxes Walk Through

To get a feel for how testing Silverlight applications differs from standard HTML web pages or AJAX, we can test against a set of cascading combo boxes. The relationship of the three combo boxes will be somewhat similar to the AJAX RadComboBox example, but will contain different content.

1) From the Visual Studio choose **File > New > Project...** This will display the "New Project" dialog.

2) Define a new test project:

a) In the "New Project" dialog, select the "Test" project type for your language (C# or VB).

b) Select the "Test Project" template.

c) Enter a descriptive name for the test project, a location path and a Solution name.

d) Click **OK** to create the new test project.

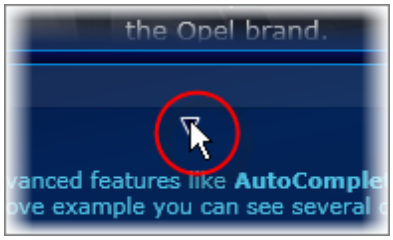
3) From the Solution Explorer, right-click the test project and select **Add > New Test...** from the context menu. This will display the "Add New Test" dialog.

a) In the "Add New Test" dialog, select the "WebAii Test" template, name the test "CascadingCombo.aii" and click **OK** to create the test.

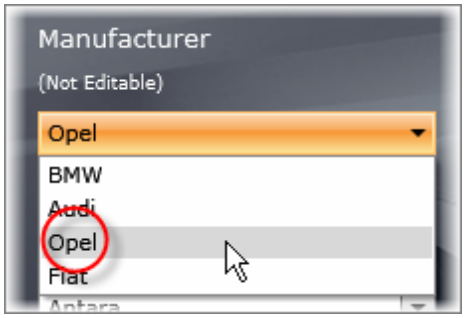
b) Locate the Record button and click it. This will display the Recording Surface and start the recording.

c) In the Recording Surface, enter "http://demos.telerik.com/silverlight/#ComboBox/FirstLook" to the browser address bar and then click the Go to Url button. This will load the RadControls for Silverlight demo application to the RadComboBox "First Look" example.

d) Click the downward pointing arrow to collapse the text description area. This will provide a little extra real estate.




4) Click the drop down arrow of the Manufacturer combo box. Select the "Opel" item from the list.

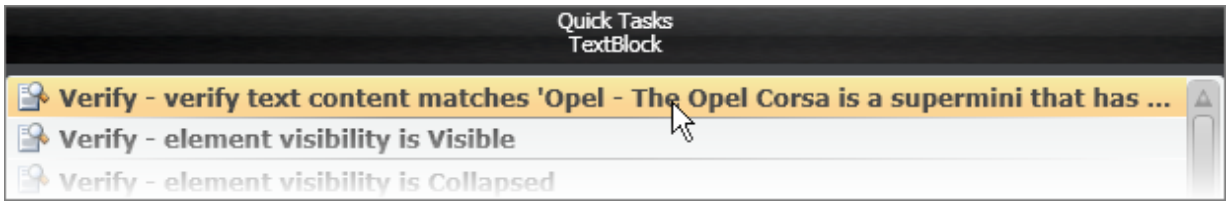



5) Press the **Tab** key.

6) Click the Highlighting button to enable it.

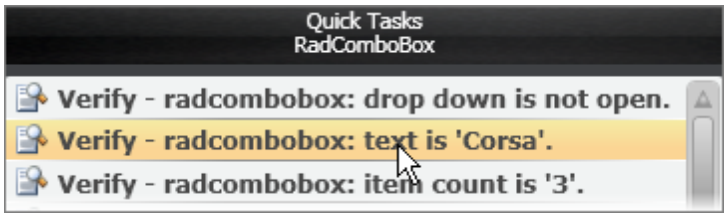
7) Hover the mouse above the car description until the Nub displays, then click the TextBlock leaf.


8) Double click the Quick Tasks item that starts "Verify - verify text content matches 'Opel - The Opel Corsa is a supermini that has...". This will add the verification as a test step.




9) Hover the mouse above the "Model" drop down until the Nub displays and click the RadComboBox leaf.


10)Double click the Quick Tasks item "Verify - radcombobox: text is 'Corsa'. This will add the verification as a test step.




11)Click the "Model" drop down arrow to open the list.

12)Click the "Model" drop down arrow a second time to close the list.

### *Notes*

Why are we opening the list? This is done to force the drop down list to load its items. The next step in this walk through will be to check the number of items in the RadComboBox. Without opening the list, the items count is zero. You can check this yourself by creating a "wait for" verification step that is valid when the number of items matches the count in the list. The test will pause at this point to wait for the number of items to match. If you manually drop down the list, the wait condition will be satisfied and the test will continue immediately.

13)Type the characters "an" into the "Model" drop down edit box. This should cause the Autocomplete to fire and automatically choose "Antara" from the list.

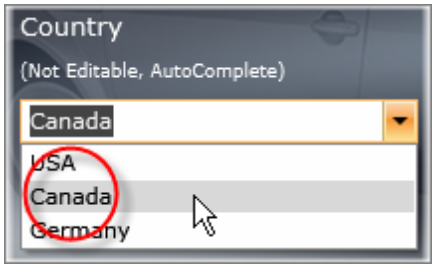
14)Hover the mouse above the "Model" drop down until the Nub displays and click the RadComboBox leaf.

15)Double click the Quick Tasks item "Verify - radcombobox: text is Antara." This will add the verification as a test step.

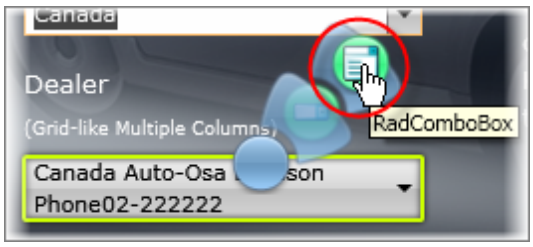
16)Hover the mouse above the car description until the Nub displays and click the TextBlock leaf.

17)Double click the Quick Tasks item that starts "Verify - verify text content matches 'Opel - The Opel Antara is a mid-size crossover...". This will add the verification as a test step.

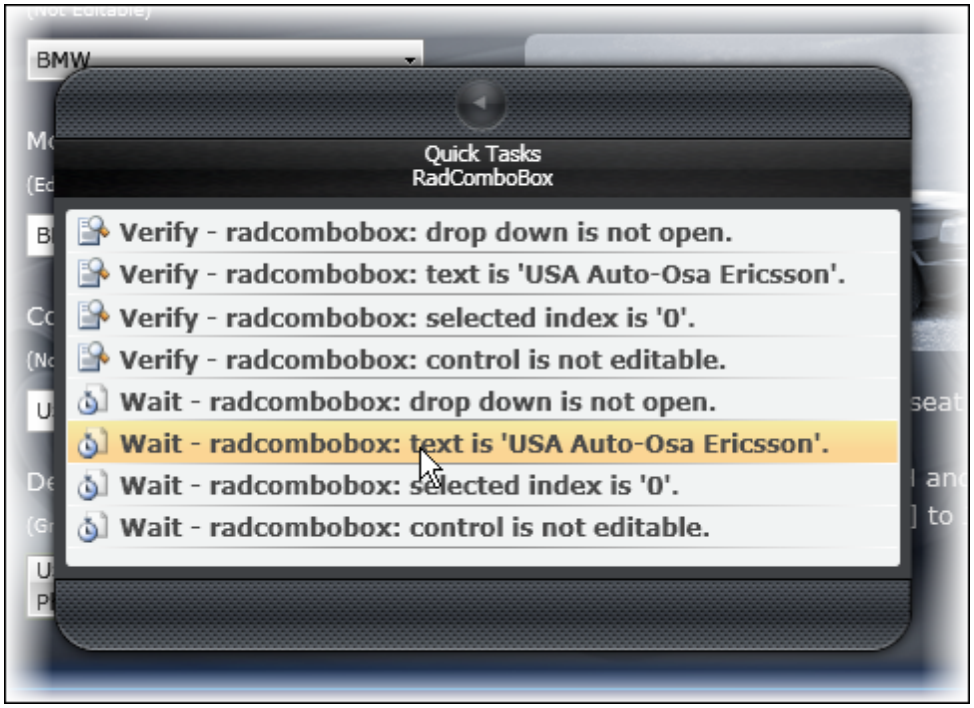
18)Click the "Country" drop down arrow to open the list. Select the "Canada" item.



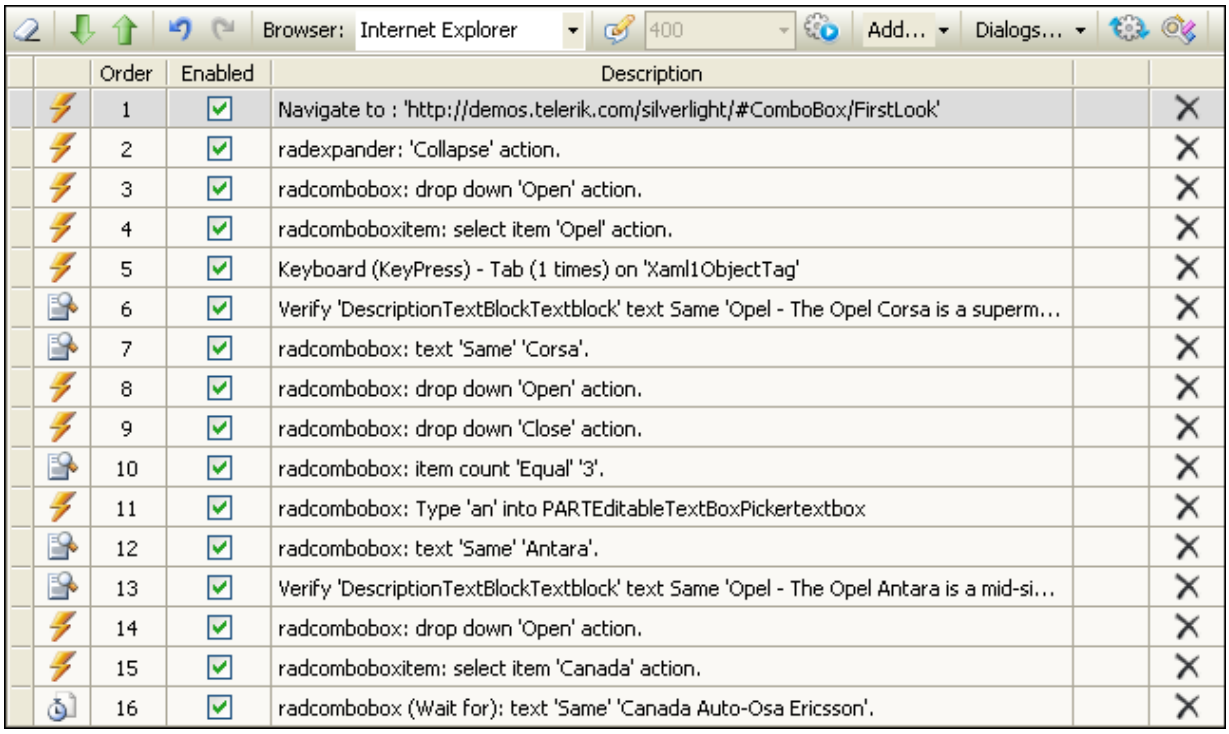

19)Hover the mouse above the "Dealer" drop down until the Nub displays and click the RadComboBox leaf.



20)Double click the Quick Tasks item "Wait - radcombobox: text is Canada Auto-Osa Ericsson." This will add the verification as a test step.

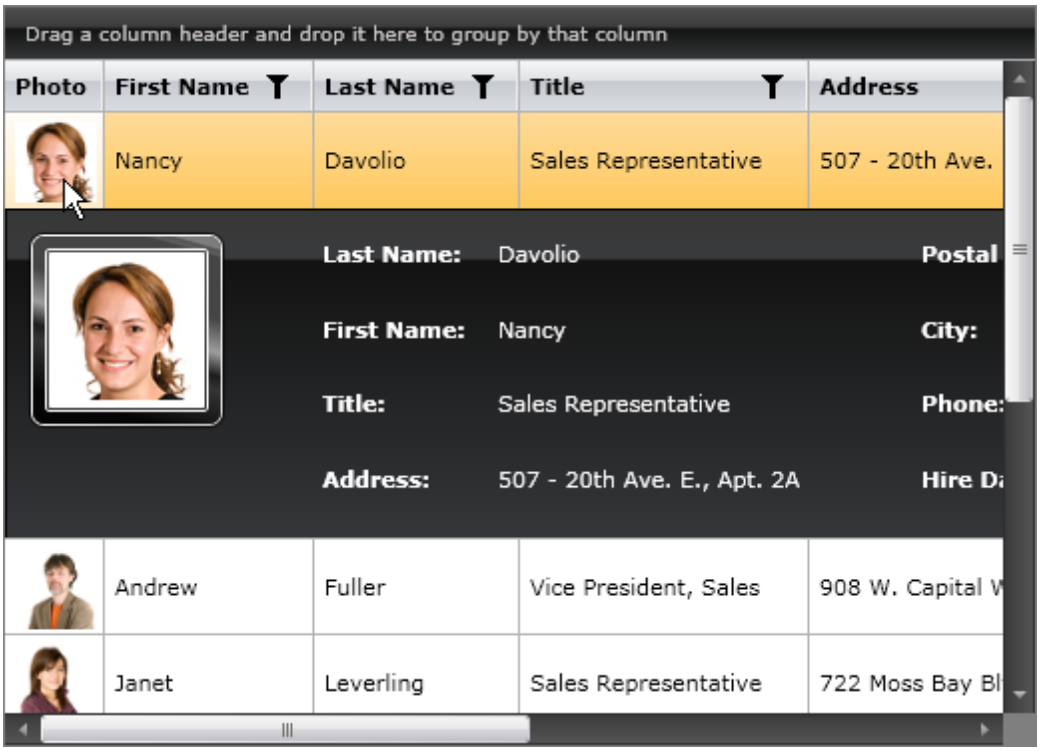



21)Click the Quick Execute button to run the test. All steps should pass.



## 9.5 RadGridView Walk Through

Like the AJAX RadGrid example, you can test multiple elements of the Silverlight RadGridView: the RadGridView as a whole, rows, cells, headers, footers and other individual Silverlight elements inside the grid (images, TextBlocks, etc.). This walk through will use the RadGridView Row Details example and test a sampling of information at different levels within the grid, i.e. the grid itself, the row, etc.




**Gotcha!**

Be aware that the RadGridView grid control is "virtualized" by default. For testing purposes this means that only the visible rows are guaranteed to exist. The reason for this is performance. The grid may need to display millions of records, so creating all rows at once would make the performance of the grid prohibitively slow. Instead, the grid loads only the rows that it needs to show at any one time. You will need to keep this in mind when designing your tests.

1) From the Visual Studio choose **File > New > Project...** This will display the "New Project" dialog.

2) Define a new test project:

a) In the "New Project" dialog, select the "Test" project type for your language (C# or VB).

b) Select the "Test Project" template.

c) Enter a descriptive name for the test project, a location path and a Solution name.

d) Click **OK** to create the new test project.

3) From the Solution Explorer, right-click the test project and select **Add > New Test...** from the context menu. This will display the "Add New Test" dialog.

a) In the "Add New Test" dialog, select the "WebAii Test" template, name the test "GridViewRows.aii" and click **OK** to create the test.

b) Locate the Record button and click it. This will display the Recording Surface and start the recording.

c) In the Recording Surface, enter "http://demos.telerik.com/silverlight/#GridView/RowDetails" to the browser address bar and then click the Go to Url button. This will load the RadControls for Silverlight demo application to the RadGridView "Row Details" example.

4) Click the Highlighting button to enable it.

5) Drag and drop the "Title" column up into the "group panel" area to group the rows by "Title" (See the Drag and Drop chapter for more detail on using the Elements Menu to control drag and drop operations).

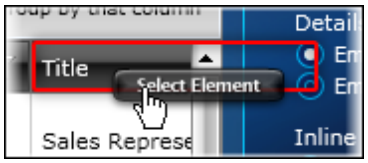
a) Hover the mouse above the "Title" column header until the Nub appears.

b) Click the Nub to open the Elements Menu. Select the Drag and Drop option from the Elements Menu.

c) To the "Do you want the drop target to be a Window or Element" prompt, click the **Element** button.

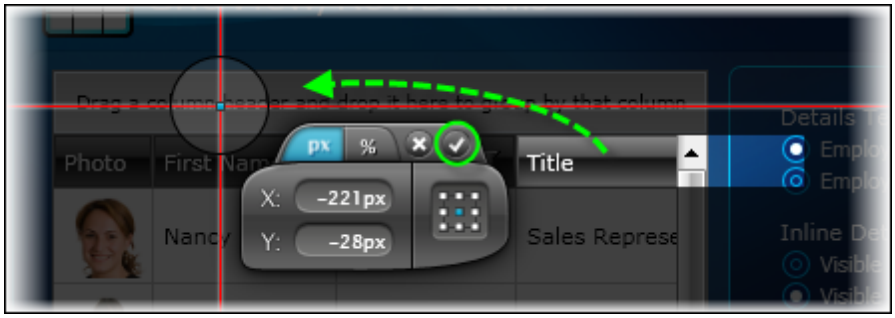
d) In the "Select an Element as the target for the drop" prompt, click the **OK** button.

e) Hover the mouse above the "Title" column until the **Select Element** button appears. Click the **Select Element** button.

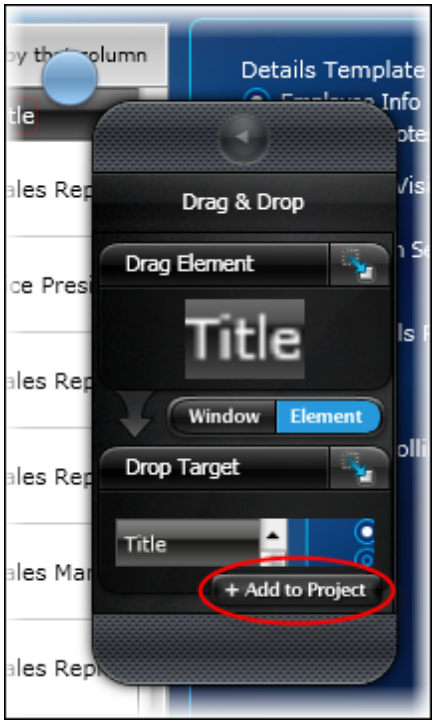



f) In the "Select the drop point by dragged..." prompt, select the **OK** button.

g) Drag the cross hairs up into the "group panel" area. Then click the checkmark button to complete the drag.

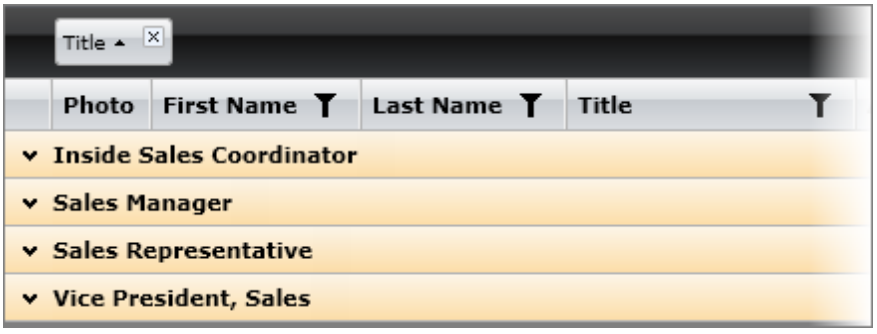


h) Click the **Add to Project** button.



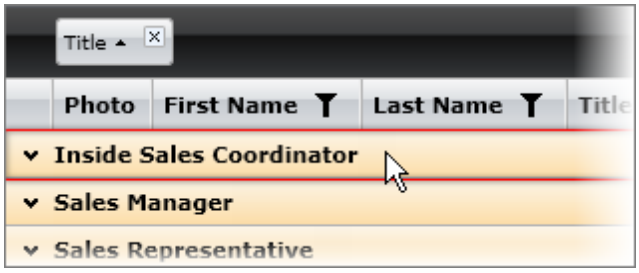

i) Pause recording momentarily, then drag the "Title" column up into the "group panel" area.


This step will cause the rows to be grouped by title and to look something like the screenshot below.

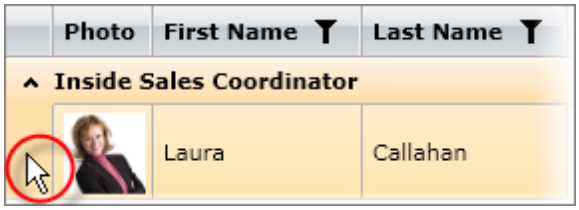



j) Turn recording back on .

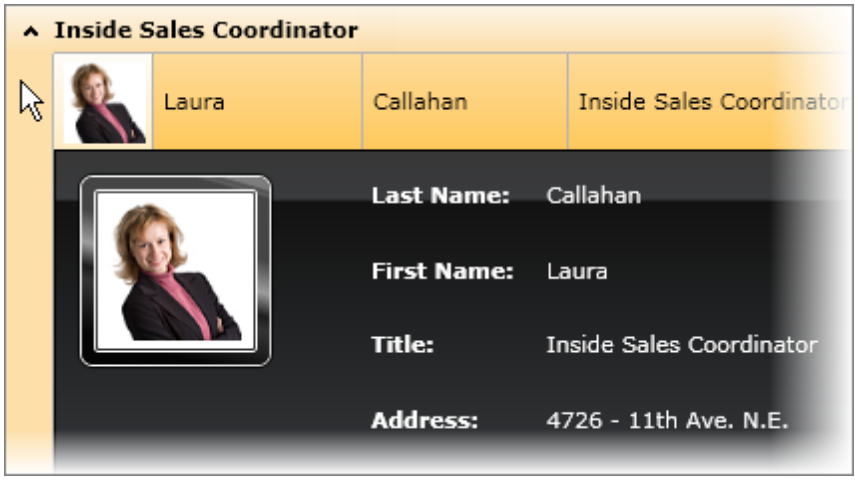
6) Click the first group row to open the group.




7) Click to the left of the row to select the entire row.



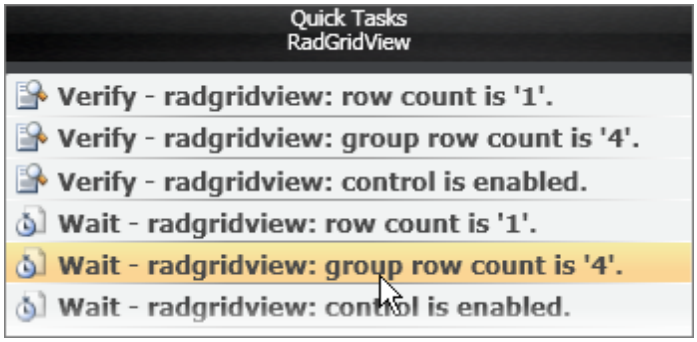

When the row is selected, an information section will expand below the row:




Now that the automation has setup the grid where the rows are grouped by title and the first row is selected and expanded, we can perform verifications against the grid, row, cell and individual Silverlight elements.

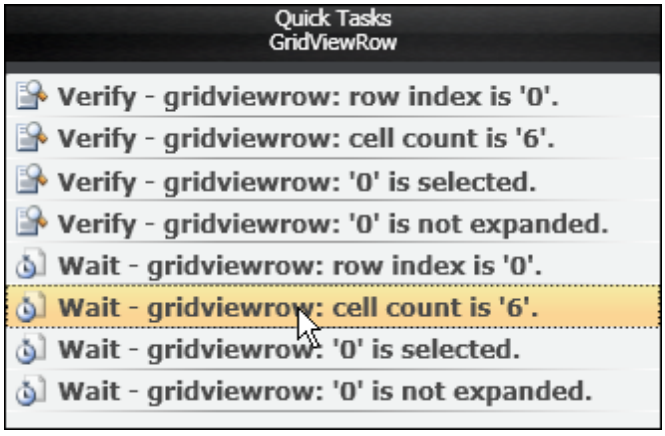
8) Hover the mouse above any part of the grid until the Nub displays and click the RadGridView leaf.

9) Double-click the Quick Tasks item "wait - radgridview: group row count is '4'." to add the verification test step.

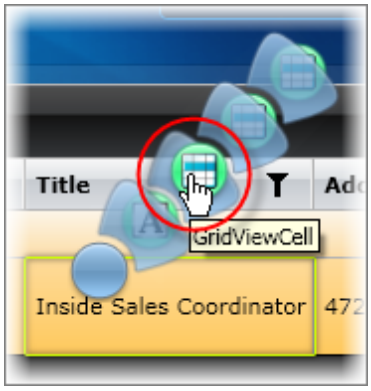



10)Hover the mouse over the first row until the Nub displays and click the GridViewRow leaf.

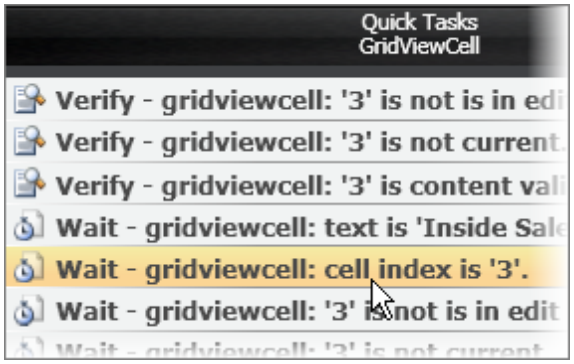

11)Double-click the Quick Tasks item "wait - gridviewrow: cell count is '6'." to add the verification test step.



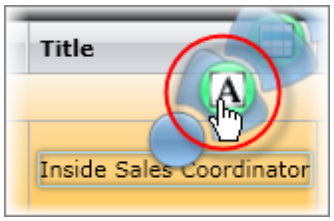
12)Hover the mouse over the "Title" cell of the first row until the Nub displays and click the GridViewCell leaf.



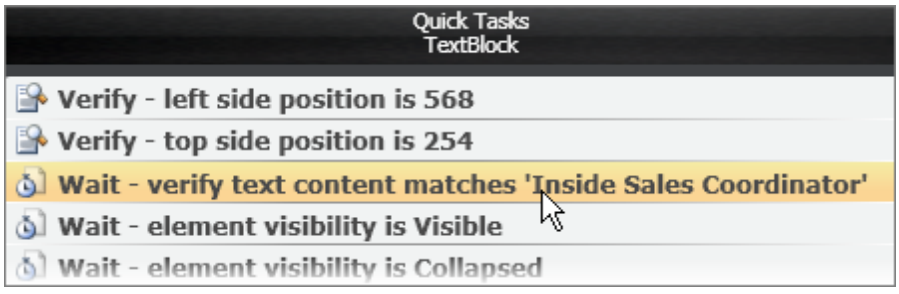

13)Double-click the Quick Tasks item "wait - gridviewcell: cell index is '3'." to add the verification test step.




14)Hover the mouse over the text in the "Title" cell of the first row until the Nub displays and click the TextBlock leaf.



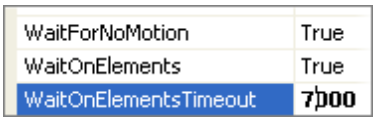

15)Double-click the Quick Tasks item "wait - verify text content matches 'Inside Sales Coordinator" to add the verification test step.



16)Press the Quick Execute button to run the test. All steps should pass.

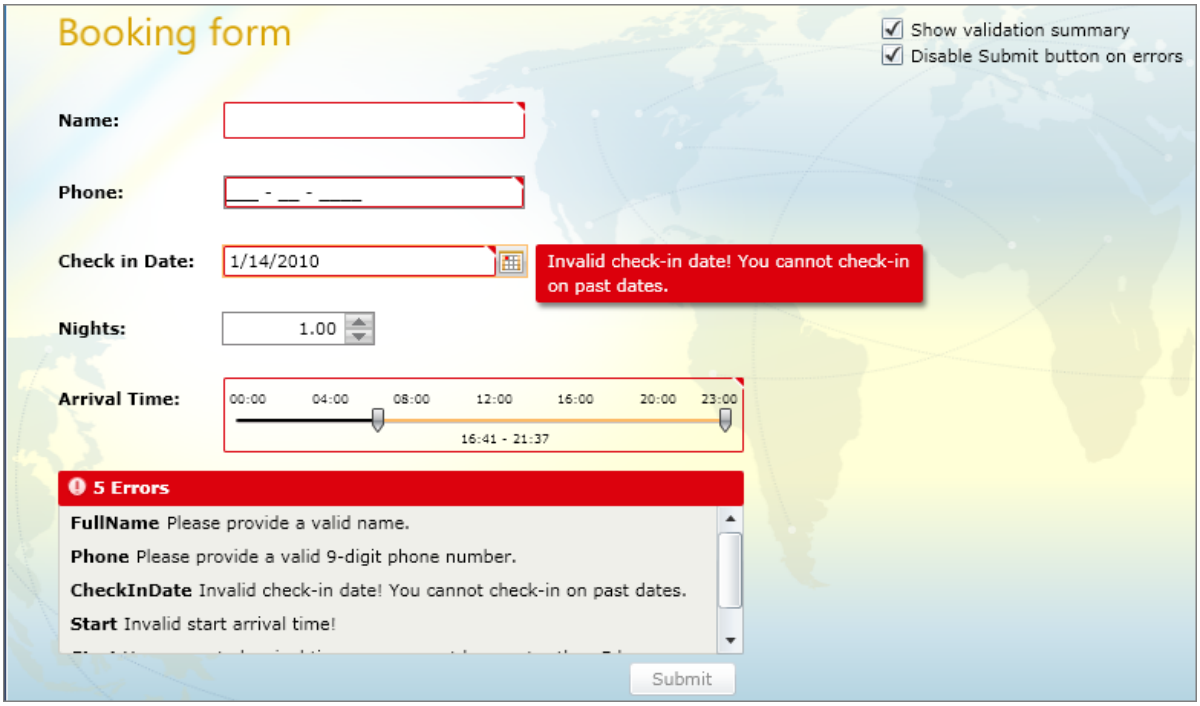
**Tip!**

Depending on your internet connection speed, computer resources and the size of the Silverlight application, loading a Silverlight application or its resources can take longer than expected and cause test steps to fail. To fix this, try bumping the "Wait on Elements" timeout property to a larger value.




## 9.6 Validation Testing Walk Through

A common quality assurance task is to verify that a series of user entries are validated correctly. This can take the form of simply pressing the "OK" button and sequentially adding entries to resolve the error messages. The walk through below uses the Telerik validation application to demonstrate automating and verifying Silverlight applications.



### 9.6.1 Test Project Setup

1) From the Visual Studio choose **File > New > Project...** This will display the "New Project" dialog.

2) Define a new test project:

   a) In the "New Project" dialog, select the "Test" project type for your language (C# or VB).

   b) Select the "Test Project" template.

   c) Enter a descriptive name for the test project, a location path and a Solution name.

   d) Click **OK** to create the new test project.

### 9.6.2 Master Test

In this part of the walk through you will create a master test that will set up the basic environment and call other tests as test steps.

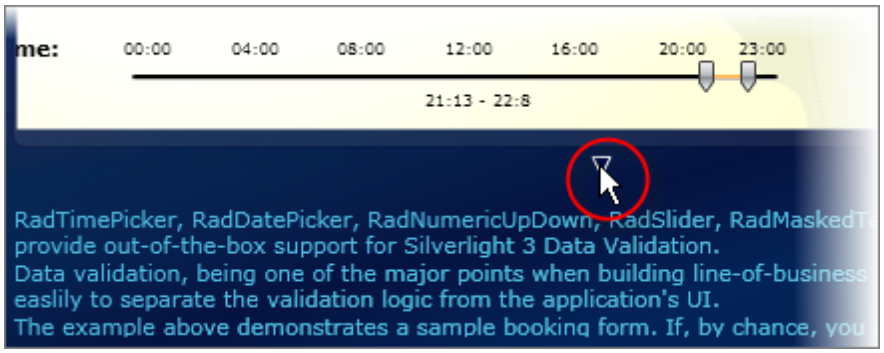
1) From the Solution Explorer, right-click the test project and select **Add > New Test...** from the context menu. This will display the "Add New Test" dialog.

2) In the "Add New Test" dialog, select the "WebAii Test" template, name the test "Validation.aii" and click **OK** to create the test.

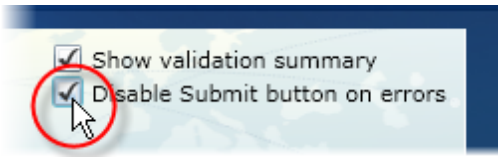
3) Locate the Record button and click it. This will display the Recording Surface.

4) In the Recording Surface, enter "http://demos.telerik.com/silverlight/#DataValidation/FirstLook" to the browser address bar and then click the Go to Url button. This will load the RadControls for Silverlight demo application to the Data Validation example.

5) Click the downward pointing arrow to collapse the text description area. This will provide a little extra real estate.

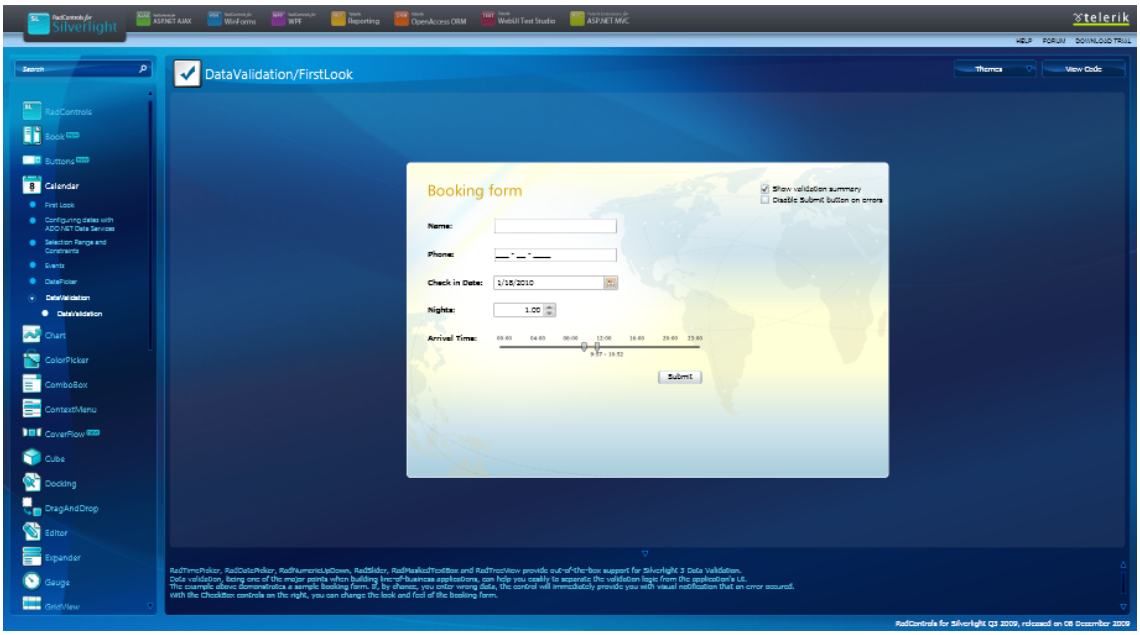



6) Locate the checkbox in the upper right corner labeled "Disable Submit button on errors" and check the box.



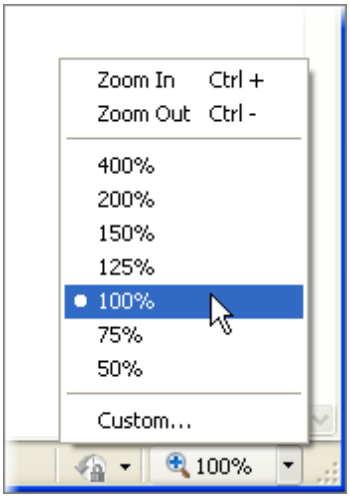
### Gotcha!

Be sure to leave the zoom level of your browser at 100%. In the current version of WebUI Test Studio, settings other than 100% cause problems in test steps that use mouse coordinates. The test steps are always recorded in the Recording Surface at 100%. The screenshot below shows the test executing in the browser with the zoom set to 50%. The mouse coordinates used to record the test at 100% zoom will not match the coordinates when the test is played back at 50% and the test steps will fail.




If you have test steps that suddenly stop working, verify the zoom level. In the Internet Explorer browser, find the Zoom control in the lower right hand corner. You can change the zoom level using the keyboard with "Ctrl +" and "Ctrl -".

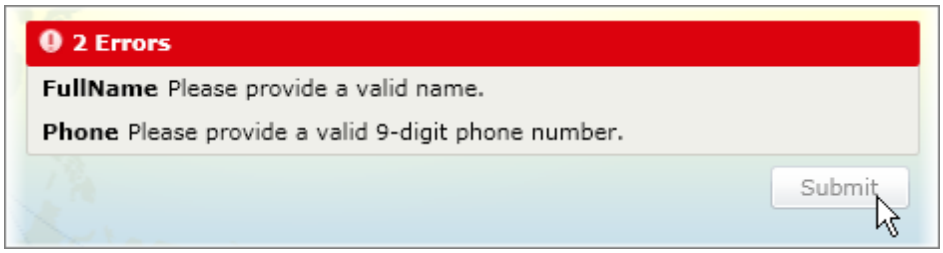
You can also use "Ctrl" plus the mouse wheel to change zoom level. This last method can occur quite accidentally and cause problems if you don't know to look for it.



### 9.6.3 Check for No Errors

Checking error conditions occurs several times throughout the test and consists of the same set of steps each time its performed. If you find yourself repeating a certain series of test steps, you should move those test steps into a separate test. This approach cuts down on the work you need to perform. You only need to write the test once and then call this test from other tests as appropriate. If you need to go back and make changes later, you only need to maintain the one test.

If you were to click the "Submit" button at this point, a validation summary would display and the "Submit" button would become disabled, as shown in the screenshot below. The two Silverlight elements are named "ValidationSummary" and "SubmitButton" respectively and can be found using the DOM Explorer. This test will check that there are no validation errors on the page. To do this, we will check that the "Submit" button is enabled and that the error validation summary is not showing.




1) From the Solution Explorer, right-click the test project and select **Add > New Test...** from the context menu. This will display the "Add New Test" dialog.

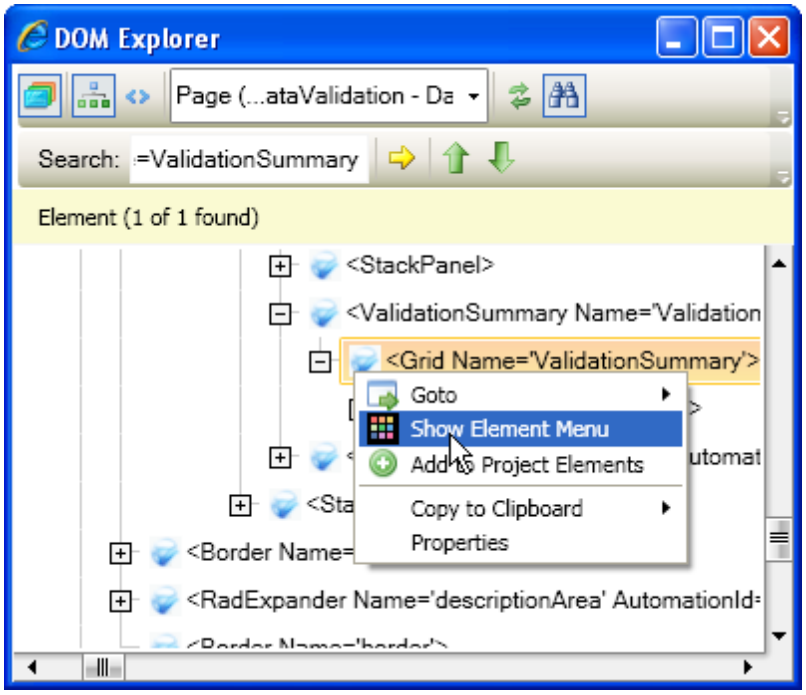
2) In the "Add New Test" dialog, select the "WebAii Test" template, name the test "Validation_CheckForNoError.aii" and click **OK** to create the test.

***Notes***

This test assumes that there are no validation errors showing on the page. If there are, click the Recording Surface Refresh button to reload the page.

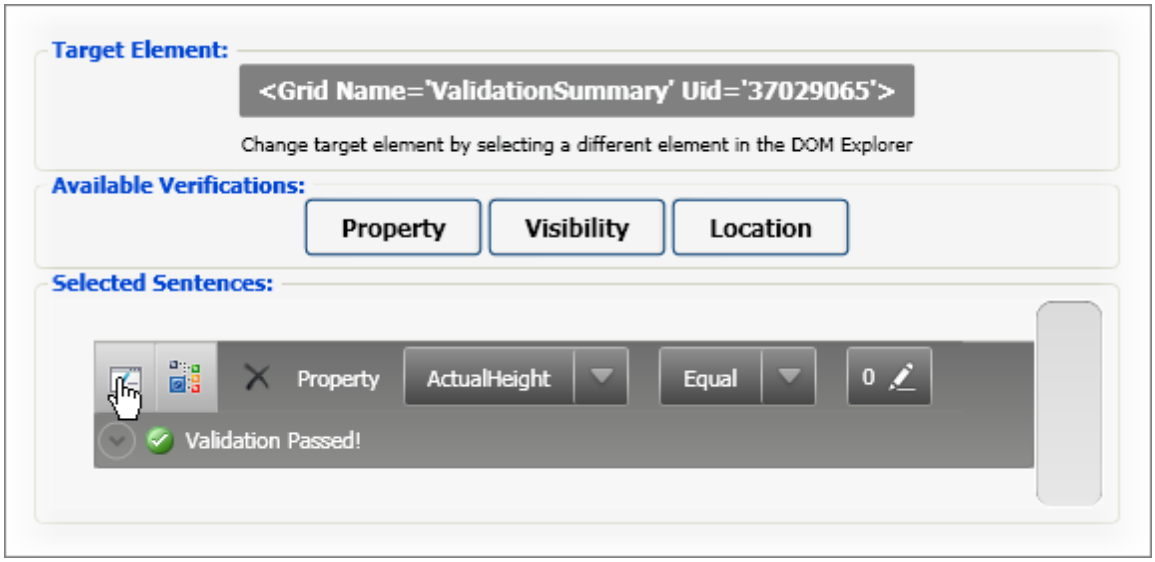
3) Navigate to the DOM Explorer, click the Search button, enter the find expression "Name=ValidationSummary" and click the "Evaluate Expression" button. This will locate a Silverlight Grid element named "ValidationSummary".

4) Still in the DOM Explorer, right-click the "ValidationSummary" and select "Show Element Menu" from the context menu. This will display the Elements Menu for the error summary.




5) In the Elements Menu, click the "Build Verification" button. This will display the Sentence Verification Builder.

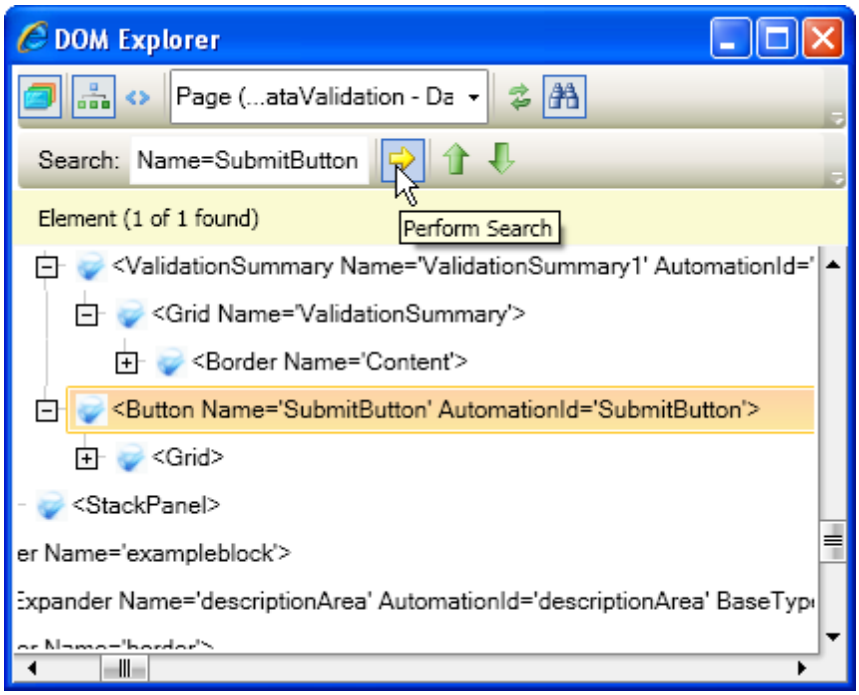
6) In the Sentence Verification Builder, click the **Property** button from the Available Verifications. Set the property to be "ActualHeight", the comparison to "Equal" and the Value to "0".




***Notes***

In this example, the "ValidationSummary" Grid element doesn't use the "Visibility" property to show and hide. Instead the element is hidden by setting its height to zero. To find this out for yourself, you will need to look at the properties of an element in the various states you are testing against. For example, you can cause the error summary element to show, then create a verification that checks Visibility, then cause the error summary to hide and check the Visibility property again. In this example, "Visibility" is "True" both when the error summary is showing *or* hidden. Reducing height to zero is a common way to hide an element, so the next step was to look at the height related properties. "ActualHeight" is zero when the validation summary is hidden and greater than zero when showing.

7) Navigate to the DOM Explorer, click the Search button, enter the find expression "Name=SubmitButton" and click the "Evaluate Expression" button. This will locate a Silverlight Grid element named "SubmitButton".



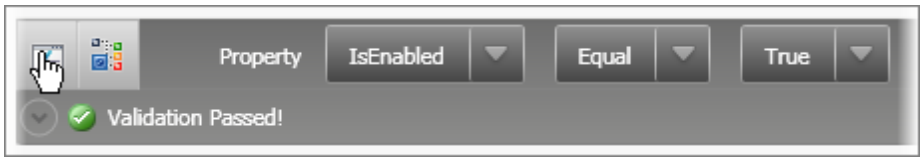

**Gotcha!**

The current version of the product does not display the Nub for disabled buttons. Instead you can use the DOM Explorer to locate the button and invoke the Elements Menu. Use the "Show the DOM Explorer" button from the main WebUI toolbar. Locate the "Submit" button using the DOM Search tool and the criteria "Name=SubmitButton". When you locate the item in the DOM Explorer, right-click the item and select "Lock on Surface" from the context menu. This will display the Elements Menu. From there you can perform any of the tasks from the Elements Menu.

8) Still in the DOM Explorer, right-click the "SubmitButton" item and select "Lock On Surface" from the context menu. This will display the Elements Menu for the "Submit" button.

9) In the Elements Menu, click the "Build Verification" button. This will display the Sentence Verification Builder.


10) In the Sentence Verification Builder, click the **Property** button from the Available Verifications. Set the property to "IsEnabled", the comparison to "Equal" and the Value to "True".




The test steps should look like those in the screenshot below:

| | Order | Enabled | Description | | |
|---|---|---|---|---|---|
| | 1 | ☑ | Verify ValidationSummaryGrid.ActualHeight 'Equal' '0' | | ✕ |
| | 2 | ☑ | Verify SubmitButtonButton.IsEnabled 'Equal' 'True' | | ✕ |

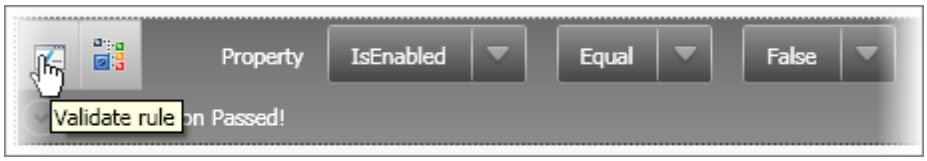
### 9.6.4 Check for Errors

This test will check if there are validation errors on the page. To do this, we will check that the "Submit" button is disabled and that the error validation summary is showing.

