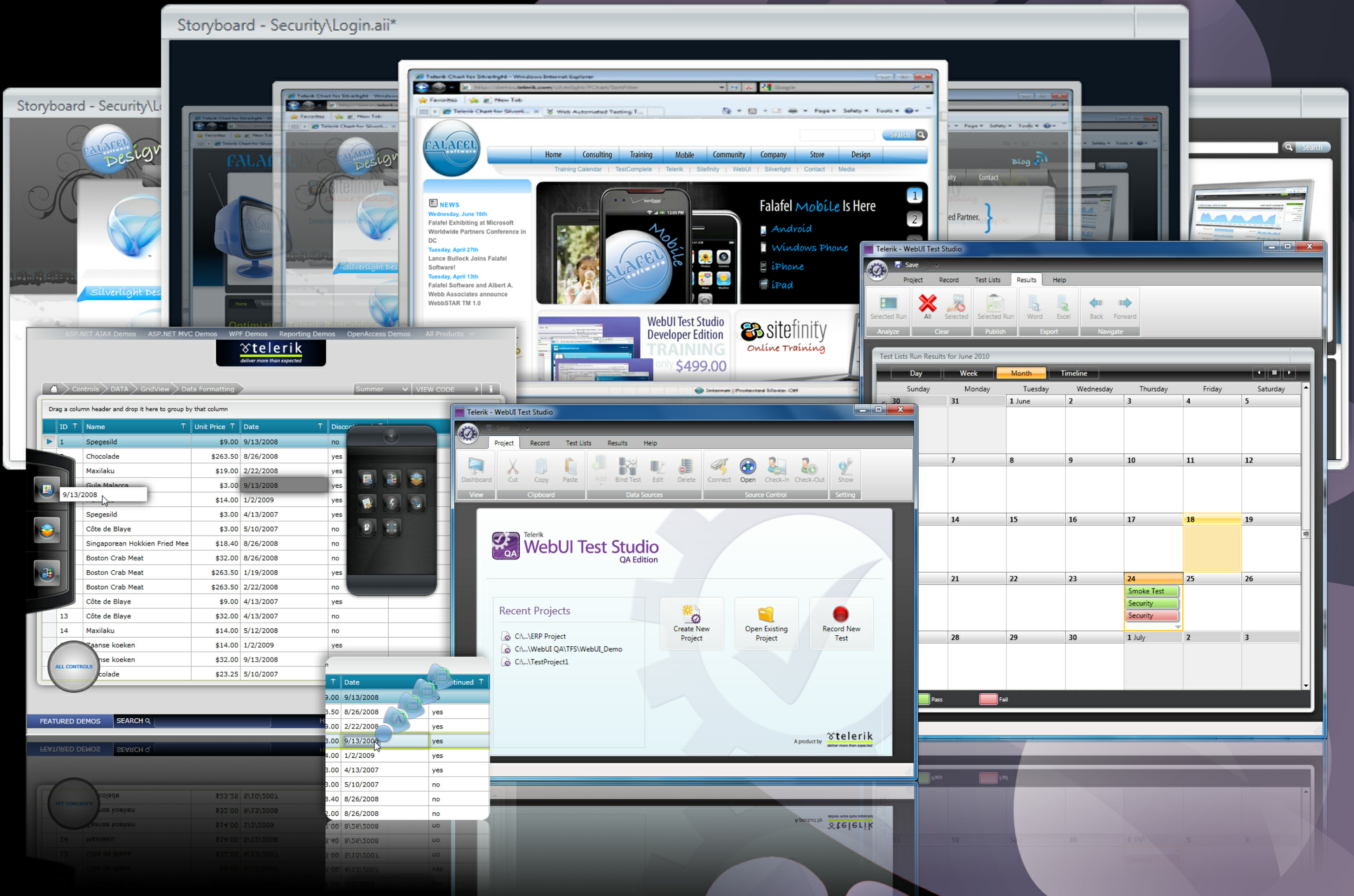
WEBUI TEST STUDIO

QA Edition

made easy



Storyboard - Security\Login.aii*
Telerik - WebUI Test Studio
WebUI Test Studio QA Edition
Recent Projects
Create New Project
Open Existing Project
Record New Test
Falafel Mobile Is Here
Test Lists Run Results for June 2010

www.falafel.com

FALAFEL software

Written by: Noel Rice and Lino Tadros

WebUI Test Studio QA Edition Made Easy

by Noel Rice and Lino Tadros

Welcome to WebUI Test Studio QA Edition Made Easy

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

WebUI Test Studio QA Edition Made Easy

While every precaution has been taken in the preparation of this document, the publisher and the author assume no responsibility for errors or omissions, or for damages resulting from the use of information contained in this document or from the use of programs and source code that may accompany it. In no event shall the publisher and the author be liable for any loss of profit or any other commercial damage caused or alleged to have been caused directly or indirectly by this document.

Printed: August 2010

Publisher
Falafel Software Inc.

Author
Noel Rice
Lino Tadros

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 QA Edition. 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, Daniel Levy, Ivo Nedkov, 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 continued 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 IV Getting Started 53

Part V Working with Projects 62

Part VI Recording Tests 77

Part VII Verification 121

Part VIII Test Organization 144

Part IX Working with Test Results 159

Part X Web Application Tests 178

Part XI Data Driven Tests 198

Part I

Introduction

1 Introduction

1.1 Who Should Read This Courseware

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

1.2 What Do You Need to Have Before Reading This Courseware

Software Setup

WebUI Test Studio QA Edition only requires:

- .NET Framework 3.5 service pack 1 or higher.
- Internet Explorer (for building and recording tests).
- One or more of the following browsers for running tests: Internet Explorer, Firefox or Safari.

1.3 How This Courseware is Organized

Installation

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

Getting Started

In this chapter you will use WebUI to create a simple test, execute the test and view the results. Along the way you'll see the major areas of WebUI.

WebUI Overview

In this chapter you take a guided tour around the product so you'll know where to find everything later. You will become familiar with major areas of the product including Projects, Record, Test lists and Results.

Recording Tests

In this chapter you will take a detailed look at how tests are recorded and the available tools used to handle specific web testing tasks. You will use the Recording Surface to save actions taken in the browser and to interact with individual web page elements. You will see how these recorded steps are managed using the Steps Pane and how to add steps manually. You will see how "Translators" are used to peer inside controls to glean information about the controls internals. You will also use WebUI to handle specialized web testing situations such as drag-and-drop and working with pop-up dialogs. Finally, you will learn how to "audition" tests in multiple browsers.

Working with Projects

In this chapter you will learn how tests, folders and other files are organized into projects. You will see how the context menu interacts with the Project Files tree view to build the project structure and to work with individual tests. You will create a simple project and open a test for recording.

Verification

In this chapter you will learn how to test, i.e. assure that certain conditions exist in the browser, using verifications. You will learn how to access verifications from more than one tool in WebUI. You will use the Sentence Verification Builder to interactively build verification rules and validate them against live web documents. You will also explore the structure of verification sentences and look at how individual verification types are used. Finally, you will use the 3D Viewer to handle verifications for multiple elements at one time.

Test Organization

In this chapter you will learn how Test Lists help get the best use of recorded tests by allowing reuse in more than one configuration. You will learn to build static test lists from existing tests and to manually set the order that the tests run. You will also learn how dynamic test lists automatically select tests from your project at the time of execution using rules about properties of the individual tests. You will see how to configure settings for all the tests within a test list.

Working with Test Results

In this chapter you will learn how to analyze and share the results of your tests. First you will see how the calendar view of test results makes it easy to see the tests that passed or failed in a given period of time. Next you will see how the TestResults panel allows you to traverse test execution results, drilling down to individual test steps and back up again. Finally, you will learn how the Step Failure Details dialog is used to work with a single failed test step.

Web Application Tests

In this chapter you will learn about some of the technologies that make web applications so easy to work with as a user, but also make testing more difficult as a Quality Assurance engineer. You'll learn how WebUI handles asynchronous updates in the browser, timing issues and Silverlight animation. In particular, you will learn how WebUI can efficiently wait for a set of conditions to occur.

Data Driven Tests

In this chapter you will learn how to drive your tests using data sources. You will learn how to add a data source, connect the data source to a test and how to bind data to properties in the test. First, you will use the built-in grid to create simple data-driven tests without needing an external data source. You will also learn how to drive tests with external data from spreadsheet files, comma delimited files, XML and database tables.

1.4 About Testing

Without automated testing tools such as WebUI you're left with "manual testing", a tiresome process where each piece of the product is carefully worked and where the tester must carefully record expected results against actual results by hand. Manual testing is certainly "real world" testing that mimics how the user will interact with the product, but the manual approach has a number of flaws:

- **Tests are only valuable when run**. Manual tests are time consuming to perform and difficult to run regularly. New bugs and regression bugs become more difficult to detect as your product grows in size and complexity.

- **Manual tests are difficult to reproduce** exactly, over a long period of time. Accurate testing requires concentration and attention to detail on the part of the tester. As the tester's mental focus varies over time, it's likely that test execution will change. If multiple people are testing the product, differences between test runs will increase.

- **Results from tests are only valuable when shared**. Publication of manual test results is not an automated process so there's a tendency to neglect regular communication. True collaboration between developers, management and other testers is next to impossible.