1) From the Solution Explorer, right-click the test project and select **Add > New Test...** from the context menu. This will display the "Add New Test" dialog.
2) In the "Add New Test" dialog, select the "WebAii Test" template, name the test "Validation_CheckForError.aii" and click **OK** to create the test.
3) Click the "Submit" button. This will display the validation summary and disable the "Submit" button.
4) Navigate to the DOM Explorer, click the Search button, enter the find expression "Name=ValidationSummary" and click the "Evaluate Expression" button. This will locate a Silverlight Grid element named "ValidationSummary".
5) Still in the DOM Explorer, right-click the "ValidationSummary" and select "Lock On Surface" from the context menu. This will display the Elements Menu for the error summary.
6) In the Elements Menu, click the "Build Verification" button. This will display the Sentence Verification Builder.
7) In the Sentence Verification Builder, click the **Property** button from the Available Verifications. Set the property to be "ActualHeight", the comparison to "Greater Than" and the Value to "0".


8) Navigate to the DOM Explorer, click the Search button, enter the find expression "Name=SubmitButton" and click the "Evaluate Expression" button. This will locate a Silverlight Grid element named "SubmitButton".
9) Still in the DOM Explorer, right-click the "SubmitButton" item and select "Lock On Surface" from the context menu. This will display the Elements Menu for the "Submit" button.
10) In the Elements Menu, click the "Build Verification" button. This will display the Sentence Verification Builder.

11) In the Sentence Verification Builder, click the **Property** button from the Available Verifications. Set the property to "IsEnabled", the comparison to "Equal" and the Value to "False".




The test steps should look like those in the screenshot below:

| Description |
| --- |
| Verify ValidationSummaryGrid.ActualHeight 'GreaterThan' '0' |
| Verify SubmitButtonButton.IsEnabled 'Equal' 'False' |

### 9.6.5 Validating for No Entry

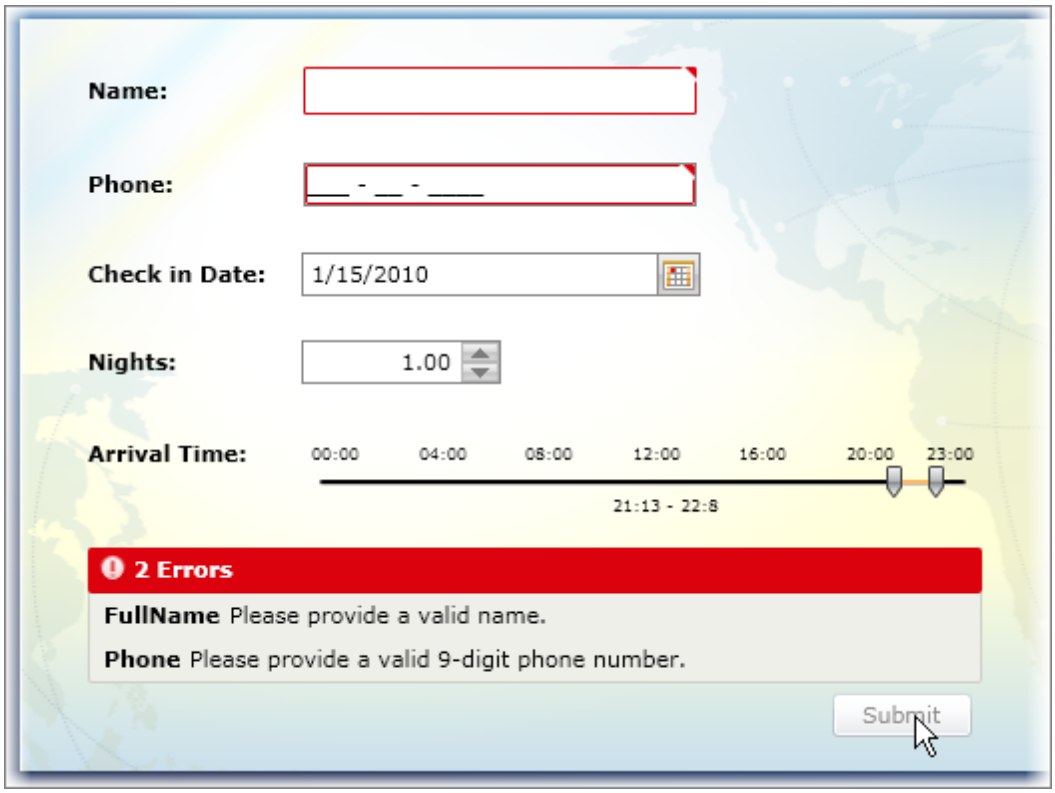
This part of the walk through will make sure that validation logic for empty fields is working correctly.

1) From the Solution Explorer, right-click the test project and select **Add > New Test...** from the context menu. This will display the "Add New Test" dialog.

2) Click the Highlighting button to enable it.

3) In the "Add New Test" dialog, select the "WebAii Test" template, name the test "Validation_NoEntry.aii" and click **OK** to create the test.

4) Click the "Submit" button. This will trigger validation errors that signal the name and phone fields are not filled in. Also notice that the "Submit" button itself is disabled. An error summary title at the bottom of the screen shows the text "2 Errors".




5) In the Steps Tab, click the **Add...** button and select "Test As Step" from the drop down list. This will open the "Select Testcase" dialog.

6) In the "Select Testcase" dialog, select "Validation_CheckForError" from the list. Click the **OK** button to add the test step.

7) Type "Bob Smith" into the "Name" text block.

8) In the Steps Tab, click the **Add...** button and select "Test As Step" from the drop down list. This will open the "Select Testcase" dialog.

9) In the "Select Testcase" dialog, select "Validation_CheckForError" from the list. Click the **OK** button to add the test step.

10) Type "123456789" into the "Phone" text block.

11)In the Steps Tab, click the **Add...** button and select "Test As Step" from the drop down list. This will open the "Select Testcase" dialog.

12)In the "Select Testcase" dialog, select "Validation_CheckForNoError" from the list. Click the **OK** button to add the test step.

The steps for this test should look like the screenshot below.

| Description |
| --- |
| Click SubmitButtonButton |
| Execute test 'Validation_CheckForError' |
| Type 'Bob Smith' into FullNameTextBoxTextbox |
| Execute test 'Validation_CheckForError' |
| Type '123456789' into PARTExtendedTextBoxExtendedtextbox |
| Execute test 'Validation_CheckForNoError' |

13)In the Solution Explorer, double-click "Validation.aii".

14)In the Steps Tab, click the **Add** button and select "Test as Step" from the drop down list. This will display the "Select testcase" dialog.

15)Select "Validation_NoEntry" from the list and click the **OK** button to create the test step.

### 9.6.6 Validate Calendar

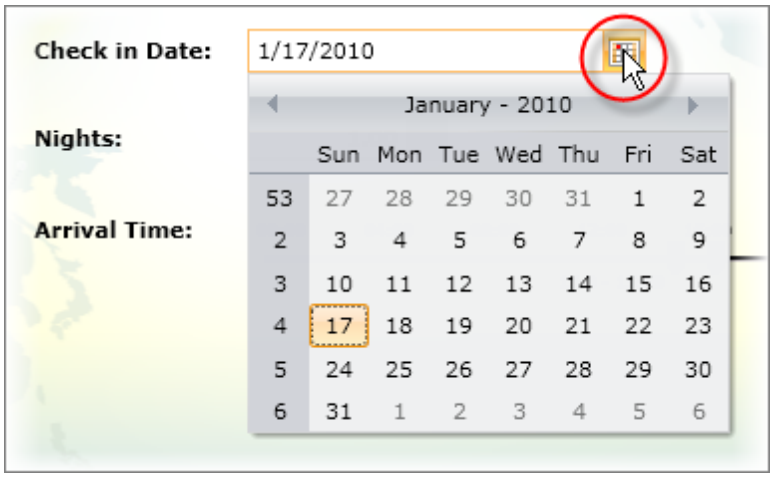
This part of the walk through will make sure that validation logic for handling the "Check In Date" is working correctly.

1) From the Solution Explorer, right-click the test project and select **Add > New Test...** from the context menu. This will display the "Add New Test" dialog.

2) In the "Add New Test" dialog, select the "WebAii Test" template, name the test "Validation_Calendar.aii" and click **OK** to create the test.

3) Locate the Record button and click it. This will display the Recording Surface.

4) Click the "Check in Date" calendar icon to open the calendar drop down.




5) Press the **Left Arrow** key. This will trigger a validation error "Invalid check-in date! You cannot check-in on past dates."

6) Click the "Check in Date" calendar icon to close the calendar.

7) In the Steps Tab, click the **Add** button and select "Test as Step" from the drop down list. This will display the "Select testcase" dialog.

8) In the "Select Testcase" dialog, select "Validation_CheckForError" from the list. Click the **OK** button to add the test step.

9) Click the "Check in Date" calendar icon to open the calendar drop down.

10)Press the **Right Arrow** key two times.

11)Click the "Check in Date" calendar icon to close the calendar.

12)In the Steps Tab, click the **Add...** button and select "Test As Step" from the drop down list. This will open the "Select Testcase" dialog.

13)In the "Select Testcase" dialog, select "Validation_CheckForNoError" from the list. Click the **OK** button to add the test step.

The test steps should look like the screenshot below:

| Description |
| --- |
| raddatepicker: calendar popup 'Open' action. |
| Press Left |
| raddatepicker: calendar popup 'Close' action. |
| Execute test 'Validation_CheckForError' |
| raddatepicker: calendar popup 'Open' action. |
| Press Right |
| Press Right |
| raddatepicker: calendar popup 'Close' action. |
| Execute test 'Validation_CheckForNoError' |

14)In the Solution Explorer, double-click "Validation.aii".

15)In the Steps Tab, click the **Add** button and select "Test as Step" from the drop down list. This will display the "Select testcase" dialog.

16)Select "Validation_Calendar" from the list and click the **OK** button to create the test step.

### 9.6.7 Validate Slider

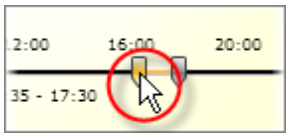
This part of the walk through will make sure that validation logic for handling the "Arrival Time" slider is working correctly. In this example we're testing that the arrival start time is later than the current time.

1) From the Solution Explorer, right-click the test project and select **Add > New Test...** from the context menu. This will display the "Add New Test" dialog.

2) In the "Add New Test" dialog, select the "WebAii Test" template, name the test "Validation_Slider.aii" and click **OK** to create the test.

3) Locate the Record button and click it. This will display the Recording Surface.

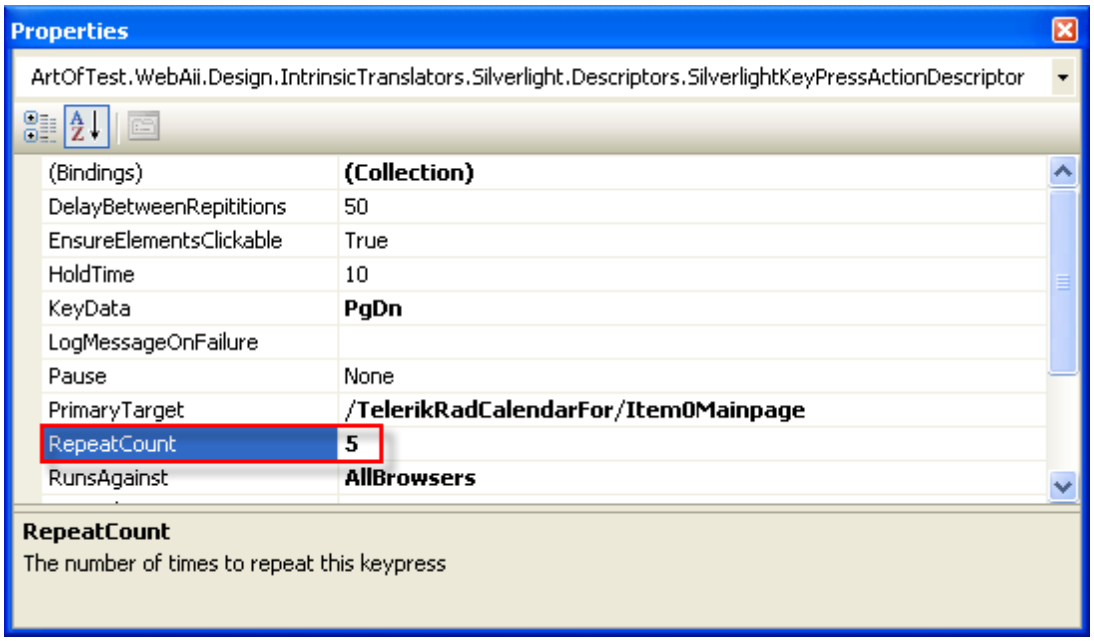
4) Click the left slider thumb to select it. This will add a test step and also create a new item in the Elements Explorer called "PathPath".




5) In the Elements Explorer, make sure that "PathPath" is selected. In the Properties pane, change the **FriendlyName** property to "StartThumb".

6) Press the **Page Down** key five times. This will generate a series of "Press Next" and "KeyPress" test steps. In the Steps Tab, delete all but the first "Press Next" step.

7) In the Properties pane, set the **RepeatCount** property to "5".




8) In the Steps Tab, click the **Add** button and select "Test as Step" from the drop down list. This will display the "Select testcase" dialog.

9) In the "Select Testcase" dialog, select "Validation_CheckForError" from the list. Click the **OK** button to add the test step.

10)Again, click the left slider thumb.

11)Press the **Page Up** key five times. This will generate a series of "Press Up" and "KeyPress" test steps. In the Steps Tab, delete all but the first "Press Up" step.

12)In the Properties pane, set the **RepeatCount** property to "5".

13)In the Steps Tab, click the **Add** button and select "Test as Step" from the drop down list. This will display the "Select testcase" dialog.

14)In the "Select Testcase" dialog, select "Validation_CheckForNoError" from the list. Click the **OK** button to add the test step.

The test steps should look like the screenshot below:

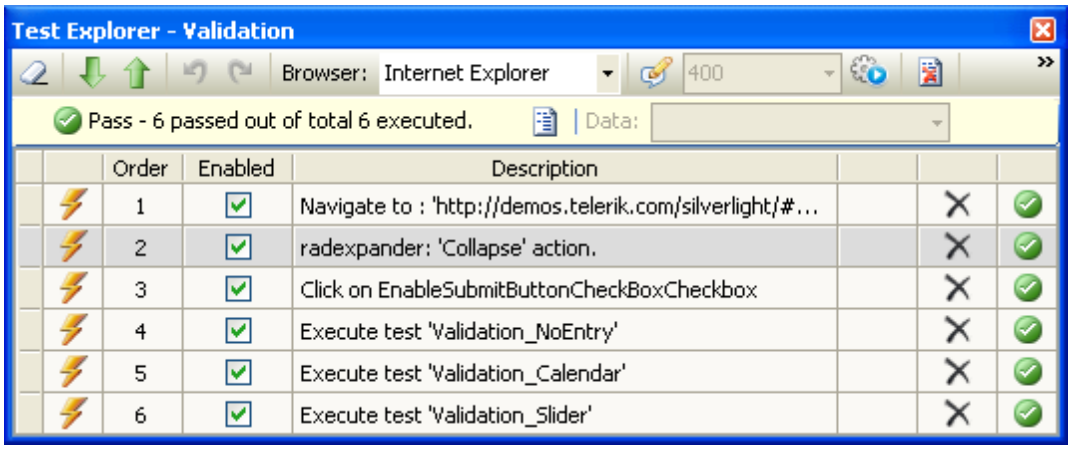
| Description |
|---|
| Click on StartThumb |
| Press Next (5 times) |
| Execute test 'Validation_CheckForError' |
| Click on StartThumb |
| Press PageUp (5 times) |
| Execute test 'Validation_CheckForNoError' |

15)In the Solution Explorer, double-click "Validation.aii".

16)In the Steps Tab, click the **Add** button and select "Test as Step" from the drop down list. This will display the "Select testcase" dialog.

17)Select "Validation_Slider" from the list and click the **OK** button to create the test step.

18)Click the Quick Execute button to run the test. All test steps should pass.

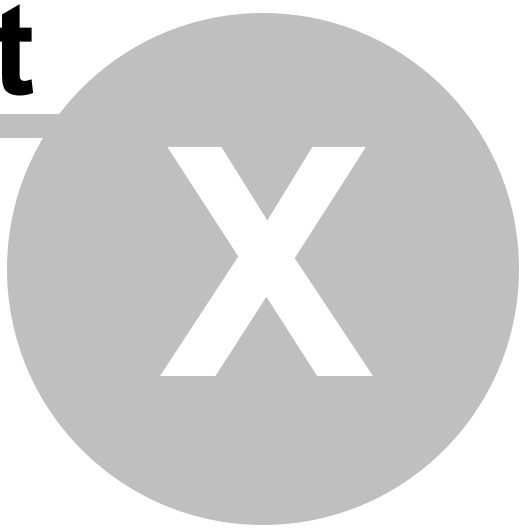


## 9.7 Wrap Up

In this chapter we talked briefly about what Silverlight is, the unique issues that arise when testing Silverlight applications and how WebUI Test Studio addresses Silverlight specific situations. You reviewed differences in the Visual Studio environment when testing a Silverlight application, paying special attention to the 3D Viewer, DOM Explorer and Elements Explorer.

You tested cascading combo boxes and RadGridView. You also tested an entry form that performs validation and has several Silverlight controls including standard TextBlock, RadSlider, RadCalendar and RadMaskedTextBox.

# Part




Handling Dialogs

# 10 Handling Dialogs

## 10.1 Objectives

In this chapter you will learn how to respond to pop-up dialogs that occur in your tests. You will learn how to handle "Win32" type dialogs including the specialized handlers for Alert, Logon, File Upload, Download and the generic "Win32" handler. You will also learn how to control HTML popup dialogs.

**Find the projects for this chapter at...**

\Courseware\Projects\<CS|VB>\Dialogs\Dialogs.sln

## 10.2 Overview

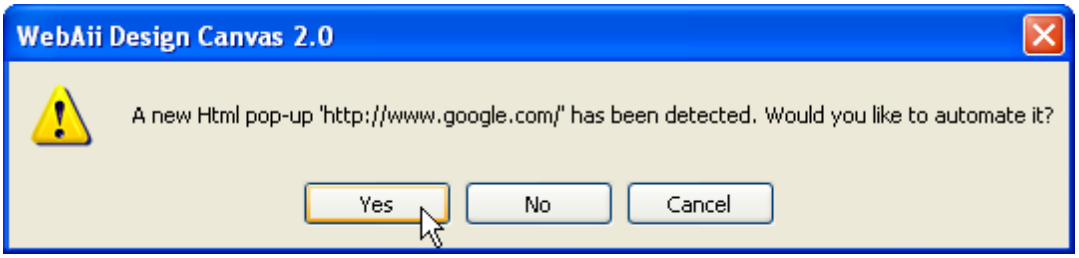
Not all test steps play out directly inside the browser. Web pages can display popup dialog windows in the form of alerts, confirmations and other web browser instances. WebUI Test Studio allows you to track and respond to dialog windows. For example, if a confirmation dialog asks if the user wants to save changes, we can automatically close the dialog as a test step by responding with "OK" or "Cancel". WebUI Test Studio can handle both HTML popup and "Win32" dialogs. HTML popups are new browser windows that are used to collect information from the user. A "Win32" dialog is not a browser window, but a dialog displayed by the Windows operating system.

***Notes***

Review the "Automation Design Canvas User Guide" chapter "Configuring Your Browser for Test Automation" for detailed instructions on how to configure your browser to prevent interference from pop-up blockers and security settings. The information covers multiple browser types and operating systems. You can find a link to this PDF at http://www.telerik.com/support/documentation-and-tutorials.aspx. You can also find the information at http://www.artoftest.com/support/webaii/topicsindex.aspx in the "Getting Started > Configuring Your Browser" section.

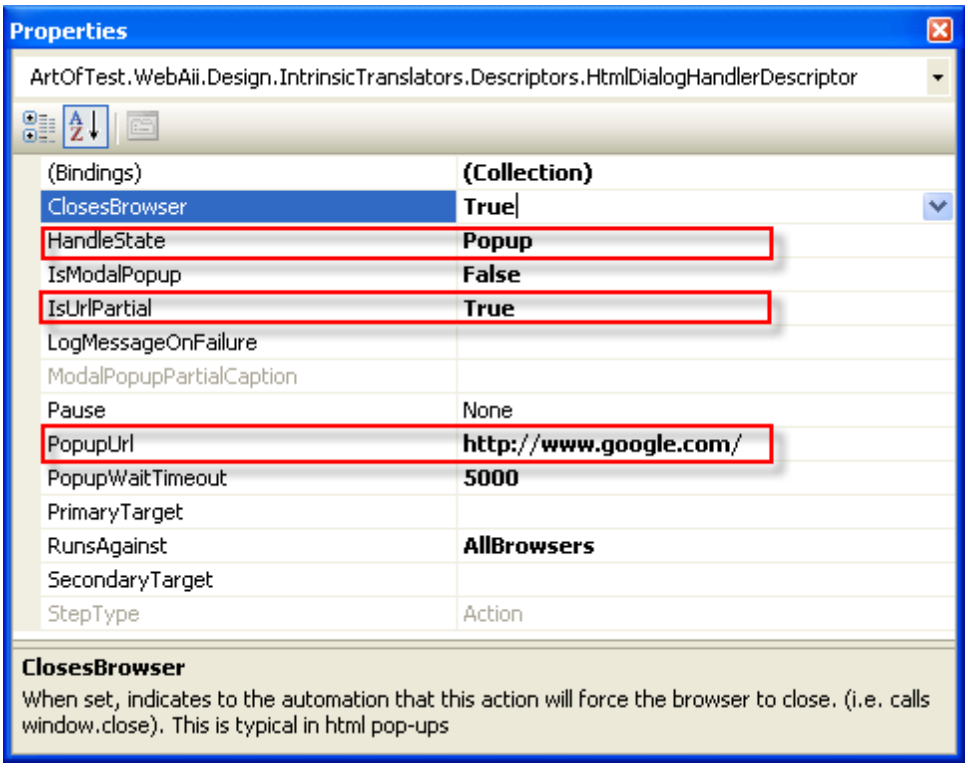
## 10.3 HTML Popups

HTML popups are detected by WebUI Test Studio automatically. When an HTML popup is about to appear, WebUI Test Studio allows you to automate the popup.

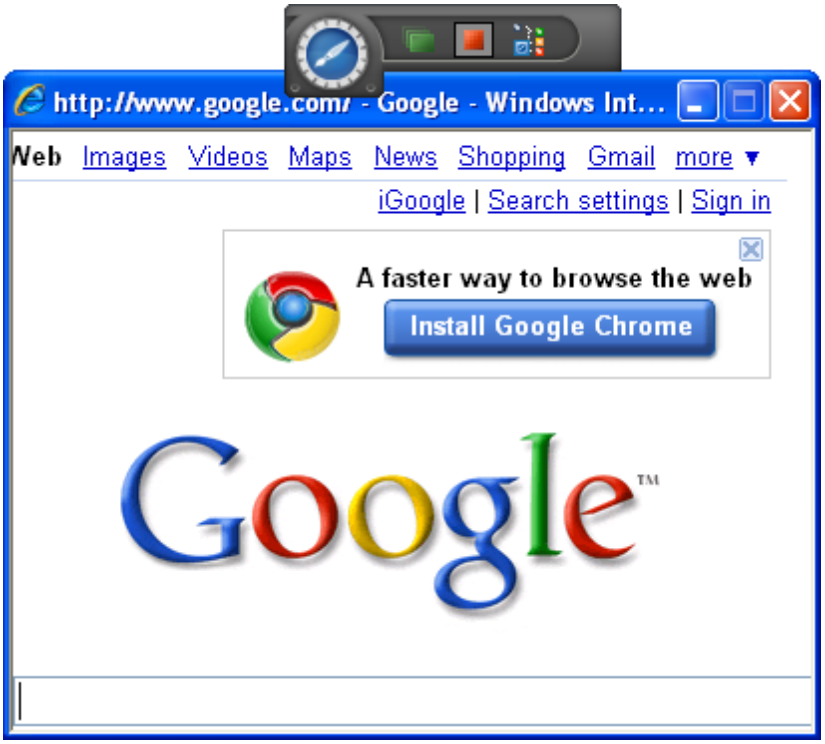



If you click the "Yes" button in the prompt, WebUI Test Studio will display the popup window along with a toolbar that includes buttons for turning on highlighting, recording and the DOM Explorer. WebUI Test Studio automatically includes test steps for connecting to, recording steps inside the new browser window and finally closing the Html popup. The sequence is typically like the test steps shown in the screenshot below. First the pop-up window is connected, then you can perform any arbitrary actions inside the new browser window and finally, the pop-up window is closed.

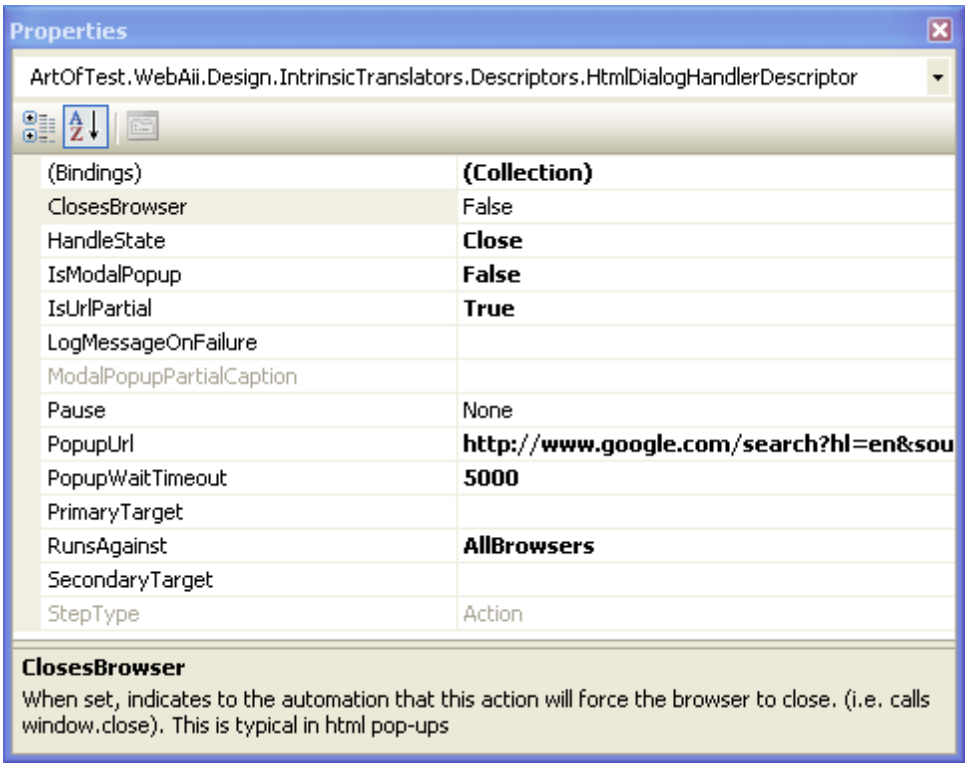
The properties for the "connecting" test step are shown in the screenshot below. The **PopupUrl** should match the address for the popup. If **IsUrlPartial** is true you can get away with only writing in part of the PopupUrl. For example, if IsUrlPartial is "True" and PopupUrl is "www.google.com", the actual url could be "www.google.com/maps". The **HandleState** property indicates that we're handling the "Popup", i.e. the state of the popup as it connects.



The screenshot below shows an HTML popup window and associated toolbar. Once the pop-up is open, you can highlight, add verifications and record test steps.

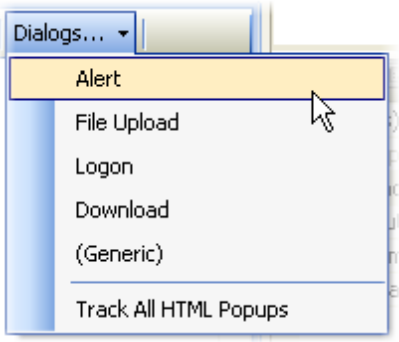


When you close the pop-up, a last test step that handles the "Close" state is added.



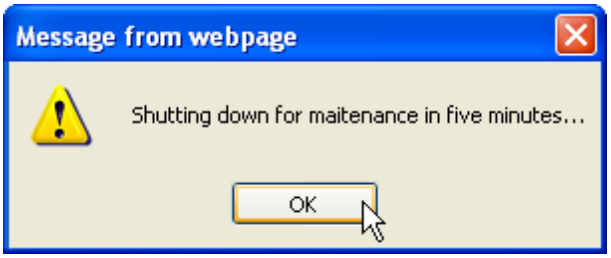

## 10.4 Win32 Dialogs

You can respond to a number of common Win32 dialogs using the Recording Surface "Dialogs" drop down list. Choosing an item from this list creates a test step that handles a particular type of dialog. The screenshot below shows the possible dialog types that can handled as test steps.

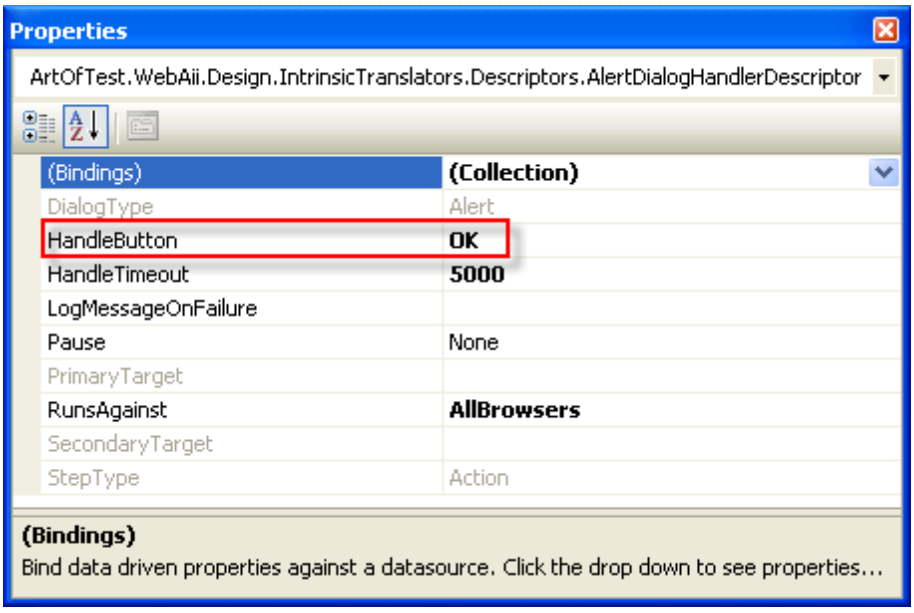


### 10.4.1 Alert

An "Alert" dialog displays a message and a single "OK" button.



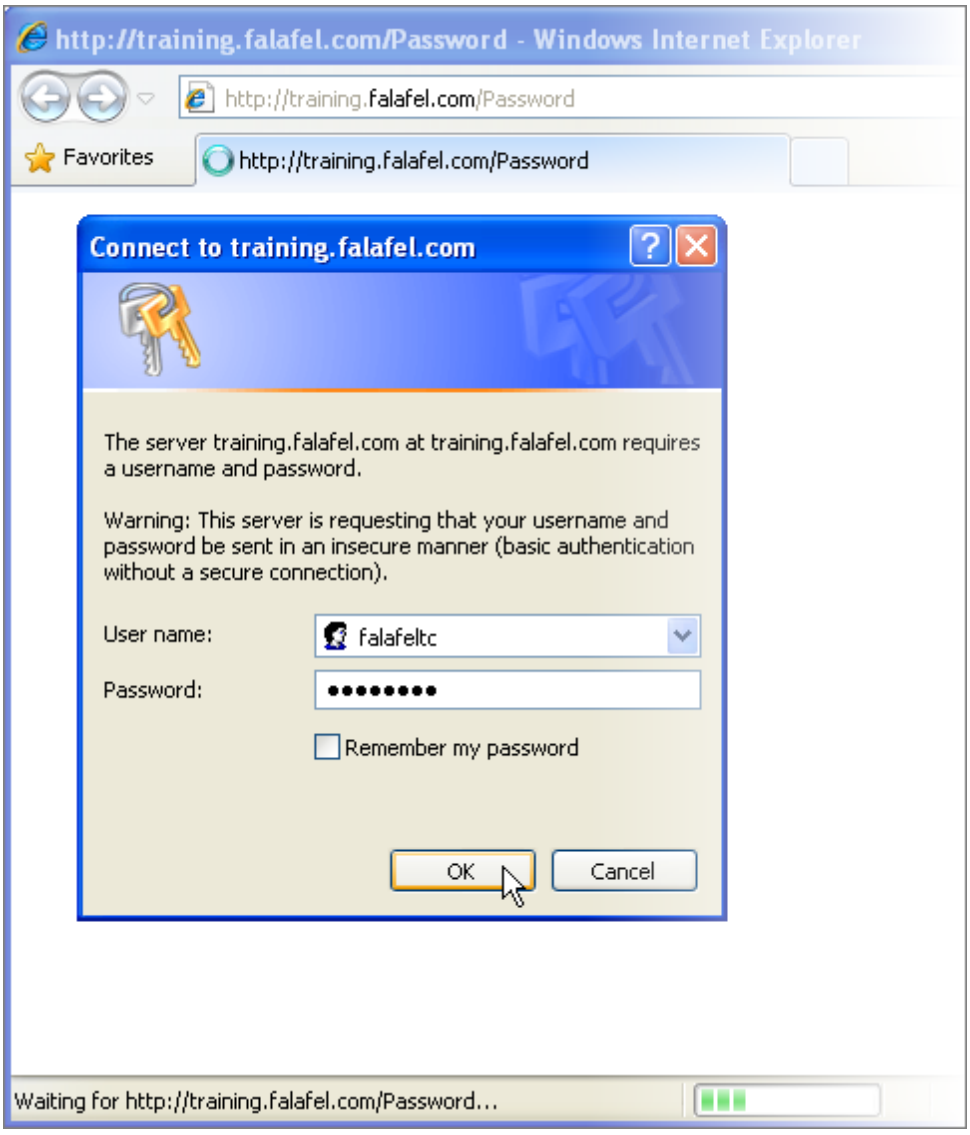

When you create a test step to handle the Alert dialog, the Properties pane includes a **HandleButton** property that should be set to "OK" or "CANCEL".



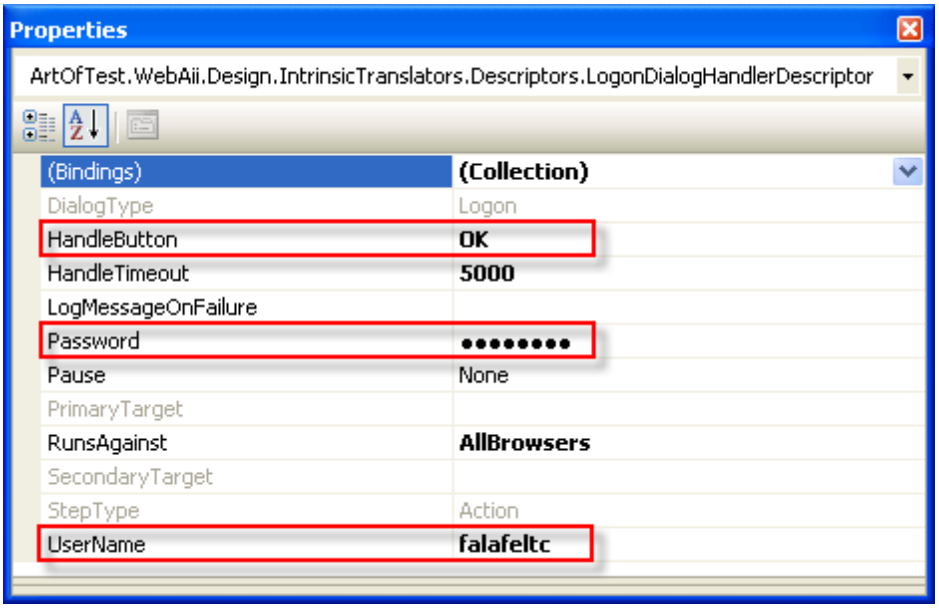

A typical set of test steps are shown in the screenshot below where some action triggers the alert to display (in this case the "Click 'PopupNotifyLink'" triggers a popup), followed by the alert handling test step.

### 10.4.2 Logon

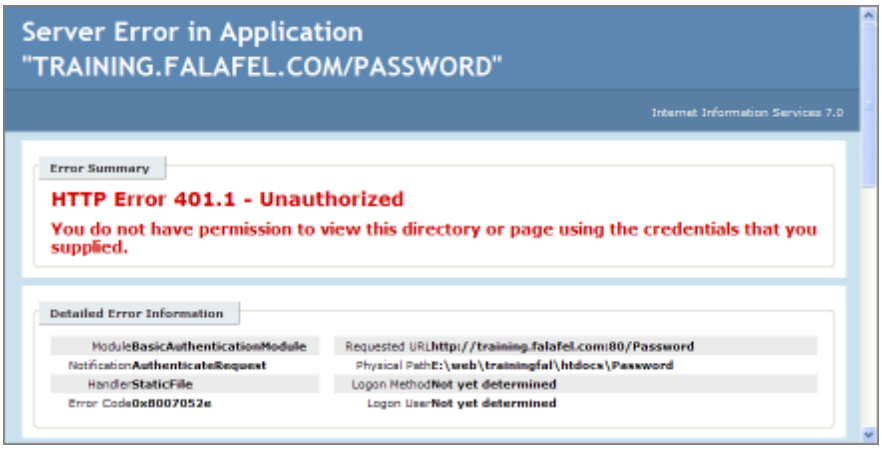
Some web pages may require a user name and password to gain access. A logon dialog displays *before* the page itself displays. The logon handler fills in the user name and password and then clicks the OK or Cancel button to close the dialog.



When you create a test step to handle a logon dialog, the Properties pane includes a **HandleButton** property that can only be set to "OK", "CANCEL" or "CLOSE". Set the **UserName** and **Password** properties to a valid logon values.

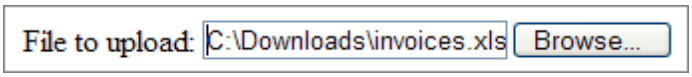



If the UserName or Password property values are incorrect, the logon dialog may display multiple times and the handler will attempt to fill in the values each time. The behavior of the page depends on how security is configured for a particular web site. For example, a typical web site may allow three logon tries before displaying an error message page like the one shown in the screenshot below:

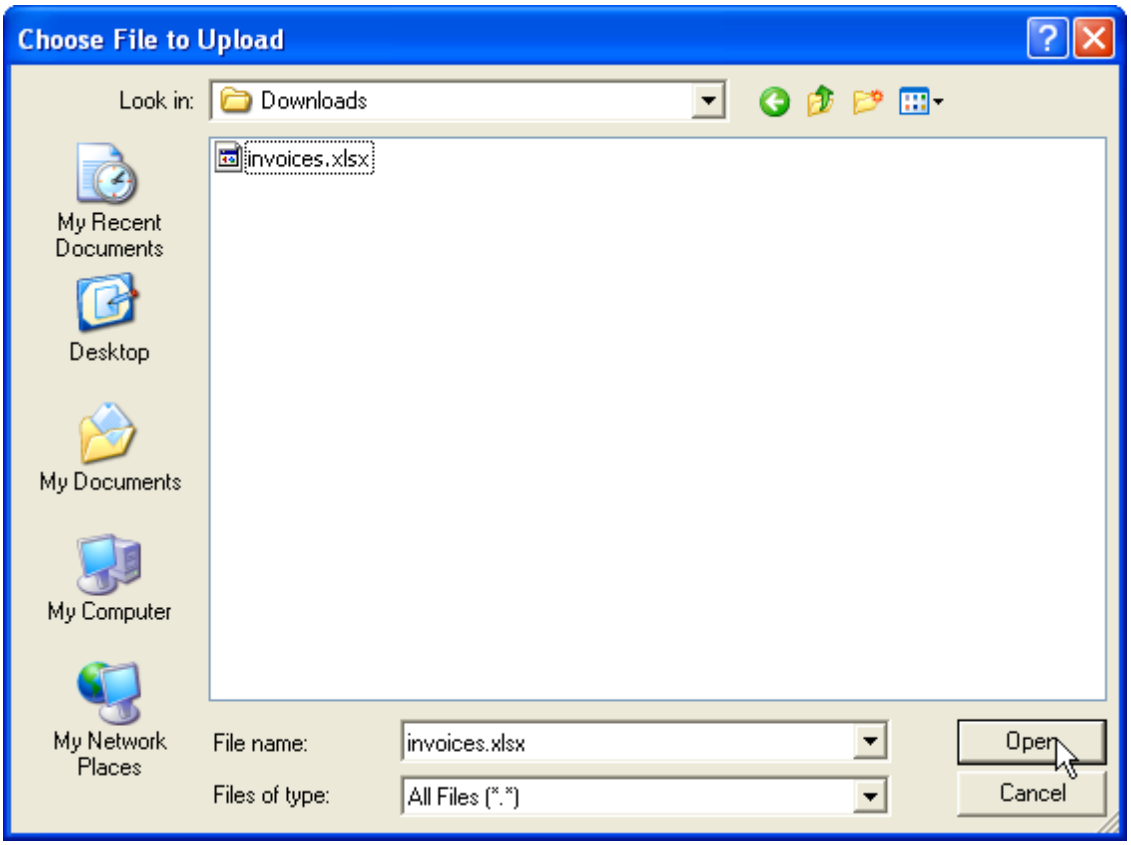


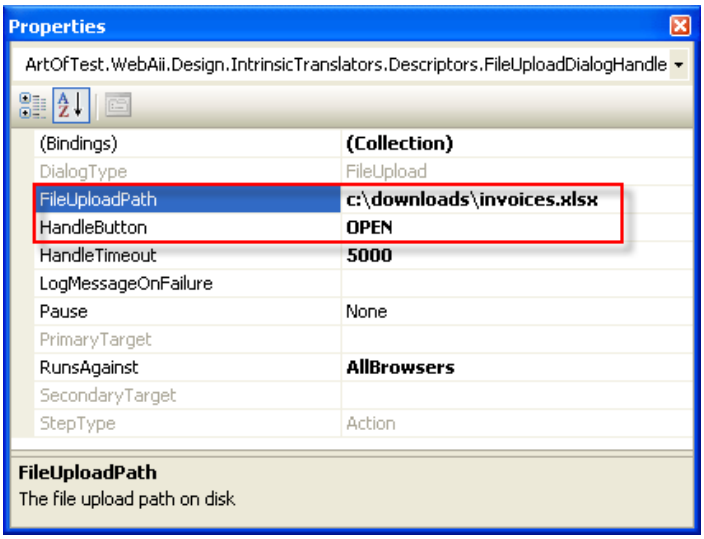
### 10.4.3 File Upload

When files need to be uploaded from the user's desktop to the server, an "Upload" control similar to the one shown in the screenshot allows the user to browse and select a valid file path.




When the user clicks the "Browse" button, a File Upload dialog displays.

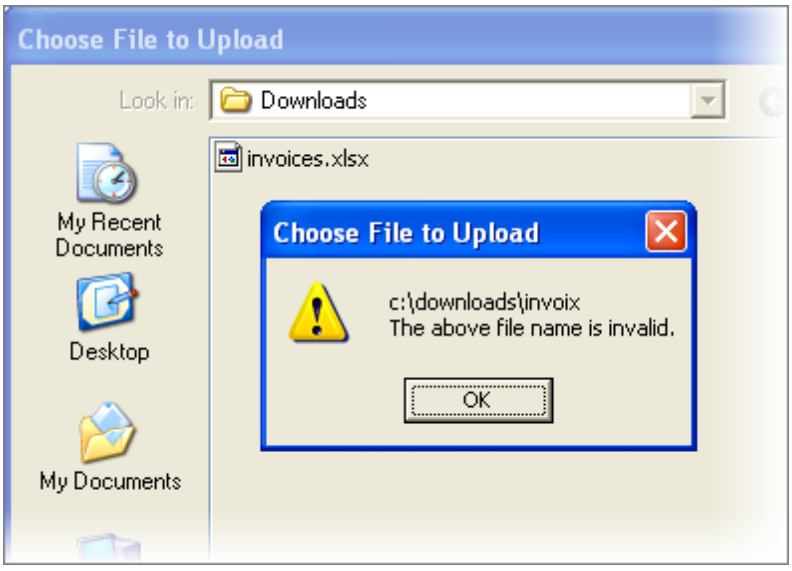





It's this dialog that gets handled by the File Upload dialog handler test step. When you create a test step to handle the File Upload dialog, the Properties pane includes a **HandleButton** property that can only be set to "OPEN", "CANCEL" or "CLOSE". Set the **FileUploadPath** property to the path of an existing file.

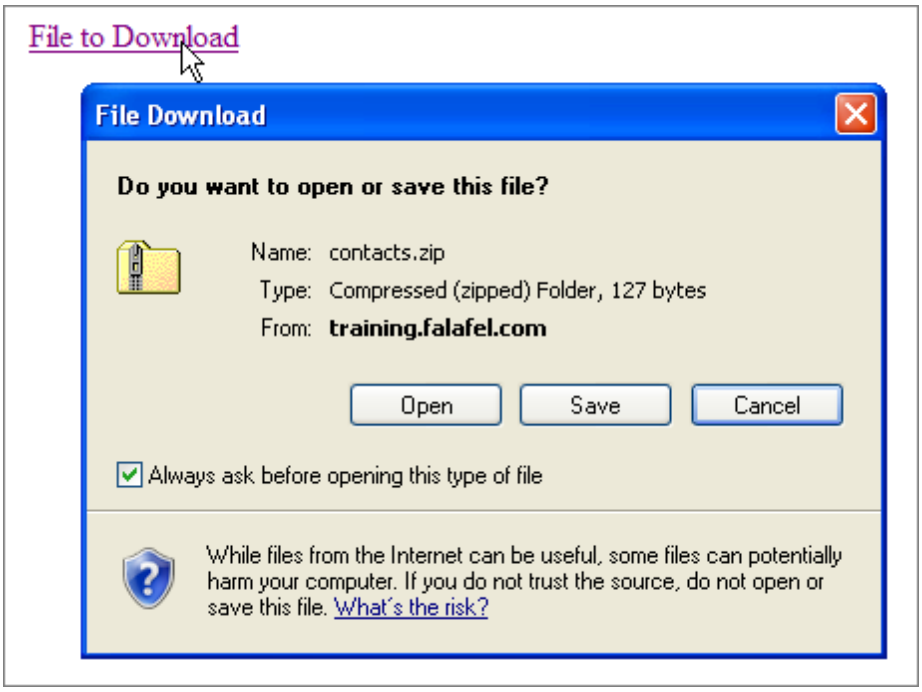
### Gotcha!

Be sure that FileUploadPath is valid and points to an existing file. If the FileUploadPath is incorrect, the test step will generate an unexpected alert dialog and cause your test to hang.

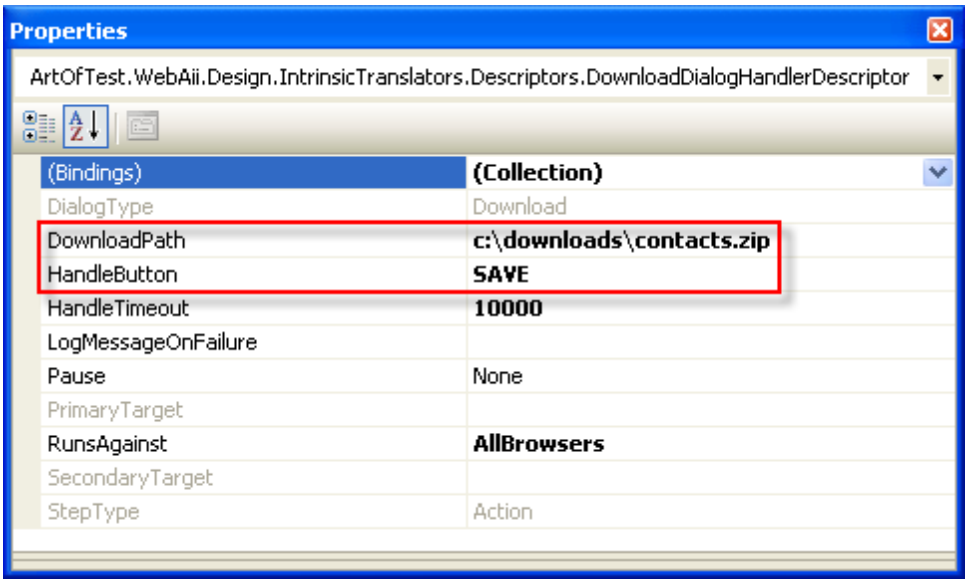


### 10.4.4 Download

When a user clicks a link to some resource on a server, such as a downloadable file, the browser displays a File Download dialog. The handler for this dialog lets you save the resource to disk.