On top of these generic issues, web testing raises other concerns:

- **Web browsers don't provide clear visibility** to what's happening on the page. The tester has no straight-forward way of consistently identifying an element on a web page and detecting changes in the element's state.

- **New technologies introduce new testing issues**. The same features that make web browsing a richer experience also present obstacles to testing, e.g. asynchronous processes, client side code running directly in the browser, animation, etc.

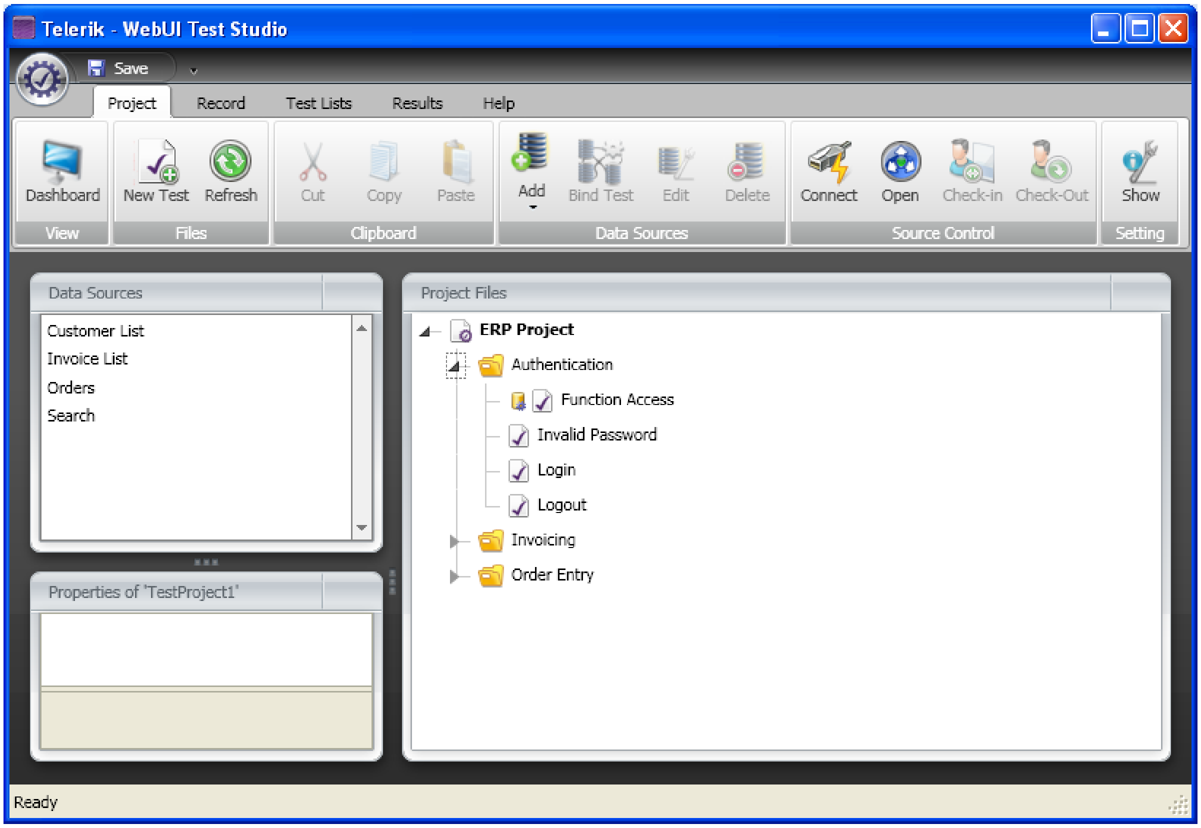
Automated testing tools present their own challenges by being complex to use and are often built to suit the style of software developers more closely than QA engineers. The user interface can resemble the dashboard of a jet cockpit with many controls and windows, but no clear functional path that assists the tester through the process. Traditional automated testing tools are also script-centric and require coding skill to use effectively.

WebUI Test Studio QA Edition addresses these issues by making tests easy to run and consistent over time and where results can be shared easily with key players in the organization. WebUI Test Studio QA Edition is built to provide deep visibility into elements on a web page and is extensible so that your tools can always handle the latest technology challenges. WebUI Test Studio QA Edition is built specifically for QA engineers, without the trappings of complex development environments.

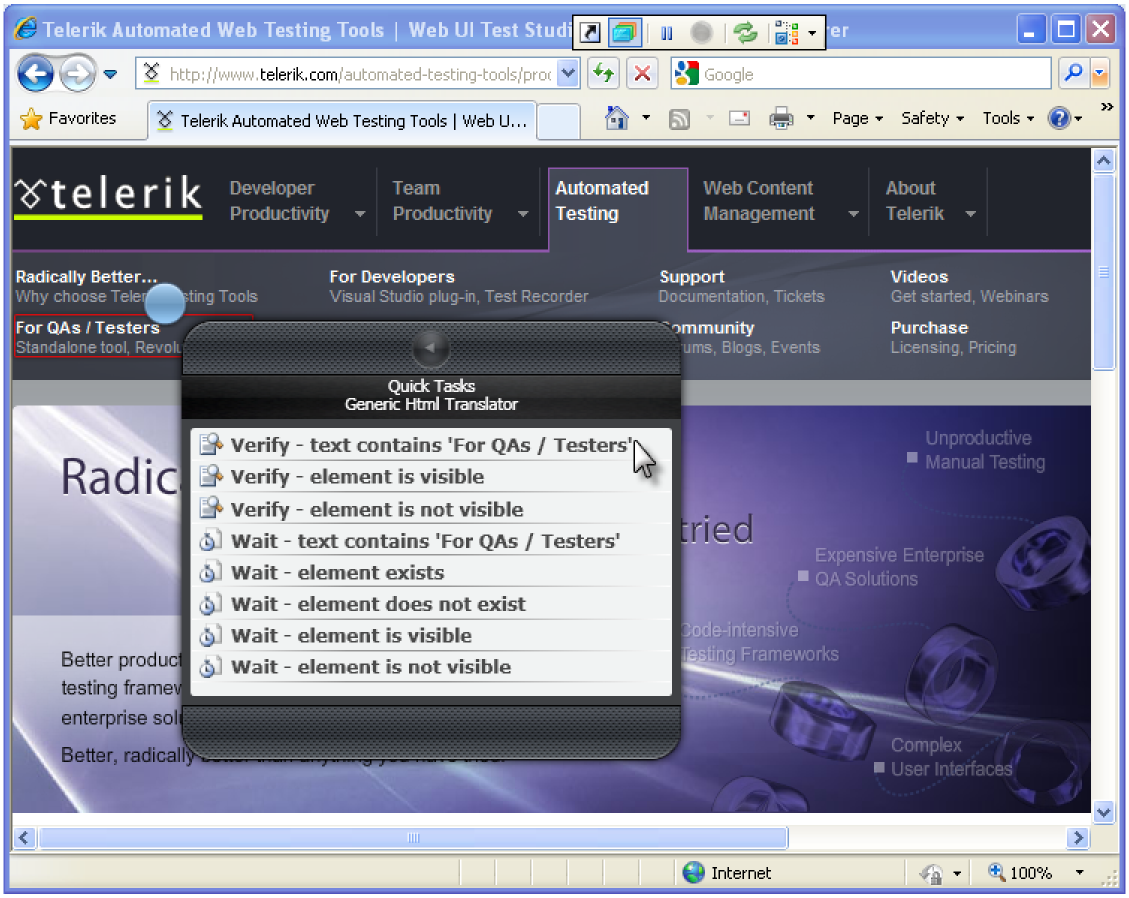
1.5 WebUI Test Studio QA Edition

WebUI Test Studio QA Edition is an automated testing tool that offers an intuitive, codeless and productive way to test any web application. Complex AJAX and Silverlight scenarios, MVC, client-side functionality, JavaScript calls, data-driven testing – WebUI covers them all. Test management and failure resolution are brought to a new level, making you more productive.

The WebUI Test Studio QA Edition user interface is task-centric and matches how a QA engineer is likely to use it. Daily tasks performed by QA engineers are laid out in a logical progression from creating test projects, to recording tests, test execution and finally, working with the test results.