When you create a test step to handle the File Upload dialog, the Properties pane includes a **HandleButton** property that can only be set to "SAVE" or "CANCEL". Set the **DownloadPath** property to the path of an existing file.

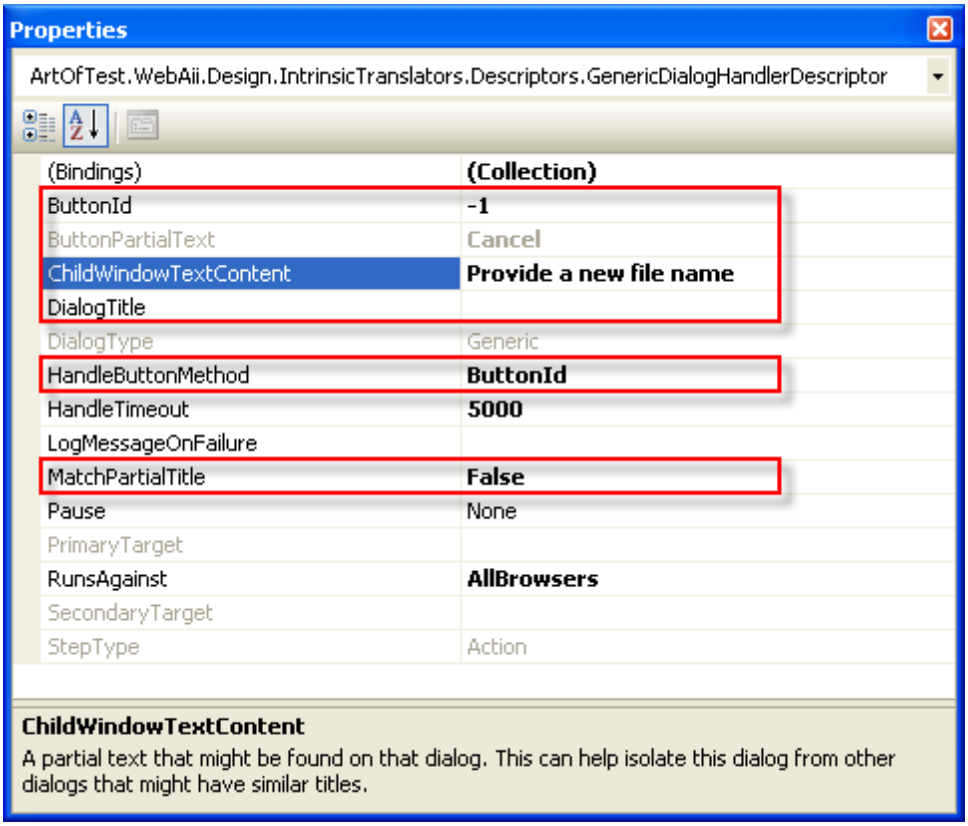


### 10.4.5 Generic

"Generic" is a customizable dialog handler that deals with "Win32" dialogs that don't have a specific handler. The key properties of the generic dialog handler help identify the dialog and identify the button used to close the dialog.

If the **MatchPartialTitle** property is "False", then the **DialogTitle** property value must match the title of the popup dialog exactly. When MatchPartialTitle is set to "True", DialogTitle can occur anywhere in the popup dialog title. In addition, the **ChildWindowTextContent** property holds text that can be found somewhere in the dialog and is used to further pinpoint the dialog.

The **HandleButtonMethod** property determines how the dialog will be closed. If you set HandleButtonMethod to the "NoneCloseDialog" setting, the dialog is closed using the close button (i.e. the little "X" in the upper right hand corner of the dialog). If you use "ButtonId", the matching **ButtonId** property must contain a number used to identify the window (you need a UI "Spy" utility to find out what the button id is). If the HandleButtonMethod is "ButtonPartialText", then a use the **ButtonPartialText** property to match some portion of the button text. For example, a ButtonPartialText property value of "Save" would match a button with the actual text "Save All".




## 10.5 Wrap Up

In this chapter you learned how to respond to pop-up dialogs that occur in your tests. You learned how to handle "Win32" type dialogs including the specialized handlers for Alert, Logon, File Upload, Download and the generic "Win32" handler. You also learned how to control HTML popup dialogs.

# Part XI

MSTest

# 11 MSTest

## 11.1 Objectives

In this chapter you'll learn how MSTest can be used to automate tests. You will learn how to run MSTest from the command line and some of the key parameters used with MSTest.

## 11.2 Overview

MSTest is a Microsoft utility that lets you run tests directly in Visual Studio or on the command line. Telerik WebUI Test Studio Developer Edition make use of MSTest as the engine behind running WebUI Tests.

***Notes***

In the QA edition, WebUI Test Studio has its own engine to run its tests without the need for MSTest to be available on the machine.

The command line version can be especially helpful when you need to automate your build and testing processes. If your organization has a "Continuous Integration" process that performs daily check-in, build and testing of your software, WebUI tests can be included to complete your testing coverage.

### 11.2.1 Running Tests From the Command Line

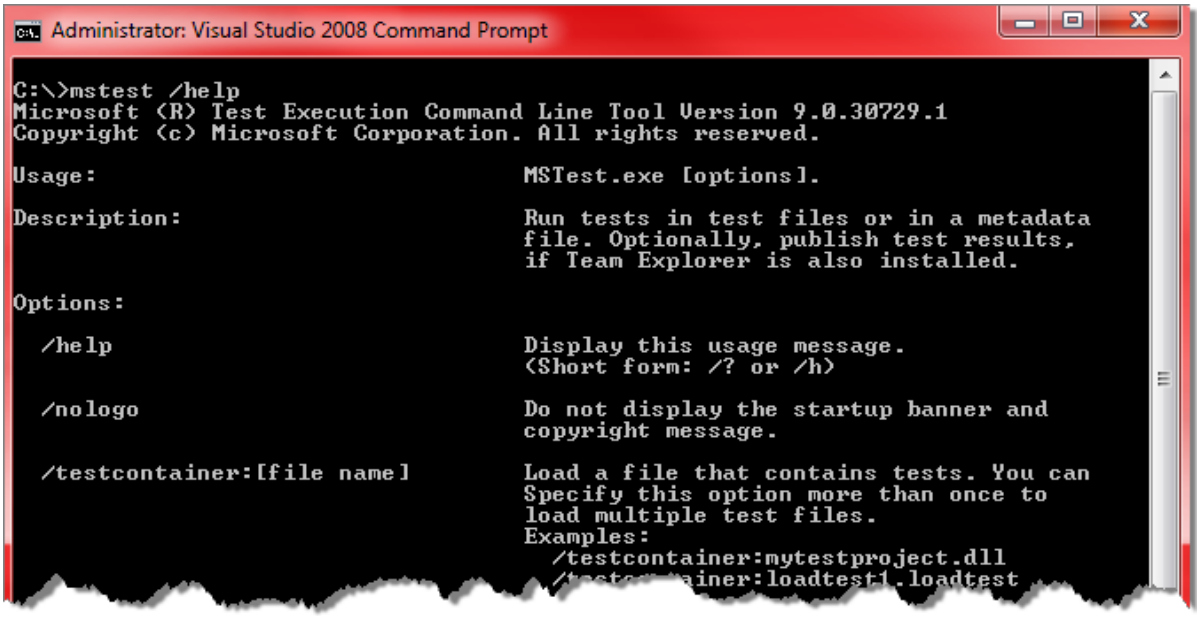
MSTest is highly configurable, but we can get by with a minimal set of parameters to get started. For example, to run the "GettingStarted" test project from the command line:

1. From the Window's "Start" menu, find the Visual Studio command prompt (usually under the Visual Studio Tools menu item).
2. In the command line, navigate, using standard DOS commands, to the directory that contains the "GettingStarted" project DLL.
3. In the command line enter **mstest.exe /testcontainer:GettingStarted.dll**. An example run of the command is shown below. Notice that it executes the test, prints a listing of the results and outputs a results file.

```
mstest.exe /testcontainer:GettingStarted.dll
Microsoft (R) Test Execution Command Line Tool Version 9.0.30729.1
Copyright (c) Microsoft Corporation. All rights reserved.

Loading GettingStarted.dll...
Starting execution...

Results            Top Level Tests
-------            ---------------
Passed             GettingStarted.UnitTest1.TestMethod1
1/1 test(s) Passed

Summary
-------
Test Run Completed.
  Passed  1
  ---------
  Total   1
Results file: ...\TestResults\falafel_WEBUI 2010-04-24 15_11_42.trx
Run Configuration: Default Run Configuration
```

## 11.2.2 Understanding Key MSTest Parameters

To get the full list of parameters available to MSTest, run "MSTest /help" from the Visual Studio command prompt.




The key command parameters that matter to WebUI Test Studio in a continuous integration environment are:

- **/testcontainer**: This option tells MSTest which DLL contains your unit test code and is required. Typically the current working directory is the root of the project. You need to take this into account and specify a path that is relative to the projects root such as '/testcontainer:bin/debug/SydneyTests.dll.
- **/testlist**: If you've taken the time to configure test lists, you can use this option to specify which list of tests to run by name. This option requires you also specify '/testmetadata:' which is the path to the projects metadata file (i.e. the .vsmdi file). The test lists defined in your project are kept in this metadata file and nowhere else. To run multiple test lists simply add this option multiple times to the command line, each time specifying one test list. You only need to specify '/testmetadata:' once however.

**Gotcha!**

If you do not specify /testlist or the '/tests:' option MSTest will run all unit tests it can find in the unit test .DLL.

- **/test:** This option tells MSTest to run one specific test found in the unit test .dll file. Unlike the '/testlist:' option, you do not need to specify '/testmetadata:' with this option. This is because you're not specifying a list, but a single specific test instead.

***Notes***

To run multiple tests, simply add this option multiple times to the command line, each time specifying a different test.

**Gotcha!**

If you do not specify this option or the '/testlist:' option MSTest will run all unit tests it can find in the unit test .DLL.

- **/resultsfile:** This option specifies the path and name of the results file to create. Normally MSTest creates the results in a uniquely named .trx file in a 'TestResults' folder. This folder is contained in the root of the project. For example: 'TestResults\agentuser_agent-pc4 2009-04-10 11_34_07.trx'. Because the name of this file is based on the logged on user ID, machine name and the date & time, it can be hard for some continuous integration systems to find this results file, especially those that run MSTest via an 'exec' task. By specifying a fixed filename, it's easier for the continuous integration systems to find the results file and pull it into the build report.

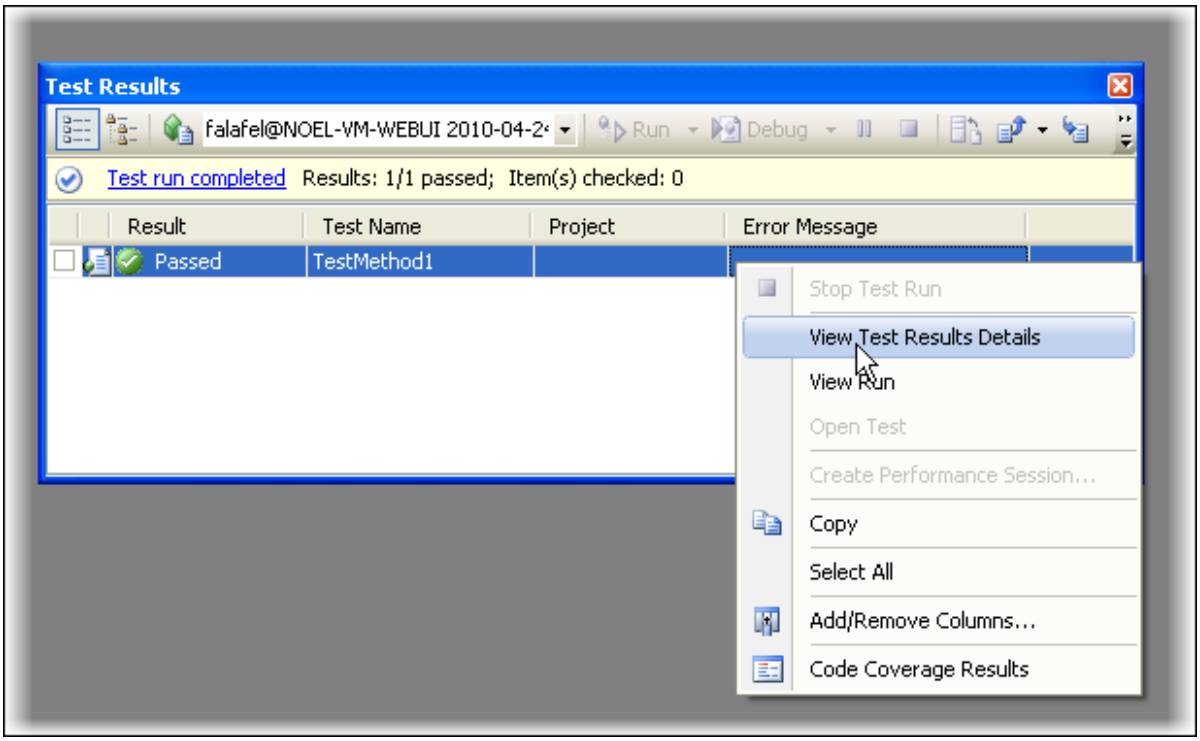
**Gotcha!**

The only drawback is that MSTest will fail (not even run tests) if the file specified already exists. Therefore you need to delete any results file that may have been left behind from previous builds. But be careful that your delete operation doesn't throw an error if the file to delete doesn't exist (e.g. a simple 'del myResults.trx' will return an error if the file doesn't exist). This can cause some automated build systems to stop if it detects the error being returned by a simple delete operation.

The "trx" results file is simply an XML file that contains the conditions that the test was run under and the results of the test. If you change the extension of the "trx" file to "xml", you can see the raw XML in the browser:

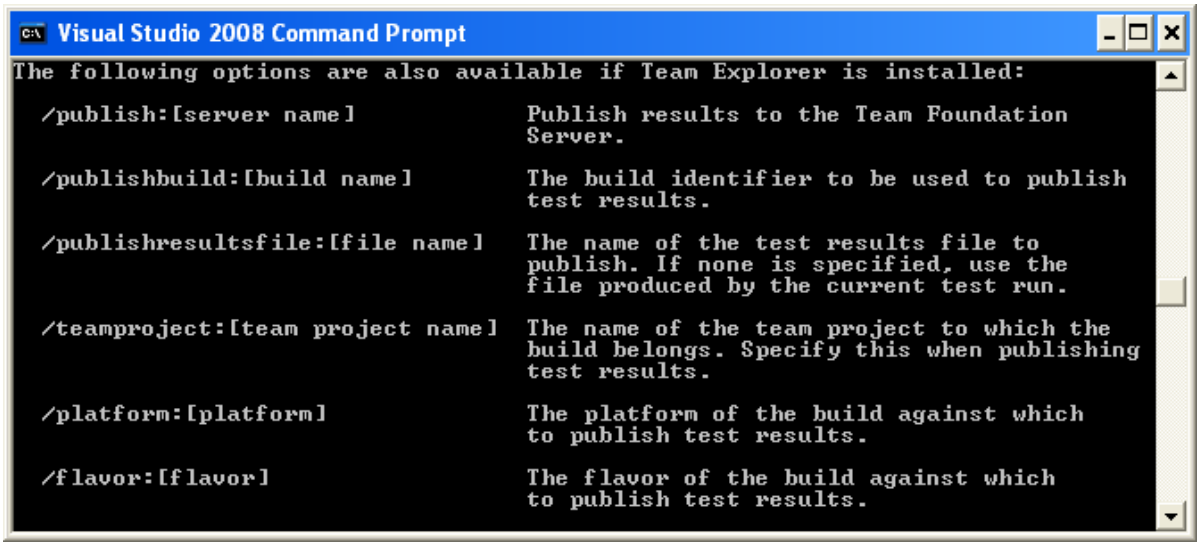
```
<?xml version="1.0" encoding="UTF-8" ?>
- <TestRun id="c9248632-850e-4d8e-9ace-9a837d1c7fb4" name="falafel@ 2010-04-24
    15:11:42" runUser=" \falafel"
    xmlns="http://microsoft.com/schemas/VisualStudio/TeamTest/2006">
  + <TestRunConfiguration name="Default Run Configuration" id="844e8566-7080-431d-9a0c-bb5babd646d7">
  + <ResultSummary outcome="Completed">
    <Times creation="2010-04-24T15:11:42.7747520-07:00" queuing="2010-04-24T15:11:48.0623552-07:00"
      start="2010-04-24T15:11:48.7333200-07:00" finish="2010-04-24T15:11:50.0351920-07:00" />
  + <TestDefinitions>
  + <TestLists>
  + <TestEntries>
  - <Results>
    - <UnitTestResult executionId="b25f0f81-c619-47f0-8f31-5077ac95e8cd" testId="1237b6b6-7bce-04b9-
        35dc-7bf4e541be77" testName="TestMethod1" computerName="
        duration="00:00:00.0173586" startTime="2010-04-24T15:11:48.9135792-07:00" endTime="2010-04-
        24T15:11:49.9250336-07:00" testType="13cdc9d9-ddb5-4fa4-a97d-d965ccfc6d4b" outcome="Passed"
        testListId="8c84fa94-04c1-424b-9868-57a2d4851a1d">
        <Output />
      </UnitTestResult>
    </Results>
  </TestRun>
```

To see the results formatted within Visual Studio, double-click the "trx" file. Visual Studio will display the file contents in the Test Results window, where you can use the IDE to drill down for greater detail.




## Integration with Team Foundation Server

If you use Team Foundation Server (TFS) there are several other options for running MSTest that will integrate with TFS directly that will allow for checking in the result file and associate the test with a specific team project, etc... The list of parameters generated by "mstest /help" will include a section of TFS specific parameters that you can reference.




## 11.3 Wrap Up

In this chapter you learned how MSTest can be used to automate tests. You also learned how to run MSTest from the command line and some of the key parameters used with MSTest.

# Part XII

Unit Testing

# 12 Unit Testing

## 12.1 Objectives

In this chapter you will learn about the purpose, advantages and types of unit testing frameworks. You will create a simple WebAii test with coded steps and data, then convert the WebAii test into a Visual Studio unit test and run it.

**Find the projects for this chapter at...**

\Courseware\Projects\<CS|VB>\UnitTesting\UnitTesting.sln

## 12.2 Overview

The goal of unit testing is to verify that each element or "unit" of an application, performs as expected. Unit tests are generally superior to manual testing:

- Automated tests are consistent, predictable and can be repeated. Running unit tests regularly helps ensure that the software works now and in the future. By contrast, manual testing tends to be less thorough over time.
- Errors are discovered early and at less cost. Problems discovered early on are usually easier to fix than those that get "baked in" to the software. Automated unit tests are more likely to be run often to catch errors early.
- Software can be changed more frequently when unit tests ensure that the software is fit to deliver. This confidence level allows easier integration of new features and cleanup ("refactoring") of aging software.
- Unit testing reduces "truck factor", i.e. if a key programmer is hit by a truck tomorrow, new programmers can take over with less fear of trashing the quality of the software. No single person "owns" any part of the software. Unit tests also guide new programmers by documenting feature requirements and grouping related features.

There are a large number of unit testing frameworks that provide a consistent model for testing and that allow test automation. Unit testing frameworks provide infrastructure for common testing tasks, such as setting up the conditions of a test, providing data to the test, reporting test conditions (e.g. "the invoice total is incorrect") and cleaning up after the test is complete.

Tests recorded in WebUI Test Studio can be converted to any of several popular unit testing frameworks, namely **NUnit**, **MbUnit**, **VsUnit** and **xUnit**.

## 12.3 Creating a Unit Test

The "Generate a Unit Test" feature takes an existing WebUI Test Studio test and creates a unit test that works for one of the supported unit test frameworks. When the unit test is created, the conversion sweeps up all the test steps, any code that we've added and any data we've defined. All this material is added to the new unit test. As an example we will walk through creating a VsUnit, i.e. Visual Studio unit test.

1) From the Visual Studio choose **File > New > Project...** This will display the "New Project" dialog.

2) Define a new test project:

   a) In the "New Project" dialog, select the "Test" project type for your language (C# or VB).

   b) Select the "Test Project" template.

   c) Enter a descriptive name for the test project, a location path and a Solution name.

   d) Click **OK** to create the new test project.

3) From the Solution Explorer, right-click the test project and select **Add > New Test...** from the context menu. This will display the "Add New Test" dialog.

4) In the "Add New Test" dialog, select the "WebAii Test" template, name the test "VsUnitTest.aii" and click **OK** to create the test.

5) In the Recording Surface, enter "http://www.google.com" to the browser address bar and then click the Go to Url button.

6) In the Steps Tab, click the **Add... > Custom Annotation** button.

7) In the Steps Tab, select the Custom Annotation test step. In the Properties pane, change the **AnnotationText** property to "This will be a coded step".

8) In the Steps Tab, right-click the Custom Annotation test step and select "Customize Step in Code" from the context menu. This test step will create a "VsUnitTest.aii.cs" file and populate it with the coded step.

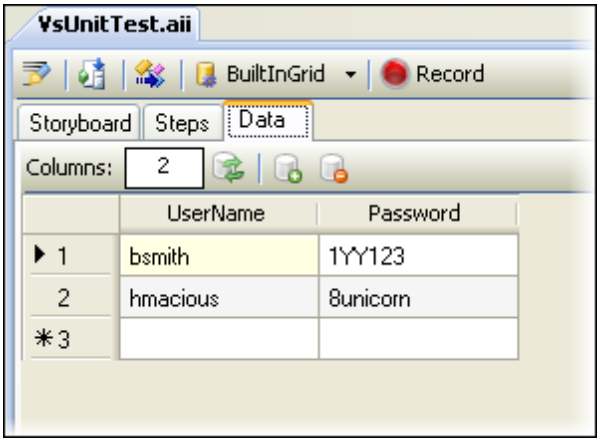
9) Navigate to the Test Tab, then click on the Data Tab tab.

10) In the Data Tab tab, set the number of columns to "2" and click the Update Columns button  .

11) Right-click the first column and select "Rename Column" from the context menu. Rename the column "UserName".

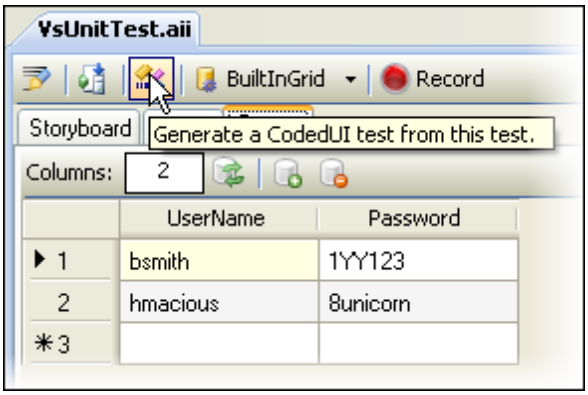
12) Right-click the second column and select "Rename Column" from the context menu. Rename the column "Password".

13)Enter two lines of random data to the table. The Data Tab tab should look something like the screenshot below.

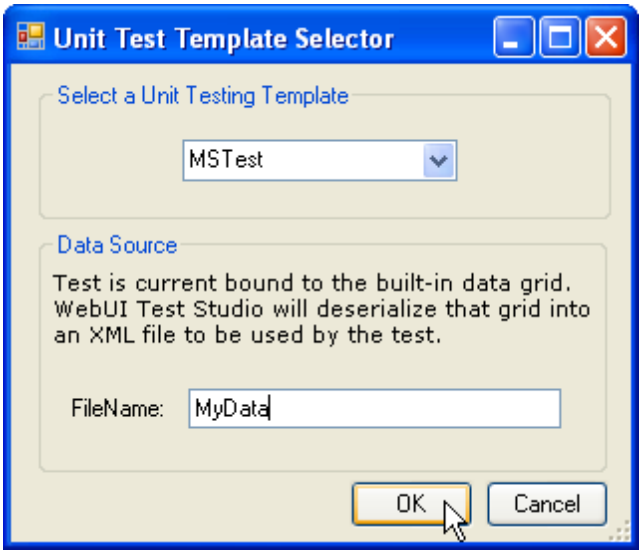



At this point we have two test steps, a coded test step and some sample data. Next, you will convert the WebUI Test Studio test into a unit test.

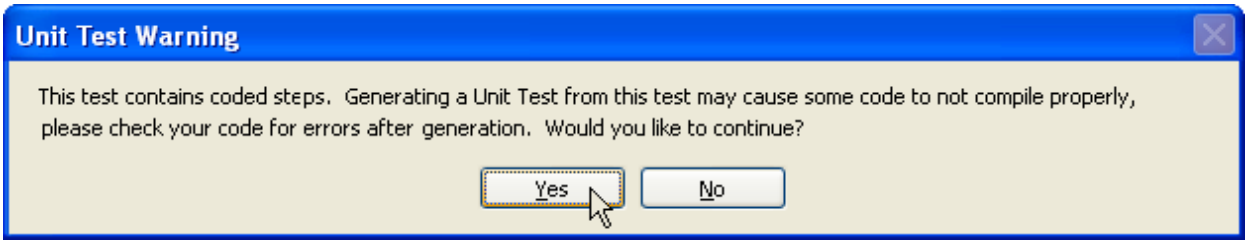
14)In the Test Tab toolbar, click the "Generate a Unit Test" button. This will display the "Unit Test Template Selector" dialog.



15)In the Unit Test Template Selector dialog, drop down the list of unit testing templates and select "MSTest". Because we have data defined, the Data Source area of the dialog is enabled. Enter "MyData" as the name of the XML file where the data will be placed. Click the **OK** button to begin the conversion.

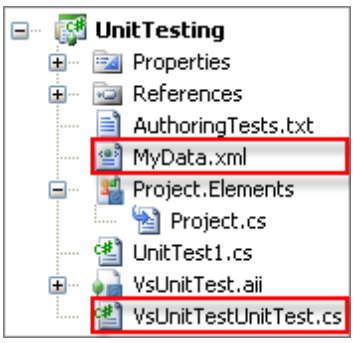



16)When you have coded steps, a warning dialog may appear. Click the **Yes** button to continue.



17)Lets review the new materials that are created in the Visual Studio Solution Explorer.

The test data is converted to an XML file shown in the screenshot below as "MyData.xml" and a new unit test file is created, named after your WebAii test name plus "UnitTest".

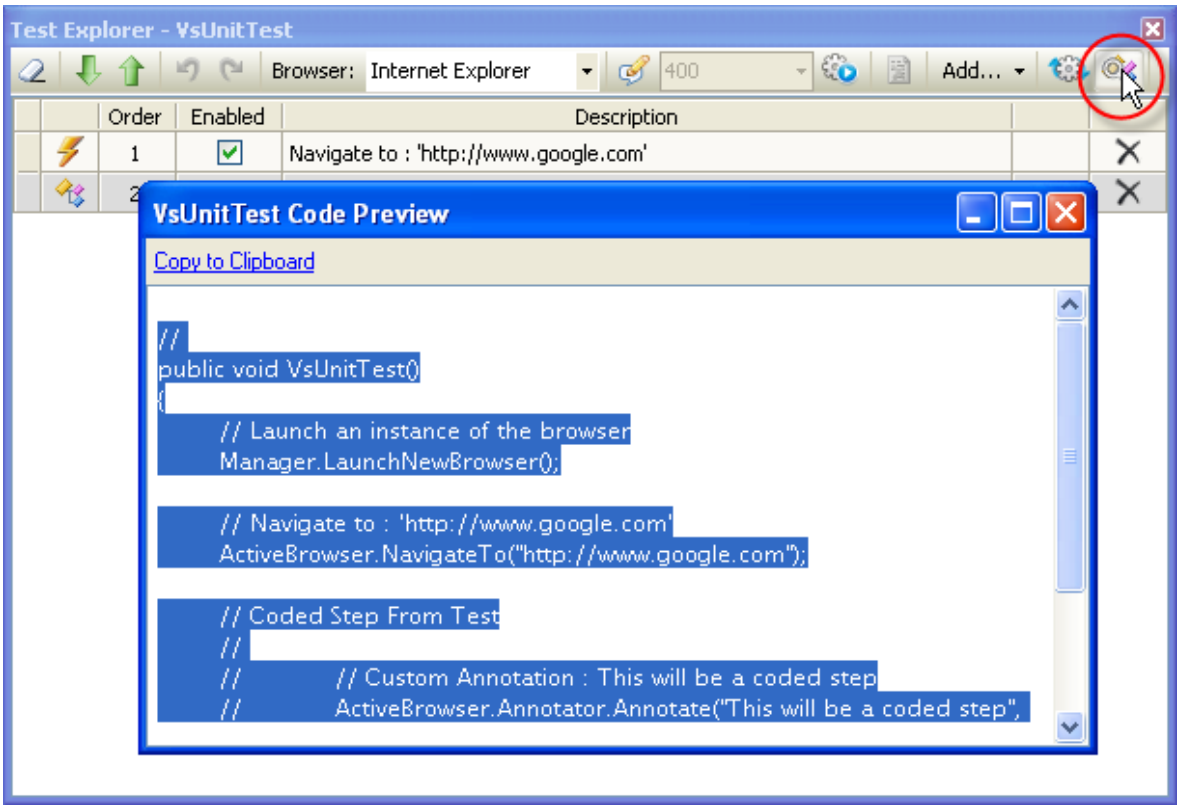



In the unit test file we find a test method named after our WebAii test "VsUnitTest". The method is annotated as a TestMethod, with parameters that point to "MyData.xml". The first step in the test (navigates to the "www.google.com") has been included in the unit test code. The Custom Annotation that was converted to a coded step is also included, but commented out. You can remove the comments but you should recompile the project to make sure the code still works.

```
[TestMethod(), DeploymentItem("UnitTesting\\MyData.xml"),
DataSource("Microsoft.VisualStudio.TestTools.DataSource.XML",
"|DataDirectory|\\MyData.xml", "WebAiiBuiltinData", DataAccessMethod.Sequential)]
public void VsUnitTest()
{
   // Launch an instance of the browser
   Manager.LaunchNewBrowser();

   // Navigate to : 'http://www.google.com'
   ActiveBrowser.NavigateTo("http://www.google.com");

   // Coded Step From Test
   //
   //       // Custom Annotation : This will be a coded step
   //       ActiveBrowser.Annotator.Annotate("This will be a coded step", 1000, ArtOfTest.Common.OffsetRefe
   //
   //
}
```

**Tip!**

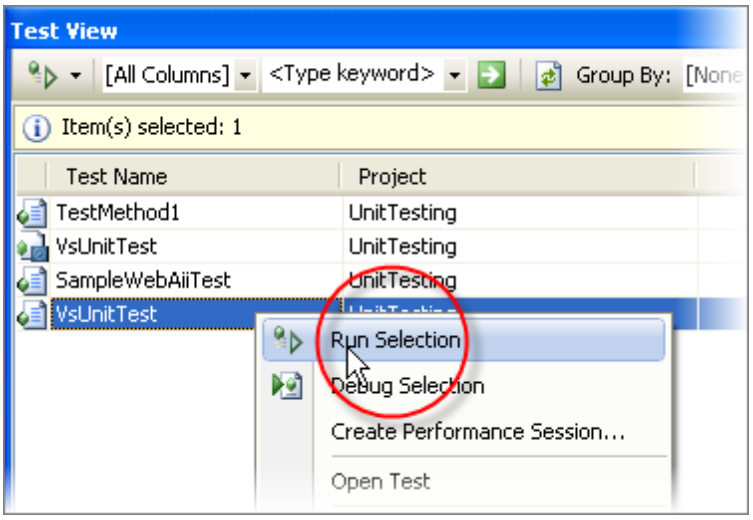
If you want to see how the code looks without actually creating the unit test, click the code preview button in the Steps Tab toolbar.



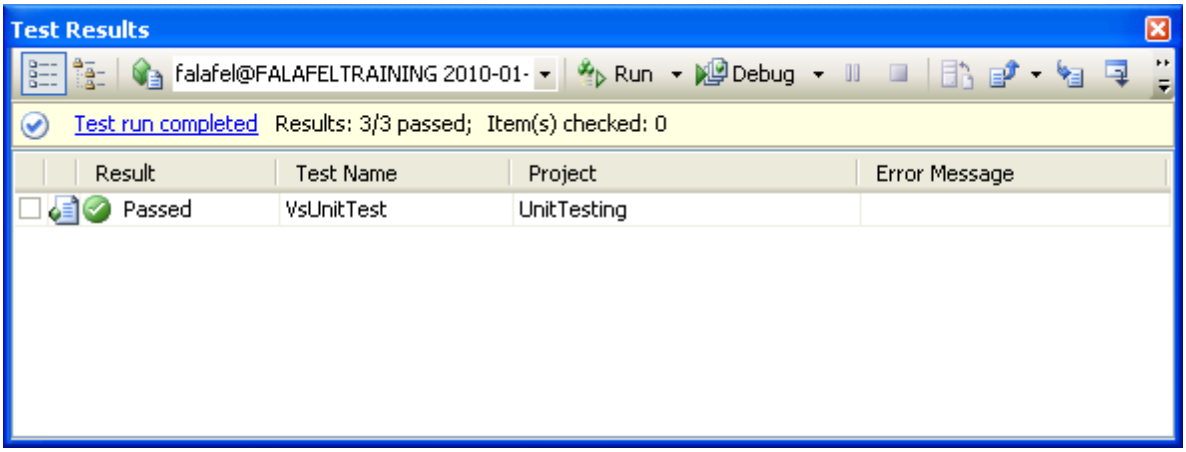

18)From the Visual Studio menu select **Test > Windows > Test Results** to open the Test Results window.

19)From the Visual Studio menu select **Test > Windows > Test View** to open the Test View window.

20) In the Test View window, right-click the unit test item (it should be the last test in the list) and select Run Selection from the context menu.




21) The unit test will begin to execute. You can observe the progress of the unit test in the Test Results window. When the unit test completes, the Test Results window will report the results. All test steps should pass.




***Notes***

If you want to use one of the unit test framework types other than VsUnit, you need to locate and download the framework, install it on your testing computer and reference the framework assemblies in your test project.

***Notes***

Be aware that once a WebAii test is converted, the connection between the WebAii test and the unit test is severed. Changes to the WebAii test are not reflected in the unit test and visa versa. To keep these in sync we recommend that you use the WebAii as a "master" test and re-generate new unit tests whenever you change the WebAii test.

## 12.4 Wrap Up

In this chapter you learned about the purpose, advantages and types of unit testing frameworks. You created a simple WebAii test with coded steps and data, then converted the WebAii test into a Visual Studio unit test and ran it.

# Part XII

Load Testing

# 13 Load Testing

## 13.1 Objectives

In this chapter you will learn the basic purpose of load testing and how a WebAii test can be made available to a Visual Studio load test. The chapter will discuss the Visual Studio load testing mechanism only as it relates to consuming WebAii tests.

**Find the projects for this chapter at…**

\Courseware\Projects\<CS|VB>\LoadTest\LoadTest.sln

## 13.2 Overview

Web applications don't run one-at-a-time for a single user in isolation. Instead, web applications run simultaneously for multiple concurrent users and have a limited set of resources. Unlike unit tests that verify individual features function as they should, ***load testing*** ensures that the application behaves well under normal and peak work loads and that the application responds to the user in a reasonable time.

WebUI Test Studio can make a recorded WebAii test usable by a Visual Studio load test. The built-in Visual Studio tools can help you model the load conditions, e.g. number of concurrent, band width resources, browsers used, think times (estimated time that a user takes to think before performing some action) and numbers of new users (new users have nothing cached in the browser yet). Visual Studio also supplies reporting and charting to interpret the test results.

Like the unit test, once a load test is created, there is no direct link between the WebAii test and the Visual Studio test. Again, the recommended best practice is to use the WebAii test as the "master" test and re-generate the Visual Studio tests as required.

## 13.3 Creating a Load Test

The general steps involved with creating a load test are:

- Recording or writing a WebAii test.
- Converting the WebAii test to a Visual Studio web test.
- Including the web test in a Visual Studio load test.

The following walk through will show the steps to create a very minimal WebAii test, use the "Convert test..." button to create a Visual Studio web test and finally add the web test to a new Visual Studio load test.

1) From the Visual Studio choose **File > New > Project...** This will display the "New Project" dialog.

2) Define a new test project:

   a) In the "New Project" dialog, select the "Test" project type for your language (C# or VB).

   b) Select the "Test Project" template.

   c) Enter a descriptive name for the test project, a location path and a Solution name.

   d) Click **OK** to create the new test project.

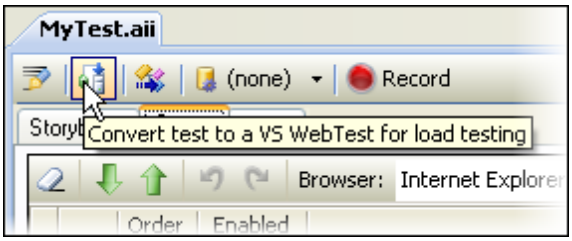
3) From the Solution Explorer, right-click the test project and select **Add > New Test...** from the context menu. This will display the "Add New Test" dialog.

4) In the "Add New Test" dialog, select the "WebAii Test" template, name the test "MyTest.aii" and click **OK** to create the test.

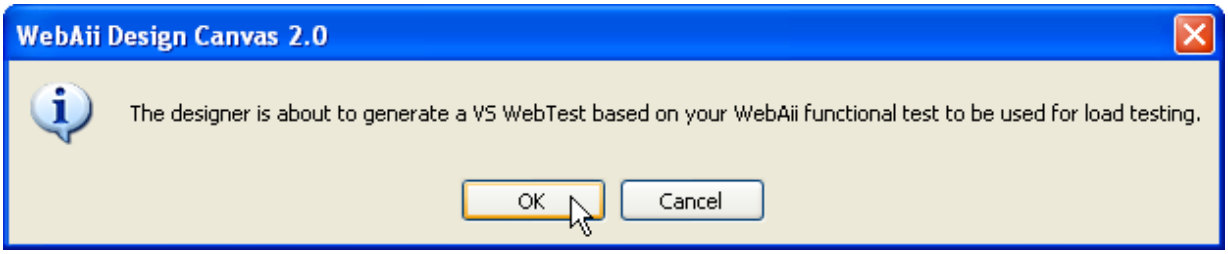
5) In the Recording Surface, enter "http://www.google.com" to the browser address bar and then click the Go to Url button.

6) Click the search button.

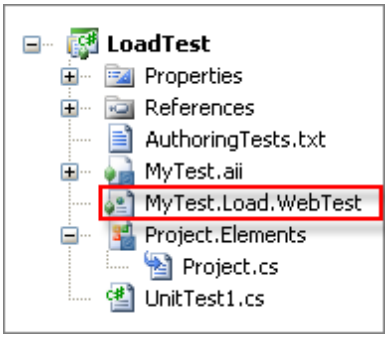
7) Navigate to the Test Tab and click the "Convert test to a VS WebTest for load testing..." button. This will cause a confirmation dialog to display.



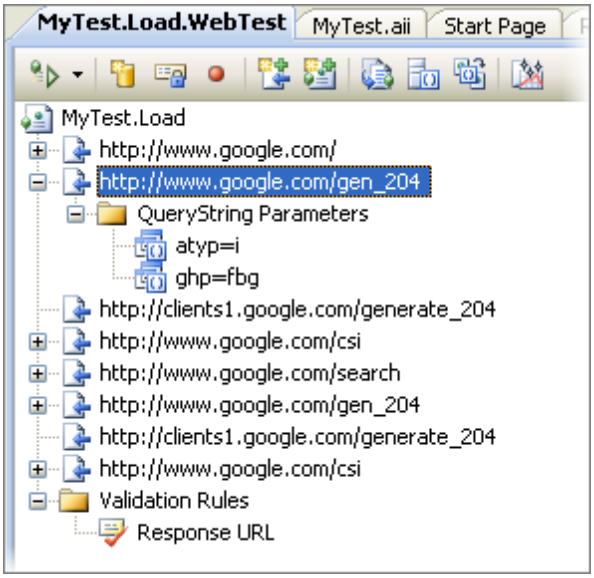
8) The confirmation dialog will notify you that the designer will generate a new Visual Studio "WebTest" based on the WebAii functional test. Click the **OK** button to continue.



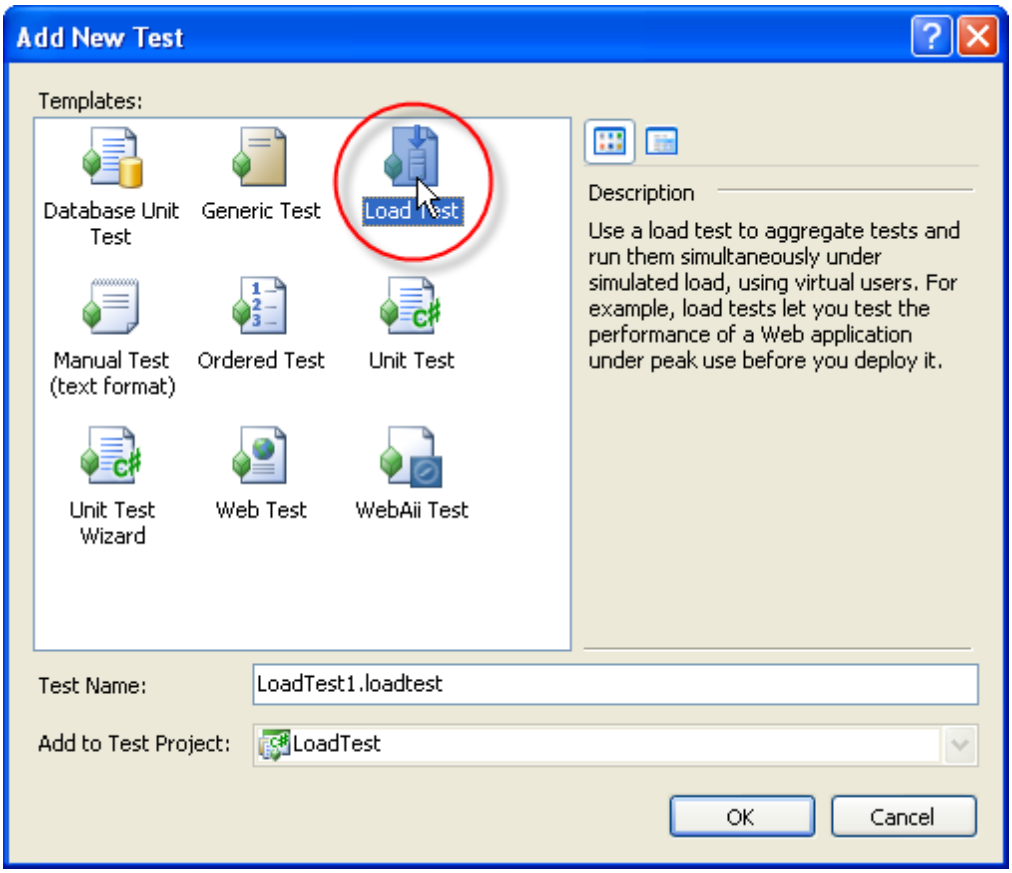

9) At this point a web browser will display, the test will be recorded by Visual Studio, parameters will be extracted for each web page and finally, the new web test file will be created.




The new load test in the Visual Studio editor shows the traffic with the web server and any parameters that were sent with the requests.

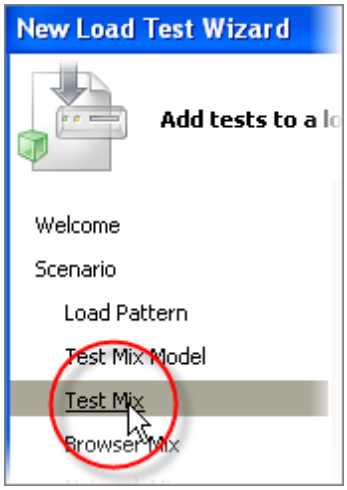


10)From the Visual Studio Solution Explorer, right-click the test project and select **Add > New Test...** from the context menu. This will display the Add New Test dialog. Select "Load Test" from the templates and click the **OK** button to continue. This will display the New Load Test Wizard.

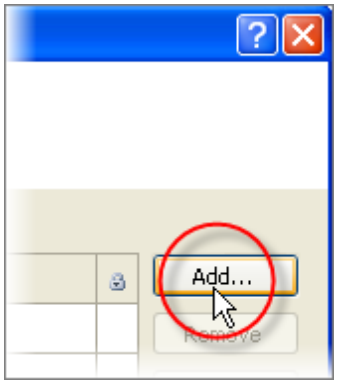



11)The New Load Test Wizard lets you select the web tests to include in the load test and to configure the load, e.g. number of users the browsers used and network bandwidth. In this case we only want to add our web test to the load test.

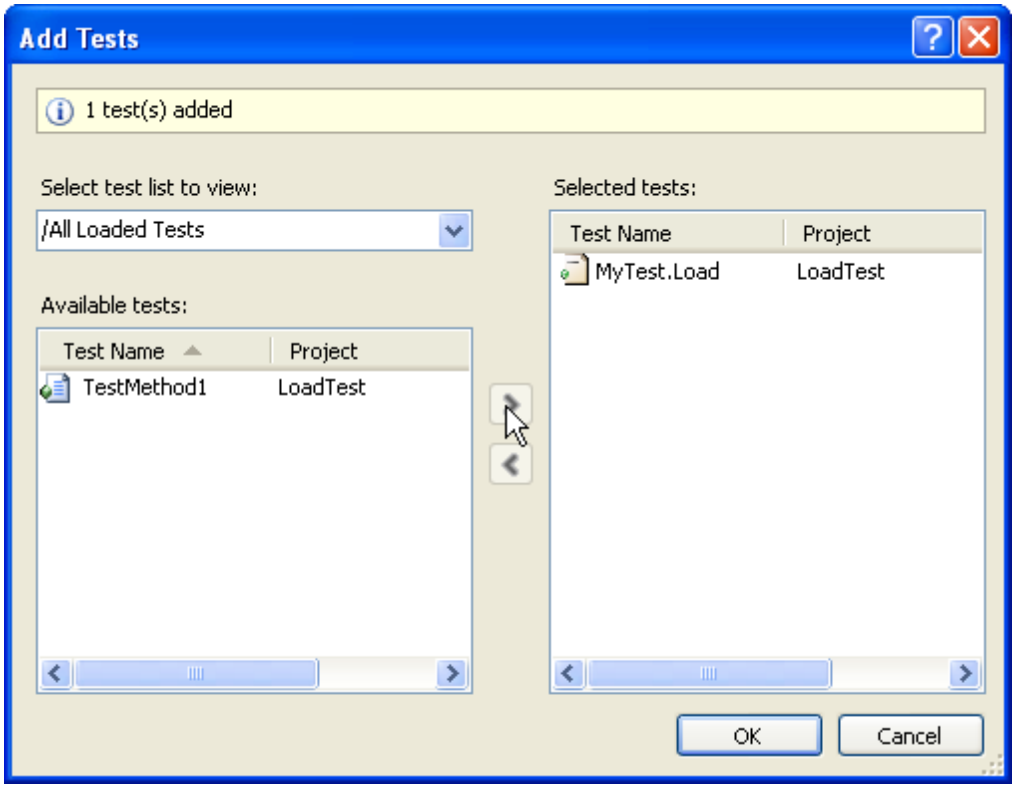
a) Click the "Test Mix" step in the tree view.



b) Click the **Add...** button on the right side of the dialog. This will display the Add Tests dialog.