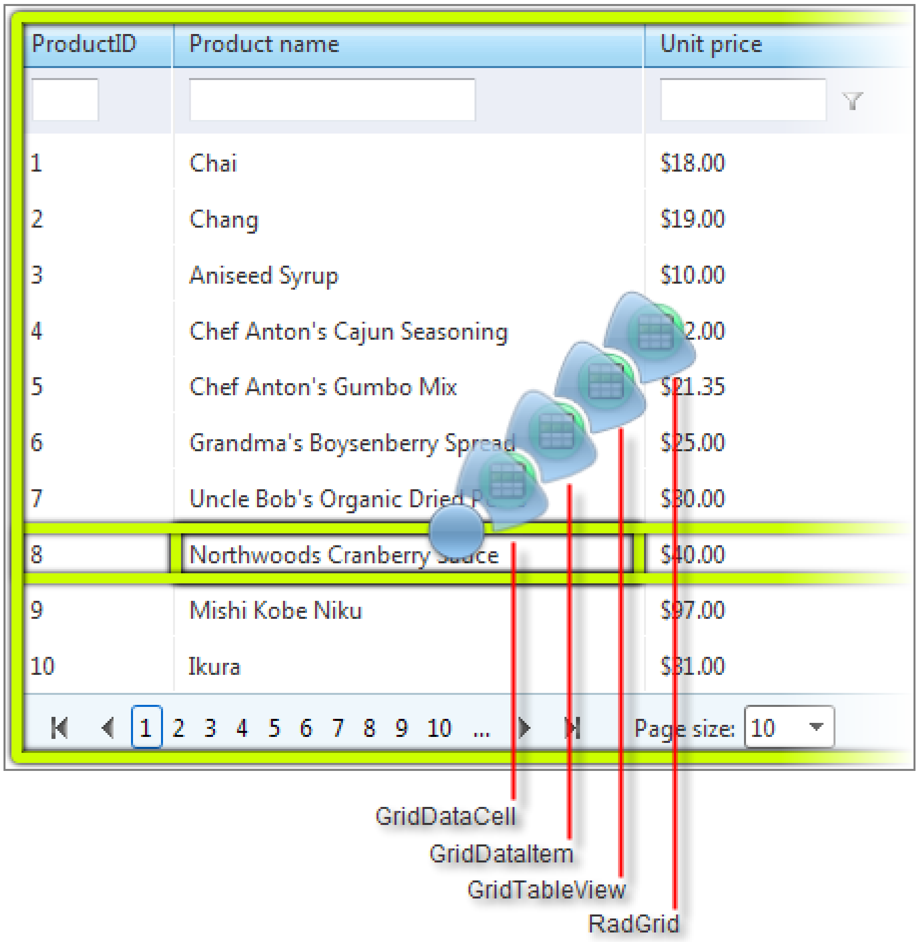


Test recording is performed directly in the browser, in the same manner as an end user would accomplish these same tasks in the browser. Navigation, text entry, clicking links, drag-and-drop, hovering the mouse are all recorded as test steps. Smart point-and-click wizards visually highlight elements and automatically generate "verifications" as test steps.


Telerik Automated Web Testing Tools | Web UI Test Studi
http://www.telerik.com/automated-testing-tools/proc
Google
Favorites
Telerik Automated Web Testing Tools | Web U...
Page
Safety
Tools
telerik
Developer Productivity
Team Productivity
Automated Testing
Web Content Management
About Telerik
Radically Better...
Why choose Teler ... sting Tools
For Developers
Visual Studio plug-in, Test Recorder
Support
Documentation, Tickets
Videos
Get started, Webinars
For QAs / Testers
Standalone tool, Revolu
ommunity
ums, Blogs, Events
Purchase
Licensing, Pricing
Quick Tasks
Generic Html Translator
Verify - text contains 'For QAs / Testers'
Verify - element is visible
Verify - element is not visible
Wait - text contains 'For QAs / Testers'
Wait - element exists
Wait - element does not exist
Wait - element is visible
Wait - element is not visible
Radic
tried
Unproductive Manual Testing
Expensive Enterprise QA Solutions
Code-intensive Testing Frameworks
Complex User Interfaces
Better product
testing framew
enterprise sol
Better, radically
Internet
100%

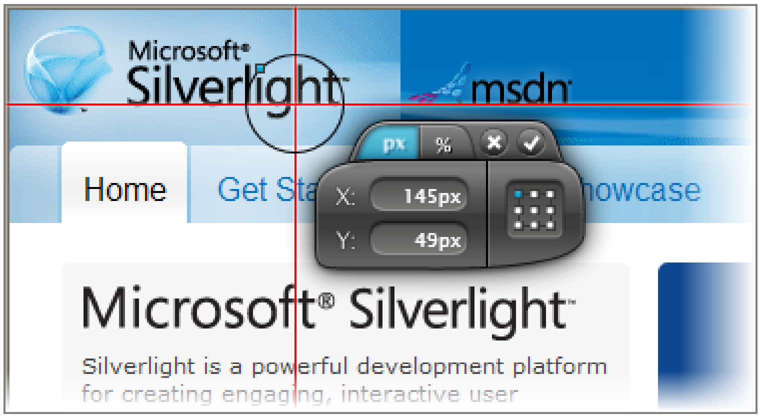
Here are just a few of the features in WebUI:

- Visibility into the internals of controls on a web page is provided by plug-in "Translators" that show the aspects of a control that can be automated and verified. For example, the translator for a RadGrid control shows grid, table, data item and cell highlighting along with menus for each. WebUI comes with the base HTML and Silverlight translators, RadControls for AJAX and RadControls for Silverlight. The Telerik model is extensible, so that WebUI can handle new controls as they come along.



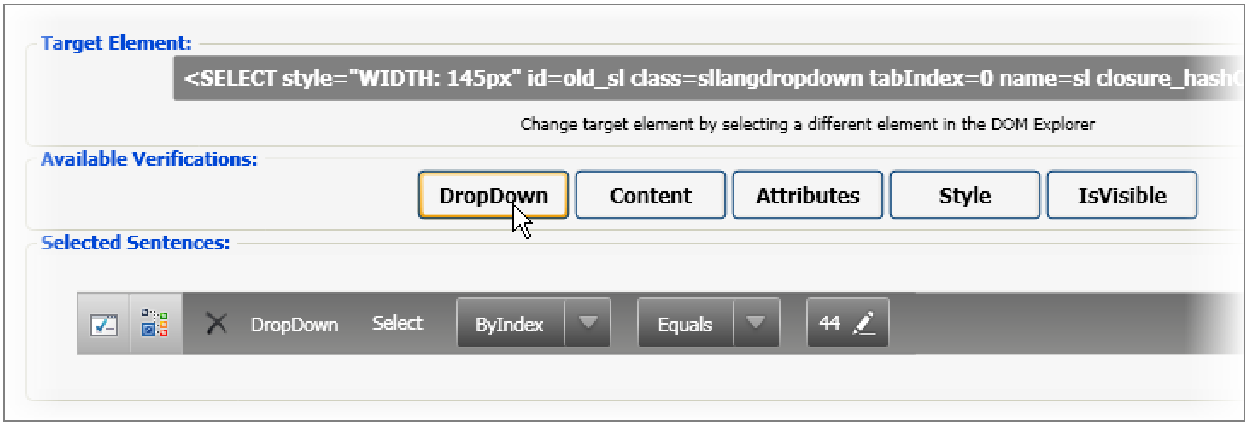
ProductID	Product name	Unit price
1	Chai	$18.00
2	Chang	$19.00
3	Aniseed Syrup	$10.00
4	Chef Anton's Cajun Seasoning	2.00
5	Chef Anton's Gumbo Mix	$21.35
6	Grandma's Boysenberry Spread	$25.00
7	Uncle Bob's Organic Dried P	$30.00
8	Northwoods Cranberry Sauce	$40.00
9	Mishi Kobe Niku	$97.00
10	Ikura	$31.00

- Drag-and-drop operations are handled automatically or you can perform precision drag-and-drop with the help of a wizard to any pixel or percentage coordinate in the entire window or within a single element.


Microsoft®
Silverlight
msdn
px
%
X: 145px
Y: 49px
Home
Get Sta
owcase
Microsoft® Silverlight
Silverlight is a powerful development platform
for creating engaging, interactive user

- WebUI will automatically record dialog handling. Testers also have the ability to insert their own automatic dialog handler verifications directly as a test step. The Test Studio provides support for Pop-up, JavaScript Alert, file upload and download, logon dialogs, and more.

- The "Sentence Based validation" tool lets you craft a wide range of verification types: attributes, styles, tables, select dropdowns, element visibility and much more. The tool guides you through choosing the verification criteria – it instantly loads the current state of the target element into the context of the selected rule.

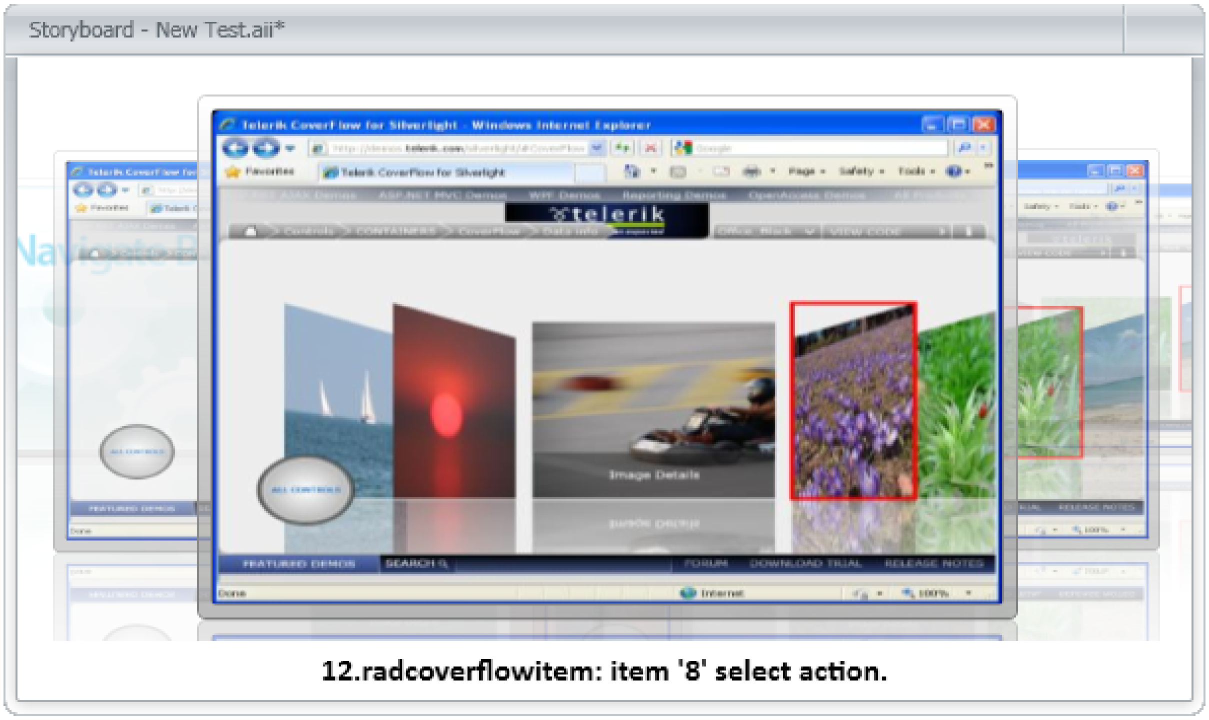

Target Element:
<SELECT style="WIDTH: 145px" id=old_sl class=sllangdropdown tabIndex=0 name=sl closure_hash