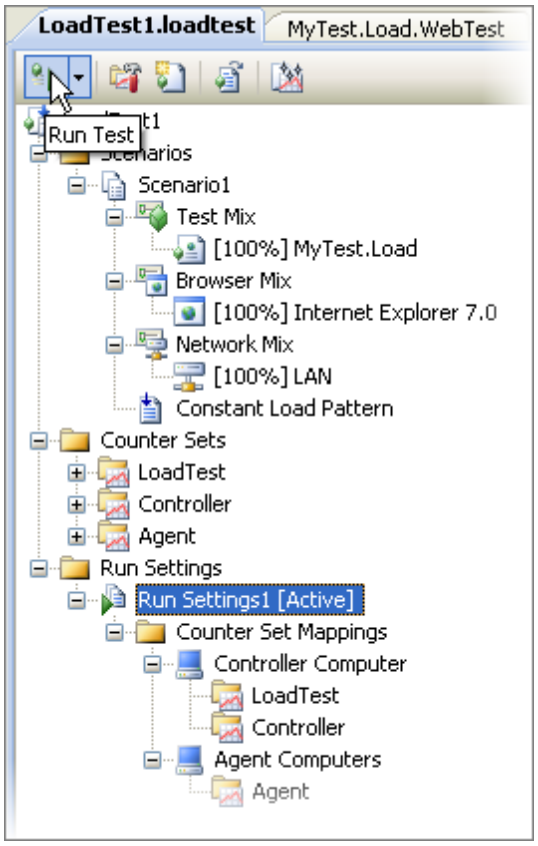

c) In the Add Tests dialog, select "MyTest.Load" from the Available Tests list on the left. Click the rightward pointing arrow to move the test to the Selected Tests list. Click the **OK** button to continue.



d) Back in the New Load Test Wizard, click the **Finish** button.

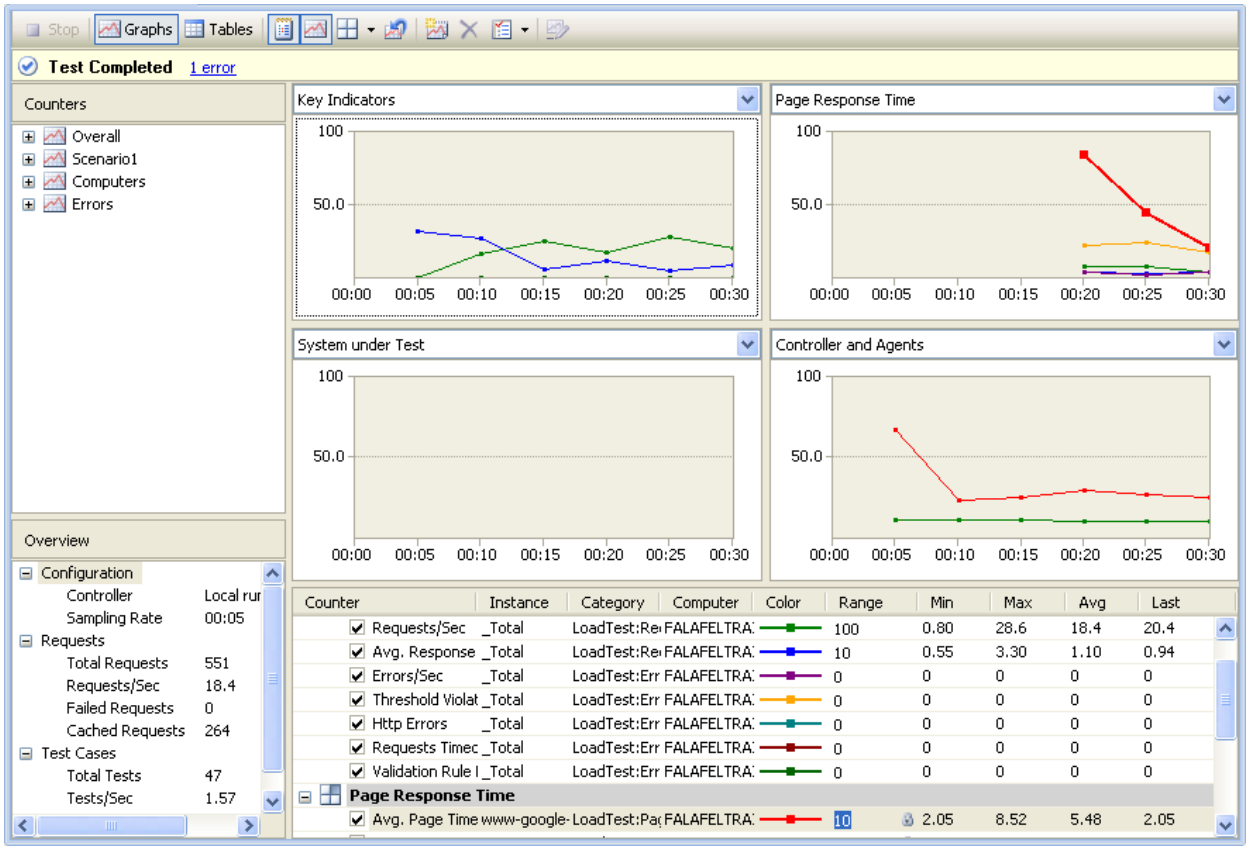
The new load test will be created and visible in the Visual Studio editor. You can tweak any of the configuration settings for the load test by selecting them in the tree view and editing the properties in the Properties pane. In particular, you can select the Run Settings node and reduce the Run Duration property from 10 minutes to some lower number.

Click the "Run Test" button to execute the load test. The load test should run for the amount of time specified in the Run Duration property. The Test Results window should show that the test passed.



e) The completed test should show graphs and tables of statistics for the completed test.

Note that the error link at the top of the page refers to a warning about a missing database connection used to store the test results. The web test itself has passed.



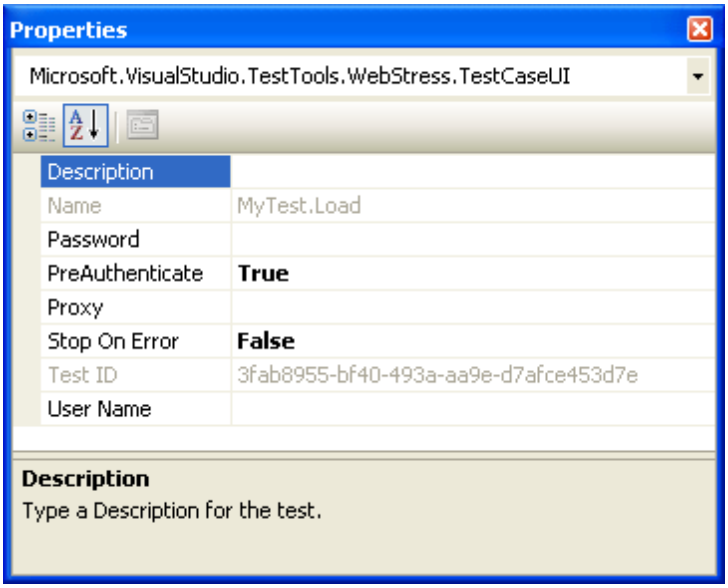

### *Notes*

Remember that once you create record the Visual Studio web test, you are "not in Kansas anymore", you are in the Visual Studio testing environment. There is no direct connection between the WebAii test and the load test.

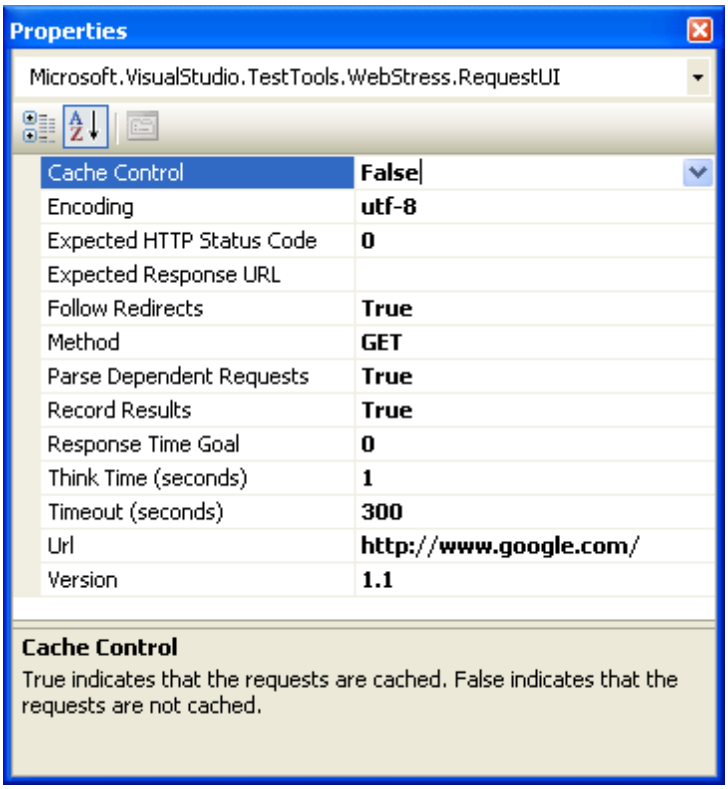
## 13.4 Web Test Step Properties

When you create a web test in Visual Studio, the editor window displays a tree view that defines the steps of the test. By default, the recorded steps should work just fine, but you may want to tweak the test settings as conditions change.

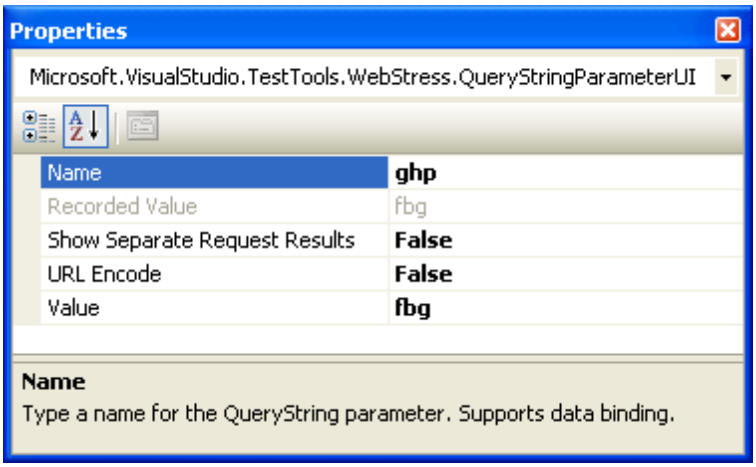

The top level node, "MyTest.Load" in the screenshot below, represents the properties for the entire test. Here you can add authentication information for the test if user name and password are required, add a description for the test, indicate a proxy by name or flag if the test should halt if there's an error.



Under the test are a series of requests that go to the web server. Each request may pass information to the server through query strings included in the Url or hidden fields embedded in the page. The content of the query strings or hidden fields are included below each web request where they appear. The properties for the web request as shown in the screenshot below can modify how the request is sent to subtly change the test conditions. For example if you disallow caching, the request will take longer than if caching is enabled.

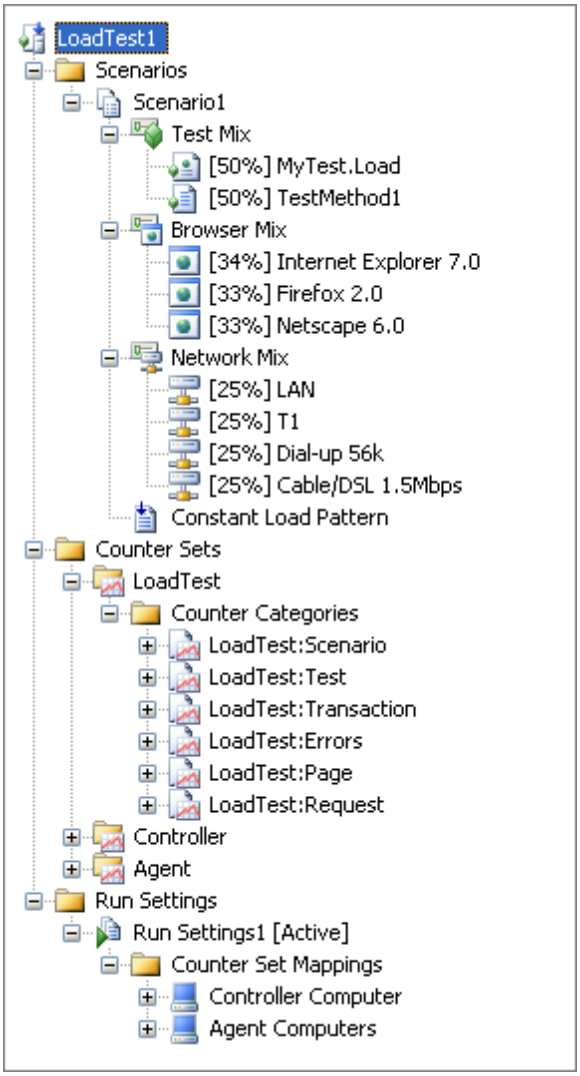



The Query String Parameters that may accompany a web request each have their own properties. Consider the Url "http://www.google.com?action=search". The Name of the query parameter is "action" and the Value is "search". The screenshot below shows another parameter named "ghp" where the value is "fbg". You can change the name or value of the query parameter or turn on URL Encoding (encoding formats a Url and substitutes legal characters for illegal characters). The "Show Separate Request Results" changes how this parameter is displayed in reports.



## 13.5 Load Test Settings

When you first create a load test the New Load Test Wizard lets you model the settings for the test. You can change these settings later when you look at the load test in the Visual Studio editor as shown in the screenshot below.

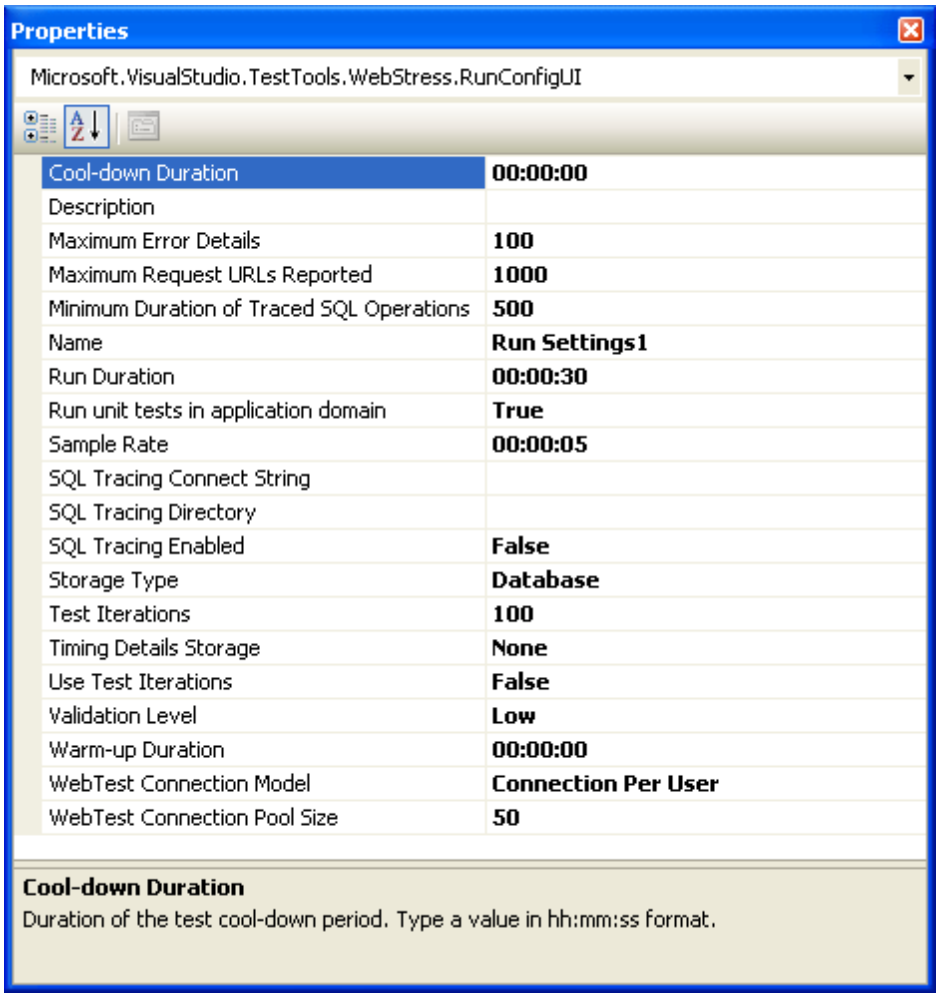



The settings are grouped into "Scenarios", "Counter Sets" and "Run Settings"

- **Scenarios**: list the elements that make up a load test including the tests themselves, the browsers used, the types of networks being tested against and the load pattern. The test, browser and network settings can be allocated by percentage to create a mix, e.g. a browser mix of IE 7, Firefox and Netscape each at roughly a 33% each. Each of these mixes can be edited by right-clicking with the mouse and selecting "Edit ... Mix" from the context menu.

  The load pattern sets the number of users making requests at any one time. A "Constant" load pattern is a fixed number of users that doesn't change during the test. A "Step" load pattern increases over time and properties of the load pattern let you set a minimum and maximum user count, number of users to add and how often. In a "Goal Based" load pattern the number of users is adjusted depending on performance criteria such as "% of Processor Time".

- **Counter Sets**: These are the categories of information being collected and are grouped by category. For example, the "Transaction" category contains "Total Transactions", "Avg. Transaction Time", and "Avg. Response Time".
- **Run Settings**: These are the properties that configure the conditions of the test, such as the length of the test or the number of test iterations, how frequently performance data is collected, warm up/cool down durations and connection information.




## 13.6 Wrap Up

In this chapter you learned the basic purpose of load testing and how a WebAii test can be made available to a Visual Studio load test. The chapter discussed the Visual Studio load testing mechanism only as it relates to consuming WebAii tests.

# Part XIV

WebAii Framework

# 14 WebAii Framework

## 14.1 Objectives

In this chapter you will learn how to perform many common automated testing operations in code to work with both Html and Silverlight based elements.

First, you will learn how to automate the browser, starting with browser navigation to web pages using both complete and relative Urls. You will also learn how to handle browser redirection.

You will learn how to locate both single elements and collections of web page elements using the Find object for the test itself and the Find object for individual elements. To fine-tune your searches you will learn about the available Find operators. You will learn how to reference elements defined in the Elements Explorer. You will also learn how jQuery is used to find elements.

You will learn how to pause test execution until certain conditions. You will wait on element existence, content, attributes, visibility, motion and custom conditions.

You will work with properties of "wrapper" objects, including the wrappers for RadControls.

Finally, you will use "Assert" objects to verify conditions on the page.

**Find the projects for this chapter at...**

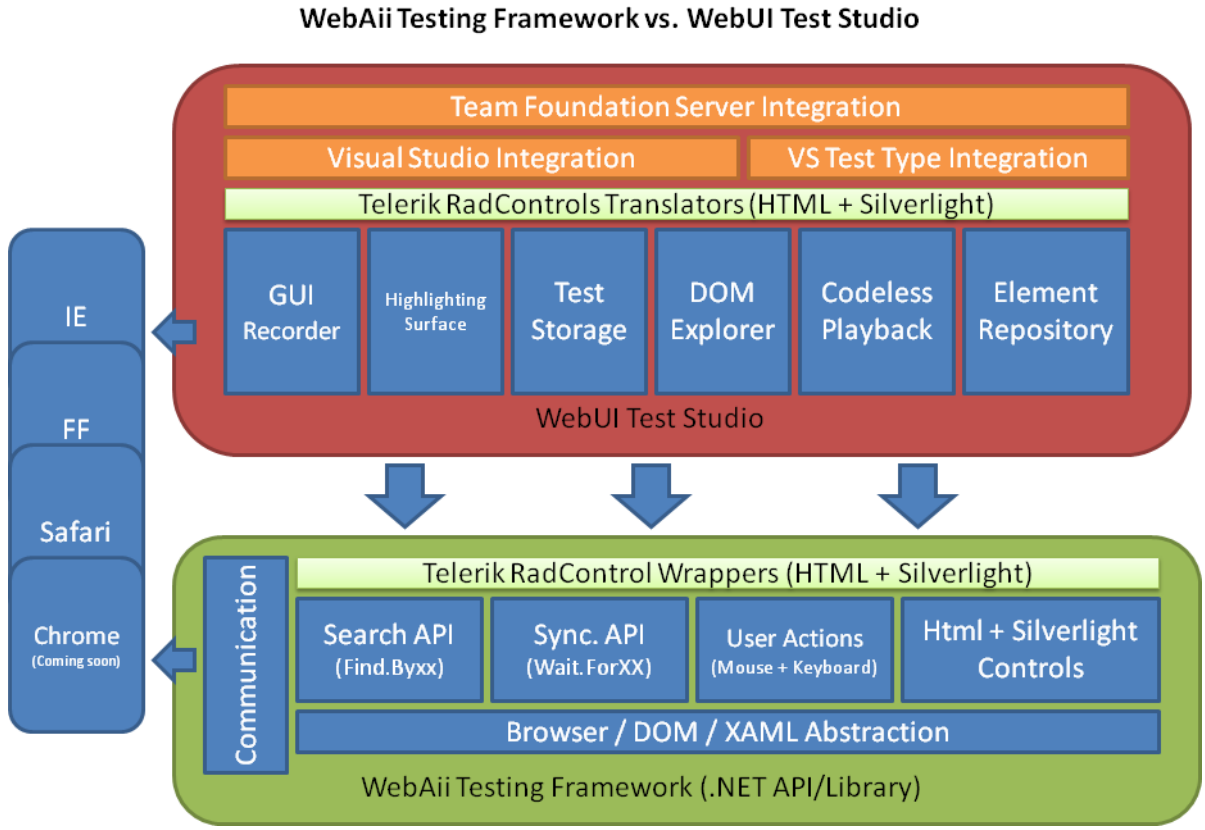
\Courseware\Projects\<CS|VB>\Framework\Framework.sln

## 14.2 Overview

There are two software testing products to consider:

- **WebAii Testing Framework:** This is a free framework that lets you write code-only tests and exercise the full functionality of the testing platform. This framework handles all the grunt work of abstracting browsers, the DOM and XAML. With the framework you can automate user actions, find elements, wait for elements and work with HTML and Silverlight controls. The Recording Surface will handle much of your testing needs and more will be possible as additional support is added to the Recording Surface, wrappers and translators. But at some point you will want to write coded tests against this framework to achieve the full potential of the tool.
- **WebUI Test Studio:** This product must be purchased and includes all the visual web testing tools we've discussed up to this point, including the Recording Surface, DOM Explorer, Steps Tab, Elements Explorer and the Test Tab with its Storyboard Tab and Data Tab areas.

The diagram below shows the relationship of the free WebAii Testing Framework to the WebUI Test Studio product. In the diagram you can see that the foundation is the WebAii Testing Framework. It provides all the base functionality used by the WebUI Test Studio. WebUI Test Studio includes all the GUI tools, the Telerik RadControls Translators and all the Visual Studio integration.



### Wrappers

Notice in the diagram that, in addition to "Translators" that let you perform actions and verifications on RadControls on the design surface, there is a set of "Wrappers" that let you work with controls using code. For example, Silverlight has an important object called "FrameworkElement" that represents all visual elements. The WebAii Testing Framework has its own version of FrameworkElement that "wraps" the original object. The wrapper allows the element to be automated and adds essential functionality such as "Find" and "Wait".

## 14.3 Getting Started Walk Through

To get started using the WebAii Framework, you first need to create the code-behind file for a test, then add test steps methods. The following walk through examples demonstrate how to perform these operations. The samples should get you up-and-running without too much background information. The sections following this walk through dig deeper into key objects that form the backbone of the WebAii API.

There are two ways to create code for a test. The first is simply to right-click a test step in the Steps Tab and select "Convert To Code" from the context menu. This creates the code-behind file and adds a test method to the file. The second technique is to create the test method manually. This next walk through demonstrates creating code steps manually.

1) From the Visual Studio choose **File > New > Project...** This will display the "New Project" dialog.

2) Define a new test project:

   a) In the "New Project" dialog, select the "Test" project type for your language (C# or VB).

   b) Select the "Test Project" template.

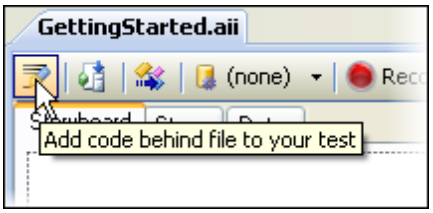
   c) Enter a descriptive name for the test project, a location path and a Solution name.

   d) Click **OK** to create the new test project.

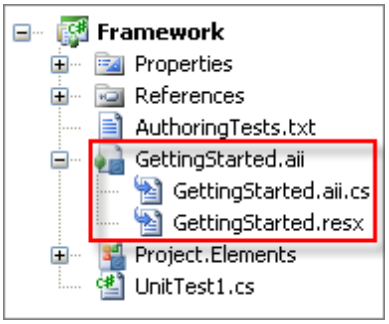
3) From the Solution Explorer, right-click the test project and select **Add > New Test...** from the context menu. This will display the "Add New Test" dialog.

4) In the "Add New Test" dialog, select the "WebAii Test" template, name the test "GettingStarted.aii" and click **OK** to create the test.

5) In the Test Tab, click the "Add Code Behind File..." button.




This will create a new code behind file in the language you've chosen for the test project type. In the screenshot below, the "GettingStarted.aii" now has a "GettingStarted.aii.cs" code behind file where we can place new coded test steps.




6) In Visual Studio, open "GettingStarted.aii.cs" file for editing.

7) Inside the "GettingStarted" test class, locate the comment "Add your test methods here..."

```
public class GettingStarted : BaseWebAiiTest
{
    [ Dynamic Pages Reference ]

    // Add your test methods here...

}
```

8) Below the comment, add a public method that doesn't return a value and that has no parameters. Name the method "MyCodedStep". **Note**: *following these steps are examples in both VB.NET and C# that can be cut and pasted.*

```
// Add your test methods here...
public void MyCodedStep()
{

}
```

9) Above the MyCodedStep() method, add a "CodedStep" attribute and pass a single string parameter "My coded step". The CodedStep attribute is required for WebUI Test Studio to display the step in the Steps Tab. The parameter can be any string that you want to show as the description of the step in the Steps Tab.

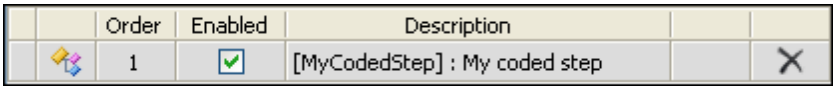
```
// Add your test methods here...
[CodedStep("My coded step")]
public void MyCodedStep()
{

}
```

Here are complete code examples for the coded step using both VB.NET and C#:


```
<CodedStep("My coded step")> _
Public Sub MyCodedStep()

End Sub
```


```
[CodedStep("My coded step")]
public void MyCodedStep()
{

}
```

10)In Visual Studio, press **Ctrl-S** to save the file (or select the Visual Studio **File > Save** option. Notice that your test step is displayed in the Steps Tab.




11)Press **F6** to build the solution (or select the Visual Studio **Build > Build Solution** menu option.

12)Click the Quick Execute button to run the test. The new coded test step should pass.

13)Inside the curly braces for the MyCodedStep() method, add the following code that writes to the test log.

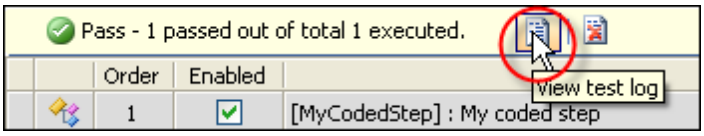

```
Log.WriteLine("This is my coded step")
```


```
Log.WriteLine("This is my coded step");
```

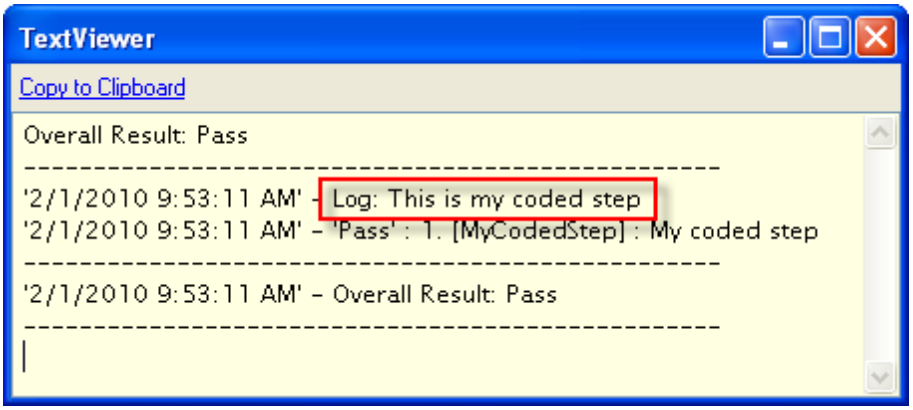
14)Press **F6** to build the solution (or select the Visual Studio **Build > Build Solution** menu option.

15)Click the Quick Execute button to run the test. The coded test step should pass.

16)From the Steps Tab, click the View Test Log button. The test log will display in its own window.




17)View the test log results. Notice the "This is my coded step" entry.



## 14.4 Common Operations

Once you have a coded step you will typically want to perform the following operations:

- Navigate to a web page.
- Locate and wait for elements.
- Automate element actions.
- Work with the current condition of an element.
- Make assertions about conditions in a test.

Unless you take special measures to implement your own base test object, tests descend from **BaseWebAiiTest**. **BaseWebAiiTest** comes with a kit of helpful methods and objects for accomplishing "bread-and-butter" tasks including:

- **Manager** is the "mother ship" of a coded WebAii Test. With it you can access the **ActiveBrowser**, handle **Desktop** input devices, launch new browsers, remove browsers, configure Settings, Log
- **ActiveBrowser** represents the browser running at any one time. The Browser type is a very rich object that has access to all methods used to automate the browser, desktop actions (i.e. controlled by mouse and keyboard), frames, Url navigation and the DOM tree, just to name a few.
- **Find** provides search routines for finding elements within a document.
- **Wait** pauses test execution until some set of conditions in the browser is met, allowing the test to continue.
- **Log** is typically used to write text to the log displayed in the Steps Tab or an external log file. Log can also be used to capture screenshots of the browser or desktop.
- **Actions** provides generic support for all browser types and includes setting check boxes, selecting from drop down lists, visually annotating on the browser surface, clicking buttons and waiting for elements.
- **ExecuteStep()** executes a test step with a particular test step name.
- **ExecuteText()** executes another test in the project as a test step.

## 14.4.1 Navigate

### 14.4.1.1 NavigateTo()

Each WebAii test always has a browser object instance called **ActiveBrowser**. You will often call the ActiveBrowser **NavigateTo()** method and pass the Url of the site you want displayed in the browser. The example below navigates to the Google site and then waits for a quarter of a second.

```
<CodedStep("Navigate to Google")> _
Public Sub MyTestStep()
  ActiveBrowser.NavigateTo("http://www.google.com")
  System.Threading.Thread.Sleep(250)
End Sub
```

```
[CodedStep("Navigate to Google")]
public void MyTestStep()
{
   ActiveBrowser.NavigateTo("http://www.google.com");
   System.Threading.Thread.Sleep(250);
}
```

### 14.4.1.2 Relative Urls

You can set the base url either in the WebUI Test Studio Settings dialog or in code. The NavigateTo() method parameter then takes only the fragment of the url past the base url. The example below sets the BaseUrl in code, then calls NavigateTo() and passes only the page name.

```
' set the base url
Manager.Settings.BaseUrl = "http://training.falafel.com/SampleWebSite"

' navigate to the elements sample site
ActiveBrowser.NavigateTo("/FindElementsSample.htm")
```

```
// set the base url
Manager.Settings.BaseUrl = "http://training.falafel.com/SampleWebSite";

// navigate to the elements sample site
ActiveBrowser.NavigateTo("/FindElementsSample.htm");
```

**Gotcha!**

Be sure to add the leading forward slash as shown in the code example above. A UriFormatException in the log may remind you:

> *"Url passed in has invalid format. If you are trying to use relative paths, please make sure your url starts with '/' or '~/'. If you are using fully qualified paths, make sure they are properly prefixed (i.e. start with 'http://', 'file://' or 'c:'). InnerException: System.UriFormatException: Invalid URI: The format of the URI could not be determined."*

#### 14.4.1.3 WaitForUrl()

You can also add a **WaitForUrl()** method call just after the NavigateTo() if you want to ensure that the Url loaded. WaitForUrl() can be useful when you need to wait for browser redirects on certain sites.

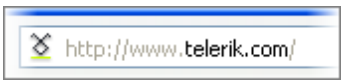
The first parameter is the Url to wait for. The next parameter, **IsPartial**, signals that this is a partial Url when set to "True". The last parameter is the number of milliseconds to wait before the operation times out. The example below navigates to the Telerik web site and then waits 10 seconds for the Url to load. The WaitForUrl() is looking for an exact match to the Url parameter.

```
<CodedStep("Navigate Test")> _
Public Sub MyTestStep()
  ActiveBrowser.NavigateTo("http://www.telerik.com")
  ActiveBrowser.WaitForUrl("http://www.telerik.com", False, 10000)
End Sub
```

```
[CodedStep("Navigate Test")]
public void MyTestStep()
{
   ActiveBrowser.NavigateTo("http://www.telerik.com");
   ActiveBrowser.WaitForUrl("http://www.telerik.com", false, 10000);
}
```

**Gotcha!**

The example above will actually fail because IsPartial is "false". The problem is very subtle in that we navigate to "http://www.telerik.com", but "http://www.telerik.com/", with the trailing slash is loaded.




To make the test step pass you need to include the trailing slash in the Url or set the IsPartial parameter to "true".

**From the Forums...**

**Question**: I'm just trying to test a simple asp.net login form on my site. When the login button is clicked I want to verify that the user navigates to the "Start" page. I check that the active browser Url contains "Start.aspx" but it always fails.

**Answer**: During the redirect from Login.aspx to Start.aspx, communication between the browser and the web server may have gone quiet just long enough for the framework to believe it's complete. The framework keeps a copy of the DOM & URL in memory and uses that for all of its testing. Some time later (maybe just a few milliseconds) communication to the web server starts up again and Start.aspx actually gets control. Unfortunately by this time the WebAii framework and the browser are now out of sync. The fix is pretty easy. Add a WaitForUrl after your Click and your Assert such that the code now looks like the example below. Notice that the WaitForUrl uses a tilde "~" to indicate partial matching. See the rest of this chapter to learn more about the Find and Assert statements.

```
' Cause the page to login
Find.ById(Of HtmlInputSubmit)("ctl00_cphMainContent_Login1_LoginButton").Click()
' Wait for the redirect to finish
ActiveBrowser.WaitForUrl("~Start.aspx", True, 5000)
Assert.IsTrue(ActiveBrowser.Url.Contains("Start.aspx"))
```

```
// Cause the page to login
Find.ById<HtmlInputSubmit>("ctl00_cphMainContent_Login1_LoginButton").Click();
// Wait for the redirect to finish
ActiveBrowser.WaitForUrl("~Start.aspx", true, 5000);
Assert.IsTrue(ActiveBrowser.Url.Contains("Start.aspx"));
```

### 14.4.2 Locate Elements

Many of the frequently asked questions on the forum deal with locating elements in the DOM or in the XAML of Silverlight applications. WebAii Testing Framework includes a **Find** *object* with an extensive set of methods for locating a single element or a collection of elements with specific characteristics. These objects and methods begin where the DOM Explorer searches left off.

#### 14.4.2.1 Finding a Single Element

The Find object methods for locating a single element start with "By" + the criteria used to locate the element. The methods include:

- **ByAttributes()**
- **ByContent()**
- **ByCustom()**
- **ByExpression()**
- **ById()**
- **ByName()**
- **ByNodeIndexPath()**
- **ByParam()**
- **ByTagIndex()**
- **ByXPath()**

##### 14.4.2.1.1 Minimal Example

To get started, here's a minimal example that searches for a single element. We have a link to the "Google" web site where the HTML markup contains the fragment below:


```
<a href="http://www.google.com">Google</a>
```

This short example navigates to a sample site that contains the Google link, locates the link and finally clicks the link. In this example we call the **ByContent()** method and pass the string "Google" that we're looking for. The second parameter is a **FindContentType** enumeration member which is set to **TextContent** for this example.

***Notes***

FindContentType can be **InnerText, InnerMarkup, OuterMarkup, TextContent** or **StartTagContent**. Refer back to the Visual Studio Integration chapter, "DOM Explorer" for more on the meaning of these settings.

```
<CodedStep("Find Elements Example")> _
Public Sub MyTestStep()
  ActiveBrowser.NavigateTo("http://training.falafel.com/SampleWebSite/FindElementsSample.htm")
  Dim element As Element = Find.ByContent("Google", FindContentType.TextContent)
  Dim anchor As HtmlAnchor = element.As(Of HtmlAnchor)()
  anchor.Click()
End Sub
```

```
[CodedStep("Find Elements Example")]
public void MyTestStep()
{
   ActiveBrowser.NavigateTo("http://training.falafel.com/SampleWebSite/FindElementsSample.htm");
   Element element = Find.ByContent("Google", FindContentType.TextContent);
   HtmlAnchor anchor = element.As<HtmlAnchor>();
   anchor.Click();
}
```

Did you notice that the Find.ByContent() method returned an **Element** object? "Element" has some basic information common to any testing element such as TextContent and basic methods such as **Focus()**. But the method **Click()** isn't available because some testing elements may not be clickable. Instead you need an object that represents the specific element you're working with. In this case we want to click on an HTML "anchor" tag. Use the Element **As()** method to convert the element to the specific type of object you want to work with. In VB.NET you pass "Of HtmlAnchor" as a parameter. In C#, pass "HtmlAnchor" between angle brackets. Then you can access the HtmlAnchor object directly and use all of its properties and methods.

```
Dim anchor As HtmlAnchor = element.As(Of HtmlAnchor)()
```

```
HtmlAnchor anchor = element.As<HtmlAnchor>();
```

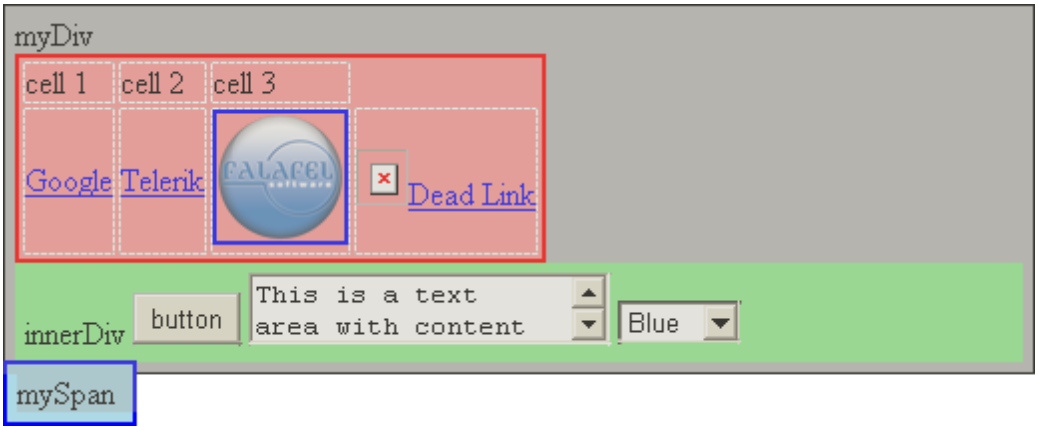
You can eliminate the As() method call by letting the Find method know that you're expecting a HtmlAnchor to be returned. Use the syntax below to have any of the Find methods automatically return the correct type.

```
Dim anchor As HtmlAnchor = Find.ByContent(Of HtmlAnchor)("Google", FindContentType.TextContent)
anchor.Click()
```


```
HtmlAnchor anchor =
   Find.ByContent<HtmlAnchor>("Google", FindContentType.TextContent);
anchor.Click();
```

14.4.2.1.2 Find Methods

Consider the HTML page in the screenshot below. It contains a little of everything including tables, divs, spans, images, links and even broken links.




The HTML markup is listed below for your reference as we go through examples of finding the elements on this page.

```
<div id="myDiv" class="OuterDiv">
myDiv
<table id="infoTable" class="TableStyle">
  <tr>
    <td>cell 1</td>
    <td>cell 2</td>
    <td>cell 3</td>
  </tr>
  <tr>
    <td><a href="http://www.google.com">
      Google</a></td>
    <td><a href="http://www.telerik.com">Telerik</a></td>
    <td>
      <a href="http://www.falafel.com">
      <img src="http://www.falafel.com/images/falafel_logo_64px.png" />
      </a>
    </td>
    <td>
      <a href="http://w.deadlink.c">
      <img src="http://w.deadlink.c/nopicture.png" />Dead Link
      </a>
    </td>
  </tr>
</table>
<div id="innerDiv" class="InnerDiv">
  innerDiv
  <input id="Button1" type="button" value="button" />
  <textarea id="TextArea1" cols="20" name="S1" rows="2">This is a text area with content
  </textarea>
  <select id="Select1" name="Select1">
    <option>Red</option>
    <option selected="selected">Blue</option>
    <option>Green</option>
  </select></div>
</div>
<span id="mySpan" class="SpanStyle">mySpan </span>
```

The examples below show additional Find methods that locate single elements.

### ByAttributes()

The **ByAttributes()** method searches for elements that match attributes in an HTML tag such as "bar=foo","class=myclass","src=~foo.gif","src=!bar". Use =~ for partial values or =! to exclude values. To match multiple attributes pass an array of string

```
Dim table As HtmlTable = Find.ByAttributes(Of HtmlTable)("class=TableStyle")
Dim byAttributeParams() As String = { "class=TableStyle", "id=infoTable" }
' or...
table = Find.ByAttributes(Of HtmlTable)(byAttributeParams)
If table IsNot Nothing Then
  Log.WriteLine(table.ID)
Else
  Log.WriteLine("table not found")
End If
```

```
HtmlTable table = Find.ByAttributes<HtmlTable>("class=TableStyle");
// or...
string[] byAttributeParams = new string[] { "class=TableStyle", "id=infoTable" };
table = Find.ByAttributes<HtmlTable>(byAttributeParams);
if (table != null)
{
   Log.WriteLine(table.ID);
}
else
{
   Log.WriteLine("table not found");
}
```

**From the Forums...**

**Question**: How can I learn the name of attributes that exist in RadControls?

**Answer**: Control wrappers provide important constants related to RadControls. For example, you can use the CssClass constant for each RadControl to locate our wrapper as shown in the code example below.

```
Dim myGrid As RadGrid = Find.ByAttributes(Of RadGrid)("class=" & GridConstants.CssGrid)
```

```
RadGrid myGrid = Find.ByAttributes<RadGrid>("class=" + GridConstants.CssGrid);
```

The naming convention for these objects is the name of the control + "Constants", i.e. TreeViewConstants, ListBoxConstants, ComboBoxConstants, etc.

### ById()

This method allows you to search for an element by its ID. The method is case insensitive. You can also prefix the ID with a tilde "~" to perform a partial search. The example below for instance, retrieves a Table element where the ID is "infoTable".

```
Dim element As Element = Find.ById("~info")
Log.WriteLine(element.OuterMarkup)
```

```
Element element = Find.ById("~info");
Log.WriteLine(element.OuterMarkup);
```

### ByName()

This method lets you perform a case-insensitive search for an element by its name. The example below locates a list box, i.e. HtmlSelect, element, select the first item in the list and then verify that the selected item text is "Red".

```
' Locate the listbox named "Select1"
Dim [select] As HtmlSelect = Find.ByName(Of HtmlSelect)("Select1")
' Select the first item in the list
[select].SelectByIndex(0)
' Verify that the selected item text is "Red"
Assert.IsTrue([select].SelectedOption.Text.Equals("Red"))
```

```
// Locate the listbox named "Select1"
HtmlSelect select = Find.ByName<HtmlSelect>("Select1");
// Select the first item in the list
select.SelectByIndex(0);
// Verify that the selected item text is "Red"
Assert.IsTrue(select.SelectedOption.Text.Equals("Red"));
```

### ByNodeIndexPath()

This method searches using a forward slash delimited list that describes the path to a target element. The path ignores the actual element tags and simply describes the hierarchy leading to that element. The indexes are zero-based, so in an HTML document, the "Head" element is at index zero and the "Body" element is at index "1". Using a subset of the reference HTML from the beginning of this section, let's look for the cell in the table with text "cell 3". The node index path would be "1/0/0/0/0/2" as shown in the HTML fragment below.

```
    . . .
      <head></head>
-->1 <body>
 -->0 <div id="myDiv" class="OuterDiv">
  -->0  <table id="infoTable" class="TableStyle">
   -->0   <tr>
            <td>cell 1</td>
            <td>cell 2</td>
    -->2   <td>cell 3</td>
    . . .
```

By using the ByNodeIndexPath() method and passing the forward slash delimited of indexes we can return the "cell 3" element.

```
Dim indexPathElement As Element = Find.ByNodeIndexPath("1/0/0/0/0/2")
If indexPathElement IsNot Nothing Then
  Log.WriteLine(indexPathElement.TagName & ": " & indexPathElement.TextContent)
End If
```

```
Element indexPathElement = Find.ByNodeIndexPath("1/0/0/0/0/2");
if (indexPathElement != null)
{
   Log.WriteLine(indexPathElement.TagName + ": " +
      indexPathElement.TextContent);
}
```

### ByTagIndex()

This method returns an element by its tag name occurrence. The first parameter is the name of the tag and the second is the occurrence number to retrieve. The example below retrieves the third "Option" tag in the document.


```
Dim [option] As HtmlOption = Find.ByTagIndex(Of HtmlOption)("Option", 2)
Log.WriteLine([option].Text)
```


```
HtmlOption option = Find.ByTagIndex<HtmlOption>("Option", 2);
Log.WriteLine(option.Text);
```

### ByXPath()

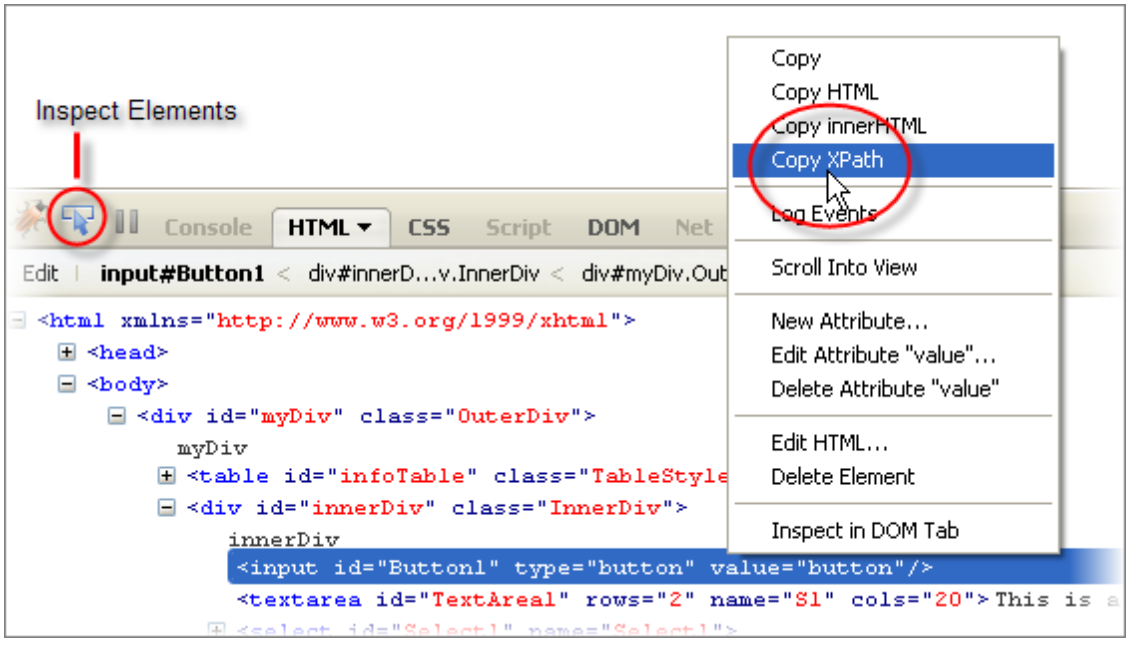
This method returns an element based on an XPath expression.

```vb
' ByXPath
Dim button As HtmlInputButton = Find.ByXPath(Of HtmlInputButton)("//*[@id=""Button1""]")
button.Focus()
```

```cs
// ByXPath
HtmlInputButton button = Find.ByXPath<HtmlInputButton>("//*[@id=\"Button1\"]");
button.Focus();
```

**Tip!**

Remember that you can use the FireFox browser plugin "Firebug" to create an XPath expression for you. You will need to locate, download and install the Firebug plugin. See getfirebug.com for more information and tutorials. When you have it running you can click the Inspect Elements button, then click the mouse on an element in the page to select it in the HTML tab. From there you can right-click and select Copy XPath from the context menu to copy the XPath expression to your clipboard. The screenshot below shows the Copy XPath that created the expression used in the example code above.



### ByCustom()

ByCustom() allows you to pass a custom method as a parameter. This custom method itself takes a parameter of the same type as that returned. In the example below, the custom method knows it has an HtmlImage parameter and looks for an element where the "src" attribute contains the characters "falafel_logo".


```
Dim image As HtmlImage = _
  Find.ByCustom(Of HtmlImage)(Function(img) img.Src.Contains("falafel_logo"))
```


```
HtmlImage image =
    Find.ByCustom<HtmlImage>(img => img.Src.Contains("falafel_logo"));
```

### ByExpression()

What happens if the element you're looking for doesn't have a unique ID or some other way to pinpoint it? The ByExpression() method allows you to follow a chain of "clauses" that describe the element you're searching for. For example, you might be looking for a text area inside two nested "div" elements. Each clause describes a name & value pair. These clauses can be separated by a pipe character "|". The pipe character is interpreted as "then". The example below first looks for an element with an ID = "myDiv", ***then,*** from that "myDiv" element the search looks for an element where the tagname = "div", ***then*** looks for an element where the textcontent contains the string "with content".

Find.ByExpression() can take either a **HtmlFindExpression** or an array of strings directly.

```
' create a series of dependant clauses
Dim findClauses() As String = { "id=myDiv", "|", "tagname=div", "|", "textcontent=~with content" }
' build a find expression
Dim findExpression As New HtmlFindExpression(findClauses)
' find the element using the expression
Dim el As Element = Find.ByExpression(findExpression)
Log.WriteLine(el.OuterMarkup)
```

```
// create a series of dependant clauses
string[] findClauses = new string[]
   { "id=myDiv", "|", "tagname=div", "|", "textcontent=~with content" };
// build a find expression
HtmlFindExpression findExpression = new HtmlFindExpression(findClauses);
// find the element using the expression
Element el = Find.ByExpression(findExpression);
Log.WriteLine(el.OuterMarkup);
```

***Notes***

Expressions also support Silverlight page searches, but instead of HtmlFindExpression, use **XamlFindExpression**.

### ByExpression() with Hierarchy Constraint

You can also place a "hierarchy constraint" on an expression. This can help find an element in a particular relation with some other element.

***Notes***

This is an advanced technique primarily aimed at WebAii developers creating wrappers and translators.

Consider the HTML sample below. There are two "div" elements, "OuterDiv" is the immediate child of the "body" tag and "InnerDiv" where the "body" tag is two elements away. We can find either "div" element based on its offset from the "body".


```
<body>
  <div id="OuterDiv" >
    <div id="InnerDiv" >
  </div>
</body>
```

To use hierarchy constraints you need two find expressions. The first looks for one or more target elements. The second serves as a reference point to the element. The hierarchy constraint defines an offset from the reference element to the target element. The offset can be one or more integers where negative numbers travel up the document towards the parent and positive numbers point downward towards the elements children. For example, "-2" points to the parent's parent, "1" to the immediate child of an element, and "-1,1" points to the first child of the parent.

Using our HTML sample above, lets say we want to find "InnerDiv". First we create a find expression where "tagname=div", and a second expression to get a reference point where "tagname=body". Then we create a **HierarchyConstraint** that takes the reference find expression with an offset of "-2". Finally, the HierarchyConstraint is added to the find expression that's looking for target "div" elements. The ByExpression() method returns the "InnerDiv" element as shown in the code sample below. If we changed the offset to "-1", the "OuterDiv" element would be returned.

```
' retrieve element where multiple possibilities
Dim expression As New HtmlFindExpression("tagname=div")

' retrieve an element to use as a reference point
Dim reference As New HtmlFindExpression("tagname=body")

' build and apply the constraint
Dim constraint As New ArtOfTest.Common.HierarchyConstraint(reference, "-2")
expression.AddHierarchyConstraint(constraint)

' locate and use the element
Dim el2 As Element = Find.ByExpression(expression)
Log.WriteLine(el2.OuterMarkup)
```

```
// retrieve element where multiple possibilities
HtmlFindExpression expression = new HtmlFindExpression("tagname=div");

// retrieve an element to use as a reference point
HtmlFindExpression reference = new HtmlFindExpression("tagname=body");

// build and apply the constraint
ArtOfTest.Common.HierarchyConstraint constraint =
   new ArtOfTest.Common.HierarchyConstraint(reference, "-2");
expression.AddHierarchyConstraint(constraint);

// locate and use the element
Element el2 = Find.ByExpression(expression);
Log.WriteLine(el2.OuterMarkup);
```

Here's a second example that finds the "Blue" option tag when its in the hierarchical relationship to the table element shown in the HTML markup below. The numbers to the left of the listing show the offsets where "0" is the starting point at the target element, -1 is the parent Select element and so on.

```
   <body>
-3   <div id="myDiv" class="OuterDiv">
 1     myDiv
 2     <table id="infoTable" class="TableStyle">
         <tr>
           <td>cell 1</td>
           <td>cell 2</td>
           <td>cell 3</td>
         </tr>
       </table>
-2     <div id="innerDiv" class="InnerDiv">
-1       <select id="Select1" name="Select1">
           <option>Red</option>
 0         <option selected="selected">Blue</option>
           <option>Green</option>
         </select></div>
     </div>
   </body>
```

The code is similar to the last example except that the offset is -3 (up to the outer div), 2 (down to the table child element).

```
' retrieve element where multiple possibilities
Dim target2 As New HtmlFindExpression("tagname=option", "textcontent=Blue")

' retrieve an element to use as a reference point
Dim reference2 As New HtmlFindExpression("tagname=table")

' build and apply the constraint
Dim constraint2 As New ArtOfTest.Common.HierarchyConstraint(reference2, "-3,2")
target2.AddHierarchyConstraint(constraint2)

' locate and use the element
Dim el3 As Element = Find.ByExpression(target2)
Log.WriteLine(el3.OuterMarkup)
```

```
// retrieve element where multiple possibilities
HtmlFindExpression target2 = new HtmlFindExpression("tagname=option", "textcontent=Blue");

// retrieve an element to use as a reference point
HtmlFindExpression reference2 = new HtmlFindExpression("tagname=table");

// build and apply the constraint
ArtOfTest.Common.HierarchyConstraint constraint2 =
   new ArtOfTest.Common.HierarchyConstraint(reference2, "-3,2");
target2.AddHierarchyConstraint(constraint2);

// locate and use the element
Element el3 = Find.ByExpression(target2);
Log.WriteLine(el3.OuterMarkup);
```

**Tip!**

Getting the correct offset can be challenging. If you're having trouble, try...

- Check that both your target and reference find expressions can be called with ByExpression().
- Keep the offsets close to the target element at first and test one element at a time as you refine the search.
- The DOM Explorer can give you some help in visualizing the parents and children of an element.

#### 14.4.2.1.3 Find Operators

For Find methods that use find clauses, here are the available operators and some examples you can use as a reference:

| Operator | Constant | Example | Description |
|---|---|---|---|
| = | Equals | class=TableStyle | class equals "TableStyle" |
| ~ | Contains | class=~Table | class contains "Table" |
| ! | NotContain | class=!Table | class does not contain "Table" |
| ^ | StartsWith | class=^Tab | class starts with "Tab" |
| ? | EndsWith | class=?yle | class ends with "yle" |
| # | RegEx | class=#.able. | class matches the RegEx expression ". able." |

14.4.2.1.4 RadControls Wrappers

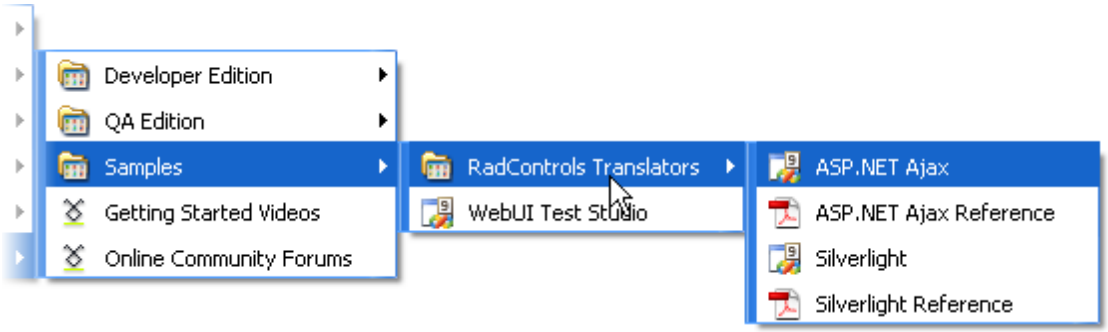
Telerik offers an extensive set of "Wrapper" controls that stand in for RadControls. These controls let you access the control in the testing setting and get at many of the methods and properties of the original control. To retrieve a RadControl "Wrapper", use one of the Find methods and specify the type. In the example below for VB.NET this is done by adding "(Of Telerik.WebAii.Controls.Html.RadTreeView)" just after the ById() call. In C#, follow the ById() method call with angle braces containing the type.

```vb
Const RADCONTROLS_DEMO As String = _
 "http://demos.telerik.com/aspnet-ajax/treeview/examples/functionality/whatsnew/defaultcs.aspx"

' navigate to the elements sample site
ActiveBrowser.NavigateTo(RADCONTROLS_DEMO)

Dim radTreeView As Telerik.WebAii.Controls.Html.RadTreeView = _
Find.ById(Of Telerik.WebAii.Controls.Html.RadTreeView)("RadTreeView1")
```

```cs
const string RADCONTROLS_DEMO =
   "http://demos.telerik.com/aspnet-ajax/treeview/examples/functionality/whatsnew/defaultcs.aspx";

// navigate to the elements sample site
ActiveBrowser.NavigateTo(RADCONTROLS_DEMO);

Telerik.WebAii.Controls.Html.RadTreeView radTreeView =
   Find.ById<Telerik.WebAii.Controls.Html.RadTreeView>("RadTreeView1");
```

**Tip!**

Be sure to check out the example solutions for translators and wrappers in both ASP.NET AJAX and Silverlight RadControls flavors. These solutions have extensive code examples that you can use to borrow for your tests and to learn from. You can find the example solutions in the Start menu under the Telerik menu item for the installed WebUI Test Studio product.



#### 14.4.2.2 Finding Multiple Elements

More than one element can satisfy a search and return a collection of objects. For this reason the Find object has a series of methods pre-pended with "AllBy", i.e. **AllByAttributes()**, **AllByContent()**, **AllByCustom()**, **AllByExpression()**, **AllByTagName()** and **AllByXPath()**. The example below demonstrates the AllByTagName() method and looks for all the links on a page by looking for only HtmlAnchor elements. Using a "ForEach" loop, the example prints the Url and page title for each link to the test log.

```
' Log the path and title of every link in the current browser page
Public Sub LogLinks(ByVal browser As Browser)

  Dim links As IList(Of HtmlAnchor) = browser.Find.AllByTagName(Of HtmlAnchor)("a")
  For Each anchor As HtmlAnchor In links
    browser.NavigateTo(anchor.HRef)
    Log.WriteLine(anchor.HRef & "  Title:""" & browser.PageTitle & """")
  Next anchor
End Sub
```

```
// Log the path and title of every link in the current browser page
public void LogLinks(Browser browser)
{

   IList<HtmlAnchor> links = browser.Find.AllByTagName<HtmlAnchor>("a");
   foreach (HtmlAnchor anchor in links)
   {
      browser.NavigateTo(anchor.HRef);
      Log.WriteLine(anchor.HRef + "  Title:\"" +
         browser.PageTitle + "\"");
   }
}
```

The Find object also has methods to return all controls or elements. The collections returned from these methods can be used in LINQ (see notes following on LINQ) statements. The example below uses the **AllElements()** method to return a collection of every element in the browser window. LINQ is used to filter out elements that are not "Style" tags and that contain text content.

```vb
Public Sub LogElements(ByVal browser As Browser)
  Dim elements As IEnumerable(Of Element) = browser.Find.AllElements()
  Dim elementsWithContent = _
    From el In elements _
    Where (Not el.TextContent.Equals(String.Empty)) _
    Where (Not el.TagName.Equals("style")) _
    Select el

  For Each element As Element In elementsWithContent
    Log.WriteLine(element.TextContent)
  Next element
End Sub
```

```cs
public void LogElements(Browser browser)
{
   IEnumerable<Element> elements = browser.Find.AllElements();
   var elementsWithContent =
      from el in elements
      where !el.TextContent.Equals(String.Empty)
      where !el.TagName.Equals("style")
      select el;

   foreach (Element element in elementsWithContent)
   {
      Log.WriteLine(element.TextContent);
   }
}
```

***Notes***

LINQ (**L**anguage **IN**tegrated **Q**uery) allows .NET languages to apply standard query syntax to all kinds of data including collections. LINQ is a big subject, beyond the scope of this tutorial, but there is plenty of material on the web including:

**101 LINQ Samples**: http://msdn.microsoft.com/en-us/vcsharp/aa336746.aspx

**MSDN: .NET Language-Integrated Query**: http://msdn.microsoft.com/en-us/library/bb308959.aspx

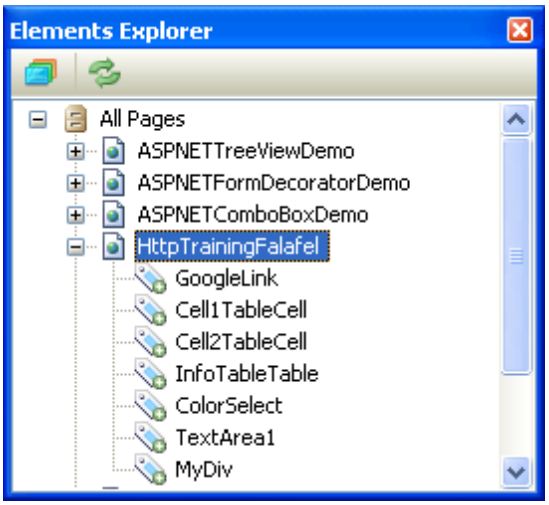
You can also pull back any controls on the page of a particular type using the **Find.AllControls()** syntax shown below. The returned collection can be filtered using LINQ syntax.

```vb
Public Sub LogControls(ByVal browser As Browser)
  Log.WriteLine("Log controls in the current browser page")
  Dim controls As IEnumerable(Of HtmlAnchor) = browser.Find.AllControls(Of HtmlAnchor)()
  Dim elementsWithContent = _
    From ctrl In controls _
    Where (Not ctrl.TextContent.Equals(String.Empty)) _
    Select ctrl

  For Each anchor As HtmlAnchor In elementsWithContent
    Log.WriteLine(anchor.TextContent)
  Next anchor
End Sub
```

```cs
public void LogControls(Browser browser)
{
   Log.WriteLine("Log controls in the current browser page");
   IEnumerable<HtmlAnchor> controls =
      browser.Find.AllControls<HtmlAnchor>();
   var elementsWithContent =
      from ctrl in controls
      where !ctrl.TextContent.Equals(String.Empty)
      select ctrl;

   foreach (HtmlAnchor anchor in elementsWithContent)
   {
      Log.WriteLine(anchor.TextContent);
   }
}
```

### 14.4.2.3 Elements Explorer

Web pages, elements and Silverlight applications defined in the Elements Explorer are "strongly typed" objects that you can use in your coded steps. Consider the screenshot below where the Elements Explorer has a page "HttpTrainingFalafel" that includes "GoogleLink" and "Cell1TableCell".




The test object itself has a Pages property that contains references to all the items in the Elements Explorer. You can access the element properties from the page reference, i.e. Pages.HttpTrainingFalafel.GoogleLink or use the page's **Get()** method. This latter technique allows you to pass a FindExpression that allows complex expressions to pinpoint the element you're looking for. The example below first uses the Get() method to retrieve an element with text content "Google". Then the page is used to access an HTML table cell.

```
<CodedStep("Elements Explorer Example")> _
Public Sub ElementsExplorerExample()
  ActiveBrowser.NavigateTo("http://training.falafel.com/SampleWebSite/FindElementsSample.htm")
  Dim googleLink As Element = Pages.HttpTrainingFalafel.Get("textcontent=google")
  Log.WriteLine(googleLink.OuterMarkup)
  Dim cell As HtmlTableCell = Pages.HttpTrainingFalafel.Cell2TableCell
  Log.WriteLine(cell.TextContent)
End Sub
```

```
[CodedStep("Elements Explorer Example")]
public void ElementsExplorerExample()
{
   ActiveBrowser.NavigateTo(
      "http://training.falafel.com/SampleWebSite/FindElementsSample.htm");
   Element googleLink =
      Pages.HttpTrainingFalafel.Get("textcontent=google");
   Log.WriteLine(googleLink.OuterMarkup);
   HtmlTableCell cell = Pages.HttpTrainingFalafel.Cell2TableCell;
   Log.WriteLine(cell.TextContent);
}
```

**Gotcha!**

Elements used in test steps cannot be deleted. The "Delete" context menu item is disabled whenever an element is used directly in a test step. In coded steps there is no such check. You can delete items that are used in coded steps and no warning will appear until you try to compile the project, then you will receive an error something like the sample below where an element called "Cell2TableCell" is missing from the Elements Explorer.

> *'Framework.Pages.HttpTrainingFalafelPage' does not contain a definition for 'Cell2TableCell' and no extension method 'Cell2TableCell' accepting a first argument of type 'Framework.Pages.HttpTrainingFalafelPage' could be found (are you missing a using directive or an assembly reference?)*

#### 14.4.2.4 Search Scope

You may want to reduce the scope of your search to children of some element, rather than search the entire document. Fortunately, each Element has its own Find and Wait objects so we can start our search much lower in the visual tree. For example, if we have several HTML Table elements on a page and search for cell elements we might return all the cells on the page, regardless of the table they occur in. But if we get a reference to just one of those tables, we can search inside the table for table cell elements. The example below does just that. Using the reference to an HTML table already defined in the elements explorer, you can use the element's Find object to search inside the table for HtmlTableCell controls.

```
<CodedStep("Finding an element using an element as a starting point")> _
Public Sub FindingFromElement()
  ' retrieve an HTML table
  Dim table As HtmlTable = Me.Pages.HttpTrainingFalafel.InfoTableTable
  ' search the table for table cells
  Dim cells As IEnumerable(Of HtmlTableCell) = table.Find.AllControls(Of HtmlTableCell)()
  ' iterate the table cells
  For Each cell As HtmlTableCell In cells
    Assert.AreNotEqual(cell.TextContent, String.Empty)
  Next cell
End Sub
```

```
[CodedStep("Finding an element using an element as a starting point")]
public void FindingFromElement()
{
   // retrieve an HTML table
   HtmlTable table = this.Pages.HttpTrainingFalafel.InfoTableTable;
   // search the table for table cells
   IEnumerable<HtmlTableCell> cells = table.Find.AllControls<HtmlTableCell>();
   // iterate the table cells
   foreach (HtmlTableCell cell in cells)
   {
      Assert.AreNotEqual(cell.TextContent, String.Empty);
   }
}
```

**Tip!**

If your search is performing slowly or bringing back more elements than are needed, consider reducing the scope of the search.

#### 14.4.2.5 jQuery Support

*"jQuery is a lightweight cross-browser JavaScript library that emphasizes interaction between JavaScript and HTML." (www.wikipedia.org)*

You can find elements using the popular **jQuery** syntax, just by adding the **ArtOfTest.WebAii.jQuery** namespace and calling the **Find.jQuery()** method.

*"This was a customer feature request ..., but in case you are a jQuery junkie, you can now search the DOM similar to how you would do it in jQuery with a strongly-typed object model in WebAii. Here is a quick example (make sure to import the ArtOfTest.WebAii.jQuery namespace first)"*


```
Assert.IsTrue(Find.jQuery().id("Button3").IdAttributeValue = "Button3")
Assert.IsTrue(Find.jQuery().attributes("id*=button", "id!=2").first().IdAttributeValue = "Button3")

' Select all textboxes that are in the even rows of the first two tables on the page
Dim allTextBoxes As IList(Of HtmlInputText) = _
Find.jQuery().tag("table").lt(2).descendant().tag("tr").even().text()
```


```
Assert.IsTrue(Find.jQuery().id("Button3").IdAttributeValue == "Button3");
Assert.IsTrue(Find.jQuery().attributes("id*=button", "id!=2").first().IdAttributeValue == "Button3");

// Select all textboxes that are in the even rows of the first two tables on the page
IList<HtmlInputText> allTextBoxes = Find.jQuery().tag("table").lt(2).descendant().tag("tr").even().text();
```

### 14.4.3 Wait for Elements

WebAii Testing Framework supports **Wait** objects that wait for some set of conditions. The conditions vary depending on the technology we're testing against. For example, the **HtmlWait** object can wait for an element to exist in the DOM, or for a set of HTML attributes to exist, a certain content to exist or even a custom set of conditions that you set up yourself. Each method also has a mirror image that waits for the conditions to ***not*** exist, e.g. ForVisibilityNot() that waits for an element to become invisible. In the Silverlight world the **VisualWait** can wait for an element to exist, be visible, be motionless or some custom set of conditions (see "Testing Silverlight Applications" for more information on waiting for Silverlight elements).

#### 14.4.3.1 WaitSync

The base WebAii Test **Wait** object is general purpose, is of type **WaitSync** and has several varieties of the **For()** method. These methods all allow you to add some custom code that tests for a condition that must be true before the wait returns. The example below uses a "generic type parameter". In this case we're looking for an **HtmlSelect**, so the VB.NET example uses the syntax "Of HtmlSelect" and the C# version uses angle braces that enclose the type, i.e. "<HtmlSelect>".

The For() method parameter expects a method name, "IsRedSelected" in the example below. Notice that the For() method only needs the name of the method, not the parameter list. "IsRedSelected" is called for each HtmlSelect in the document until the condition is met.

***Notes***

The parallel to the WaitSync object in Silverlight is the **VisualWait** object. See the "Testing Silverlight Applications" section for more information.

The HtmlSelect parameter in the "IsRedSelected" method has an **AssertSelect()** method that in turn has a **SelectedText()** method. SelectedText() compares the text in the select item against a string and returns "True" if they match. Once a match occurs, the Wait.For() operation is satisfied and returns. The second parameter to the Wait.For() method is "ColorSelect", the actual object we're waiting on. The last Wait.For() parameter is a timeout value expressed in milliseconds. In this example we're waiting on "ColorSelect" for ten seconds.


```
<CodedStep("Wait for Red to be selected from drop down")> _
Public Sub WaitForHtmlElements()
  Const SAMPLE_SITE As String = "http://training.falafel.com/SampleWebSite/FindElementsSample.htm"

  ' navigate to the elements sample site
  ActiveBrowser.NavigateTo(SAMPLE_SITE)

  Dim ColorSelect As HtmlSelect = Pages.HttpTrainingFalafel.ColorSelect
  Wait.For(Of HtmlSelect)(AddressOf IsRedSelected, ColorSelect, 10000)
End Sub

Private Function IsRedSelected(ByVal [select] As HtmlSelect) As Boolean
  Return [select].AssertSelect().SelectedText(ArtOfTest.Common.StringCompareType.Contains, "Red")
End Function
```

```
[CodedStep("Wait for Red to be selected from drop down")]
public void WaitForHtmlElements()
{
   const string SAMPLE_SITE =
      "http://training.falafel.com/SampleWebSite/FindElementsSample.htm";

   // navigate to the elements sample site
   ActiveBrowser.NavigateTo(SAMPLE_SITE);

   HtmlSelect ColorSelect = Pages.HttpTrainingFalafel.ColorSelect;
   Wait.For<HtmlSelect>(IsRedSelected, ColorSelect, 10000);
}

private bool IsRedSelected(HtmlSelect select)
{
   return select.AssertSelect().SelectedText(
      ArtOfTest.Common.StringCompareType.Contains, "Red");
}
```

***Notes***

By default the Wait object's **AutoCheckResult** property is "True", so Wait automatically checks for timeouts or errors once the wait is done. You can actually get the specific result code yourself if you wish by looking at the **SyncWaitResult** enumeration property. SyncWaitResult stores a **WaitResultType** property of **NotSet, ConditionMet, TimedOut, ErrorAbort** and **ElementNotFound**. SyncWaitResult can also contains an Error string property that you can reference if the WaitResultType is ErrorAbort.

### 14.4.3.2 Wait

While Wait.For() is a nice general purpose tool, you'll typically use the more technology and task-specific **HtmlWait** available as an object of an HTML element. HtmlWait methods are much simpler to use and offers a set of methods used to find elements.

- **ForExists(), ForExistsNot()**
- **ForAttributes(), ForAttributesNot()**
- **ForContent(), ForContentNot()**
- **ForCondition()**

These methods closely parallel similar methods for the Find object. Both Find and Wait rely on identifying an element to satisfy the search or the wait condition. We can cover the basics by showing a wait for a particular element to exist. The example methods take no parameters, but you could pass a timeout parameter if you wish.


```
Dim ColorSelect As HtmlSelect = Pages.HttpTrainingFalafel.ColorSelect
ColorSelect.Wait.ForExists()
```


```
HtmlSelect ColorSelect = Pages.HttpTrainingFalafel.ColorSelect;
ColorSelect.Wait.ForExists();
```

Most of the methods also have a mirror "Not" version:


```
ColorSelect.Wait.ForExistsNot()
```


```
ColorSelect.Wait.ForExistsNot();
```

The next few examples will use an HTML snippet that contains a TextArea element attributes for id, columns and rows. The snippet also encloses its own text content.


```
<textarea id="TextArea1" cols="20" name="S1" rows="2">This is a text area with content</
textarea>
```

To wait for an HTML object that has certain attributes, call **Wait.ForAttributes()** and pass an array of name/value pairs. The example below snags the TextArea element that has "rows" and "cols" attributes with particular values.

```
Dim TextArea1 As HtmlTextArea = Pages.HttpTrainingFalafel.TextArea1
Dim attributeParams() As String = { "rows=2", "cols=20" }
TextArea1.Wait.ForAttributes(attributeParams)
```

```
HtmlTextArea TextArea1 = Pages.HttpTrainingFalafel.TextArea1;
string[] attributeParams = new string[] { "rows=2", "cols=20" };
TextArea1.Wait.ForAttributes(attributeParams);
```

The example below waits until the contents of the TextArea are ***not*** "This is a text area with content". Running this example against the HTML snippet above, the test will not continue until the text is modified. You can actually change the text interactively in the browser while the test runs as long as you make the change before the wait times out. Notice that the first parameter is a **FindContentType** enumeration that specifies what part of the element we're looking at and can include values **InnerText**, **InnerMarkup**, **OuterMarkup**, **TextContent** and **StartTagContent**.

```
TextArea1.Wait.ForContentNot(FindContentType.InnerText, _
 "This is a text area with content")
```

```
TextArea1.Wait.ForContentNot(FindContentType.InnerText,
   "This is a text area with content");
```

The **ForCondition()** method lets you "go to town" and make your wait condition be as elaborate as necessary to suit your purposes. Pass ForCondition() a method that will contain your custom logic, an "invertCondition" Boolean that can reverse the custom method's return value, the control to check and a time out value. The custom method takes a control to check and any custom object you want to pass. This custom object could be your own custom class, a primitive type (e.g. string, int, double) or you could pass a null value. For example, here's a class definition that will hold comparison information.

```vb
Public Class MyObject
  Private privateID As String
  Public Property ID() As String
    Get
      Return privateID
    End Get
    Set(ByVal value As String)
      privateID = value
    End Set
  End Property
  Private privateCheckString As String
  Public Property CheckString() As String
    Get
      Return privateCheckString
    End Get
    Set(ByVal value As String)
      privateCheckString = value
    End Set
  End Property
  Private privateRows As Integer
  Public Property Rows() As Integer
    Get
      Return privateRows
    End Get
    Set(ByVal value As Integer)
      privateRows = value
    End Set
  End Property
  Private privateCols As Integer
  Public Property Cols() As Integer
    Get
      Return privateCols
    End Get
    Set(ByVal value As Integer)
      privateCols = value
    End Set
  End Property
End Class
```

```cs
public class MyObject
{
   public string ID { get; set; }
   public string CheckString { get; set; }
   public int Rows { get; set; }
   public int Cols { get; set; }
}
```

...and here's the Wait.ForCondition() method call that consumes "MyObject". We create and populate "MyObject" with values we want to use when checking the condition, then call ForCondition(), passing the method "MyCheckForCondition", "False" to indicate we don't want to invert the condition, the "MyObject" instance and finally a timeout value of five seconds.

In the "MyCheckForCondition()" method we perform some safety checking to make sure the control and custom objects are the types we expect. The type safety checking at the top of "MyCheckForCondition()" makes sure that the code doesn't raise errors when we try to access properties from the control or custom object. If these checks pass, the method continues and can make assumptions about the available properties. Finally, the method returns true only if all the property values in "MyObject" match the values for the control.

```
<CodedStep("Wait for HTML Elements demo")> _
Public Sub WaitForHtmlElements()
  Const SAMPLE_SITE As String = _
"http://training.falafel.com/SampleWebSite/FindElementsSample.htm"

  ' navigate to the elements sample site
  ActiveBrowser.NavigateTo(SAMPLE_SITE)

  ' get a reference to the text area
  Dim TextArea1 As HtmlTextArea = Pages.HttpTrainingFalafel.TextArea1

  ' Create and initialize the custom object
  Dim myObject As New MyObject() With { _
.ID = "TextArea1", .Cols = 20, .Rows = 2, .CheckString = "This is a text area with content"}

  ' wait for the condition, passing the custom method
  TextArea1.Wait.ForCondition(AddressOf MyCheckForCondition, False, myObject, 5000)
End Sub

Public Function MyCheckForCondition( _
ByVal controlToCheck As ArtOfTest.WebAii.Controls.Control, _
ByVal anyCustomObject As Object) As Boolean
  ' convert to actual types
  Dim textArea As HtmlTextArea = TryCast(controlToCheck, HtmlTextArea)
  Dim myObject As MyObject = TryCast(anyCustomObject, MyObject)

  If (textArea Is Nothing) OrElse (myObject Is Nothing) Then
    Return False
  End If

  ' check the data in the custom object against control properties
  Dim result As Boolean = textArea.Text.Equals(myObject.CheckString) AndAlso _
textArea.ID.Equals(myObject.ID) AndAlso textArea.Cols.Equals(myObject.Cols) AndAlso _
textArea.Rows.Equals(myObject.Rows)

  Return result
End Function
```

```
[CodedStep("Wait for HTML Elements demo")]
public void WaitForHtmlElements()
{
   const string SAMPLE_SITE =
      "http://training.falafel.com/SampleWebSite/FindElementsSample.htm";

   // navigate to the elements sample site
   ActiveBrowser.NavigateTo(SAMPLE_SITE);

   // get a reference to the text area
   HtmlTextArea TextArea1 = Pages.HttpTrainingFalafel.TextArea1;

   // Create and initialize the custom object
   MyObject myObject = new MyObject() {
      ID = "TextArea1",
      Cols = 20,
      Rows = 2,
      CheckString = "This is a text area with content"
   };

   // wait for the condition, passing the custom method
   TextArea1.Wait.ForCondition(MyCheckForCondition, false, myObject, 5000);
}

public bool MyCheckForCondition(ArtOfTest.WebAii.Controls.Control controlToCheck,
   Object anyCustomObject)
{
   // convert to actual types
   HtmlTextArea textArea = controlToCheck as HtmlTextArea;
   MyObject myObject = anyCustomObject as MyObject;

   if ((textArea == null) || (myObject == null))
      return false;

   // check the data in the custom object against control properties
   bool result =
      textArea.Text.Equals(myObject.CheckString) &&
      textArea.ID.Equals(myObject.ID) &&
      textArea.Cols.Equals(myObject.Cols) &&
      textArea.Rows.Equals(myObject.Rows);

   return result;
}
```

This is just one example of what can be done with ForCondition(). It's really a catch-all that can use any logic you might need and can leverage any functionality available in the .NET platform. A few other examples that ForCondition() might satisfy are:

- Access your company's database or other data store to get comparison values.
- Access values available from a network.
- Use a third party API (Application Programming Interface) or library to determine the condition results.

#### 14.4.3.3 HtmlWait

HtmlWait extends Wait and adds methods that wait for an element to be visible and for elements with particular styles:

- **ForVisible(), ForVisibleNot()**
- **ForStyles(), ForStylesNot()**

The check for visibility is very simple and takes no parameters (although you can add a timeout if you wish):

```
ColorSelect.Wait.ForVisible()
```

```
ColorSelect.Wait.ForVisible();
```

The wait for styles is a bit more complex, includes an array of name/value pairs describing styles and can include a Boolean flag that indicates if "computed styles" are used. "Computed styles" consider the entire chain of cascading styles, not just the style explicitly set on the element.

Consider this fragment of style and a "div" element that uses that style:

```
<head>
  <title></title>
  <style type="text/css">
    .OuterDiv
    {
      border-color: Black;
      border-width: 1px;
      border-style: solid;
    }
  </style>
</head>
<body>
  <div id="myDiv" class="OuterDiv">
  . . .
```

The code below waits for a div with a particular set of styles. Set up each element of your ForStyles() array with a style attribute name, a colon ":" and then the style attribute value. The example below looks for a particular set of border color, width and style values.

```
Dim borderStyles() As String = { _
"border-color:Black", "border-width:1px", "border-style:solid" }
Dim MyDiv As HtmlDiv = Pages.HttpTrainingFalafel.MyDiv
MyDiv.Wait.ForStyles(borderStyles)
```

```
string[] borderStyles = new string[]
{
   "border-color:Black",
   "border-width:1px",
   "border-style:solid"
};
HtmlDiv MyDiv = Pages.HttpTrainingFalafel.MyDiv;
MyDiv.Wait.ForStyles(borderStyles);
```

**Gotcha!**

The syntax for styles uses a colon ":" to separate the style attribute name from the value, not the equal sign. Be sure to use the correct syntax with the colon delimiter or this method will wait "till the cows come home...", or at least until the method times out.

### 14.4.4 Work With Element Properties

Once you've navigated to a page, located your elements or waited till the elements were in some particular state, you can finally work with the properties of the element. Depending on the object that encapsulates a particular element, you can get or set any available properties to that object. For example, the HtmlTextArea lets you set the Text property to a new value:

```
' get a reference to the text area
Dim TextArea1 As HtmlTextArea = Pages.HttpTrainingFalafel.TextArea1
' set the Text value
TextArea1.Text = "A new value"
```

```
// get a reference to the text area
HtmlTextArea TextArea1 = Pages.HttpTrainingFalafel.TextArea1;
// set the Text value
TextArea1.Text = "A new value";
```

Many of the HTML wrappers will allow you to retrieve tag-specific properties, but may not have any properties that can be set. The example below retrieves a link and records the HtmlAnchor properties.

```vb
' get a link and record HtmlAnchor properties
Dim anchor As HtmlAnchor = Pages.HttpTrainingFalafel.GoogleLink
Const anchorFormat As String = _
"HtmlAnchor Href: {0}  Name: {1}  Target: {2}  Title: {3}"
Log.WriteLine(String.Format( _
anchorFormat, anchor.HRef, anchor.Name, anchor.Target, anchor.Title))
```

```cs
// get a link and record HtmlAnchor properties
HtmlAnchor anchor = Pages.HttpTrainingFalafel.GoogleLink;
const string anchorFormat =
   "HtmlAnchor Href: {0}  Name: {1}  Target: {2}  Title: {3}";
Log.WriteLine(String.Format(
   anchorFormat, anchor.HRef, anchor.Name, anchor.Target, anchor.Title));
```

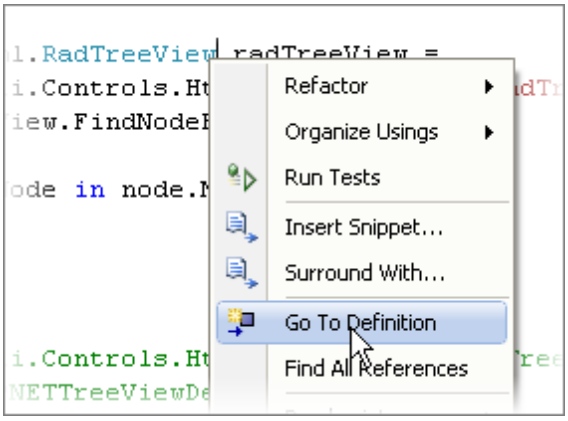
The "wrapper" objects for the RadControls suite have extensive capabilities particular to the type of control. The example below finds a **RadTreeView** control from the RadControls demo page.


The RadTreeView is first located using the **Find.ById()** method. The RadTreeView control has a **FindNodeByText()** method that retrieves the "Zanzibar" node and disables all its child nodes.

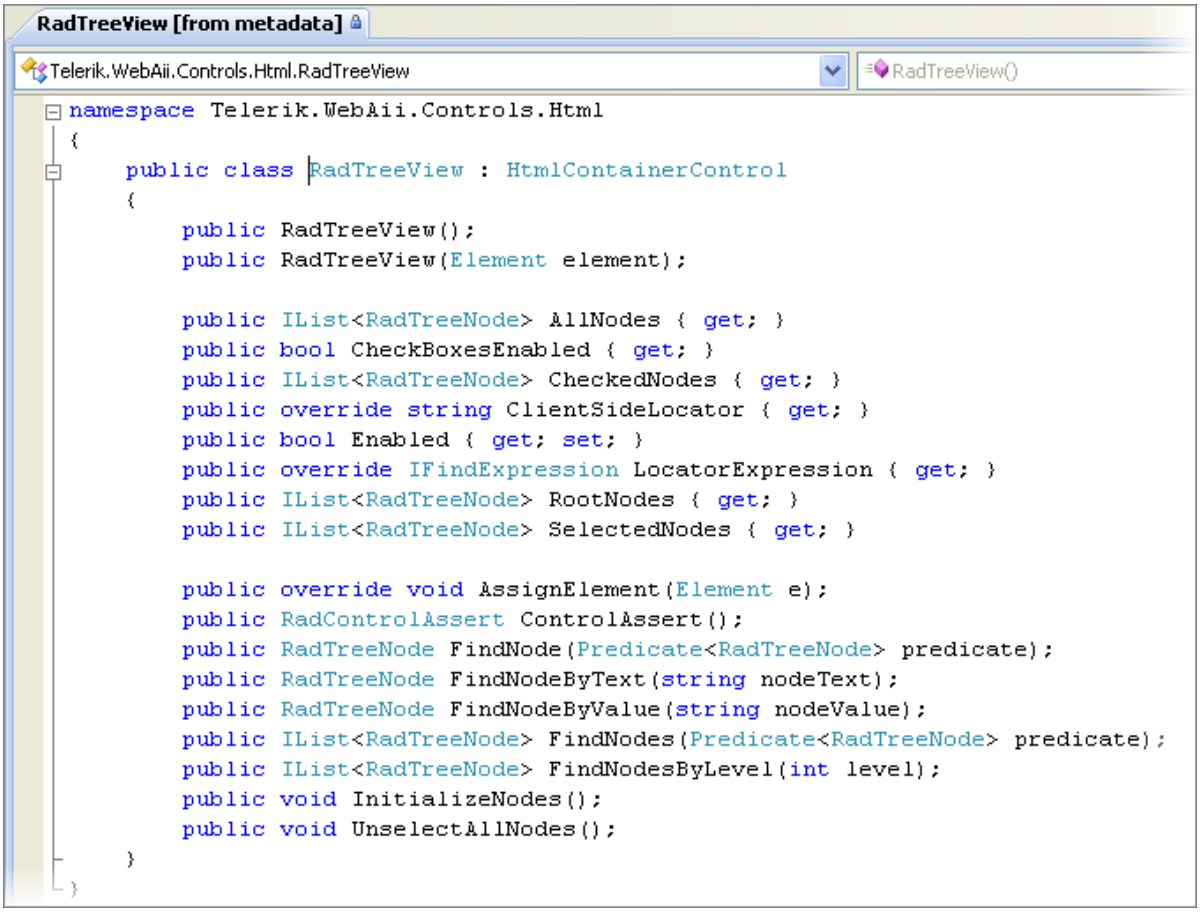
```vb
Const RADCONTROLS_DEMO As String = _
 "http://demos.telerik.com/aspnet-ajax/treeview/examples/functionality/whatsnew/defaultcs.aspx"

' navigate to the elements sample site
ActiveBrowser.NavigateTo(RADCONTROLS_DEMO)

Dim radTreeView As Telerik.WebAii.Controls.Html.RadTreeView = _
Find.ById(Of Telerik.WebAii.Controls.Html.RadTreeView)("RadTreeView1")
Dim node As RadTreeNode = radTreeView.FindNodeByText("Zanzibar")
node.Expand()
For Each childNode As RadTreeNode In node.Nodes
  childNode.Disable()
Next childNode
```

```
const string RADCONTROLS_DEMO =
   "http://demos.telerik.com/aspnet-ajax/treeview/examples/functionality/whatsnew/defaultcs.aspx";

// navigate to the elements sample site
ActiveBrowser.NavigateTo(RADCONTROLS_DEMO);

Telerik.WebAii.Controls.Html.RadTreeView radTreeView =
   Find.ById<Telerik.WebAii.Controls.Html.RadTreeView>("RadTreeView1");
RadTreeNode node = radTreeView.FindNodeByText("Zanzibar");
node.Expand();
foreach (RadTreeNode childNode in node.Nodes)
{
   childNode.Disable();
}
```

To discover the available methods and properties yourself, right-click the class you want to know more about and select "Go To Definition" item from the context menu.




You'll see a list signatures for all the control's methods and properties.




## 14.4.5 Make Assertions

Most of what we've done up till now in this chapter has nothing to do with testing. We've automated the browser, navigated, located elements and automated controls. To actually test something you have to verify that some condition is true or false. We use **assertions** to verify some state or condition in the application. The Microsoft Visual Studio testing API (Application Programming Interface) provides a basic **Assert** object that has methods to help check the state of objects and in particular, to make comparisons between objects. WebAii provides a whole family of assert objects that make setting up these checks easier. **AssertAttribute**, **AssertContent** and **AssertStyle** can be used against a number of HTML objects and for your convenience, there are element-specific assertions **AssertTable**, **AssertSelect** and **AssertCheck.**

#### 14.4.5.1 Assert

The Microsoft Visual Studio testing API **Assert** object is a "work horse" object that can be used if other, special-purpose assertion objects aren't available.

```
ActiveBrowser.NavigateTo("http://training.falafel.com/SampleWebSite/FindElementsSample.htm")

Dim colorSelect As HtmlSelect = Pages.HttpTrainingFalafel.ColorSelect
Assert.IsTrue(colorSelect.Parent(Of HtmlDiv)() IsNot Nothing)
Assert.IsInstanceOfType(colorSelect, GetType(HtmlSelect))
Assert.AreEqual(colorSelect.SelectedIndex, 1)
```

```
ActiveBrowser.NavigateTo(
    "http://training.falafel.com/SampleWebSite/FindElementsSample.htm");

HtmlSelect colorSelect = Pages.HttpTrainingFalafel.ColorSelect;
Assert.IsTrue(colorSelect.Parent<HtmlDiv>() != null);
Assert.IsInstanceOfType(colorSelect, typeof(HtmlSelect));
Assert.AreEqual(colorSelect.SelectedIndex, 1);
```

### From the Forums...

**Question**: Is there is any way to test the order of elements on a web page so that if the developer changes the order of elements on a webpage, that the test will fail?

**Answer**: Yes there are a couple of methods you can use to verify the order of elements. The quick method would be to select the element and craft a verification step that would verify the InnerMarkup or the InnerText equals a specific value. For example, if you have a "<UL>" element that contains a number of "<LI>" elements, you would locate the UL element in DOM explorer, right click on it and select Record Options, select Verification, then construct a verification of the InnerText or the InnerMarkup is exact to some value.

The other method is to add a code behind method that does something like this:

```
<CodedStep("Verify 'InnerMarkup'...")> _
Public Sub webaiitest1_CodedStep()
  Dim ul_1 As HtmlControl = Pages.Demo__CKEditor.ul_1
  ul_1.Wait.ForExists(10000)
  Assert.AreEqual(7, ul_1.BaseElement.ChildNodes.Count, "Selection menu count is incorrect")
  Assert.AreEqual("Choose sample", ul_1.BaseElement.ChildNodes(0).InnerText)
  Assert.AreEqual("Editor with all features", ul_1.BaseElement.ChildNodes(1).InnerText)
  Assert.AreEqual("Interface color", ul_1.BaseElement.ChildNodes(2).InnerText)
  Assert.AreEqual("Multi-language interface", ul_1.BaseElement.ChildNodes(3).InnerText)
  Assert.AreEqual("Custom toolbar", ul_1.BaseElement.ChildNodes(4).InnerText)
  Assert.AreEqual("Skins", ul_1.BaseElement.ChildNodes(5).InnerText)
  Assert.AreEqual("", ul_1.BaseElement.ChildNodes(6).InnerMarkup)
End Sub
```

```
[CodedStep(@"Verify 'InnerMarkup'...")]
public void webaiitest1_CodedStep()
{
   HtmlControl ul_1 = Pages.Demo__CKEditor.ul_1;
   ul_1.Wait.ForExists(10000);
   Assert.AreEqual(7, ul_1.BaseElement.ChildNodes.Count, "Selection menu count is incorrect");
   Assert.AreEqual("Choose sample", ul_1.BaseElement.ChildNodes[0].InnerText);
   Assert.AreEqual("Editor with all features", ul_1.BaseElement.ChildNodes[1].InnerText);
   Assert.AreEqual("Interface color", ul_1.BaseElement.ChildNodes[2].InnerText);
   Assert.AreEqual("Multi-language interface", ul_1.BaseElement.ChildNodes[3].InnerText);
   Assert.AreEqual("Custom toolbar", ul_1.BaseElement.ChildNodes[4].InnerText);
   Assert.AreEqual("Skins", ul_1.BaseElement.ChildNodes[5].InnerText);
   Assert.AreEqual("", ul_1.BaseElement.ChildNodes[6].InnerMarkup);
}
```

### From the Forums...

Here is another Assert example from the forums, just as a reference to how Assert is used when verifying the contents of a RadDatePicker control.