Change target element by selecting a different element in the DOM Explorer

Available Verifications:

Selected Sentences:
DropDown Select ByIndex Equals 44

- Testers can quickly explore and understand the parent/child relationship of any element on the web page using the Storyboard. WebUI provides a full set of pre-built sentence verifications for each element.



Storyboard - New Test.aii*

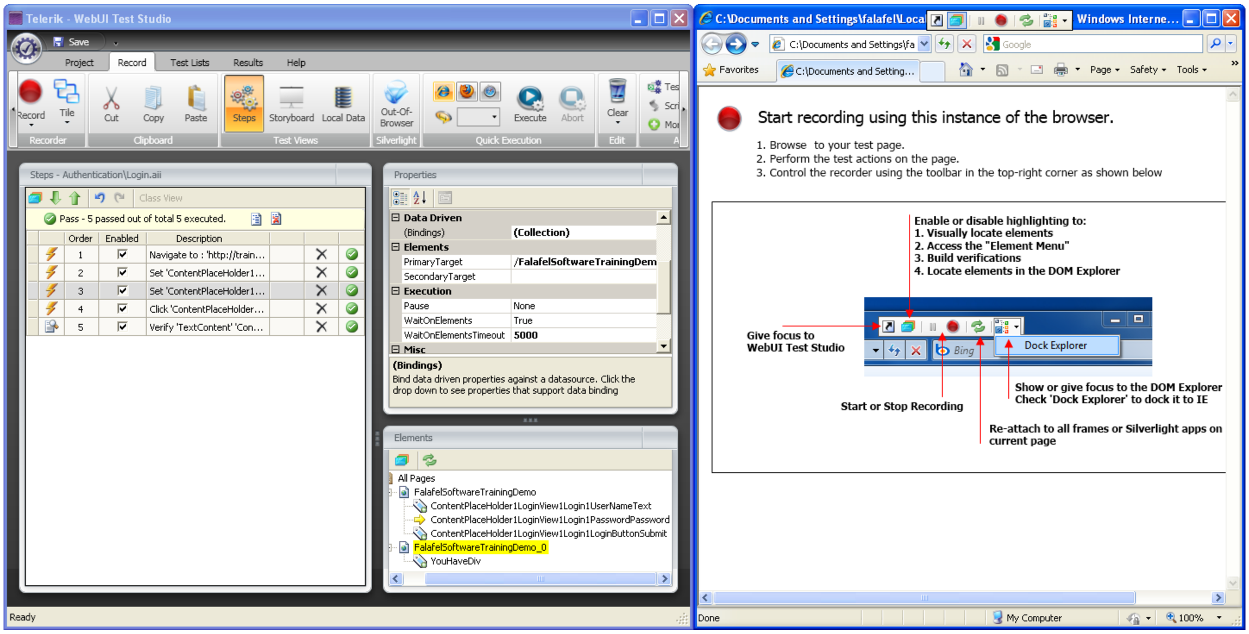
12.radcoverflowitem: item '8' select action.

- Once the test is executed and the results are analyzed, test results can be directly published to the selected project build by instantly connecting to Team Foundation Server (TFS). You also have the option to export test results to Word or Excel. In either case, there is no need to leave WebUI.

WebUI Editions

WebUI comes in two flavors: **WebUI Test Studio QA Edition** and **WebUI Studio Developer Edition**.

- **WebUI Test Studio QA Edition** is a standalone testing application plus a test recorder that works right in the browser. As the name implies, **WebUI Test Studio QA Edition** can be used by QA engineers to build, execute and evaluate tests. This courseware explains how to be productive with **WebUI Test Studio QA Edition**.



Pass - 5 passed out of total 5 executed.

Order	Enabled	Description
1	☑	Navigate to : 'http://train...
2	☑	Set 'ContentPlaceHolder1...
3	☑	Set 'ContentPlaceHolder1...
4	☑	Click 'ContentPlaceHolder...
5	☑	Verify 'TextContent' 'Con...

Data Driven	
(Bindings)	**(Collection)**
Elements	
PrimaryTarget	**/FalafelSoftwareTrainingDem**
SecondaryTarget	
Execution	
Pause	None
WaitOnElements	True
WaitOnElementsTimeout	**5000**
Misc	

(Bindings)
Bind data driven properties against a datasource. Click the drop down to see properties that support data binding

All Pages
FalafelSoftwareTrainingDemo
ContentPlaceHolder1LoginView1Login1UserNameText
ContentPlaceHolder1LoginView1Login1PasswordPassword
ContentPlaceHolder1LoginView1Login1LoginButtonSubmit
FalafelSoftwareTrainingDemo_0
YouHaveDiv

Start recording using this instance of the browser.

1. Browse to your test page.
2. Perform the test actions on the page.
3. Control the recorder using the toolbar in the top-right corner as shown below

Enable or disable highlighting to:
1. Visually locate elements
2. Access the "Element Menu"
3. Build verifications
4. Locate elements in the DOM Explorer

Give focus to WebUI Test Studio

Start or Stop Recording

Show or give focus to the DOM Explorer
Check 'Dock Explorer' to dock it to IE

Re-attach to all frames or Silverlight apps on current page

- **WebUI Studio Developer Edition** is a Visual Studio "plugin" that developers can use to perform testing directly from Visual Studio. This product also includes the test recorder for recording test automation and validation using the browser.

Test projects can be exchanged between editions using TFS. **WebUI Test Studio QA Edition** can also export projects in a Visual Studio compatible form.

Part II

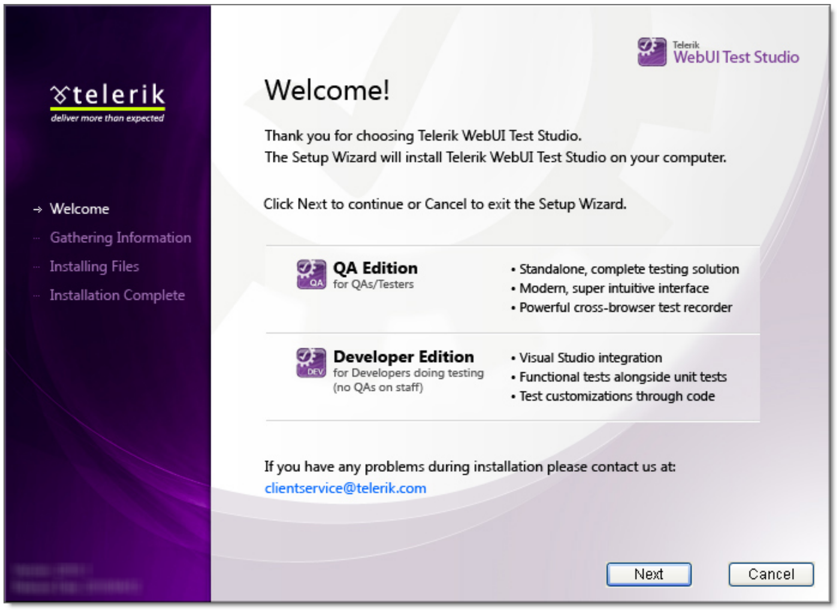
Installation

2 Installation

2.1 Install WebUI

1) To install WebUI Test Studio QA 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.



Welcome!

Thank you for choosing Telerik WebUI Test Studio.
The Setup Wizard will install Telerik WebUI Test Studio on your computer.

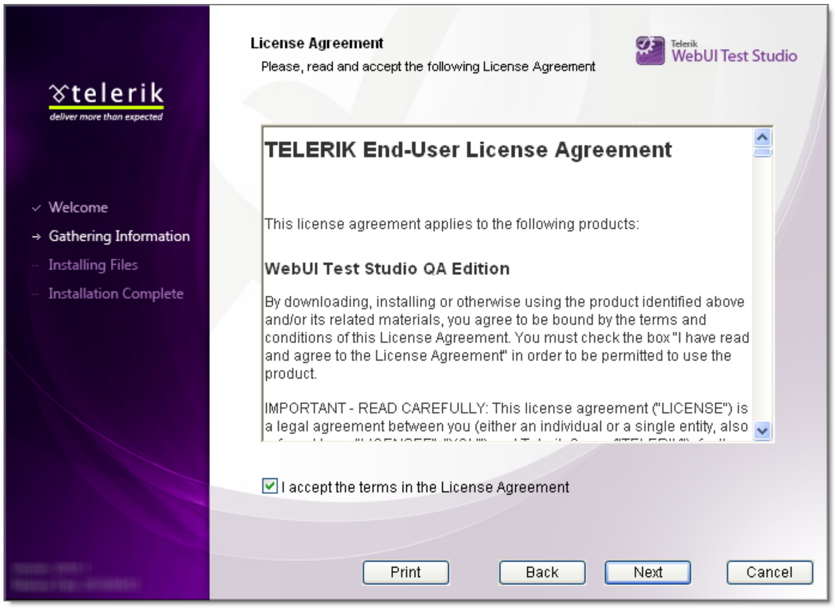
Click Next to continue or Cancel to exit the Setup Wizard.