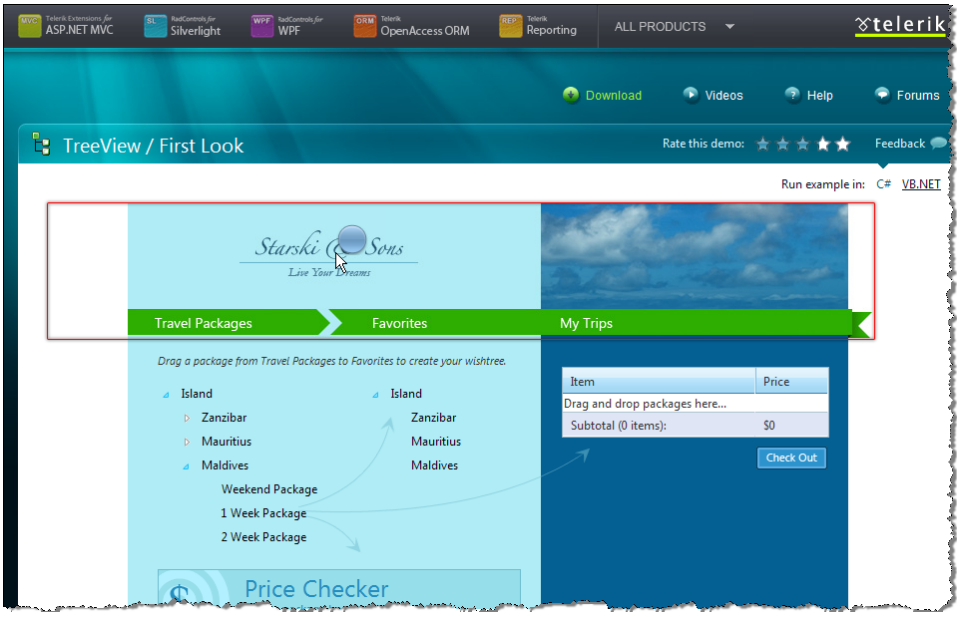
```
Manager.LaunchNewBrowser()
ActiveBrowser.NavigateTo("http://demos.telerik.com/aspnet-ajax/calendar" & _
 "/examples/datepicker/custompopup/defaultcs.aspx")

Dim picker As Telerik.WebAii.Controls.Html.RadDatePicker = _
Find.ById(Of Telerik.WebAii.Controls.Html.RadDatePicker)("RadDatePicker1_wrapper")
CType(picker.DateInput, RadDateInput).InputValue = _
DateTime.Today.AddDays(3).ToShortDateString()
Assert.AreEqual(DateTime.Today.AddDays(3), picker.SelectedDate)
```

```
Manager.LaunchNewBrowser();
ActiveBrowser.NavigateTo(
   "http://demos.telerik.com/aspnet-ajax/calendar" +
   "/examples/datepicker/custompopup/defaultcs.aspx");

Telerik.WebAii.Controls.Html.RadDatePicker picker =
   Find.ById<Telerik.WebAii.Controls.Html.RadDatePicker>("RadDatePicker1_wrapper");
((RadDateInput)picker.DateInput).InputValue =
   DateTime.Today.AddDays(3).ToShortDateString();
Assert.AreEqual(DateTime.Today.AddDays(3), picker.SelectedDate);
```

### 14.4.5.2 AssertAttribute

The AssertAttribute object checks if a particular attribute exists and if the value of the attribute matches a particular value. The **Exists()** method simply takes the name of the attribute and returns True if that attribute exists. The **Value()** method also passes the attribute name as the first parameter but also passes a StringCompareType enumeration value (e.g. Contains, Exact, RegEx, etc.) and the value that the attribute should contain.

The screenshot below shows an image element that forms a banner at the top of a page...




Here is the Html markup for the image element "<src>" tag and its contents.

```
<img src="Img/top.jpg" alt="Starski &amp;amp; Sons Travel Agency" style="float: left; padding-left: 80px" />
```

The example below retrieves the **HtmlImage** representation of the element. Then, using the HtmlImage, the example code checks that the "src" attribute exists and finally that the "src" attribute contains the value "top.jpg".

```
<CodedStep("Make Assertions")> _
Public Sub MakeAssertions()
  ActiveBrowser.NavigateTo("http://demos.telerik.com/aspnet-ajax/treeview/" & _
"examples/functionality/whatsnew/defaultcs.aspx")

  Dim topImage As HtmlImage = Pages.ASPNETTreeViewDemo.TopImage
  topImage.Wait.ForVisible()
  topImage.AssertAttribute().Exists("src")
  topImage.AssertAttribute().Value("src", _
ArtOfTest.Common.StringCompareType.Contains, "top.jpg")
End Sub
```

```
[CodedStep("Make Assertions")]
public void MakeAssertions()
{
   ActiveBrowser.NavigateTo(
      "http://demos.telerik.com/aspnet-ajax/treeview/" +
      "examples/functionality/whatsnew/defaultcs.aspx");

   HtmlImage topImage = Pages.ASPNETTreeViewDemo.TopImage;
   topImage.Wait.ForVisible();
   topImage.AssertAttribute().Exists("src");
   topImage.AssertAttribute().Value("src",
      ArtOfTest.Common.StringCompareType.Contains, "top.jpg");
}
```

### 14.4.5.3 AssertStyle

An Html element can have a mass of styles, too many possibilities to easily track. The **AssertStyle** object has a series of methods that group style possibilities into reasonable subsets. The AssertStyle **ColorAndBackground()** method, for example, takes a **HtmlStyleColorAndBackground** enumeration parameter that lists all the styles you might be looking for, e.g. Color, BackgroundColor, BackgroundImage, etc.

The AssertStyle methods you'll want to use are **Box()**, **ColorAndBackground()**, **Display()**, **Font()**, **List()** and **Text()**. The parameters for each of these methods may only include the type of stye and the string value you're comparing. Or you can call one of the method overloads to use additional parameters:

- **HtmlStyleType**: This can be **Inline** to look only at the style defined in the tag or **Computed** to include everything in the cascaded style chain.
- **StringCompareType**: This parameter defines how the string value is compared to the style and can be **Exact** (i.e. matches exactly and is case sensitive), **Same** (i.e. matches but is not case sensitive), **Contains**, **NotContain**, **StartsWith**, **EndsWith** and **RegEx**.

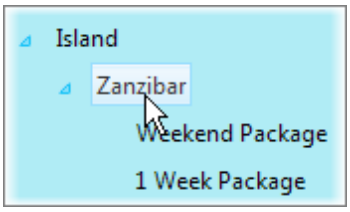
The example below uses the Box() method to verify that the left padding of an element is "80" pixels.


```
topImage.AssertStyle().Box(HtmlStyleBox.PaddingLeft, "80px", HtmlStyleType.Computed, _
ArtOfTest.Common.StringCompareType.Contains)
```


```
topImage.AssertStyle().Box(HtmlStyleBox.PaddingLeft,
   "80px", HtmlStyleType.Computed, ArtOfTest.Common.StringCompareType.Contains);
```

### 14.4.5.4 AssertContent

The **AssertContent** object has a set of methods that all take a compare type (e.g. Exact, Contains, RegEx, etc) and a string to compare against. Each method returns True if the object and the string match, otherwise an **AssertException** is thrown. You can use the AssertContent object to compare against **InnerMarkup**, **InnerText**, **OuterMarkup**, **StartTagContent**, and **TextContent**. The example below uses the AssertContent object from an HTML span element to verify that the TextContent contains "Zanzibar".




```
<CodedStep("Make Assertions")> _
Public Sub MakeAssertions()
  ActiveBrowser.NavigateTo("http://demos.telerik.com/aspnet-ajax/treeview/" _
& "examples/functionality/whatsnew/defaultcs.aspx")
  Dim htmlSpan As HtmlSpan = Pages.ASPNETTreeViewDemo.ZanzibarSpan

  htmlSpan.Wait.ForVisible()
  htmlSpan.AssertContent().TextContent( ArtOfTest.Common.StringCompareType.Contains, "Zanzibar")
End Sub
```

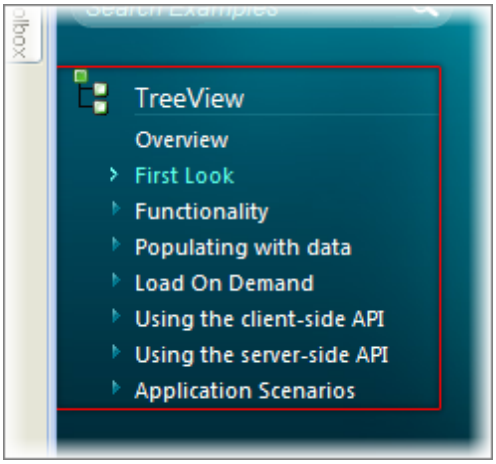
```
[CodedStep("Make Assertions")]
public void MakeAssertions()
{
   ActiveBrowser.NavigateTo(
      "http://demos.telerik.com/aspnet-ajax/treeview/" +
      "examples/functionality/whatsnew/defaultcs.aspx");
   HtmlSpan htmlSpan = Pages.ASPNETTreeViewDemo.ZanzibarSpan;

   htmlSpan.Wait.ForVisible();
   htmlSpan.AssertContent().TextContent(
    ArtOfTest.Common.StringCompareType.Contains, "Zanzibar");
}
```

#### 14.4.5.5 AssertTable

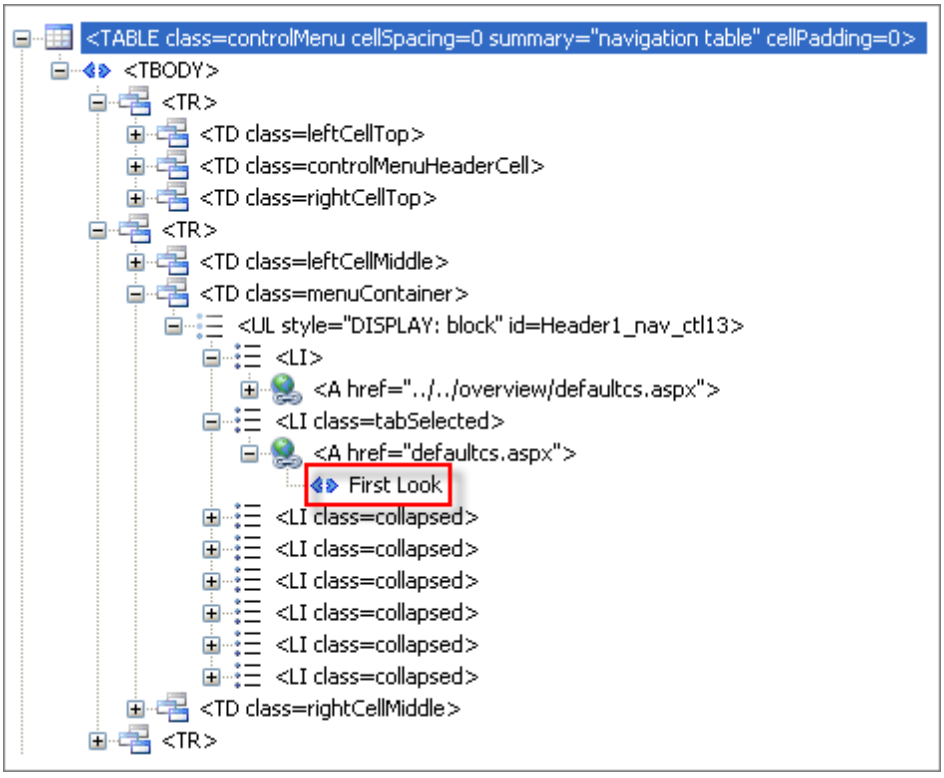
**AssertTable** lets you verify content and the number of columns and rows in an HtmlTable element.

- **ColumnCount(), RowCount():** Pass these methods a **NumberCompareType** of **Equals**, **LessThan**, **GreaterThan**, **LessThanOrEqual**, **GreaterThanOrEqual** or **NotEqual**. The last parameter is the number of rows or columns to compare.
- **ColumnRange(), RowRange():** This method determines if the number of columns or rows is within a certain range. The first parameter is a **NumberRangeCompareType** that can be **InRange** or **OutsideRange**. The last two parameters are the lower and upper ends of the range.
- **Contains():** This method verifies that the table includes certain text somewhere within the tag. It uses a **StringCompareType** that can be **Exact**, **Same**, **Contains**, **NotContain**, **StartsWith**, **EndsWith** and **RegEx**

Consider the RadTreeView in the screenshot below:




This same treeview rendered as an Html table in the browser, produces the elements shown below in the DOM Explorer. Notice the number of row "<TR>" tags, the number of column "<TD>" tags and finally, notice the content "First Look" inside the second row.




The example below verifies that a particular HtmlTable has two or more columns, that the table has at least one, no more than five rows, and that the table contains the string "First Look".

```
ActiveBrowser.NavigateTo("http://demos.telerik.com/aspnet-ajax/treeview/" & _
 "examples/functionality/whatsnew/defaultcs.aspx")

Dim treeViewTable As HtmlTable = Pages.ASPNETTreeViewDemo.TreeViewTable
treeViewTable.AssertTable().ColumnCount(ArtOfTest.Common.NumberCompareType.GreaterThanOrEqual,
treeViewTable.AssertTable().RowRange(ArtOfTest.Common.NumberRangeCompareType.InRange, 1, 5)
treeViewTable.AssertTable().Contains(ArtOfTest.Common.StringCompareType.Contains, "First Look")
```

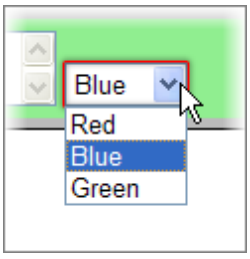
```
ActiveBrowser.NavigateTo(
    "http://demos.telerik.com/aspnet-ajax/treeview/" +
    "examples/functionality/whatsnew/defaultcs.aspx");

HtmlTable treeViewTable = Pages.ASPNETTreeViewDemo.TreeViewTable;
treeViewTable.AssertTable().ColumnCount(ArtOfTest.Common.NumberCompareType.GreaterThanOrEqual,
treeViewTable.AssertTable().RowRange(ArtOfTest.Common.NumberRangeCompareType.InRange, 1, 5);
treeViewTable.AssertTable().Contains(ArtOfTest.Common.StringCompareType.Contains, "First Look");
```

### 14.4.5.6 AssertSelect

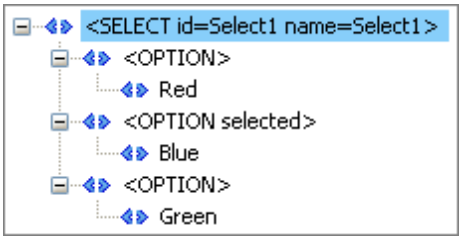
An HtmlSelect element is a drop down list that contains some set of options for the user to select from. The **AssertSelect** object lets you verify the number of items, confirm the selected item and check against the content of the options. AssertSelect has methods for:

- **ItemCountIs()**: This method verifies the total number of options in the list. Pass this method a NumberCompareType (e.g. LessThanOrEqual, etc.) and a numeric value.
- **SelectedIndex()**: This method assures that a particular option in the list was selected by the user. Pass this method a NumberCompareType and a numeric value.
- **SelectedText()**: This method checks the text of the option selected by the user. Pass a StringCompareType (e.g. Same, Exact, etc.) and a string to compare against.
- **SelectedValue()**: This method checks the Value attribute portion of the selected item. Pass a StringCompareType and a string to compare against.
- **TextExists()**, **TextExistsNot()**: This method verifies that a string is contained (or not contained) somewhere in the options list.
- **ValueExists()**, **ValueExistsNot()**: This method verifies that a string is contained (or not contained) somewhere in an options Value attribute.

The screenshot below shows an HtmlSelect with options for "Red", "Blue" and "Green" and where the "Blue" option is selected.




This same element in the DOM Explorer looks like the screenshot below.



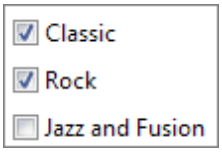

The code example checks that the list has at least, but no more than three items. The selected item index is "1" (the list is zero-based, so the second item index is "1"). The selected item text is "blue", using the case insensitive **Same** string compare type. None of the options in the list have a Value defined, so we can compare the selected value to "String.Empty". Finally, the list of options contains the text "Red".

```vb
ActiveBrowser.NavigateTo("http://training.falafel.com/SampleWebSite/FindElementsSample.htm")

Dim colorSelect As HtmlSelect = Pages.HttpTrainingFalafel.ColorSelect
colorSelect.AssertSelect().ItemsCountIs(ArtOfTest.Common.NumberCompareType.LessThanOrEqual, 3)
colorSelect.AssertSelect().SelectedIndex(ArtOfTest.Common.NumberCompareType.Equals, 1)
colorSelect.AssertSelect().SelectedText(ArtOfTest.Common.StringCompareType.Same, "blue")
colorSelect.AssertSelect().SelectedValue(ArtOfTest.Common.StringCompareType.Exact, String.Empty)
colorSelect.AssertSelect().TextExists("Red")
```

```cs
ActiveBrowser.NavigateTo(
    "http://training.falafel.com/SampleWebSite/FindElementsSample.htm");

HtmlSelect colorSelect = Pages.HttpTrainingFalafel.ColorSelect;
colorSelect.AssertSelect().ItemsCountIs(
   ArtOfTest.Common.NumberCompareType.LessThanOrEqual, 3);
colorSelect.AssertSelect().SelectedIndex(ArtOfTest.Common.NumberCompareType.Equals,
   1);
colorSelect.AssertSelect().SelectedText(ArtOfTest.Common.StringCompareType.Same, "blue");
colorSelect.AssertSelect().SelectedValue(ArtOfTest.Common.StringCompareType.Exact, String.Empty);
colorSelect.AssertSelect().TextExists("Red");
```

### 14.4.5.7 AssertCheck

The "Check" in "AssertCheck" refers to verifying the value of a checkbox. The AssertCheck object has two methods of interest, **IsTrue()** and **IsFalse()**. The example below retrieves an **HtmlInputCheckBox** element for an input check box element labeled "Classic". The example verifies that the input element has been checked.




```vb
ActiveBrowser.NavigateTo( _
"http://demos.telerik.com/aspnet-ajax/formdecorator/examples/default/defaultcs.aspx")

Dim checkboxClassic As HtmlInputCheckBox = _
Pages.ASPNETFormDecoratorDemo.CheckboxClassic
checkboxClassic.Wait.ForExists()
checkboxClassic.AssertCheck().IsTrue()
```

```
ActiveBrowser.NavigateTo(
"http://demos.telerik.com/aspnet-ajax/formdecorator/examples/default/defaultcs.aspx");

HtmlInputCheckBox checkboxClassic =
   Pages.ASPNETFormDecoratorDemo.CheckboxClassic;
checkboxClassic.Wait.ForExists();
checkboxClassic.AssertCheck().IsTrue();
```

## 14.5 Testing Silverlight Applications

The first critical step to allow coded steps against Silverlight applications is to set the RequiresSilverlight parameter of the CodedStep attribute to "True". After that, you need a reference to a SilverlightApp instance so you can access key objects including:

**VisualTree**: Gets the serialized visual tree for this application.

**Find**: Use this to locate elements in the Silverlight visual tree. The SilverlightApp.Find object is the parallel to the Find object used to search HTML documents. SilverlightApp.Find supports some of the same methods as its HTML counterpart including **ByName()** and **ByCustom()**, and also supports Silverlight specific methods **ByAutomationID()**, **ByText()** and **ByType()**.

**Desktop**: The Desktop object used for real Keyboard/Mouse automation .

**OwnerBrowser**, **Host**, **Plugin**: Objects representing the browser, the Html tag hosting this Silverlight App and the actual Silverlight plugin object.

To get a list of Silverlight application instances, call the ActiveBrowser **SilverlightApps()** method. If there is only a single Silverlight application, you can pick off the first instance. You can also reference into the SilverlightApps() array by name or use the name defined in the Elements Explorer.

Here's a short example that demonstrates the use of the RequiresSilverlight parameter, retrieving the SilverlightApp instance and performing a ByName() find.

```
<CodedStep("Find Silverlight Elements Example", RequiresSilverlight := True)> _
Public Sub FindingSilverlightElements()
  Const SILVERLIGHT_DEMO_SITE As String = "http://demos.telerik.com/silverlight/#Home"

  ActiveBrowser.NavigateTo(SILVERLIGHT_DEMO_SITE)
  Dim silverlightApp As SilverlightApp = ActiveBrowser.SilverlightApps()(0)

  Dim frameworkElement As FrameworkElement = silverlightApp.Find.ByName("topLeftLogo")
  Log.WriteLine(frameworkElement.AutomationId)
End Sub
```

```
[CodedStep("Find Silverlight Elements Example", RequiresSilverlight = true)]
public void FindingSilverlightElements()
{
   const string SILVERLIGHT_DEMO_SITE =
      "http://demos.telerik.com/silverlight/#Home";

   ActiveBrowser.NavigateTo(SILVERLIGHT_DEMO_SITE);
   SilverlightApp silverlightApp = ActiveBrowser.SilverlightApps()[0];

   FrameworkElement frameworkElement =
      silverlightApp.Find.ByName("topLeftLogo");
   Log.WriteLine(frameworkElement.AutomationId);
}
```

**Gotcha!**

If you forget to include the RequiresSilverlight parameter and to set its value to "True", your test will fail when you call the SilverlightApps() method. When you see a "Timeout trying to connect to Silverlight App" error like the example below, be sure to check the RequiresSilverlight parameter.

```
Exception thrown executing coded step: '[MySilverlightTest] : Silverlight Example'.
InnerException:
System.TimeoutException: Timeout trying to connect to Silverlight App.
  at ArtOfTest.WebAii.Silverlight.SilverlightApp.WaitUntilExtensionCreated(String extensionCall)
  at ArtOfTest.WebAii.Silverlight.SilverlightApp.Connect(Int32 timeout)
  at ArtOfTest.WebAii.Silverlight.SilverlightApp.Connect()
  at ArtOfTest.WebAii.Silverlight.SilverlightAppsList.get_Item(Int32 index)
  at Framework.FindElements.MySilverlightTest() in C:\Courseware\Projects\CS\Framework\FindElements.aii
```

### 14.5.1 Finding Silverlight Elements

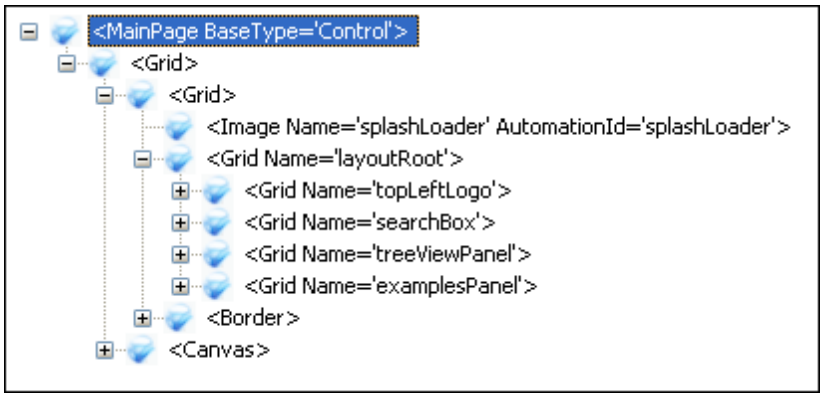
While the HTML "Find" object is from the ArtOfTest.WebAii.Core namespace, the Silverlight Find object is a VisualFind descendant from the ArtOfTest.WebAii.Silverlight namespace. The syntax for using the object is similar. These Find methods return FrameworkElement, the base type in Silverlight that represents a visual element.

- **ByAutomationId()**, **AllByAutomationId()**: Searches the visual tree for elements with a specific automation id. The example below returns a Silverlight Button with an AutomationId of "searchButton" and asserts that the width is "16".

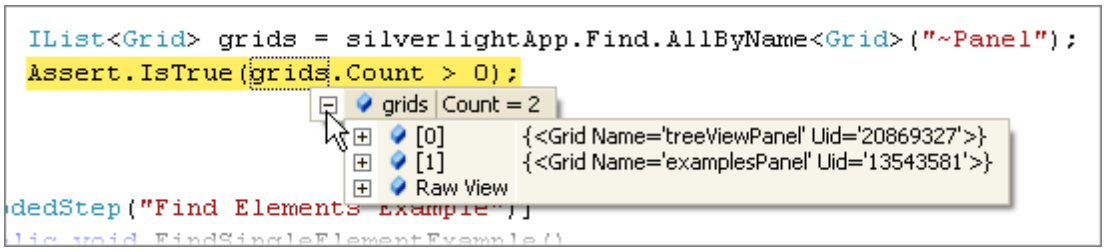
```
Dim searchButton As Button = silverlightApp.Find.ByAutomationId(Of Button)("searchButton")
Assert.AreEqual(searchButton.Width, 16)
```

```
Button searchButton =
   silverlightApp.Find.ByAutomationId<Button>("searchButton");
Assert.AreEqual(searchButton.Width, 16);
```

- **ByName()**, **AllByName()**: We can search for one or more Silverlight elements that match a name or part of a name and we can also filter by the type of control. Given the DOM Explorer in the screenshot below, two of the Grid elements contain "Panel" in the Name attribute.




We can use the AllByName() method, entering "Grid" between the angle braces to specify the control type and "~Panel" to get every element that contains "Panel". The screenshot of the test running in the debugger (see the "Debugging" chapter for more information) shows that we return two Grids that match those we found in the DOM Explorer. The code in both VB.NET and C# is listed below that returns the list of Grid elements and then verifies that the count of Grids is greater than zero.




```
Dim grids As IList(Of Grid) = silverlightApp.Find.AllByName(Of Grid)("~Panel")
Assert.IsTrue(grids.Count > 0)
```

```
IList<Grid> grids = silverlightApp.Find.AllByName<Grid>("~Panel");
Assert.IsTrue(grids.Count > 0);
```

- **ByText()**, **AllByText()**: These methods look for any TextBlock elements that match a given text string. The example below looks for all TextBlock elements that contain "RadControls" and then asserts that one or more elements were returned.

```
Dim elements As IList(Of TextBlock) = silverlightApp.Find.AllByText("~RadControls")
Assert.IsTrue(elements.Count > 0)
```

```
IList<TextBlock> elements = silverlightApp.Find.AllByText("~RadControls");
Assert.IsTrue(elements.Count > 0);
```

- **ByType()**, **AllByType()**: These methods look for elements of a given type. The example below looks for all Image elements.

```
Dim images As IList(Of Image) = silverlightApp.Find.AllByType(Of Image)()
Assert.IsTrue(images.Count > 0)
```

```
IList<Image> images = silverlightApp.Find.AllByType<Image>();
Assert.IsTrue(images.Count > 0);
```

- **ByCustom()**, **AllByCustom()**: These methods find elements using custom logic that you design. AllByCustom(), for instance, allows you to pass a custom method as a parameter. In the example below, the custom method has a signature that passes a FrameworkElement as a parameter and returns a Boolean. You can populate the custom method with any logic that works for you as long as the signature matches. Return "True" for any element you want to include in the search results. The example returns all elements where the Opacity property is smaller than "0.5".

```vb
<CodedStep("Find Silverlight Elements Example", RequiresSilverlight := True)> _
Public Sub FindingSilverlightElements()
  Const SILVERLIGHT_DEMO_SITE As String = "http://demos.telerik.com/silverlight/#Home"

  ActiveBrowser.NavigateTo(SILVERLIGHT_DEMO_SITE)
  Dim silverlightApp As SilverlightApp = ActiveBrowser.SilverlightApps()(0)

  Dim faintElements As IList(Of FrameworkElement) = silverlightApp.Find.AllByCustom(AddressOf CustomLo
  Assert.IsTrue(faintElements.Count = 0)
End Sub

Public Function CustomLogic(ByVal element As FrameworkElement) As Boolean
  Return element.Opacity < 0.5
End Function
```

```cs
[CodedStep("Find Silverlight Elements Example", RequiresSilverlight = true)]
public void FindingSilverlightElements()
{
   const string SILVERLIGHT_DEMO_SITE =
      "http://demos.telerik.com/silverlight/#Home";

   ActiveBrowser.NavigateTo(SILVERLIGHT_DEMO_SITE);
   SilverlightApp silverlightApp = ActiveBrowser.SilverlightApps()[0];

   IList<FrameworkElement>  faintElements = silverlightApp.Find.AllByCustom(CustomLogic);
   Assert.IsTrue(faintElements.Count == 0);
}

public bool CustomLogic(FrameworkElement element)
{
   return element.Opacity < 0.5;
}
```

### Gotcha!

Custom methods can also help you bypass "problem" elements or parts of the document that you want to ignore. The example below skips any element that has a XAML tag that reads "#text#".

```vb
Public Function CustomLogic(ByVal element As FrameworkElement) As Boolean
  If element.XamlTag.Equals("#text#") Then
    Return False
  End If
  Return element.Opacity < 0.5
End Function
```

```cs
public bool CustomLogic(FrameworkElement element)
{
   if (element.XamlTag.Equals("#text#"))
   {
      return false;
   }
   return element.Opacity < 0.5;
}
```

### 14.5.2 Find Strategies

The VisualFind object used in Silverlight element searches has a **Strategy** property used to set the behavior of the Find object when searching. Strategy can be:

- **AlwaysWaitForElementsVisible**: Wait for the element to be visible.
- **WhenNotVisibleReturnElementProxy**: Return the element proxy when not visible or null. An element proxy "stands in" for a real element when used in a Wait operation. The proxy contains information on how to find the element in the visual tree.
- **WhenNotVisibleReturnNull**: Return null when the element is not visible or null.
- **WhenNotVisibleThrowException**: Throw an exception when the element is null.

Here is an example of Strategy in use that looks for an element that doesn't exist. Due to the Strategy setting of "WhenNotVisibleReturnNull", the Find ByName() method returns null and the test succeeds. If the Strategy had been set to **WhenNotVisibleThrowException** the test would have failed.

```
<CodedStep("Test Find Strategy", RequiresSilverlight := True)> _
Public Sub UseFindStrategy()
  Const SILVERLIGHT_DEMO_SITE As String = "http://demos.telerik.com/silverlight/#Home"

  ActiveBrowser.NavigateTo(SILVERLIGHT_DEMO_SITE)
  Dim silverlightApp As SilverlightApp = ActiveBrowser.SilverlightApps()(0)

  silverlightApp.Find.Strategy = FindStrategy.WhenNotVisibleReturnNull
  Dim frameworkElement As FrameworkElement = silverlightApp.Find.ByName("invalidElement")
  If frameworkElement Is Nothing Then
    Log.WriteLine("element is not visible")
  End If
End Sub
```

```
[CodedStep("Test Find Strategy", RequiresSilverlight = true)]
public void UseFindStrategy()
{
   const string SILVERLIGHT_DEMO_SITE =
    "http://demos.telerik.com/silverlight/#Home";

   ActiveBrowser.NavigateTo(SILVERLIGHT_DEMO_SITE);
   SilverlightApp silverlightApp = ActiveBrowser.SilverlightApps()[0];

   silverlightApp.Find.Strategy =
      FindStrategy.WhenNotVisibleReturnNull;
   FrameworkElement frameworkElement =
      silverlightApp.Find.ByName("invalidElement");
   if (frameworkElement == null)
      Log.WriteLine("element is not visible");
}
```

### 14.5.3 Wait for Elements

Waiting for Silverlight elements is more challenging than handling standard web or even AJAX based applications. The wait operation not only needs to take existence and visibility into account, the wait should also let us know if the element is being animated. The **VisualWait** object takes care of all these situations, and also allows you to craft a custom method for those really sticky situations where the stock methods won't do.

VisualWait has methods **ForExists()**, **ForExistsNot()**, **ForVisible()**, **ForVisibleNot()**, **ForNoMotion()** and **For()**. The **ForNoMotion()** method waits for an element to cease animating. You can just pass a number of milliseconds that the element must be stationary before the method returns. Optionally you can pass the number of milliseconds to wait before starting to check if the element has stopped moving, and you can also pass a timeout value.

The example below demonstrates retrieving a TextBlock, although we could be using any Silverlight element. Then we perform a series of Wait operations. First, **ForExists()** tests that the element is present on the page, then **ForVisible()** ensures that the element is able to be seen. The ForNoMotion() method waits half a second before checking the element's motion, then waits for the element to be stationary for a entire second. If the element isn't stationary an entire second, the method times out after thirty seconds.

The last method in this example demonstrates passing a custom method to **Wait.For()**. This method should take a Silverlight FrameWorkElement and return a Boolean. The custom method in the example checks that the ActualHeight property is greater than five.


```
<CodedStep("Waiting for Silverlight Elements Example", RequiresSilverlight := True)> _
Public Sub WaitForSilverlightElements()
  Const SILVERLIGHT_DEMO_SITE As String = "http://demos.telerik.com/silverlight/#Home"

  ActiveBrowser.NavigateTo(SILVERLIGHT_DEMO_SITE)
  Dim silverlightApp As SilverlightApp = ActiveBrowser.SilverlightApps()(0)

  Dim frameworkElement As FrameworkElement = silverlightApp.Find.ByName("topLeftLogo")
  frameworkElement.Wait.ForExists()
  frameworkElement.Wait.ForVisible()
  frameworkElement.Wait.ForNoMotion(500, 1000, 30000)
  frameworkElement.Wait.For(AddressOf MyCustomSilverlightWait)
End Sub

Public Function MyCustomSilverlightWait(ByVal element As FrameworkElement) As Boolean
  Return element.ActualHeight > 5
End Function
```

```
[CodedStep("Waiting for Silverlight Elements Example", RequiresSilverlight = true)]
public void WaitForSilverlightElements()
{
   const string SILVERLIGHT_DEMO_SITE =
      "http://demos.telerik.com/silverlight/#Home";

   ActiveBrowser.NavigateTo(SILVERLIGHT_DEMO_SITE);
   SilverlightApp silverlightApp = ActiveBrowser.SilverlightApps()[0];

   FrameworkElement frameworkElement =
      silverlightApp.Find.ByName("topLeftLogo");
   frameworkElement.Wait.ForExists();
   frameworkElement.Wait.ForVisible();
   frameworkElement.Wait.ForNoMotion(500, 1000, 30000);
   frameworkElement.Wait.For(MyCustomSilverlightWait);
}

public bool MyCustomSilverlightWait(FrameworkElement element)
{
   return element.ActualHeight > 5;
}
```

## 14.6 Automating the Browser

Earlier we looked at navigating the browser, but really you can automate any button you might click in the browser and many other actions not directly available from the browser user interface. The rich browser automation API includes all the usual interface browser actions such as forward, back, stop (cancel), refresh and close. You can also resize and position the browser window, toggle full screen mode, minimize, maximize and even scroll the browser content.

### Clearing the Browser Cache

At the start of your test you may need to set the browser to a known state. This may involve clearing the temp files cache, history or cookies. Use the Browser **ClearCache()** method and pass one of the **BrowserCacheType** enumeration members:


```
ActiveBrowser.ClearCache(BrowserCacheType.TempFilesCache)
```


```
ActiveBrowser.ClearCache(BrowserCacheType.TempFilesCache);
```

### Getting Browser Information

If you need to take browser differences into account you can use the **BrowserType** enumeration to get the general browser flavor, e.g. Internet Explorer, FireFox, etc and the **Version** property to get the specific browser edition.

```
Log.WriteLine("Browser Type: " & ActiveBrowser.BrowserType.ToString())
Log.WriteLine("Browser Version: " & ActiveBrowser.Version)

Select Case ActiveBrowser.BrowserType

  Case BrowserType.InternetExplorer
      ' handle IE specific scenarios
      Exit Select
  Case BrowserType.FireFox
      ' handle FF specific scenarios
      Exit Select
  Case BrowserType.Safari
      ' handle Safari specific scenarios
      Exit Select
End Select
```

```
Log.WriteLine("Browser Type: " + ActiveBrowser.BrowserType.ToString());
Log.WriteLine("Browser Version: " + ActiveBrowser.Version);

switch (ActiveBrowser.BrowserType)
{

   case BrowserType.InternetExplorer:
     {
       // handle IE specific scenarios
       break;
     }
   case BrowserType.FireFox:
     {
       // handle FF specific scenarios
       break;
     }
   case BrowserType.Safari:
     {
       // handle Safari specific scenarios
       break;
     }
};
```

### Automate the Browser User Interface

Use the **NavigateTo(), GoBack()**, **GoForward(), Refresh()** and **Stop()** methods to automate the browser user interface, effectively mimicking the browser tool bar. The example below navigates between two different web sites, then uses the back and forward to move between the web sites and print the **PageTitle** for each location. Finally, the **Stop()** method is called to cancel the browser's current action.

```
Const TELERIK_DEMOS As String = _
 "http://demos.telerik.com/aspnet-ajax/controls/examples/default/defaultcs.aspx"

Const TELERIK_TESTING As String = _
"http://www.telerik.com/products/web-testing-tools.aspx"

ActiveBrowser.NavigateTo(TELERIK_DEMOS)
Log.WriteLine("Navigated to " & ActiveBrowser.PageTitle)
ActiveBrowser.NavigateTo(TELERIK_TESTING)
Log.WriteLine("Navigated to " & ActiveBrowser.PageTitle)
ActiveBrowser.GoBack()
Log.WriteLine("Navigated back to " & ActiveBrowser.PageTitle)
ActiveBrowser.GoForward()
Log.WriteLine("Navigated forward to " & ActiveBrowser.PageTitle)
ActiveBrowser.Stop()
Log.WriteLine("Stop the browser's current navigation action")
```

```
const string TELERIK_DEMOS =
   "http://demos.telerik.com/aspnet-ajax/controls/examples/default/defaultcs.aspx";

const string TELERIK_TESTING =
   "http://www.telerik.com/products/web-testing-tools.aspx";

ActiveBrowser.NavigateTo(TELERIK_DEMOS);
Log.WriteLine("Navigated to " + ActiveBrowser.PageTitle);
ActiveBrowser.NavigateTo(TELERIK_TESTING);
Log.WriteLine("Navigated to " + ActiveBrowser.PageTitle);
ActiveBrowser.GoBack();
Log.WriteLine("Navigated back to " + ActiveBrowser.PageTitle);
ActiveBrowser.GoForward();
Log.WriteLine("Navigated forward to " + ActiveBrowser.PageTitle);
ActiveBrowser.Stop();
Log.WriteLine("Stop the browser's current navigation action");
```

### Reloading the Browser Window and DOM Tree

Use the **Refresh()** method to reload the browser. Call the **RefreshDomTree()** to update the view of the DOM when items may have changed since loading the page.

**From the Forums...**

**Question**: I can't find a list box element that I can see exists. Why can't I find this element?

**Answer:** If there is any dynamic changes to the DOM when displaying the List Box, you can call **ActiveBrowser.RefreshDomTree();** to update the Frameworks view of the DOM when the List Box becomes visible as in:

```
ActiveBrowser.RefreshDomTree()
Dim listBox1 As RadListBox = Find.ById(Of RadListBox)("ctl00_DefaultContent_lstAvailableFields")
Dim item_Session As RadListBoxItem = listBox1.Items(2)
```

```
ActiveBrowser.RefreshDomTree();
RadListBox listBox1 = Find.ById<RadListBox>("ctl00_DefaultContent_lstAvailableFields");
RadListBoxItem item_Session = listBox1.Items[2];
```

You can do likewise with the frames using **Frames.RefreshAllDomTrees();**

## Manipulating the Browser Window

You can fully control the browser window size and position as well as scroll the browser content within the window. The example below first toggles up to full screen and back again, then minimizes and maximizes the browser window. The last section of code in the example resizes and repositions the browser window, then scrolls the browser contents.


```
Log.WriteLine("Toggle to Full Screen")
ActiveBrowser.ToggleFullScreen()
Log.WriteLine("Toggle back from Full Screen")
ActiveBrowser.ToggleFullScreen()

Log.WriteLine("Minimize the window")
ActiveBrowser.ContentWindow.Minimize()
Log.WriteLine("Maximize the window")
ActiveBrowser.ContentWindow.Maximize()

Log.WriteLine("Resize browser to 300x300 and place 10 pixels from upper right")
ActiveBrowser.ResizeContent(10, 10, 300, 300)
Log.WriteLine("Scroll browser 50 pixels to the right and down")
ActiveBrowser.ScrollBy(50, 50)
Log.WriteLine("Scroll browser 50 pixels to the left and up")
ActiveBrowser.ScrollBy(-50, -50)
```


```
Log.WriteLine("Toggle to Full Screen");
ActiveBrowser.ToggleFullScreen();
Log.WriteLine("Toggle back from Full Screen");
ActiveBrowser.ToggleFullScreen();

Log.WriteLine("Minimize the window");
ActiveBrowser.ContentWindow.Minimize();
Log.WriteLine("Maximize the window");
ActiveBrowser.ContentWindow.Maximize();

Log.WriteLine("Resize browser to 300x300 and place 10 pixels from upper right");
ActiveBrowser.ResizeContent(10, 10, 300, 300);
Log.WriteLine("Scroll browser 50 pixels to the right and down");
ActiveBrowser.ScrollBy(50, 50);
Log.WriteLine("Scroll browser 50 pixels to the left and up");
ActiveBrowser.ScrollBy(-50, -50);
```

### Browser Events

You can be notified when the browser DOM tree is refreshed and when the browser is closed by subscribing to the **DomRefreshed** and **Closing** events respectively. The example below iterates the BrowserType enumeration, skipping any unsupported browser types, launches a browser for each browser type, saves off a reference to the instance of the browser and finally hooks up a Closing event handler. A second "For Each" loop iterates the open browsers and closes them all.

The Closing event handler is defined at the end of the code listing and simply logs that the close occurred.

```
<CodedStep("Assorted Browser methods")> _
Public Sub BrowserTricks()
  Dim openedBrowsers As List(Of Browser) = New List(Of Browser)()
  For Each browserType As BrowserType In System.Enum.GetValues(GetType(BrowserType))
    ' skip unsupported browsers
    If (browserType Is BrowserType.NotSet) OrElse (browserType Is BrowserType.Designer) Then
      Continue For
    End If
    Log.WriteLine("Launch a new browser instance: " & browserType.ToString())
    Manager.LaunchNewBrowser(browserType)
    openedBrowsers.Add(ActiveBrowser)
    ' hook the browser closing event to a handler
    AddHandler ActiveBrowser.Closing, AddressOf ActiveBrowser_Closing
  Next browserType

  For Each browser As Browser In openedBrowsers
    Log.WriteLine("Shut down: " & browser.BrowserType.ToString())
    browser.Close()
  Next browser
End Sub

Private Sub ActiveBrowser_Closing(ByVal sender As Object, ByVal e As EventArgs)
  Log.WriteLine("Browser closing for " & (TryCast(sender, Browser)).BrowserType.ToString())
End Sub
```

```
[CodedStep("Assorted Browser methods")]
public void BrowserTricks()
{
   List<Browser> openedBrowsers = new List<Browser>();
   foreach (BrowserType browserType in Enum.GetValues(typeof(BrowserType)))
   {
      // skip unsupported browsers
      if ((browserType == BrowserType.NotSet) || (browserType == BrowserType.Designer))
         continue;
      Log.WriteLine("Launch a new browser instance: " + browserType.ToString());
      Manager.LaunchNewBrowser(browserType);
      openedBrowsers.Add(ActiveBrowser);
      // hook the browser closing event to a handler
      ActiveBrowser.Closing += new EventHandler(ActiveBrowser_Closing);
   }

   foreach (Browser browser in openedBrowsers)
   {
      Log.WriteLine("Shut down: " + browser.BrowserType.ToString());
      browser.Close();
   }
}

void ActiveBrowser_Closing(object sender, EventArgs e)
{
   Log.WriteLine("Browser closing for " + (sender as Browser).BrowserType.ToString());
}
```

## 14.7 Walk Through

In this walk through you can put into practice some of the techniques that have been presented so far including navigating, locating elements, waiting for elements and making assertions. The walk through navigates to a web page with a "Wizard" interface. The example fills out information and clicks "Next" to move through the steps, verifying content along the way.


1) From the Visual Studio choose **File > New > Project...** This will display the "New Project" dialog.

2) Define a new test project:

a) In the "New Project" dialog, select the "Test" project type for your language (C# or VB).

b) Select the "Test Project" template.

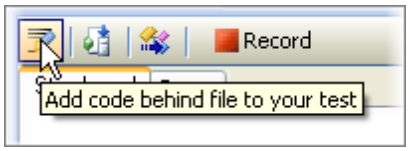
c) Enter a descriptive name for the test project, a location path and a Solution name.

d) Click **OK** to create the new test project.

3) From the Solution Explorer, right-click the test project and select **Add > New Test...** from the context menu. This will display the "Add New Test" dialog.

4) In the "Add New Test" dialog, select the "WebAii Test" template, name the test "Wizard.aii" and click **OK** to create the test.

5) Click the **Add Code Behind** button to create the code behind file.




6) In the code behind file add a new WizardTest() method, compete with the **CodedStep** attribute:

```
<CodedStep("Test a Wizard dialog")> _
Public Sub WizardTest()

End Sub
```

```
[CodedStep("Test a Wizard dialog")]
public void WizardTest()
{

}
```

7) Add a constant to hold the path to the web page:

```
Const WIZARD_URL As String = _
"http://demos.telerik.com/aspnet-ajax/tabstrip/" & _
"examples/applicationscenarios/wizard/defaultcs.aspx"
```

```
const string WIZARD_URL =
   "http://demos.telerik.com/aspnet-ajax/tabstrip/" +
   "examples/applicationscenarios/wizard/defaultcs.aspx"
```

8) Add code to launch a new browser and navigate to the page. Optionally, you can add a Log.WriteLine() to trace the course of execution in your log file.

```
Log.WriteLine("Navigate to the page")
Manager.LaunchNewBrowser()
ActiveBrowser.NavigateTo(WIZARD_URL)
```

```
Log.WriteLine("Navigate to the page");
Manager.LaunchNewBrowser();
ActiveBrowser.NavigateTo(WIZARD_URL);
```

9) Add code to find the RadTabStrip control. The **RadTabStrip** wrapper has an **AllTabs** collection that we can index into. Get a reference to the first tab and make an assertion that the first tab is selected, using the **Assert.AreEqual()** method. Again, you can use the Log.WriteLine() method to track where you're at in the test.

```
Log.WriteLine("Get a reference to the tab strip and verify we're on the first tab")
Dim tabStrip As RadTabStrip = Find.ById(Of RadTabStrip)("RadTabStrip1")
Assert.AreEqual(True, tabStrip.AllTabs(0).Selected)
```

```
Log.WriteLine("Get a reference to the tab strip and verify we're on the first tab");
RadTabStrip tabStrip = Find.ById<RadTabStrip>("RadTabStrip1");
Assert.AreEqual(true, tabStrip.AllTabs[0].Selected);
```

10) Add the code below to get references to the "First" Html input text box and set the **Value** property to "First". Then get a reference to the "Last" Html input text box and enter "Last" as the Value property. Finally, get a reference to the Html input check box and call the **Check()** method to set the check.

```
Log.WriteLine("Fill in first, last and check the terms checkbox")
Dim firstName As HtmlInputText = Find.ById(Of HtmlInputText)("~firstNameTextBox")
firstName.Value = "First"
Dim lastName As HtmlInputText = Find.ById(Of HtmlInputText)("~lastNameTextBox")
lastName.Value = "Last"
Dim agree As HtmlInputCheckBox = Find.ById(Of HtmlInputCheckBox)("~termsCheckBox")
agree.Check(True, True)
```

```
Log.WriteLine("Fill in first, last and check the terms checkbox");
HtmlInputText firstName = Find.ById<HtmlInputText>("~firstNameTextBox");
firstName.Value = "First";
HtmlInputText lastName = Find.ById<HtmlInputText>("~lastNameTextBox");
lastName.Value = "Last";
HtmlInputCheckBox agree = Find.ById<HtmlInputCheckBox>("~termsCheckBox");
agree.Check(true, true);
```

***Notes***

Notice that the Find.ById() methods above use the "~", i.e. "contains" operator.

11)Find the "Next" button using the **Find.ById()** method and call the returned HtmlInputButton **Click()** method. Call the **Wait.For()** method to make sure that the second tab has shown up.