QA Edition for QAs/Testers
- Standalone, complete testing solution
- Modern, super intuitive interface
- Powerful cross-browser test recorder

Developer Edition for Developers doing testing (no QAs on staff)
- Visual Studio integration
- Functional tests alongside unit tests
- Test customizations through code

If you have any problems during installation please contact us at:
clientservice@telerik.com

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.



License Agreement

Please, read and accept the following License Agreement

TELERIK End-User License Agreement

This license agreement applies to the following products:

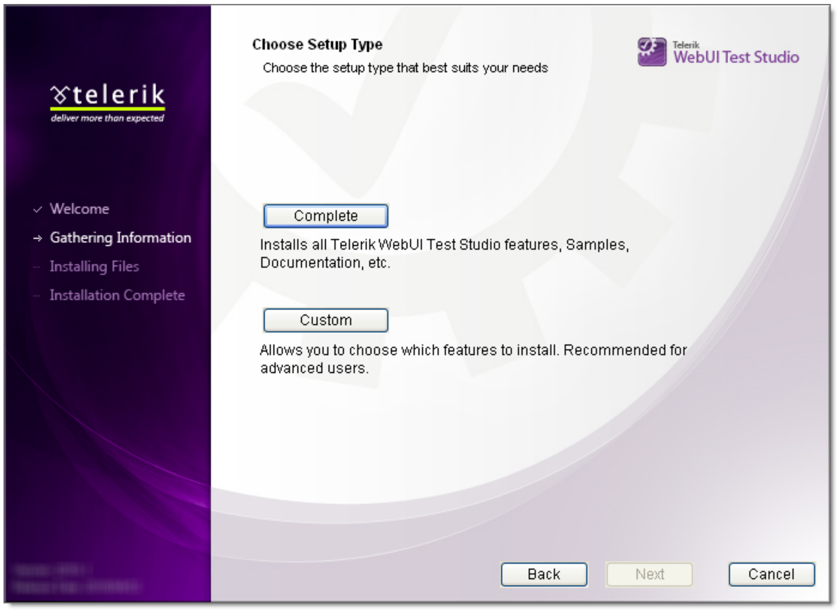
WebUI Test Studio QA Edition

By downloading, installing or otherwise using the product identified above and/or its related materials, you agree to be bound by the terms and conditions of this License Agreement. You must check the box "I have read and agree to the License Agreement" in order to be permitted to use the product.

IMPORTANT - READ CAREFULLY: This license agreement ("LICENSE") is a legal agreement between you (either an individual or a single entity, also

☑ I accept the terms in the License Agreement

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



Choose Setup Type

Choose the setup type that best suits your needs

Complete

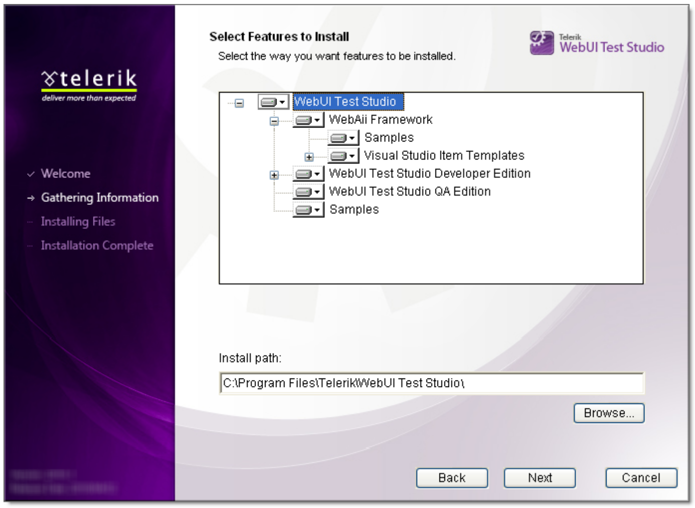
Installs all Telerik WebUI Test Studio features, Samples, Documentation, etc.

Custom

Allows you to choose which features to install. Recommended for advanced users.

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.



Select Features to Install

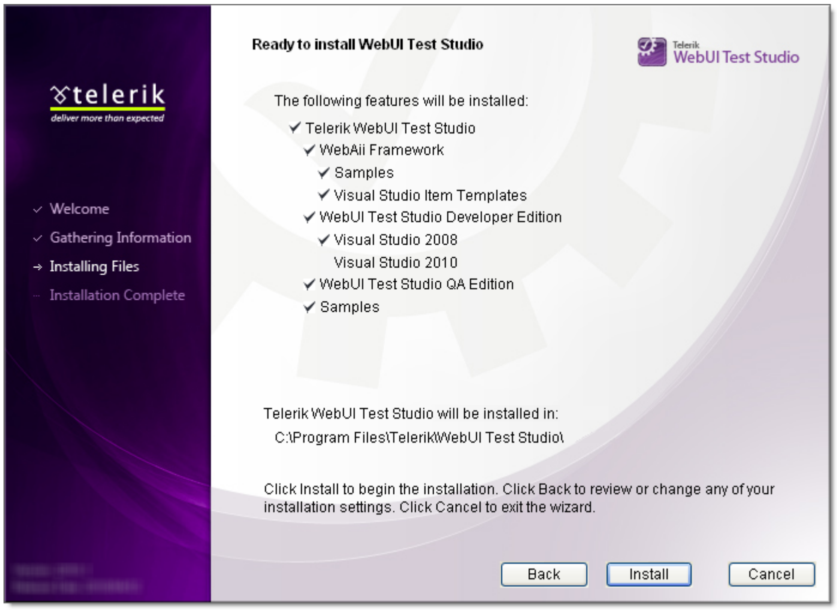
Select the way you want features to be installed.

- WebUI Test Studio
  - WebAii Framework
    - Samples
    - Visual Studio Item Templates
  - WebUI Test Studio Developer Edition
  - WebUI Test Studio QA Edition
  - Samples

Install path:

C:\Program Files\Telerik\WebUI Test Studio\

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 WebUI Test Studio QA Edition to your computer.



Ready to install WebUI Test Studio

The following features will be installed:

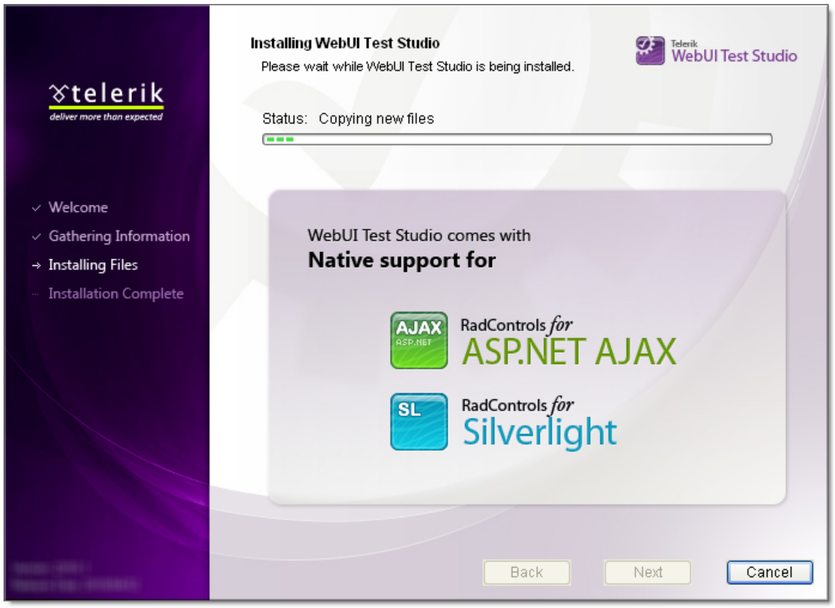
- ✓ Telerik WebUI Test Studio
  - ✓ WebAii Framework
    - ✓ Samples
    - ✓ Visual Studio Item Templates
  - ✓ WebUI Test Studio Developer Edition
    - ✓ Visual Studio 2008
    - Visual Studio 2010
  - ✓ WebUI Test Studio QA Edition
  - ✓ Samples

Telerik WebUI Test Studio will be installed in:

C:\Program Files\Telerik\WebUI Test Studio\

Click Install to begin the installation. Click Back to review or change any of your installation settings. Click Cancel to exit the wizard.

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


telerik
deliver more than expected
Welcome
Gathering Information
Installing Files
Installation Complete
Telerik WebUI Test Studio
AJAX ASP.NET
RadControls for ASP.NET AJAX
SL
RadControls for Silverlight
Back
Next
Cancel

Installing WebUI Test Studio

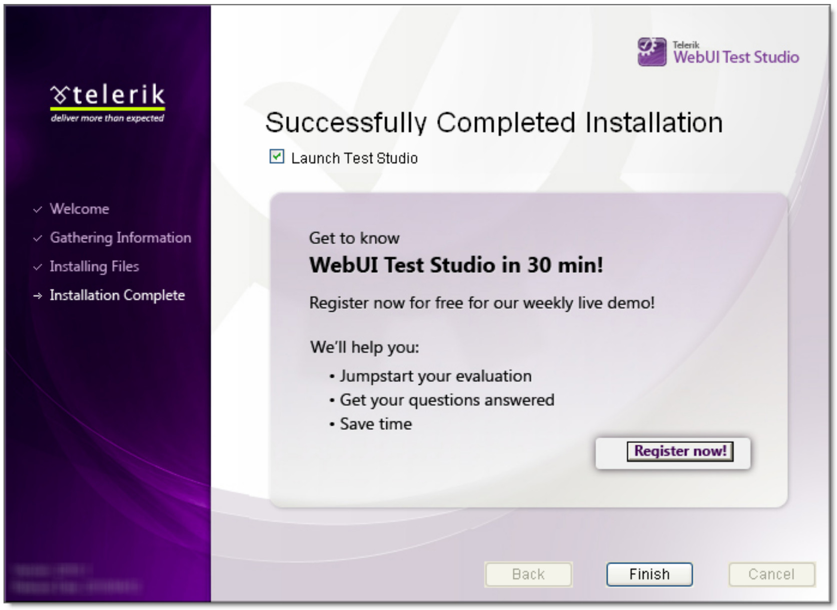
Please wait while WebUI Test Studio is being installed.

Status: Copying new files

WebUI Test Studio comes with

Native support for

7) When WebUI Test Studio QA 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.



Successfully Completed Installation

☑ Launch Test Studio

Get to know

WebUI Test Studio in 30 min!

Register now for free for our weekly live demo!

We'll help you:

- Jumpstart your evaluation
- Get your questions answered
- Save time

WebUI Test Studio QA Edition will now be available from the Windows Start menu or from a shortcut placed on the desktop.

Part III

WebUI Overview

3 WebUI Overview

3.1 Objectives

In this chapter you take a guided tour of the product so you'll know where to find everything later. You will become familiar with major areas of the product including Projects, Record, Tests lists and Results.

3.2 Overview

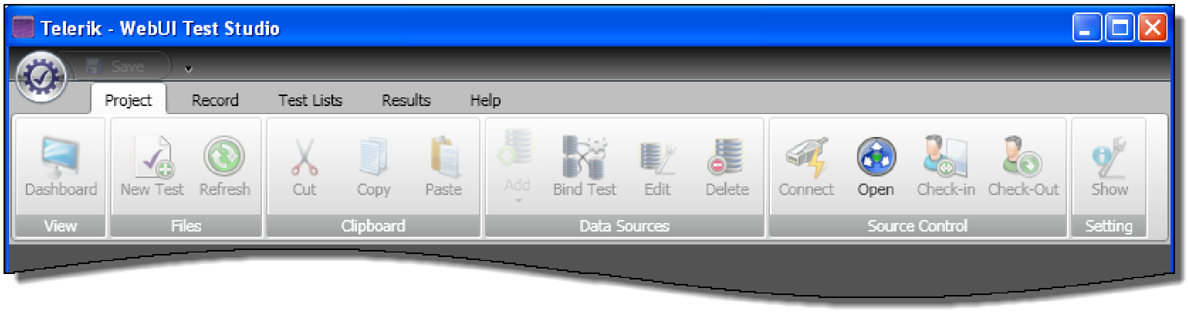
Before looking at *how* the user WebUI interface creates tests, let's take a high level view of what WebUI is doing conceptually. The main tasks that WebUI performs are:

- **Creating tests --** Tests consist of automating browser actions and verifying that browser elements are in the state we expect them to be.
- **Organizing tests** -- Tests can be organized into lists so that larger test "suites" can be built.
- **Running tests --** WebUI executes the tests in one of several supported browsers.
- **Evaluating Test Results** -- The results of the test can be analyzed and communicated to the team.

3.3 Guided Tour

When you first open WebUI you will see that there's not much fat on the user interface -- just what you need to be productive and nothing else. The Welcome Screen gets you started by letting you create new projects, open existing projects or record new tests. The ribbon bar at the top of the screen contains tabs for major functions you need to perform, in roughly the order you need to use them:

- **Project Tab**: Use this tab to manage projects, project files and data sources. You can also hook up to Microsoft's Team Foundation Server (TFS), see a "Dashboard" view of important testing activity and tweak your user settings.
- **Record Tab**: You will spend much of your time in the Record Tab building, editing and executing your tests. Additional functions on this tab let you automate parts of the test that can't be triggered by actions in the Recording Surface, e.g. handling dialogs, adding screen capture, adding other tests as steps, etc.
- **Test Lists Tab**: This tab allows you to work with multiple tests as a single entity. With Test Lists you can apply settings to all tests at one time, reorder tests and execute all the tests in the list as a single unit. The Dynamic List feature builds a test list on-the-fly during execution, based on properties and attributes of the test.
- **Results Tab**: This tab tracks the outcome of your tests, showing Pass/Fail results in a handy graphic calendar or timeline display. Use this tab to analyze the test results in place, publish to TFS or export the test results to Excel or Word.
- **Help Tab**: This tab contains support options including How-To Videos, Community Forums and Support Tickets. The tab also hosts an application log for use in trouble-shooting WebUI issues.



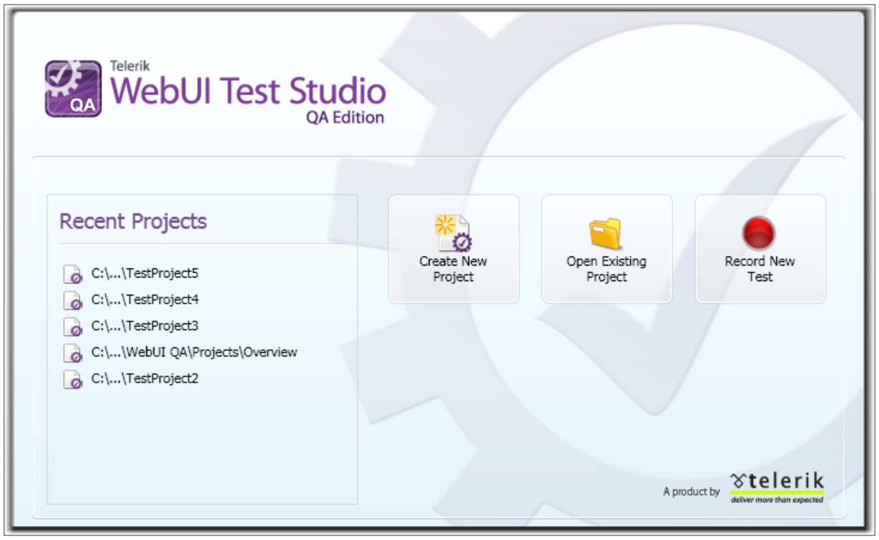
Project Tab

The Project Tab is the typical starting point for work done in WebUI. When you first open WebUI you will see a Welcome Screen. After opening a project you will be able to work with source control to check out files, organize test files, bind test data and adjust project settings.

Project Tab Views

Welcome Screen

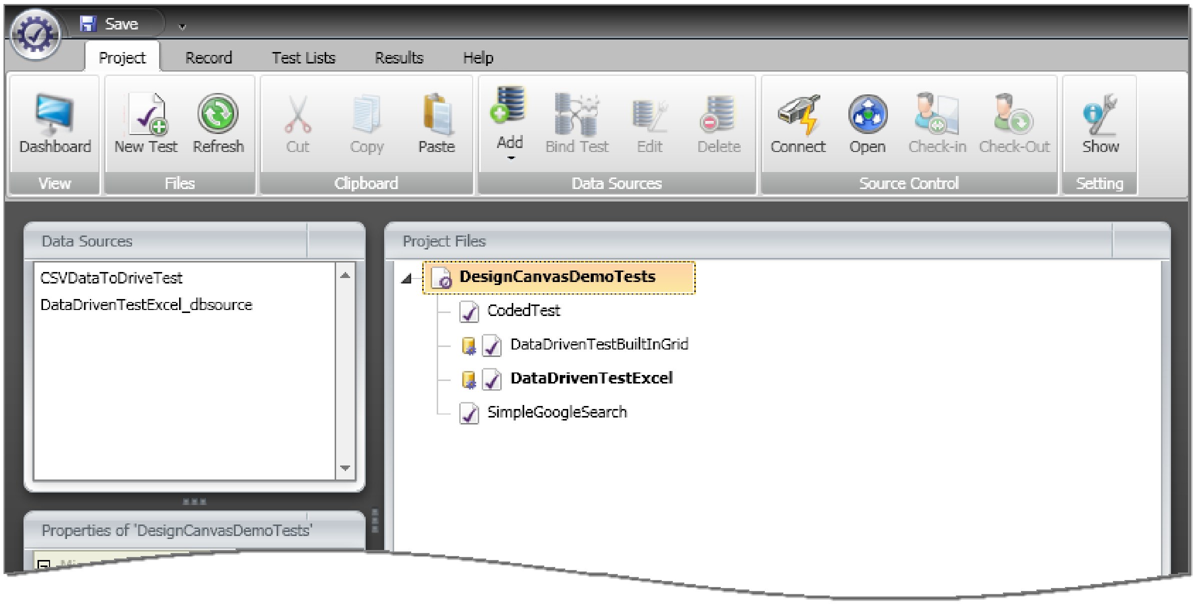
From the Welcome Screen you can create a new project or open an existing project. The **Recent Projects** list contains links that will open existing projects. You can also record a new test directly from the Welcome Screen. Clicking the **Record New Test** button will create a new project and test for you automatically.