Notice that we're passing a custom method (a Microsoft supplied "Func" type in fact), the tab strip and the particular tab we want to wait for. Later, we'll code the **TabSelected** method.

```
Log.WriteLine("Click Next button and wait")
Dim buttonPersonalNext As HtmlInputButton = _
Find.ById(Of HtmlInputButton)("PersonaluserControl_nextButton")
buttonPersonalNext.Click()
Wait.For(Of RadTabStrip, RadTab)(TabSelected, tabStrip, tabStrip.AllTabs(1), 10000)
```

```
Log.WriteLine("Click Next button and wait");
HtmlInputButton buttonPersonalNext =
   Find.ById<HtmlInputButton>("PersonaluserControl_nextButton");
buttonPersonalNext.Click();
Wait.For<RadTabStrip, RadTab>(TabSelected, tabStrip, tabStrip.AllTabs[1], 10000);
```

12)Again, get a reference to the "Next" button and call its **Click()** method. Call the **Wait.For()** method to make sure that the third tab has shown up.

```
Log.WriteLine("Click Next button again and wait")
Dim buttonEducationNext As HtmlInputButton = _
Find.ById(Of HtmlInputButton)("EducationuserControl_nextButton")
buttonEducationNext.Click()
Wait.For(Of RadTabStrip, RadTab)(TabSelected, tabStrip, tabStrip.AllTabs(2), 10000)
```

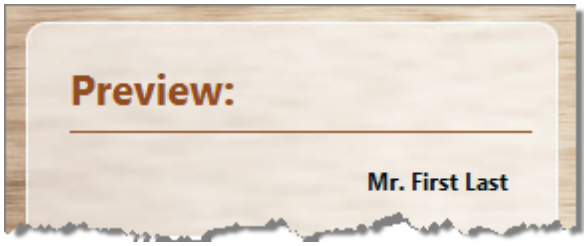
```
Log.WriteLine("Click Next button again and wait");
HtmlInputButton buttonEducationNext =
   Find.ById<HtmlInputButton>("EducationuserControl_nextButton");
buttonEducationNext.Click();
Wait.For<RadTabStrip, RadTab>(TabSelected, tabStrip, tabStrip.AllTabs[2], 10000);
```

13)Use the **Find.By()** method to get the "Finish" button and again call its **Click()** method.

```vb
Log.WriteLine("Click the finish button")
Dim buttonFinish As HtmlInputButton = _
 Find.ById(Of HtmlInputButton)("ProfessionaluserControl_nextButton")
buttonFinish.Click()
```

```cs
Log.WriteLine("Click the finish button");
HtmlInputButton buttonFinish =
Find.ById<HtmlInputButton>("ProfessionaluserControl_nextButton");
buttonFinish.Click();
```

14) In this last step of the method, use the **Find.ByAttributes** method to first get the "previewWrapper" DIV element, then get the HtmlSpan that contains the "preview" first name label. Finally, wait for the label to contain the text content "First".




```vb
Log.WriteLine("Get the first name span verify content")
Dim previewWrap As HtmlDiv = Find.ByAttributes(Of HtmlDiv)("class=previewWrapper")
Dim firstNameSpan As HtmlSpan = _
previewWrap.Find.ByAttributes(Of HtmlSpan)("id=~firstNameLabel")
firstNameSpan.Wait.ForContent(FindContentType.TextContent, "First")
```

```cs
Log.WriteLine("Get the first name span verify content");
HtmlDiv previewWrap = Find.ByAttributes<HtmlDiv>("class=previewWrapper");
HtmlSpan firstNameSpan = previewWrap.Find.ByAttributes<HtmlSpan>("id=~firstNameLabel");
firstNameSpan.Wait.ForContent(FindContentType.TextContent, "First");
```

15) Earlier, we coded a Wait.For() to accept a method called "TabSelected". We need to code the "TabSelected" method now. The signature of the method should return a Boolean. The first parameter to the method should be a **RadTabStrip** type (from the Telerik.WebAii.Controls.Html namespace). The second parameter should be a **RadTab** type, (also from the Telerik.WebAii.Controls.Html namespace). The method returns the RadTab **Selected** property value.

```
' Custom method used in Wait.For()
Public Function TabSelected(ByVal tabStrip As RadTabStrip, ByVal tab As RadTab) As Boolean
  Return tab.Selected
End Function
```

```
// Custom method used in Wait.For()
public bool TabSelected(RadTabStrip tabStrip, RadTab tab)
{
   return  tab.Selected;
}
```

16)Execute the test. The test should navigate to the wizard page, fill in some values, click "Next" through the wizard and finally, verify the "First" name in the "Preview" label. All steps should pass.

## 14.8 Wrap Up

In this chapter you learned how to perform many common automated testing operations in code to work with both Html and Silverlight based elements.

First, you learned how to automate the browser, starting with browser navigation to web pages using both complete and relative Urls. You also learned how to handle browser redirection.

You learned how to locate both single elements and collections of web page elements using the Find object for the test itself and the Find object for individual elements. To fine-tune your searches you learned about the available Find operators. You learned how to reference elements defined in the Elements Explorer. You also learned how jQuery is used to find elements.

You learned how to pause test execution to wait for element existence, content, attributes, visibility, motion and custom conditions.

You worked with properties of "wrapper" objects, including the wrappers for RadControls.

Finally, you used "Assert" objects to verify conditions on the page.

# Part XV

Data Driven Testing

# 15 Data Driven Testing

## 15.1 Objectives

In this chapter you will learn how to drive tests using the built-in data grid. You will also learn how to access data from external spreadsheet, XML and database data sources. You will learn how to access and modify data connections. Finally, you will learn how to access the test Data object in code.

**Find the projects for this chapter at...**

\Courseware\Projects\<CS|VB>\DataDriven\DataDriven.sln

## 15.2 Overview

Using the WebUI Test Studio user interface alone, you can drive tests with data from the built-in grid. You can also use an external data source such as XML, CSV spreadsheet file or database table. Your database can be anything that Visual Studio can hook up to, which is virtually unlimited and includes Oracle, MS SQL, Access and ODBC.

***Notes***

Not all of these database types are right out of the Visual Studio box. For some database types, you may need to do some research and install a "data source provider".

What we mean by "driving" a test with data is that we let the test know where a table of data exists and then use the data wherever required in the test. For example, if we have a test of a login dialog, the data be a user name and password. If we were driving a test of several web searches, we could supply the value to search for, the value being compared and even the page that performs the search.

Both the built-in grid and external data sources all drive tests the same way, i.e. a one-way trip straight through the test, one iteration of the test per row of data. During each iteration, the test can access columns from a single row of the data. For example, consider the table of user names and passwords below. The second iteration of the test can access the user name "nuygen" and password "@lmost".

| | User Name | Password |
|---|---|---|
| | bsmith | xxbox!! |
| ----> | nuygen | @lmost |
| | nigelt | fl@r3 |

There are two parts to driving a test with data:

- Define a connection. This lets the test know where the data is. You need to define a "connection string" that tells the test what kind of data is needed and the specifics of how you're going to get at that data. The only exception to this rule is the built-in grid. WebUI Test Studio already knows about the built-in grid data, so you don't have to define a connection string in this case.
- Access the data. You need to identify some data item that you want to use. Data is assigned to properties in the test through a process called "**binding**". Binding lets the test know what column should be used and where. For example, we can get a value from the "User Name" column and assign it to a text box in a login dialog.

In the following sections you'll learn how to build several different flavors of connection strings and how to bind particular columns to properties in the test.

***Notes***

Without using code you cannot loop or branch the testing logic.

**From the Forums...**

**Question**: I've created a test to Boundary Test a series of fields on a Page, by entering a value in one field and clicking the save button. Now the web page acts slightly different in several cases:-

- If the data is in the boundaries it just saves the page.

- If the data is below the lower boundary it displays a Dialog Box with a message.

- If the data is above the upper boundary it displays a Dialog Box with a different message.

Currently I have recorded entering a value and clicking Save. Now if I data-drive this recording, it will work whenever the data is valid (lower boundary; lower boundary + 1; mid boundary; upper boundary - 1; upper boundary) - when no dialog is displayed.

But how should I cope with the two cases (lower boundary - 1 and upper boundary + 1), WITHOUT going to Code Behind?

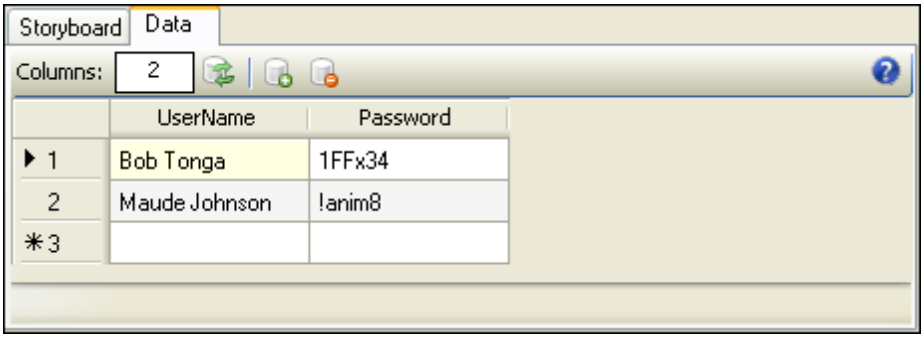
**Answer**: In tests where you are expecting some of the data rows to make the web page react normally and some of the data rows to cause errors on the web page, code behind is your only option. There is no conditional type branching in WebUI Test Studio without using code.

What you can do instead is separate your two (or three or four) testing scenarios into separate tests. Have one test in which you expect all the data rows to not generate errors. Have another test in which all the data rows should generate error code type 1. Have yet another test in which all the data rows generate error code type 2, etc.

One advantage to this approach is that you separate your individual test cases into individual test scripts which can be run independently on and as needed bases rather than having to go through every single row just to verify error type 1 is generated at the right time.

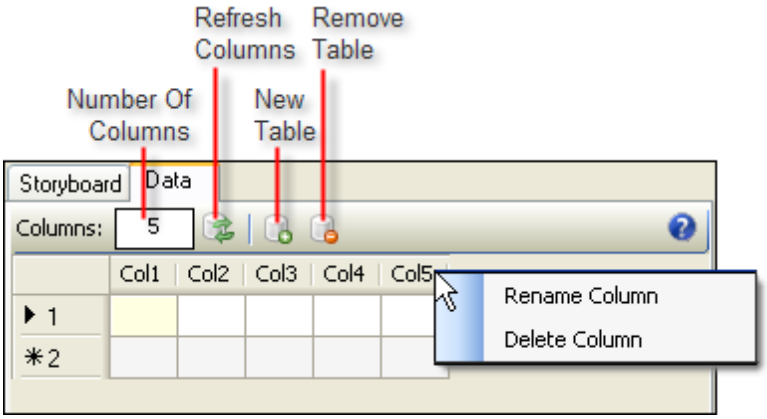
## 15.3 The Built-In Grid

The Data Tab tab allows you to build simple, ad-hoc, data-driven tests without needing to connect to an external data source, such as the login example data shown below. You can find the Data Tab inside the Test Tab next to the Storyboard Tab.




The main point is that this functionality is very simple and is not expected to connect to an external database, have multiple tables or provide fine tune control over looping or branching.

The interface for the Data Tab tab allows you to change the **Number Of Columns** in the table, **Refresh Columns** to reflect the current number of columns, create a **New Table** (this option over-writes any previous table) and to **Remove** the **Table**. Right-clicking the column headings displays a context menu that lets you rename and delete columns.



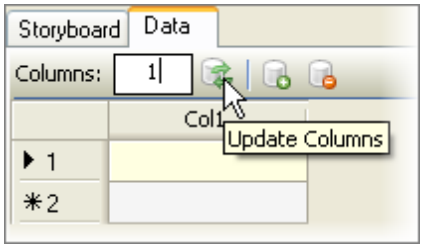
### 15.3.1 Walk Through

This example extends the Walk Through project in the "Getting Started" chapter. Instead of checking against a single hard-coded string "WebAii", this test will check against several different strings from the data table.

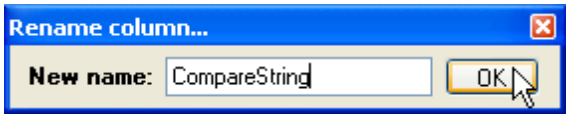
1) Start with the "Getting Started, Walk Through" project or a copy.

2) In the Data Tab tab, click the **New Data Table** button ( ).

3) Enter "1" in the **Columns** edit box and click the **Update Columns** button. Click **OK** to accept and close the confirmation dialog.

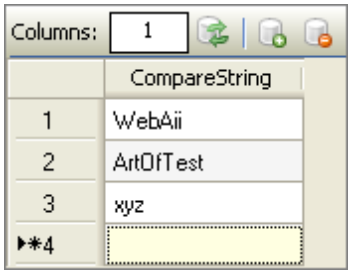



4) Right-click "Col1" and select "Rename Column" from the context menu. Enter "CompareString" as the new column name and click **OK** to close the Rename column... dialog.



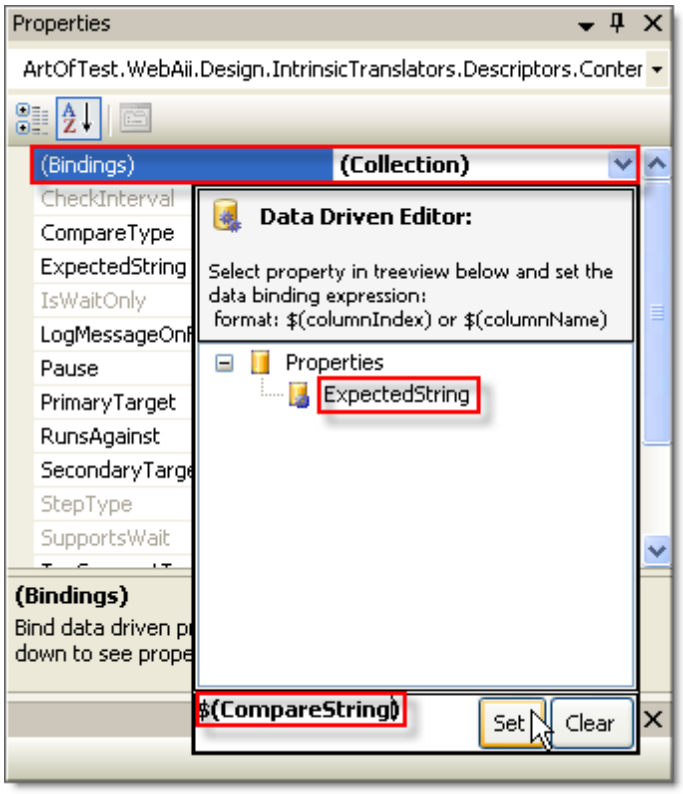

5) In the first row of the table, enter "WebAii" in the "CompareString" column and press **Enter**.

6) Enter two more "CompareString" rows with text "ArtOfTest" and "xyz". The data table should now look like the screenshot below:




7) In the Steps Tab, select the last test step "Verify 'InnerText' 'Contains'...".

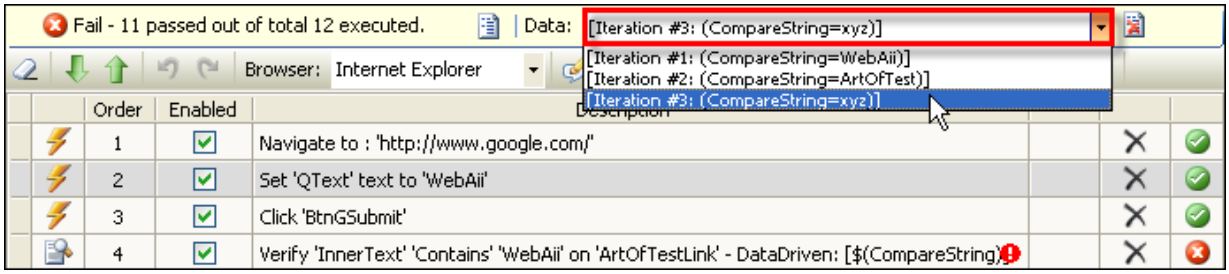
8) In the Properties pane, locate the "(Bindings)" property and click the arrow button to open the drop down menu. Select the **ExpectedString** property, enter "$(CompareString)" in the value entry text box and click the **Set** button.




In the Steps Tab, the test step description is changed to add "DataDriven" and "$CompareString" as shown in the screenshot below.


9) Click the Steps Tab Quick Execute button (  ) to run the test. Notice that the test will run three times, once for every row of data in your table.

10) View the summary results in the Steps Tab which reads "Fail - 11 passed out of total 12 executed". There were 4 test steps, executed three times, one for each data row. Locate the drop down list next to the "Data:" label, drop down the list and select the third iteration. The data "WebAii" and "ArtOfTest", in the first two iterations, both existed in the element text you were testing. The data "xyz" in the last iteration did not exist in the element text and so the test failed.

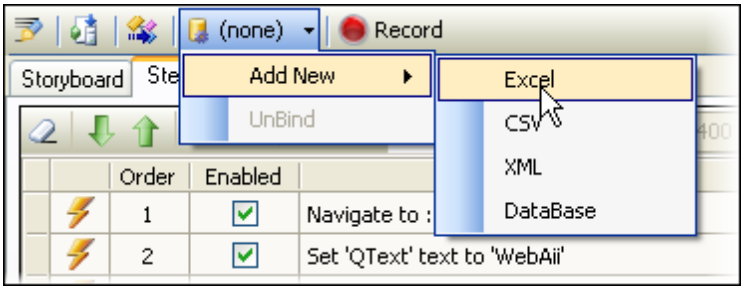


11)If you read the log you'll see the data used to drive each iteration:

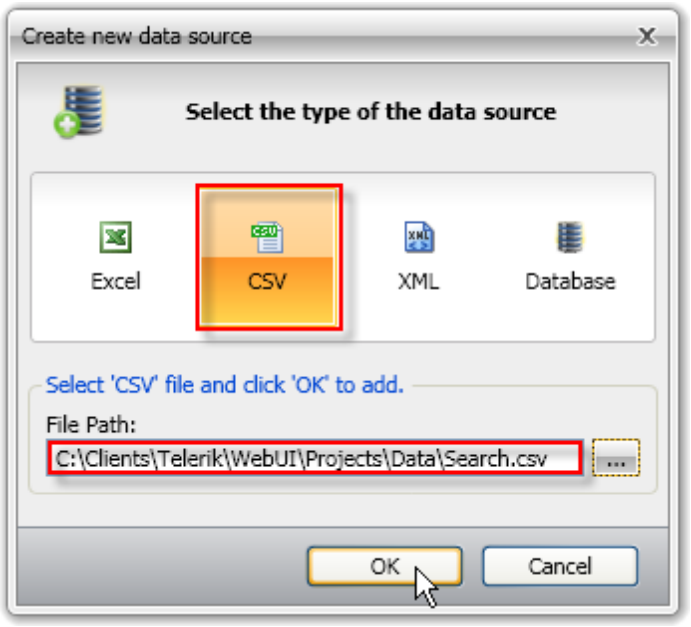
## 15.4 Connecting to External Data

You're not restricted to the built-in grid. You can get at any external data available using Microsoft's connectivity mechanisms. Out of the box, you can connect to standard "*.csv" spreadsheet files, Excel "*.xls" files , XML files and database tables. The database tables can include MS SQL, Oracle, Access and ODBC. The connectivity options are not limited to these few choices. For all practical purposes, you can connect to any data that you're likely to find.

To connect to data use the database button to drop down a menu that allows you to "UnBind" an existing database connection or to add a connection (see the screenshot below).



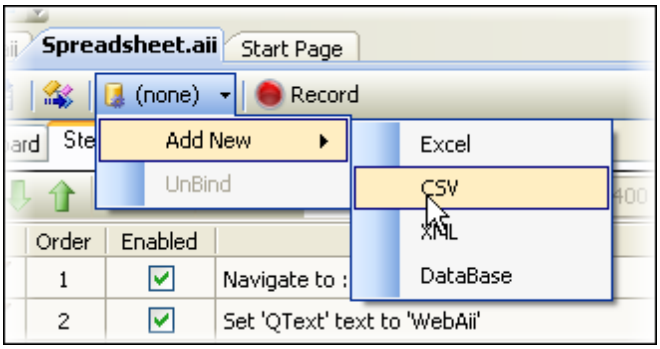

Clicking any of the database connection possibilities (i.e. Excel, CSV, XML or Database), brings up the "Create new data source" dialog. The user interface for the dialog will differ slightly based on the type of data source you choose.



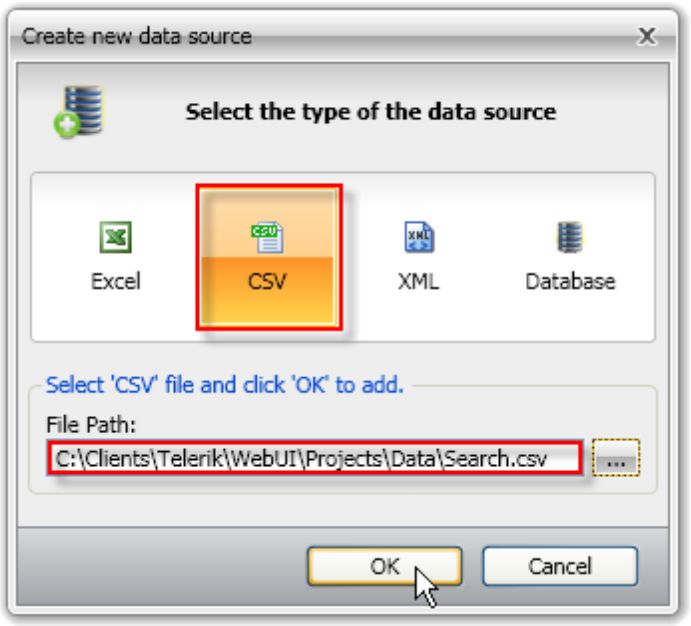
### 15.4.1 Spreadsheet Files

We can drive the entire test using an external spreadsheet using either a Comma Separated Value file (*.csv) or an Excel format file (*.xls, *.xlsx). Both type of files can be created in Excel. *.csv files can actually be created in any spreadsheet application or directly in Notepad if you follow the *.csv formatting conventions (each line with comma separated values, the same number of commas in each line). If you supply a spreadsheet with the same column names as the built-in grid example, you don't have to change the test steps.

1. Click the database button to drop down the list of connection choices. Click **Add New > CSV** from the drop down menu. This action will display the "Create new data source" dialog.




2. In the "Create new data source" dialog, select the "CSV" data source type icon. Use the browse button ... to locate a "*.csv" file and click the **OK** button. In this example, the "Search.csv" file has the same data as the built-in grid example.

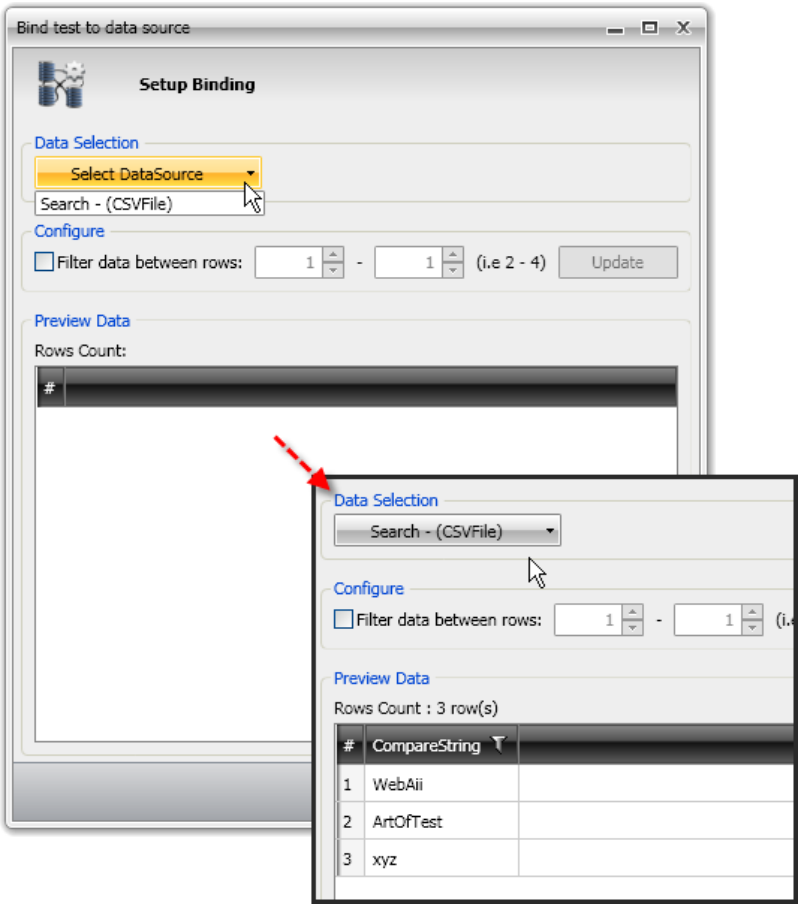


3. A confirmation dialog asking if you want to bind appears. Click the **Yes** button to continue.


4. The "Bind test to data source" dialog displays. Click the **Select DataSource** drop down list and select the "Search - (CSVFile)" item. Your data will display in the Preview Data section of the dialog. Click the **OK** button to complete the binding.

***Notes***

Also notice in the "Configure" section of the dialog that you can select the "Filter data between rows" check box, enter the starting and ending row number and click the Update button. This update limits the number of rows being bound to the test.



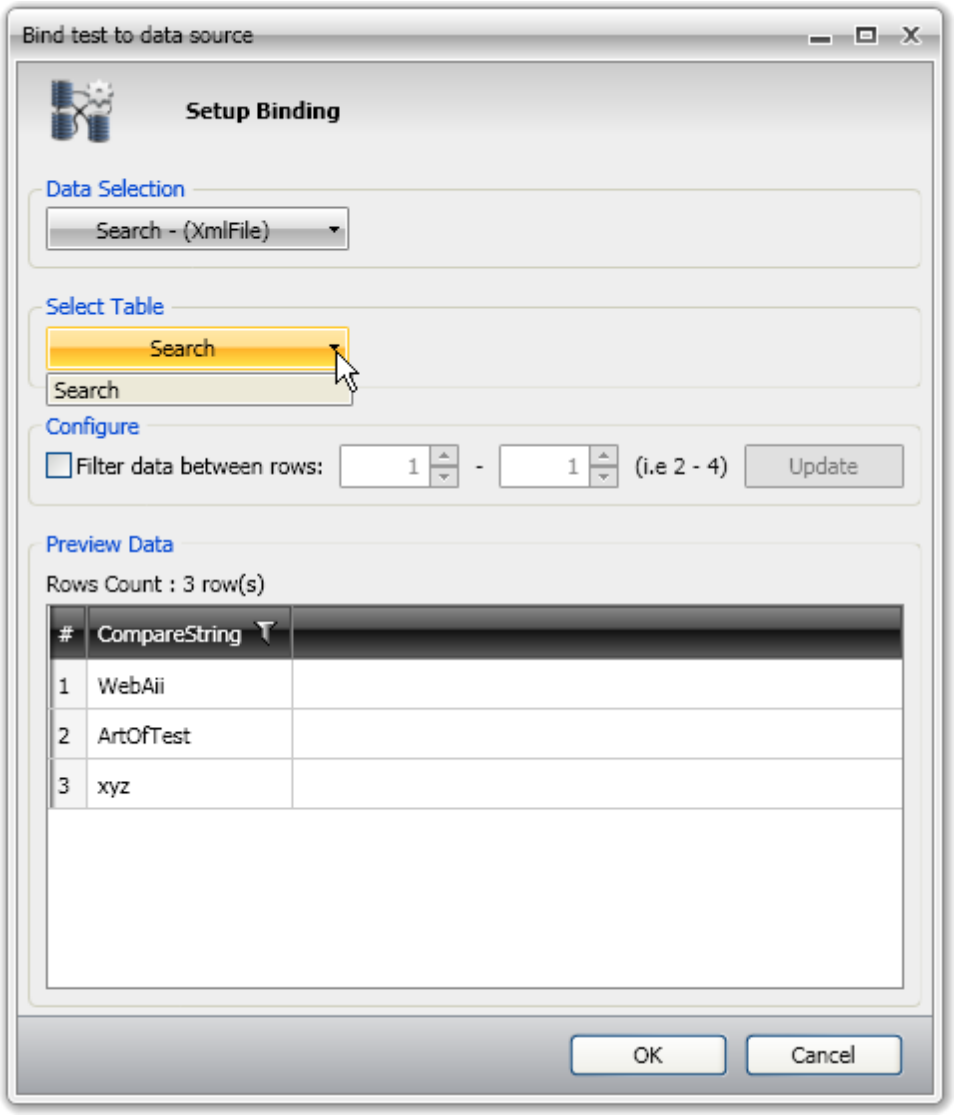

The database button will show that the test is now bound to the "Search.csv" file. You can rerun the test and get the same results as the "Built-in Grid" example, but in this case the data will be coming from the csv file.


## 15.4.2 XML Files

Driving your test from **XML** (Extended Markup Language) data is similar to using a spreadsheet, but instead of choosing the "CSV" or "Excel" options you choose the "XML File" option. Here is an XML sample that closely matches the structure of our previous list of searches.

```
<Searches>
  <Search CompareString="WebAii" />
  <Search CompareString="ArtOfTest" />
  <Search CompareString="xyz" />
</Searches>
```

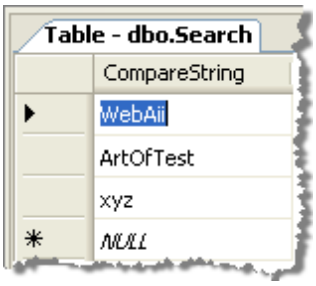
The preview of the data in the wizard is slightly different in that you can use the "Table" drop down to drill down into the XML hierarchy and select some group of data. This example uses very simple data and there is only the one "Search" table available.




Once again, if the column names match the data bindings from the built-in test, then the test steps don't need to be changed. Running this test should return the same results as the built-in grid example.

### 15.4.3 Database Tables

Once again, you can modify the data connection of the test to use the "Database" option, you can get at just about any external data available today. The data source possibilities include, but are certainly not limited to, MS SQL, Oracle, Access and ODBC. In this example we'll use a MS SQL table called "Search" that has a single "CompareString" column. Like the earlier examples, once the connection is configured, you don't need to change any of the test steps from the "Built-In Grid Walk Through" example.

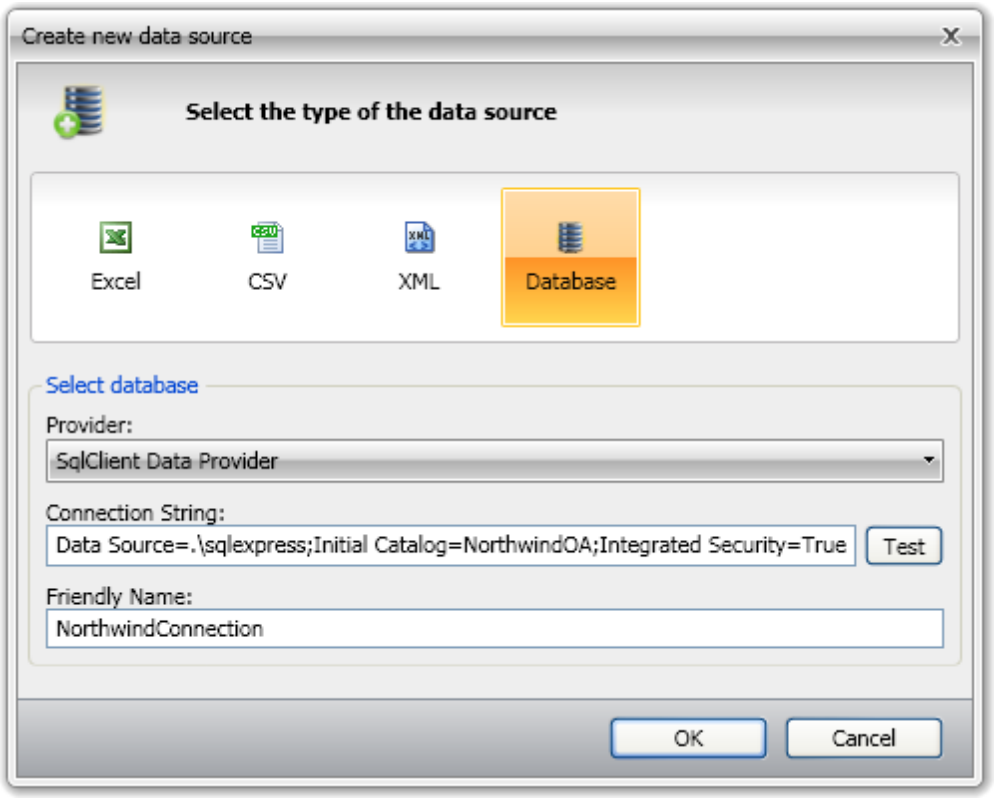



***Notes***

To run this example yourself, you'll need to create a table called "Search" in your own MS SQL database. You can use the SQL script shown below if you have a suitable utility, such as SQL Server Management Studio, installed to run the script.

```
DROP TABLE Search
CREATE TABLE Search(
  [CompareString] [nvarchar](50) NOT NULL
)
INSERT INTO Search VALUES('WebAii')
INSERT INTO Search VALUES('ArtOfTest')
INSERT INTO Search VALUES('xyz')
```

Clicking the database connection button **Add > Database** displays the same "Create new data source" dialog used in the earlier spreadsheet, but the parameters require you to enter a Provider, Connection String and Friendly Name. The Provider drop down list may take a moment the first time you use it to collect all the data source providers present on your system. In the screenshot below we have a "SqlClient Data Provider".

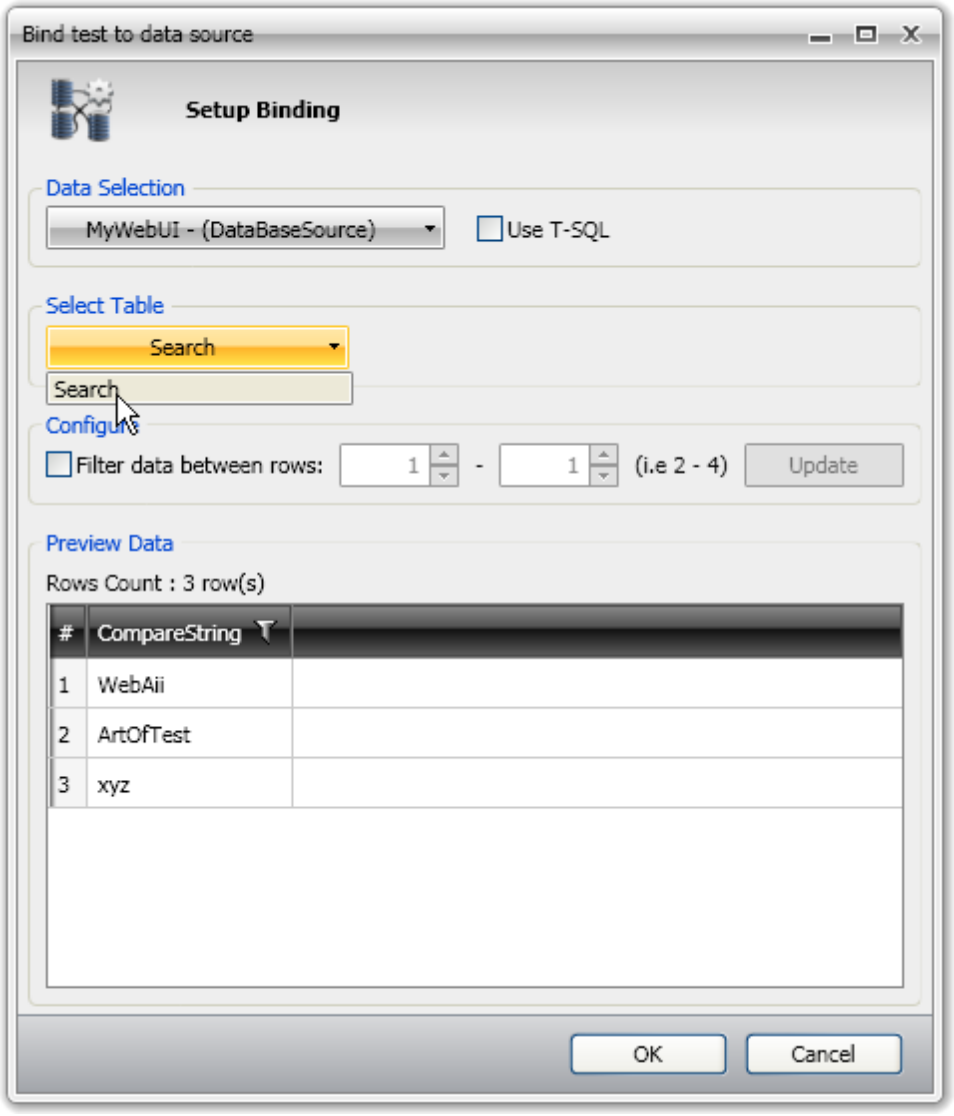



The Connection String must be entered manually. Finally, enter a Friendly Name for the connection

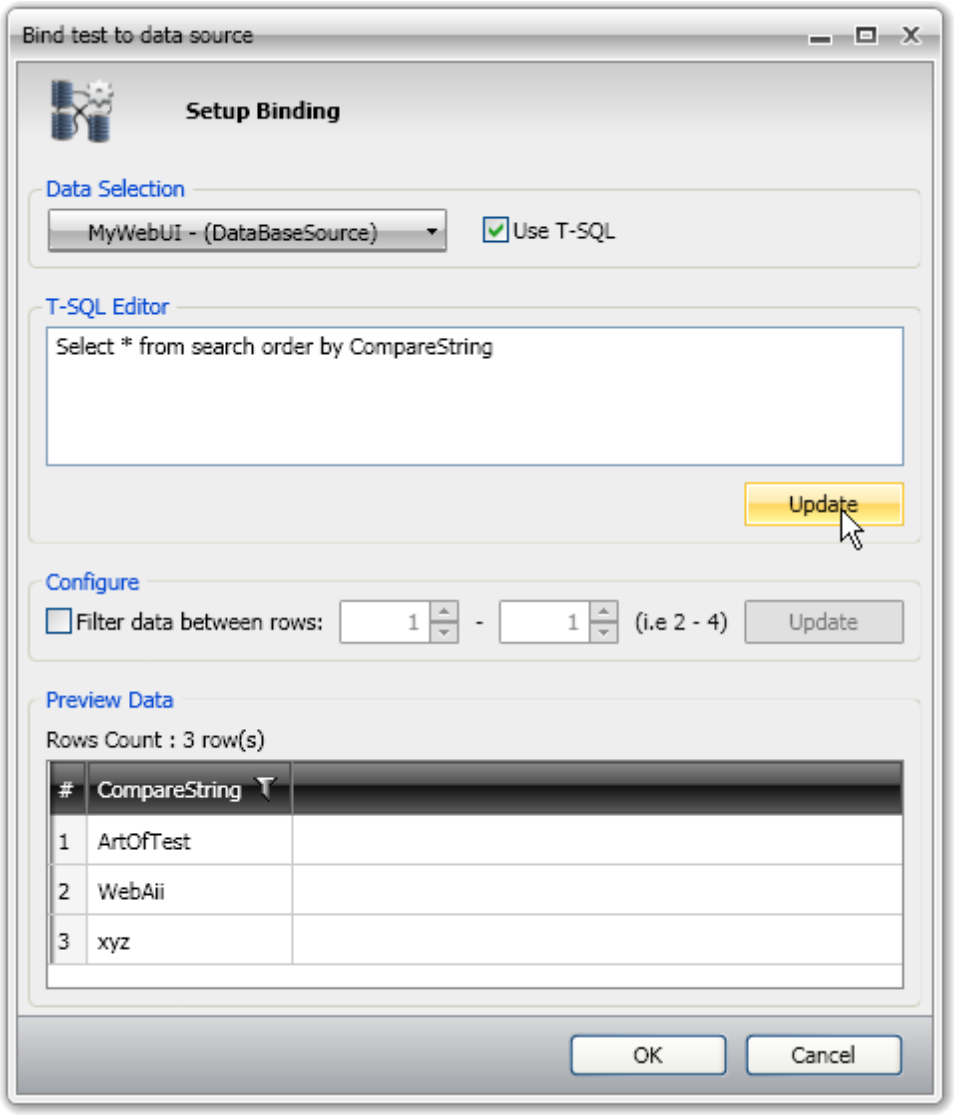
**Gotcha!**

As of this writing, WebUI Test Studio does not have a built-in connection string creation dialog, so you will need to compose your own. See the web site http://www.connectionstrings.com/ for more information on creating connection strings.

In the "Bind test to data source" dialog, you will need to select your data base source from the "Data Selection" drop down list. Choose a table from the "Select Table" drop down list and, once again, the "Preview Data" grid will display the list of records



You can also enable the "Use T-SQL" checkbox if you want to use SQL to tailor the exact data set you want returned. The screenshot below shows all the rows of the "Search" table are returned in alphabetical order.




And yet again, if the column names match the data bindings from the built-in test, the test should return the same results as the built-in grid example.

## 15.5 Using Code to Access Data

If we wanted to simply reproduce the test from the previous "Database Tables" example we could use something like the code below. Most of the code should look familiar except the line that uses the test's **Data** object. Data is a **TestData** type and can be return an object either by indexing using an integer or the using the name of the column. The example below indexes into Data using "CompareString".

One other change below is that we're skipping the assert if the CompareString is "xyz". What this is telling you is that not only can you put in a simple "If" statement in the code, but you can add conditional logic of any complexity.

```
<CodedStep("Drive test with data using code")> _
Public Sub DataDriveTest()
  ActiveBrowser.NavigateTo("http://www.google.com/")
  Pages.Google.QText.Text = "WebAii"
  Pages.Google.BtnGSubmit.Click(False)

  ' get the data for this iteration
  Dim compareString As String = Data("CompareString").ToString()

  ' ignore the "xyz" compare string
  If (Not compareString.Equals("xyz")) Then
    Pages.WebAiiGoogleSearch.ArtOfTestLink.AssertContent().InnerText( _
ArtOfTest.Common.StringCompareType.Contains, compareString)
  End If
End Sub
```

```
[CodedStep("Drive test with data using code")]
public void DataDriveTest()
{
  ActiveBrowser.NavigateTo("http://www.google.com/");
  Pages.Google.QText.Text = "WebAii";
  Pages.Google.BtnGSubmit.Click(false);

  // get the data for this iteration
  string compareString = Data["CompareString"].ToString();

  // ignore the "xyz" compare string
  if (!compareString.Equals("xyz"))
  {
    Pages.WebAiiGoogleSearch.ArtOfTestLink.AssertContent().InnerText(
      ArtOfTest.Common.StringCompareType.Contains, compareString);
  }
}
```

***Notes***

Be aware that you can also connect to data without using the WebUI Test Studio Data object using one of many mechanisms available from .NET such as ADO.NET or web services.

## 15.6 Advanced Scenarios

What if your test should be driven by data that doesn't fit the "one row at a time" scenario? What if the data isn't a standard data source and can't be reached using a connection string? Here are a few examples:

- Your organization still uses a legacy system that can be reached only through custom software.
- You need to check that certain parts of the network are up and running in a particular configuration.
- You need to test that certain documents are available at an FTP (File Transfer Protocol) site.
- The test must branch, loop or perform other complicated logic depending on the data.

***Notes***

Be aware that scenarios like these above may well require developer-level coding skills. The line between developers and QA engineers begins to blur at this point, but its important to understand what the possibilities are.

All of these scenarios can be handled in code. In these situations you would not define a data connection, but instead perform all your logic directly in the code.

Here's a brief example that uses SQL Management Objects (SMO) to get a list of SQL servers and check that they have "failover" capability. The point here is not to show you how SMO works or how to iterate through a data table. The main idea is that you can use any software available in the .NET world to use in your test code.

The example code shows an SmoApplication object finding available SQL servers and returning a DataTable object containing the results. The example code iterates the rows in the DataTable, looking at the "IsClustered" column. An assertion checks that IsClustered is true and if not, fails with a message that includes the server name.

```
<CodedStep("Verify Failover Capability for Sql Servers")> _
Public Sub VerifySqlServers()
  Const fmt As String = "Server {0} is not failover capable"

  Dim table As System.Data.DataTable = SmoApplication.EnumAvailableSqlServers()
  For Each row As System.Data.DataRow In table.Rows
    Dim isFailoverReady As Boolean = Convert.ToBoolean(row("IsClustered"))
    Assert.IsTrue(isFailoverReady, String.Format(fmt, row("Name").ToString()))
  Next row
End Sub
```

```
[CodedStep("Verify Failover Capability for Sql Servers")]
public void VerifySqlServers()
{
   const string fmt =
      "Server {0} is not failover capable";

   System.Data.DataTable table =
      SmoApplication.EnumAvailableSqlServers();
   foreach (System.Data.DataRow row in table.Rows)
   {
      bool isFailoverReady = Convert.ToBoolean(row["IsClustered"]);
      Assert.IsTrue(isFailoverReady,
         String.Format(fmt, row["Name"].ToString()));
   }
}
```

Here's an example excerpt from the log when the test is run:

```
Exception thrown executing coded step: '[VerifySqlServers] : Verify Sql Servers'.
InnerException:
Microsoft.VisualStudio.TestTools.UnitTesting.AssertFailedException: Assert.IsTrue failed.
Server WEBUITRAINING\SQLEXPRESS is not failover capable
  at Microsoft.VisualStudio.TestTools.UnitTesting.Assert.HandleFail(String assertionName, String message,
  at Microsoft.VisualStudio.TestTools.UnitTesting.Assert.IsTrue(Boolean condition, String message)
  at DataDriven.API_Example.VerifySqlServers() in C:\Courseware\Projects\CS\DataDriven\API_Example.aii.cs
---------------------------------------------------
'3/2/2010 9:32:28 AM' - Detected a failure. Step is marked 'ContinueOnFailure=True' continuing test execution.
```

## 15.7 Wrap Up

In this chapter you learned how to drive tests using the built-in data grid. You also learned how to access data from external spreadsheet, XML and database data sources. You learned how to access and modify data connections. Finally, you learned how to access the test Data object in code.

# Part XVI

Test Regions

# 16 Test Regions

## 16.1 Objectives

In this chapter you will learn how Test Regions are used to solve element identification, maintenance and performance problems inherent in complex web pages. You will learn how to access test regions in code and how to drill down to child elements of test regions. You will also see how the TestRegion ASP.NET control makes it easy to keep track of regions at design time.

## 16.2 Overview

### What is a TestRegion?

A Test Region is an innovative technology that "sections" markup code (XML, HTML, XHTML, etc...) to make automated tests more performant and easier to maintain.

One of the main obstacles between Development and Testing teams is the fact that code changes and implementations grow and get more complicated. QA teams find it difficult to maintain automated tests that count on specific IDs, Tags or even specific location of markup code in a file.

Think of a web page that contains thousands of lines of HTML markup with complicated element structures. Most of the tags are generated dynamically by a development tool compiler (like in ASP.NET). The QA tester is obligated to use element IDs or Tags for the outer elements while hoping that the names of the inner elements do not change in future executions. This approach is just not possible when working with the powerful controls being released everyday from companies like Telerik and others in the ASP.NET AJAX and Silverlight markets.

Another problem is the fact that if the tester uses something like XPath, the tester quickly finds out that they have to read the entire DOM each time they want to reference a particular object. If the Development team changes the hierarchy of elements in the DOM structure, the test will probably fail. This kind of testing is very fragile and requires heavy maintenance by a tester to make sure all changes on the R&D side are covered and manipulated on the QA side as well.