C:\...\TestProject5
C:\...\TestProject4
C:\...\TestProject3
C:\...\WebUI QA\Projects\Overview
C:\...\TestProject2

Open Project

An open project displays panes for **Data Sources**, **Project Files** and **Properties**. The screenshot below shows how the Project Tab looks when a project is loaded.

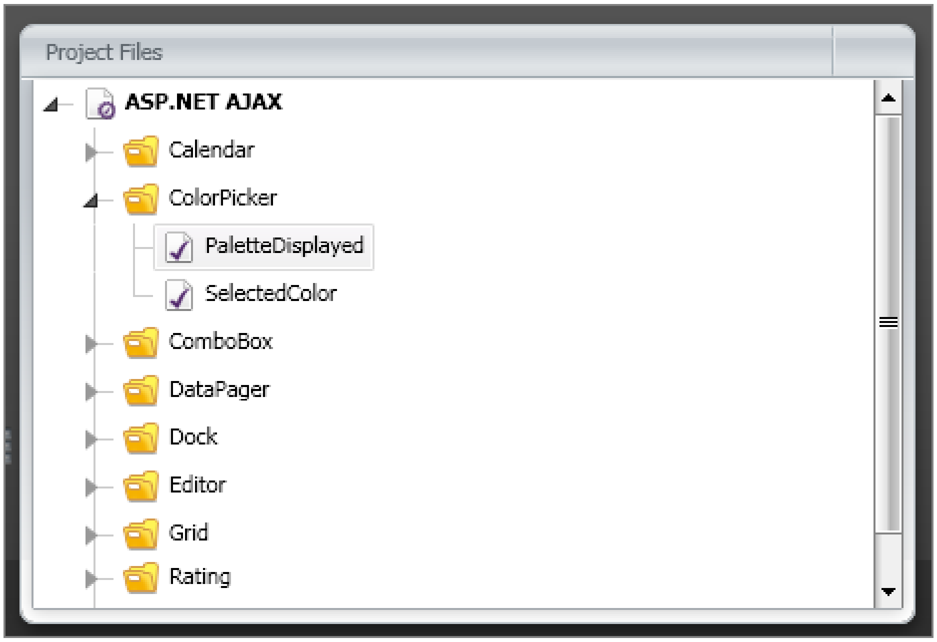


CSVDataToDriveTest
DataDrivenTestExcel_dbsource

DesignCanvasDemoTests
CodedTest
DataDrivenTestBuiltInGrid
DataDrivenTestExcel
SimpleGoogleSearch

Project Files

The **Project Files** pane has a tree view that holds the current WebUI project, folders and individual tests. The screenshot below shows the tests from the "ASP.NET AJAX" sample that ship with WebUI.

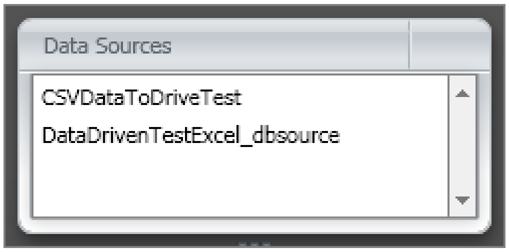


Project Files

ASP.NET AJAX
Calendar
ColorPicker
PaletteDisplayed
SelectedColor
ComboBox
DataPager
Dock
Editor
Grid
Rating

Data Sources

The **Data Sources** Pane lists data sources that can be associated with or "bound" to your test. The "Data Driven Tests" chapter will cover data binding in-depth. For now, know that items in the Data Sources Pane can retrieve data from comma delimited text "CSV" files, Excel files, XML files and database tables including MS SQL, Oracle and ODBC. The data from these sources can be used to supply properties for test automation and verification. The screenshot below shows entries for "CSV" and "Excel" data sources.

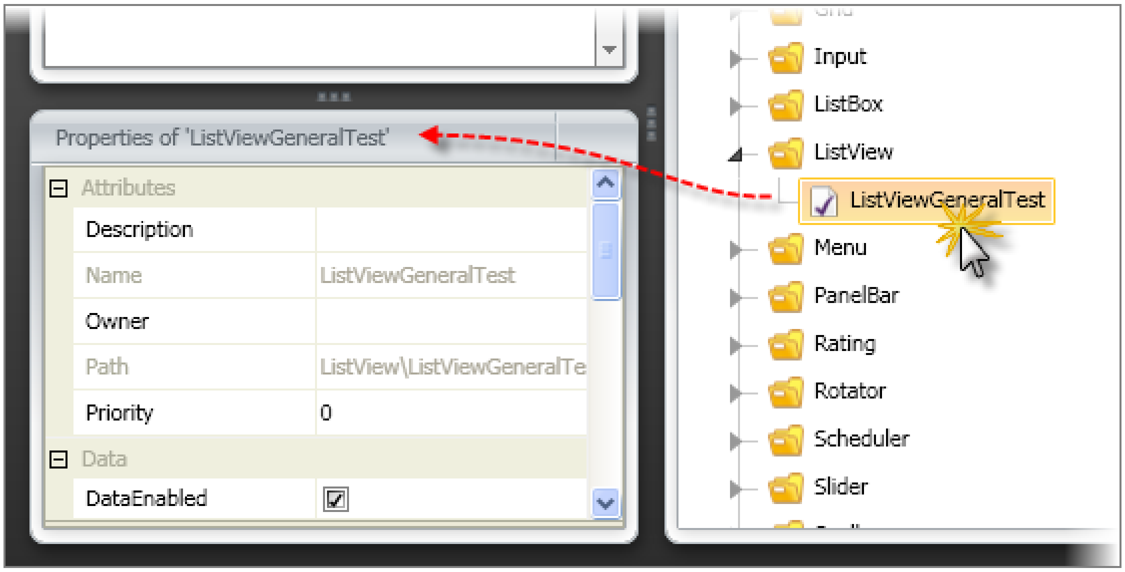


Data Sources

CSVDataToDriveTest
DataDrivenTestExcel_dbsource

Properties

The **Properties** Pane reflects the attributes of anything that you click on in the Project Pane and in some cases allow you to change these properties. For example, if you click a test node in the Project Files Pane, the "Name" and "Path" of the test will display as read-only, but other properties like "Description", "Owner" and "Priority" allow you to enter values directly.



Properties of 'ListViewGeneralTest'

⊟ Attributes

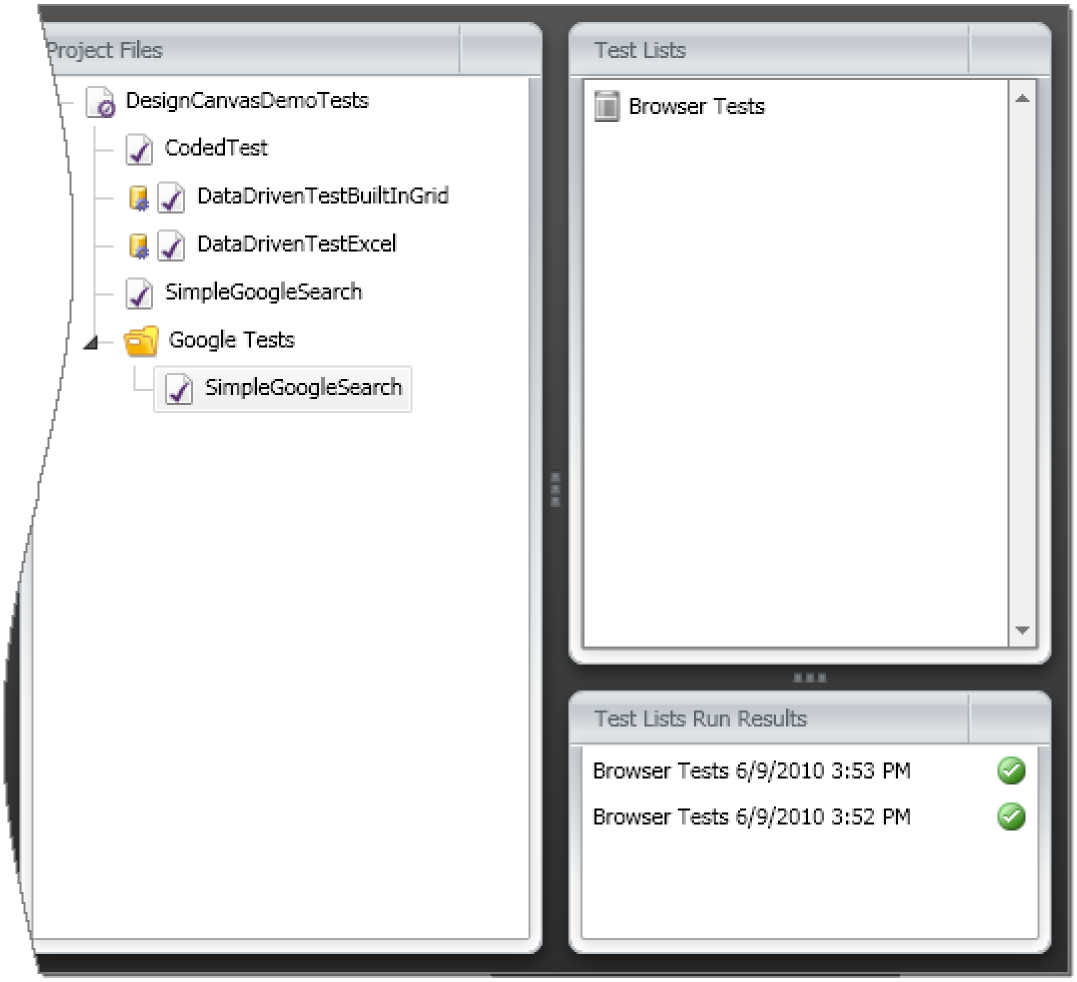
Description	
Name	ListViewGeneralTest
Owner	
Path	ListView\ListViewGeneralTe
Priority	0