Take for example the simple HTML code below:

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
  <title>Falafel Software Locations</title>
</head>
<body>
   <table>
  <tr>
     <td>Falafel Software Inc.Home Page.</td>
  </tr>
   </table>
  <div>
   <input type="text" id="State" />
   <input type="text" id="zipcode" />
   <!-- Falafel offices in the area -->
   <table>
    <tr>
     <td>Office Name</td>
     <td>Address</td>
     <td>Tel.#</td>
    </tr>
    <tr>
     <td>Office Name</td>
     <td>Address</td>
     <td>Tel.#</td>
    </tr>
   </table>
  </div>
</body>
</html>
```

If the development team decides to add more rows in the table of offices or change the hierarchy of containment in the HTML DOM, the tests against this simple page will fail. The tests will fail whether they use IDs, Tags or XPath to identify the element.

***Notes***

TestRegion's methodology and identification system to building testability into markup applications, at the time of this writing, is "Patent Pending" at the US Patent and Trade Office.

## 16.3 TestRegion Sample

### Using TestRegions

Now, let's see what TestRegions bring to the table. First, the syntax of TestRegions is recommended to be a simple markup like:

```
<testregion id="MyRegion">...your markup goes here...</testregion>
```

The TestRegion technology is currently in a Patent Pending state, so the preceding example is only a recommendation. Not many browsers or tools will recognize the syntax, so for now you should use comment syntax like so:

```
<!-- testregion id="MyRegion" -->...your markup goes here...<!--/testregion -->
```

**Gotcha!**

You see that ending close of /testregion above? There is no space before the "/". If you write it this way, "<!-- /testregion -->", that includes the space, it will waste some debugging time and half your hair will be pulled out before you figure out the problem :(

So to use the previous HTML example with TestRegions, we would do something like:

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
  <title>Falafel Software Locations</title>
</head>
<body>
   <table>
  <tr>
     <td>Falafel Software Inc.Home Page.</td>
  </tr>
   </table>
  <div>
 <!--testregion id="FalafelInput"-->
    <input type="text" id="State" />
    <input type="text" id="zipcode" />
 <!--/testregion -->

<!--testregion id="FalafelLocation"-->
<!-- Falafel offices in the area -->
    <table>
      <tr>
        <td>Office Name</td>
        <td>Address</td>
        <td>Tel.#</td>
      </tr>
      <tr>
        <td>Office Name</td>
        <td>Address</td>
        <td>Tel.#</td>
      </tr>
    </table>
 <!--/testregion -->
  </div>
</body>
</html>
```

Now that the markup is "sectioned" using TestRegions, it is extremely easy in WebUI Test Studio to reference just these regions in code like the following:

```
Dim FalafelInput As TestRegion = Manager.ActiveBrowser.Regions("FalafelInput")
Dim FalafelLocation As TestRegion = Manager.ActiveBrowser.Regions("FalafelLocation")
```

```
TestRegion FalafelInput = Manager.ActiveBrowser.Regions["FalafelInput"];
TestRegion FalafelLocation = Manager.ActiveBrowser.Regions["FalafelLocation"];
```

There are two ways to access elements within a TestRegion:

1. Using the "Element" property. I don't recommend this way as it uses the DOM hierarchy inside of the TestRegion to get to the element. So if the R&D team changes the hierarchy, using this property on the TestRegion object will fail.
2. Using the "Find" object. This is the most robust way of finding elements inside of TestRegions. Bear in mind that the search takes place only inside of that specific TestRegion and NOT in the entire DOM of the page.

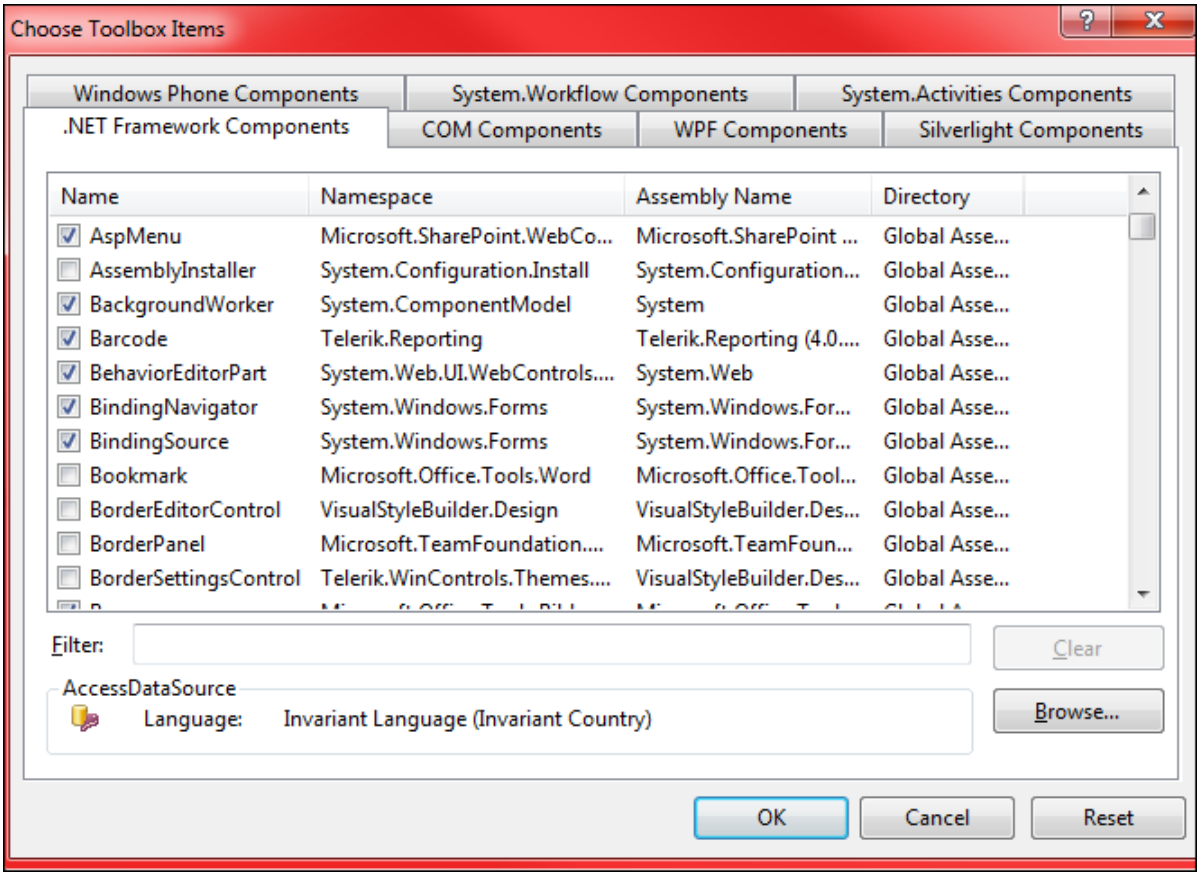
```
Dim FalafelOffice As Element = FalafelLocation.Find.ByTagIndex("table", 0)
Dim OfficeZip As Element = FalafelInput.Find.ByXPath("/input[2]")
```

```
Element FalafelOffice = FalafelLocation.Find.ByTagIndex("table", 0);
Element OfficeZip = FalafelInput.Find.ByXPath("//input[2]");
```

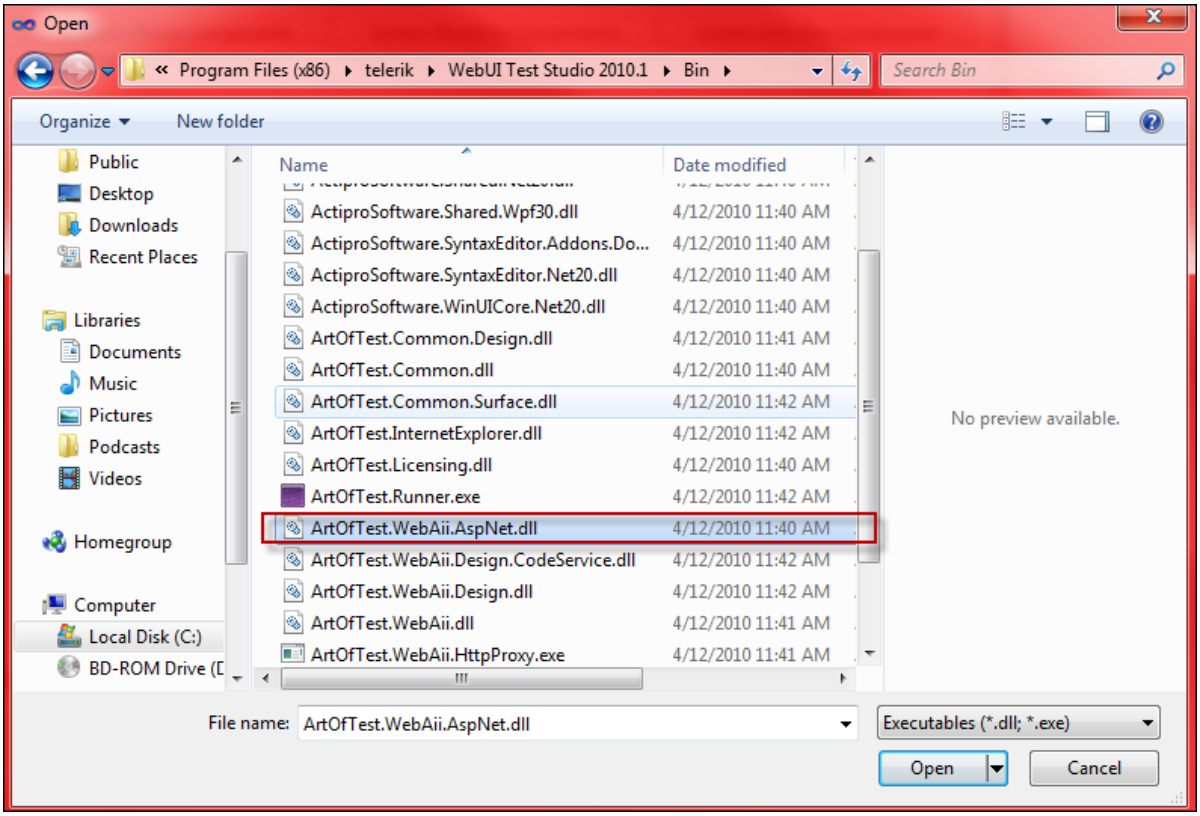
To find elements, you can use all the methods described in the "WebAii Framework" chapter, such as ByID (), ByTagIndex(), ByXPath(), ByName(), ByContent(), etc...

## 16.4 TestRegion in ASP.NET

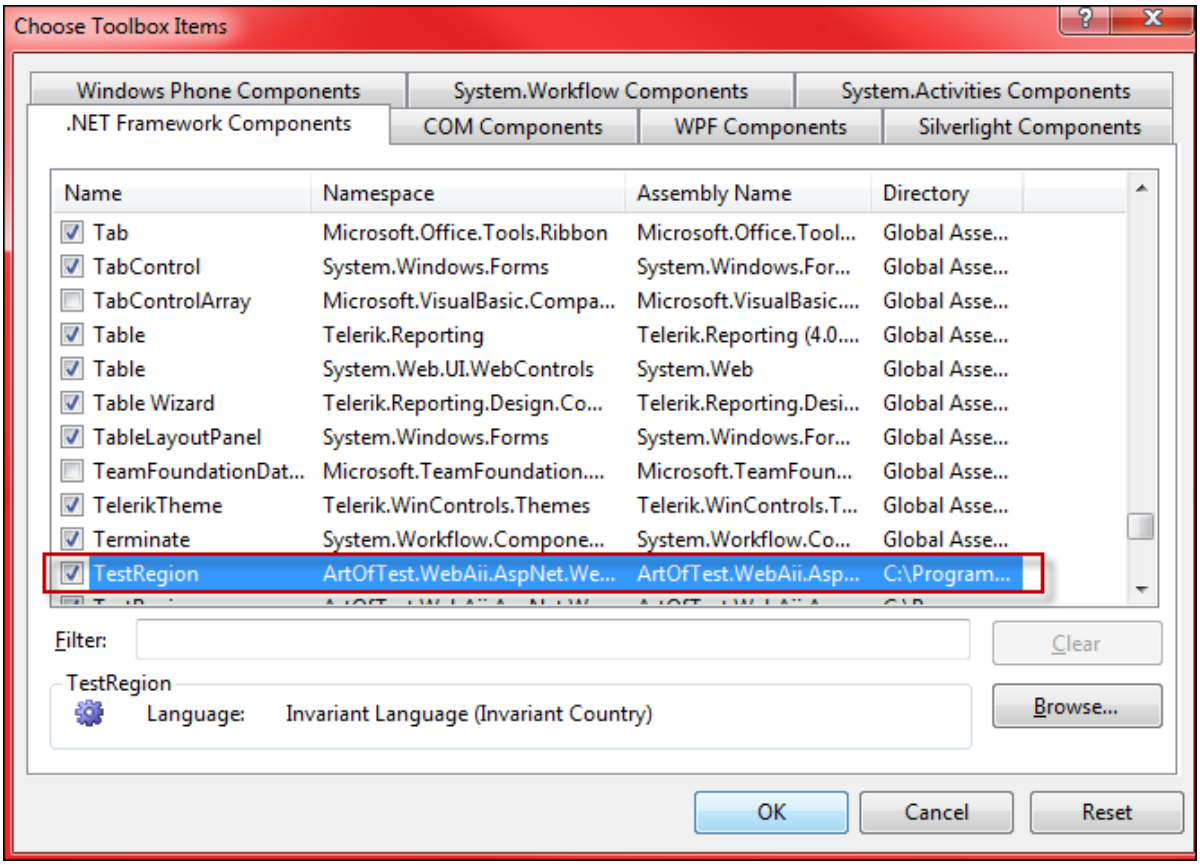
WebUI Test Studio makes it even easier to use TestRegions within ASP.NET applications by providing a TestRegion Control in Visual Studio that encapsulates a section of markup code. To Add the control to the Visual Studio Toolbox, first you need to launch the "Choose Toolbox items" dialog from the Visual Studio toolbox:



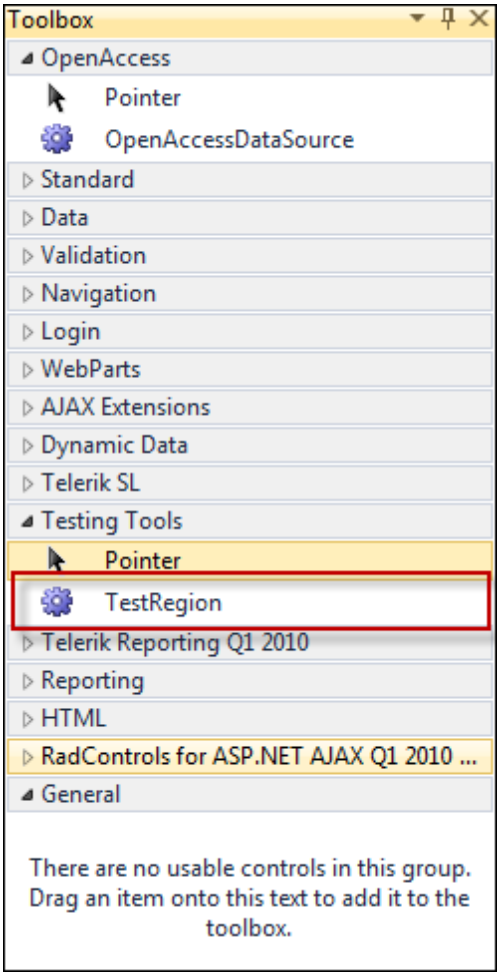
Then you will need to navigate to the "bin" directory where WebUI Test Studio has been installed on your machine.  If you chose the default installation, that will be under the Program files (86) directory. The DLL we are looking for is "ArtOfTest.WebAii.AspNet.dll" as shown below:



As soon as the DLL is loaded, a new control under the ".NET Framework Components" will appear in the list called "TestRegion". Check that control and add it to the Visual Studio Toolbox.



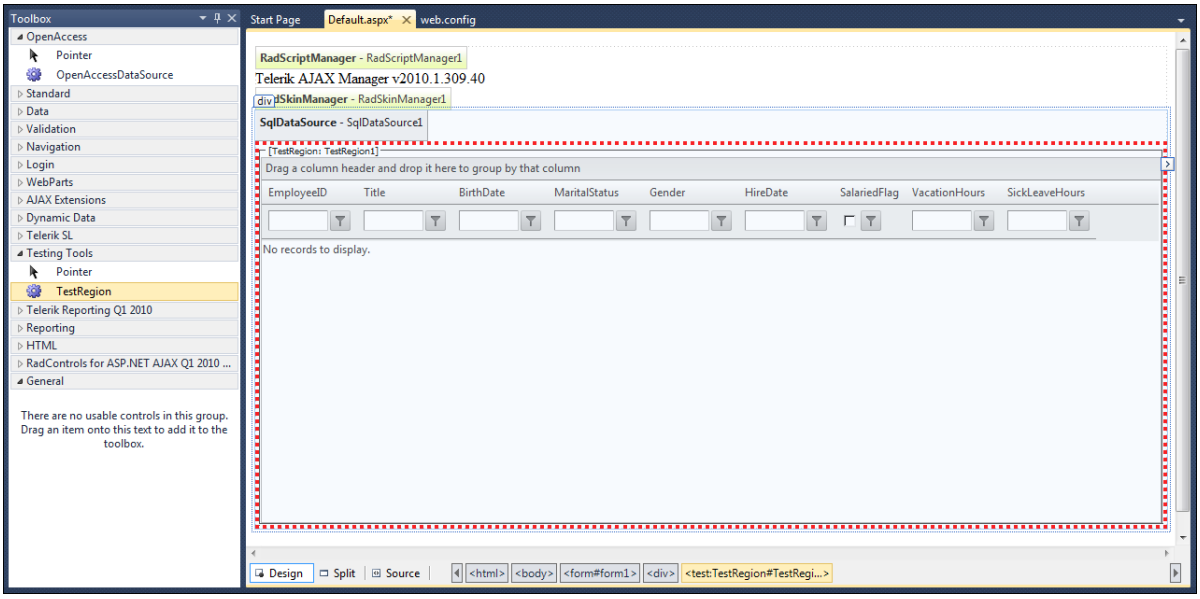
The control will show up in the Toolbox under whichever Tab you had opened when you launched the dialog. In the screenshot below, a new tab called "Testing Tools" has been created to contain the new TestRegion Control.



Now, anytime you need to encapsulate a specific complex control or a bunch of controls together for easy access in the DOM using the TestRegion control, just drop the control in your source code and place all the markup of your controls inside the TestRegion open and close tags.

```
<telerik:RadAjaxManager ID="RadAjaxManager1" runat="server">
</telerik:RadAjaxManager>
<telerik:RadSkinManager ID="RadSkinManager1" runat="server" Skin="Hay">
</telerik:RadSkinManager>
<div>
    <asp:SqlDataSource ID="SqlDataSource1" runat="server" ConnectionString
        SelectCommand="SELECT EmployeeID, Title, BirthDate, MaritalStatus,
    </asp:SqlDataSource>
    <test:TestRegion ID="TestRegion1" runat="server">
        <telerik:RadGrid ID="RadGrid1" runat="server" AllowFilteringByColu
            AllowSorting="True" DataSourceID="SqlDataSource1" GridLines="N
            <ClientSettings>...</ClientSettings>
            <MasterTableView ...>...</MasterTableView>
            <HeaderContextMenu EnableAutoScroll="True">
            </HeaderContextMenu>
        </telerik:RadGrid>
    </test:TestRegion>
</div>
```

The Control renders itself as an HTML comment at runtime in order to satisfy the requirements of all browsers. But at design time in Visual Studio, the control renders a red dotted line around each TestRegion so you can easily recognize your regions on the screen during test development.




## 16.5 Wrap Up

In this chapter you learned how Test Regions are used to solve element identification, maintenance and performance problems inherent in complex web pages. You learned how to access test regions in code and how to drill down to child elements of test regions. You also saw how the TestRegion ASP.NET control makes it easy to keep track of regions at design time.

# Part XVI

Debugging

# 17 Debugging

## 17.1 Objectives

In this chapter you will learn basic techniques to get you started debugging your coded tests. You will learn how to set breakpoints, step through your code and examine your variables.

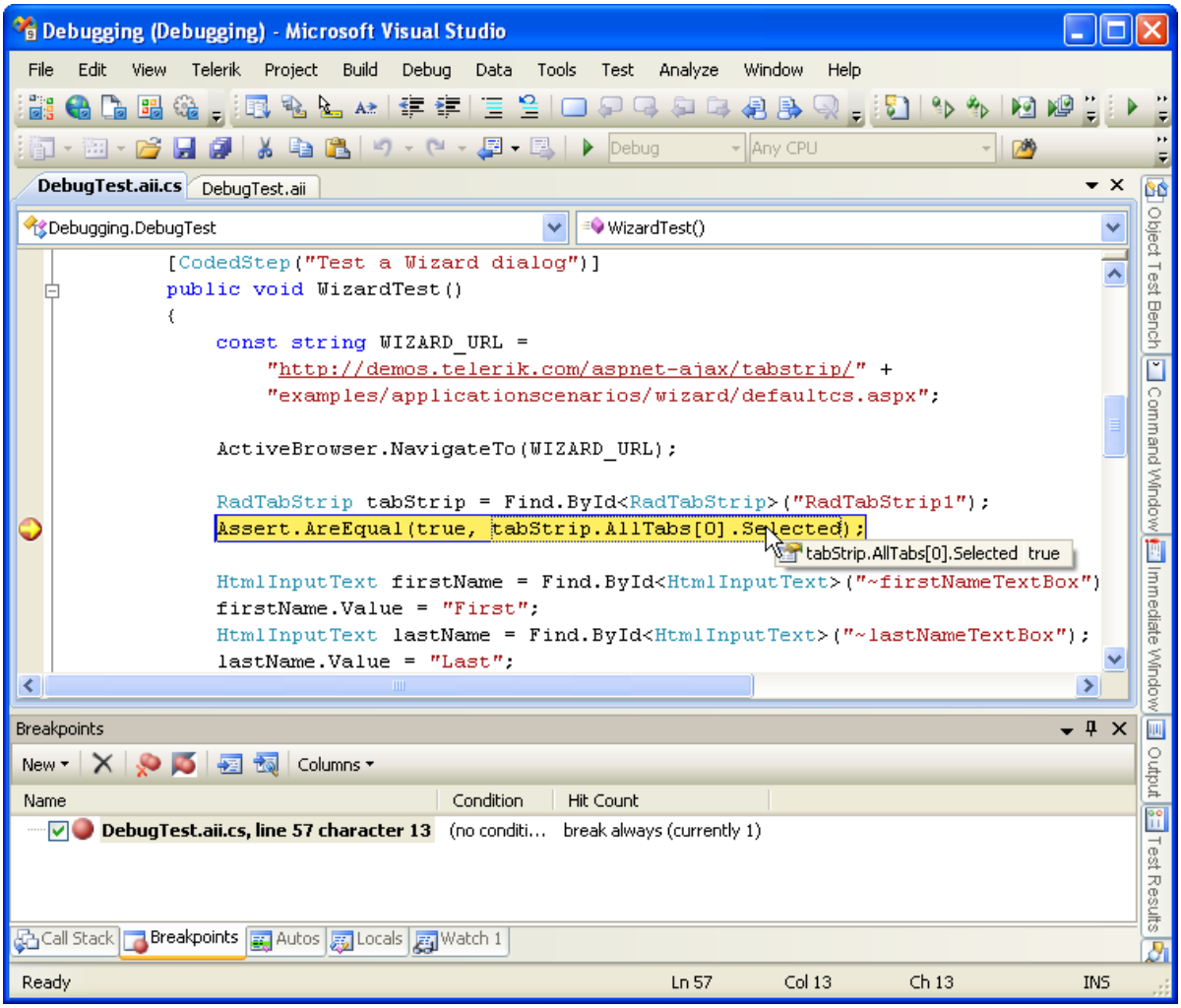
**Find the projects for this chapter at…**

\Courseware\Projects\<CS|VB>\Debugging\Debugging.sln

## 17.2 Overview

"Debugging? I'm a Quality Assurance engineer and debugging is for programmers". Actually, no. Once you have code of any sort, including code-driven tests, you need tools to determine why code doesn't perform as expected. That's where the debugger comes in. The debugger allows you to step one line of code at a time, examine the variables to see what values they contain and to pause or "break" when certain conditions exist in the code.

The image below shows a test being debugged "in the wild". In following sections we'll look at some of the key pieces you can use to step through your test code and examine all the variables and expressions you'll find there.

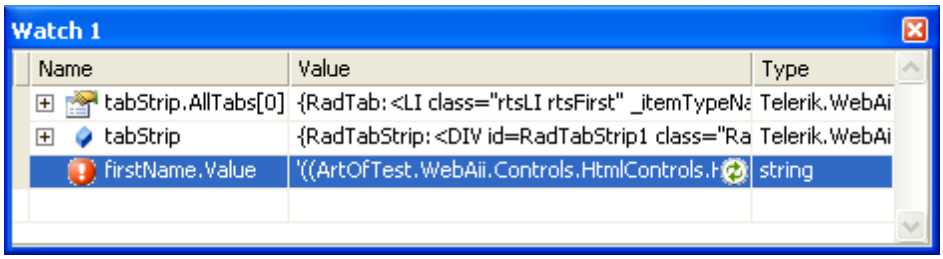



### *Notes*

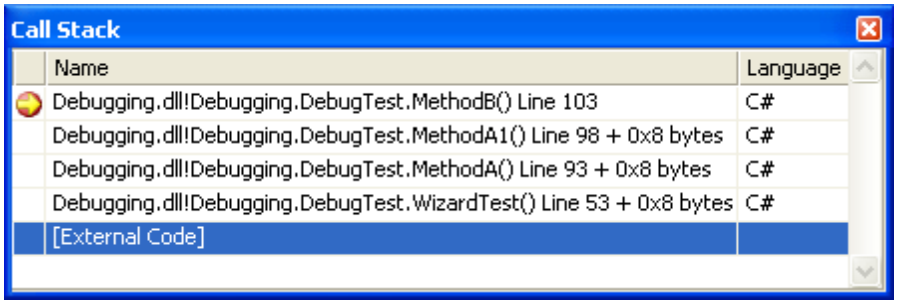
WebAii leverages Visual Studio debugging tools so tests must be initiated from the Test View window or Debugging menu. You will not be able to debug tests using the Quick Execute button to run the test.

While this certainly won't be an exhaustive look at debugging in Visual Studio, we will take a quick look at the most powerful features that you're most likely to need.

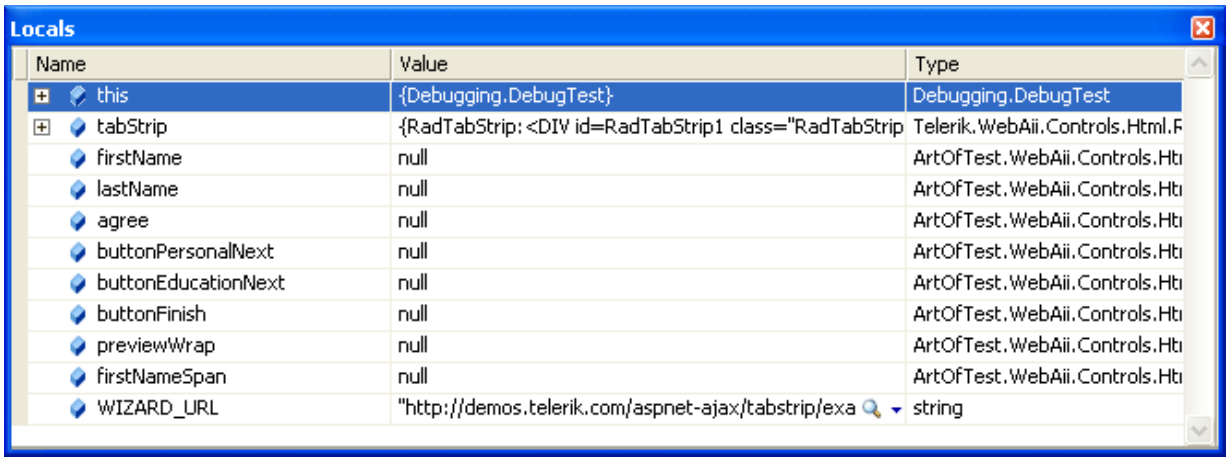
- **Breakpoints** are locations in the code that you can mark where execution should stop. When you debug the test, the code will stop running at the marked line, the line will be highlighted and you will be able to use debugging features to examine variable values and expressions. The previous screenshot shows a breakpoint on the line starting with "Assert.AreEqual...". The Breakpoints window shows line position and other information about the breakpoint.
- **Watches** are variables or expressions that you want to view as they change during the execution of your code. The Visual Studio Watch window stores multiple watches that can be viewed when execution stops at a break point.



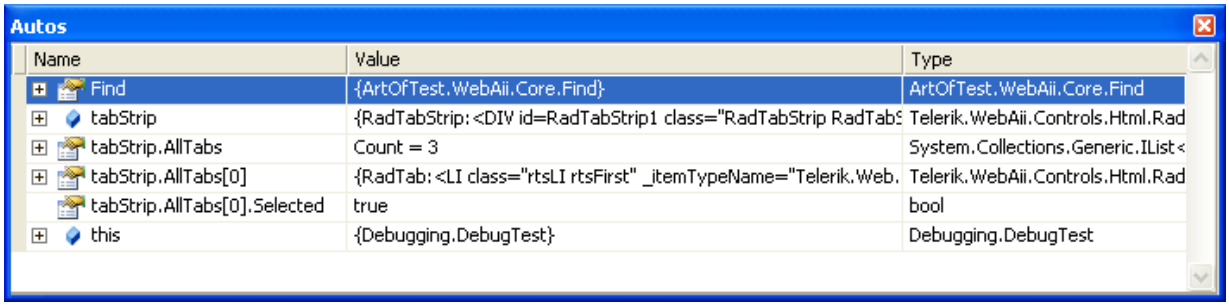

- The **Call Stack** shows you the chain of methods that are calling other methods. The screenshot below shows that WizardTest() was the first method called and that it called MethodA() and MethodA() called MethodA1().



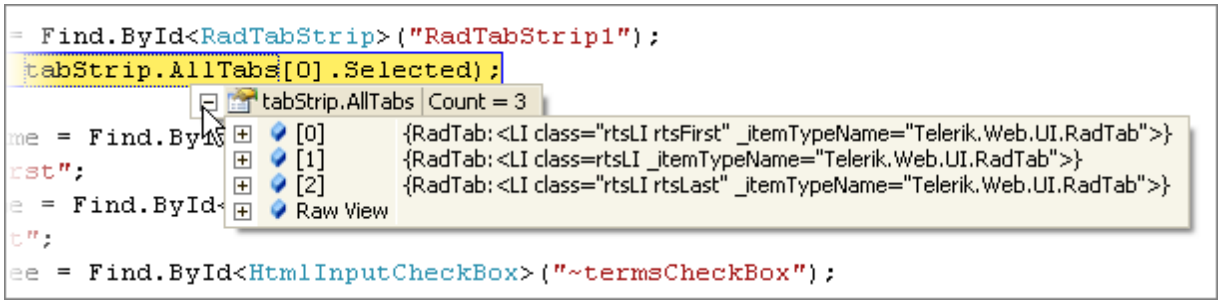
- **Locals** displays variables that are visible from the method you're debugging. The screenshot below shows the Locals window listing all the objects that can be seen when breaking at a line within the coded test method. The "this" object refers to the test itself.



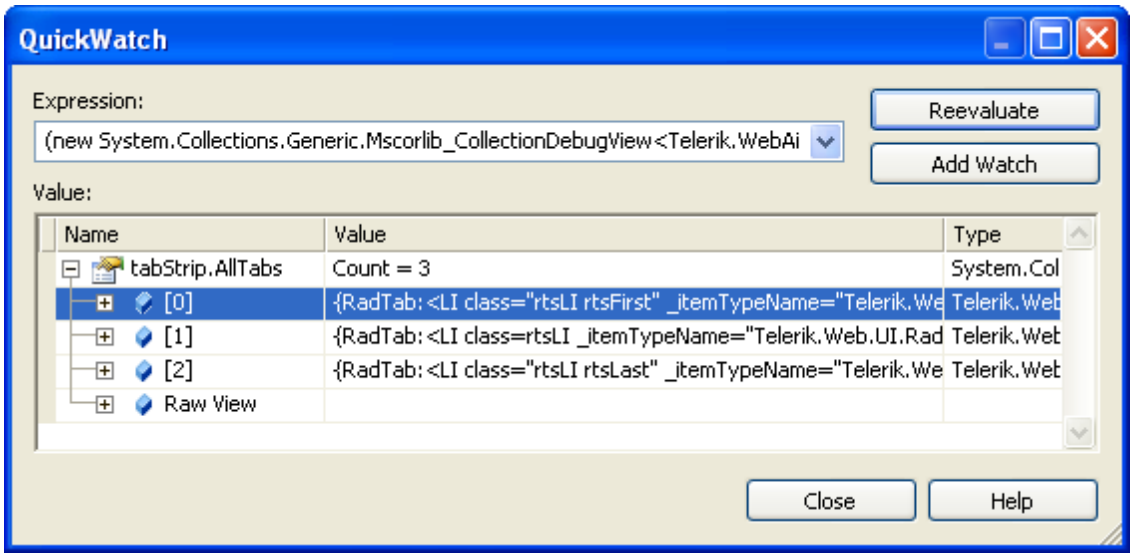

- **Autos** displays variables "in play" during the current statement. This is something like the Locals window, but is focused to the specific line that execution is stopped on. On this line we have access to the Find object, a RadTabStrip and a RadTab. The Autos window also shows us the "this" object that represents the test itself.




- While **Tool tips** are not a window unto themselves, they're an important part of debugging. When execution stops at a particular line of code, you can let the mouse hover over a variable and the tool tip will show the current value. In addition, you can "drill down" into the variable and get information about the object's base types and also see the objects children.



- Use the **QuickWatch** window to evaluate variables and expressions. Like tool tips, QuickWatch lets you expand objects with children. You can also enter new expressions and evaluate them on the fly. Notice the **Add Watch** button that lets you add the current expression to the Watch window. You can get to this window by right-clicking in the editor during a debug session and selecting QuickWatch from the context menu.



## 17.3 Debugging Walk Through

To keep things small and manageable, the "bug" for this project will be highly contrived, but will allow us to navigate the code using many of the available debugging tools.

1) From the Visual Studio choose **File > New > Project...** This will display the "New Project" dialog.

2) Define a new test project:

   a) In the "New Project" dialog, select the "Test" project type for your language (C# or VB).

   b) Select the "Test Project" template.

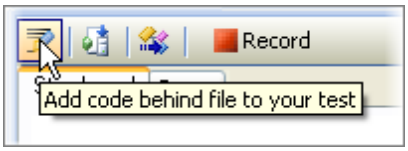
   c) Enter a descriptive name for the test project, a location path and a Solution name.

   d) Click **OK** to create the new test project.

3) From the Solution Explorer, right-click the test project and select **Add > New Test...** from the context menu. This will display the "Add New Test" dialog.

4) In the "Add New Test" dialog, select the "WebAii Test" template, name the test "DebugTest.aii" and click **OK** to create the test.

5) Click the **Add Code Behind** button to create the code behind file.




6) In the code behind file add the code below.

   The new code will provide raw material to debug through. The coded step takes a series of three numbers, each number is doubled, then the doubled number is squared and added to a variable called "total". On the first iteration of the "for" loop, "total" should be "4", then "20" the second time through and finally "total" should equal "56".

```
<CodedStep("Work with Debugging")> _
Public Sub DebuggingTest()
  Dim numbers() As Integer = { 1, 2, 3 }
  Dim total As Integer = 0

  ' result should be:
  '  first iteration:  4  --(2 * 2)
  '  second iteration: 20 --((4 * 4) + 4)
  '  third iteration:  56 --((6 * 6) + 20)
  For Each number As Integer In numbers
    total = total + DoubleAndSquare(number)
  Next number
  Assert.IsTrue(total < 100)
  Assert.AreEqual(56, total)
End Sub

Public Function DoubleAndSquare(ByVal number As Integer) As Integer
  Dim doubledNumber As Integer = number + 2
  Dim result As Integer = Square(doubledNumber)
  Return result
End Function

Public Function Square(ByVal number As Integer) As Integer
  Dim result As Integer = number * number
  Return result
End Function
```

```
[CodedStep("Work with Debugging")]
public void DebuggingTest()
{
   int[] numbers = { 1, 2, 3 };
   int total = 0;

   // result should be:
   //  first iteration:  4  --(2 * 2)
   //  second iteration: 20 --((4 * 4) + 4)
   //  third iteration:  56 --((6 * 6) + 20)
   foreach (int number in numbers)
   {
      total = total + DoubleAndSquare(number);
   }
   Assert.IsTrue(total < 100);
   Assert.AreEqual(56, total);
}

public int DoubleAndSquare(int number)
{
   int doubledNumber = number + 2;
   int result = Square(doubledNumber);
   return result;
}

public int Square(int number)
{
   int result = number * number;
   return result;
}
```

7) In the Steps Tab, click the **Quick Execute**  button to run the test. The test should fail.

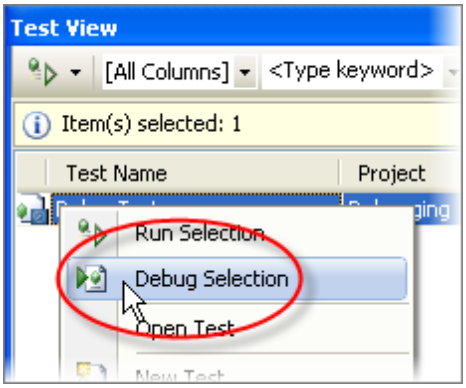
8) View the log to see where the test failed. The log should show an AssertFailedException where the AreEqual() method expects "56" but the actual value is "50".

". . .Microsoft.VisualStudio.TestTools.UnitTesting.AssertFailedException: Assert.AreEqual failed. Expected:<56>. Actual:<50>. . ."

Now let's use the debugger to hunt down the problem...

9) In the code editor, click the "gutter" area to the left of the "for each" loop. A red circle should appear to indicate a breakpoint has been placed on this line.

10)In the Test View window, right-click the test and select **Debug Selection** from the context menu.




***Notes***

If the Test View window is not open, you can open it from the Visual Studio menu **Test > Windows > Test View**. Also know that you can debug the test directly using the Visual Studio menu **Test > Debug** item.

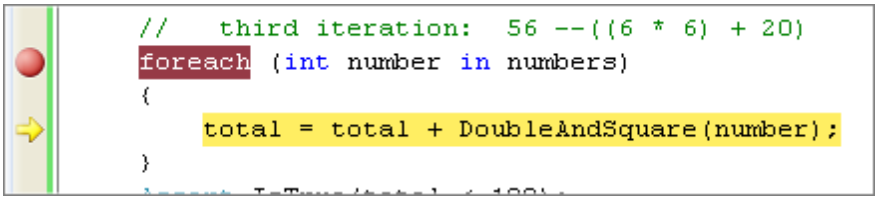
11)The execution of the test should stop on the line where you placed your breakpoint. The visual cue is a yellow arrow and highlighting of the line as shown in the screenshot below. Press **F10** to continue to the next line.


***Notes***

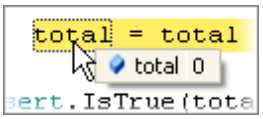
You can also step through your debugging code using the Visual Studio Debug menu items:

- **F10**: "Step Over", i.e. only step through items in this method
- **F11**: "Step Into", i.e. if this method calls another method, step into the called method.
- **Shift-F11**: "Step Out" backs out from the method you're in to a method that called it.
- **F5**: "Continue" runs through the code until it finishes the test or hits a breakpoint.
- **Shift-F5**: "Stop Debugging" ends execution of the test and returns you back to design mode.

12)Continue to press **F10** until the line "total = total + ..." is highlighted.

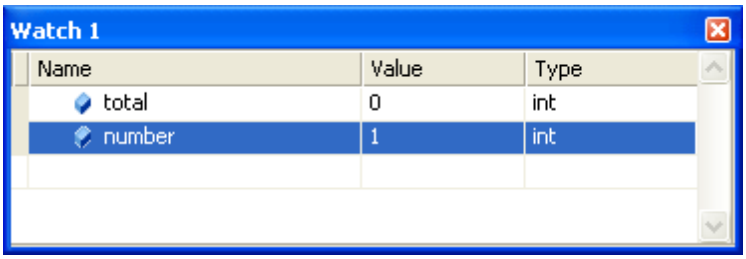



13)Hover the mouse above the "total" variable. The tool tip should display that the value is currently "0".

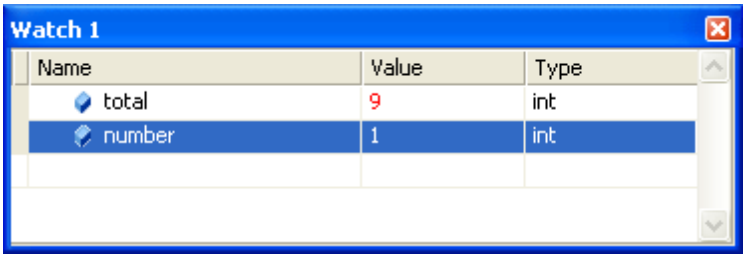



14)Right-click the "total" variable and select **Add Watch** from the context menu.

15)Right-click the "number" variable and select **Add Watch** from the context menu. The Watch window should look like the screenshot below (if the Watch window isn't visible, bring it up using the Visual Studio menu **Debug > Windows > Watch > Watch 1**).

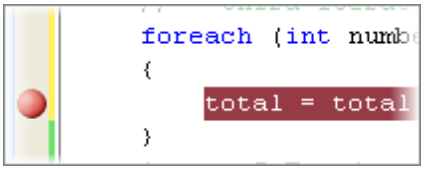



16)Press **F10** to continue to the next line. Notice the Watch window. The "Total" Value is highlighted in red to indicate that it's changed from its previous value. The value is now "9", although the value should be "4".



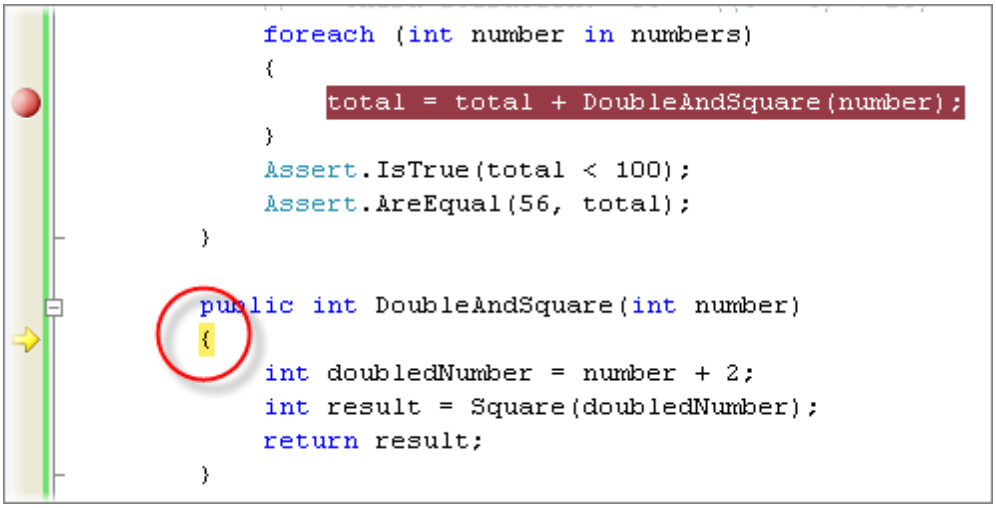

17)Press **Shift-F5** to end debugging.

18)Place a breakpoint on the line "total = total + ". Remove any other breakpoints you have defined.



19)In the Test View window, right-click the test and select **Debug Selection** from the context menu to run the test in debug mode again.

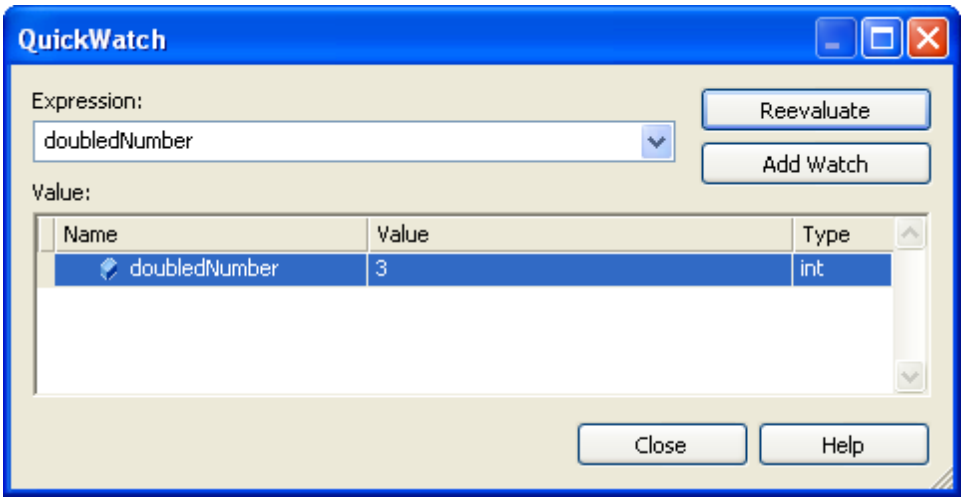
20)When the test execution stops on the breakpoint, press **F11** to continue *step into* the first line of the DoubleAndSquare() method.




21)Press **F10** until you reach the line "int result = Square...".

22)Right-click the "doubledNumber" variable and select **QuickWatch** from the context menu.

We should have a value of "2" but the Value is actually "3". The reason for that appears to be that "doubledNumber gets the number "+ 2", not "* 2", so we're adding two instead of multiplying by two.




23)Press **Shift-F5** to end debugging.

24)In the code editor, change the DoubleAndSquare() method so that "doubledNumber" is assigned "number * 2". The code for DoubleAndSquare() should now look like the example below.

```
Public Function DoubleAndSquare(ByVal number As Integer) As Integer
  Dim doubledNumber As Integer = number * 2
  Dim result As Integer = Square(doubledNumber)
  Return result
End Function
```

```
public int DoubleAndSquare(int number)
{
   int doubledNumber = number * 2;
   int result = Square(doubledNumber);
   return result;
}
```

25)Build the project and rerun the test. The test should now pass.

This only scratches the surface when it comes to debugging in Visual Studio. To continue learning more about the extensive capabilities of the Visual Studio debugger, start at the MSDN article Building, Debugging and Testing at http://msdn.microsoft.com/en-us/library/d8k88a0k.aspx..

## 17.4 Wrap Up

In this chapter you learned some basic techniques to get you started debugging your coded tests. You learned how to set breakpoints, step through your code and examine your variables.

# Part XVI

Support and Services

# 18 Support and Services

### Support

Read the community forums, watch informative videos, see the latest blogs or send a Support Ticket to the excellent Telerik support team, all at this link: http://www.telerik.com/automated-testing-tools/support.aspx.

### Services

Telerik's worldwide network of partners can provide your organization with training and services to help you ramp up more quickly or help with your existing automated testing projects. Go to www.Telerik.com/partners to find a partner that fits your needs.

### WebUI Training and Services from Falafel Software

The authors of this book are from Falafel Software, Telerik's premier services partner. Falafel has a wide range of services ranging from WebUI training and consulting to large-scale custom enterprise application development. The professionals from Falafel Software are great to work with and we hear nothing but effusive praise about them from our customers. Here's a description of some of their WebUI related services:

### WebUI Training from Falafel Software

- **Training Summit**: If your team only has a few individuals in need of training, this open-enrollment option is the most cost effective solution.
- **Online Training**: For companies that have team members in multiple locations or in situations where onsite training is not feasible, online training is a great option.
- **Onsite Training**: A highly knowledgeable Falafel Software trainer will come to you and provide your team with an enlightening 3-5 day class. This is the best way to ramp up quickly.

### WebUI Consulting from Falafel Software

On a deadline? Need assistance from the Pros? Let Falafel provide you with world-class consulting for all your WebUI needs. Falafel's consultants have been working with WebUI since the very beginning and are the highest qualified individuals to assist you.

### WebUI Consulting Express

Need help right now? Purchase pre-paid online consulting from the Falafel Store and you'll have a WebUI Consultant working with you live over the phone and via GotoMeeting so you can virtually work shoulder to shoulder to get you going quickly.

http://store.falafel.com/p-56-telerik-consulting-express.aspx

For more info on Falafel Software, go to www.falafel.com or call 1-888-GOT-FALAFEL (1-888-468-3252).

# Index

## - " -



## - . -



## - / -



## - 3 -



## - A -



## - B -

# - C -



# - D -



# - E -

# - F -



# - G -



# - H -

# - I -



# - J -



# - L -



# - M -



# - N -



# - O -

# - P -



# - Q -



# - R -



# - S -

# - T -



# - U -



# - V -



# - W -

# - X -



# - Z -


FALAFEL
software
www.falafel.com