⊟ Data

DataEnabled	☑

The Dashboard View

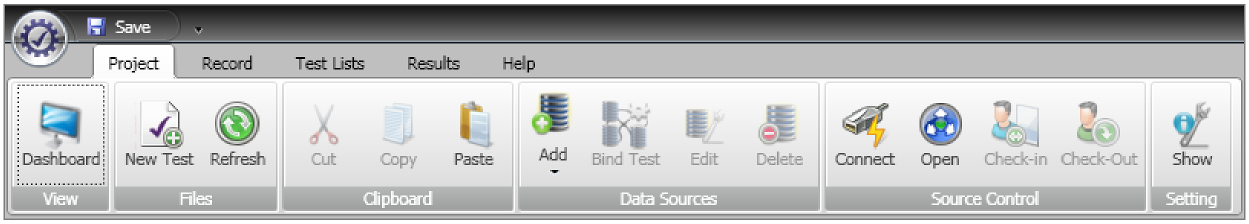
With the Dashboard feature on, two new panes display to the right side of Project Files: **Tests Lists** and **Test Lists Run Results**. These panes show at-a-glance that we have a Test List called "Browser Tests" and that two executions of "Browser Tests" have been run. Double-click the entry to navigate to the Test List Screen where you can edit or run the Test List. Likewise, you can double-click entries in the Test Lists Run Results.



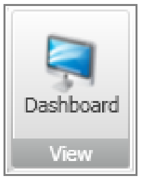
Browser Tests 6/9/2010 3:53 PM ✓
Browser Tests 6/9/2010 3:52 PM ✓

Project Tab Tools

The ribbon bar has sections of tools for **View**, **Clipboard**, **Data Sources**, **Source Control** and **Settings**.

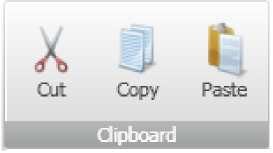


View



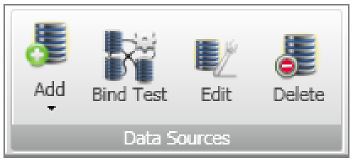
The **View** section of the ribbon bar contains only the **Dashboard** button. Toggling the Dashboard button on allows you to get a snapshot view of all your testing activity at one time.

Clipboard



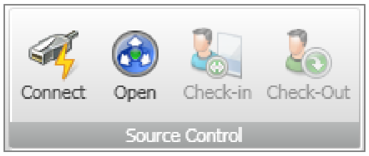
The **Clipboard** section of the ribbon bar allows you to **Cut**, **Copy** and **Paste** folders and tests. These same options are also available from the context menu of the Project Files Pane.

Data Sources



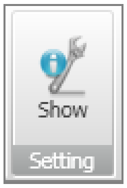
The **Data Sources** buttons are context sensitive to the currently selected items in the Data Sources and Project Files panes. These buttons allow you to create and configure data sources so that data can be consumed by your tests.

Source Control



The **Source Control** buttons allow you to work with Team Foundation Server (TFS) so that you can effectively version-control your tests. The **Connect** button displays a Connection Setup dialog that allows you to reach any TFS server over standard HTTP or HTTPS secure connection.

Setting



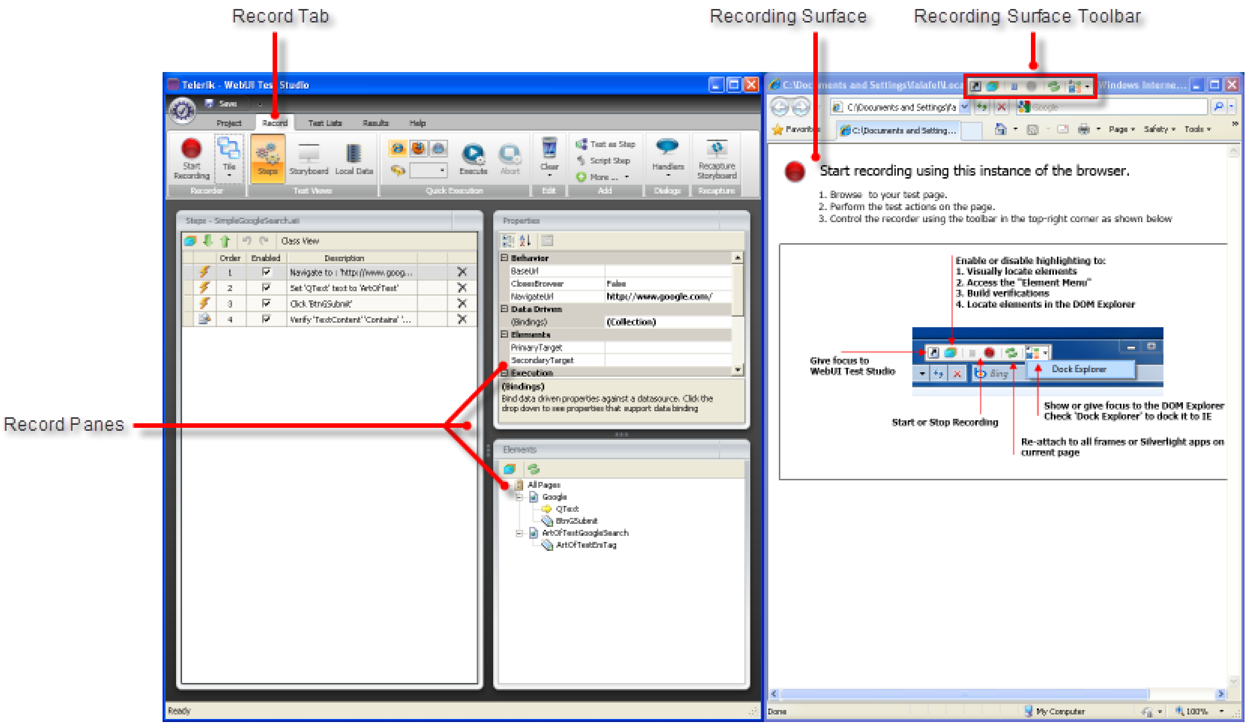
The **Setting** section has a single button **Show** that displays the **User Settings** dialog. Use this dialog to configure how tests are recorded, fine tune how web page elements are located, extend WebUI with new "translators" to provide special internal knowledge of web page controls and get information about the current WebUI installation. Many of these settings can be quite specialized, so expect to use the defaults as you get started with WebUI and use the settings as you find need for them.

Record Tab

The Record Tab works in tandem with the **Recording Surface**, a special Internet Explorer browser instance that records all your actions in the browser as test steps. The Recording Surface packs quite a bit of functionality and includes a floating toolbar, an "Elements Menu" and a Common Tasks Menu.

Using the tools in the Record Tab and the Recording Surface, you can perform just about any test building task including:

- Automate any of the browser functions e.g. "forward", "go to url", etc.
- Automate actions within the browser e.g. clicking buttons, entering keystrokes, etc.
- Handle pop-up dialogs such as alerts or login dialogs.
- Navigate and organize test steps.
- Examine and manipulate web page element properties.
- Capture images of the browser or desktop.
- Add specialized test steps to execute other tests, delay a test, clear the browser cookies, wait for a URL to open and add custom annotations to the test.
- Try out your test by running it in any of the supported browsers.



Order	Enabled	Description
1	☑	Navigate to : 'http://www.goog...
2	☑	Set 'QText' text to 'ArtOfTest'
3	☑	Click 'BtnGSubmit'
4	☑	Verify 'TextContent' 'Contains' '...

Property	Value
Behavior	
BaseUrl	
CloseBrowser	False
NavigateUrl	https://www.google.com/
Data Driven	
(Bindings)	(Collection)
Elements	
PrimaryTarget	
SecondaryTarget	
Execution	

(Bindings)
Bind data driven properties against a datasource. Click the drop down to see properties that support data binding

Start recording using this instance of the browser.

1. Browse to your test page.
2. Perform the test actions on the page.
3. Control the recorder using the toolbar in the top-right corner as shown below

Enable or disable highlighting to:
1. Visually locate elements
2. Access the "Element Menu"
3. Build verifications
4. Locate elements in the DOM Explorer

Give focus to WebUI Test Studio

Start or Stop Recording

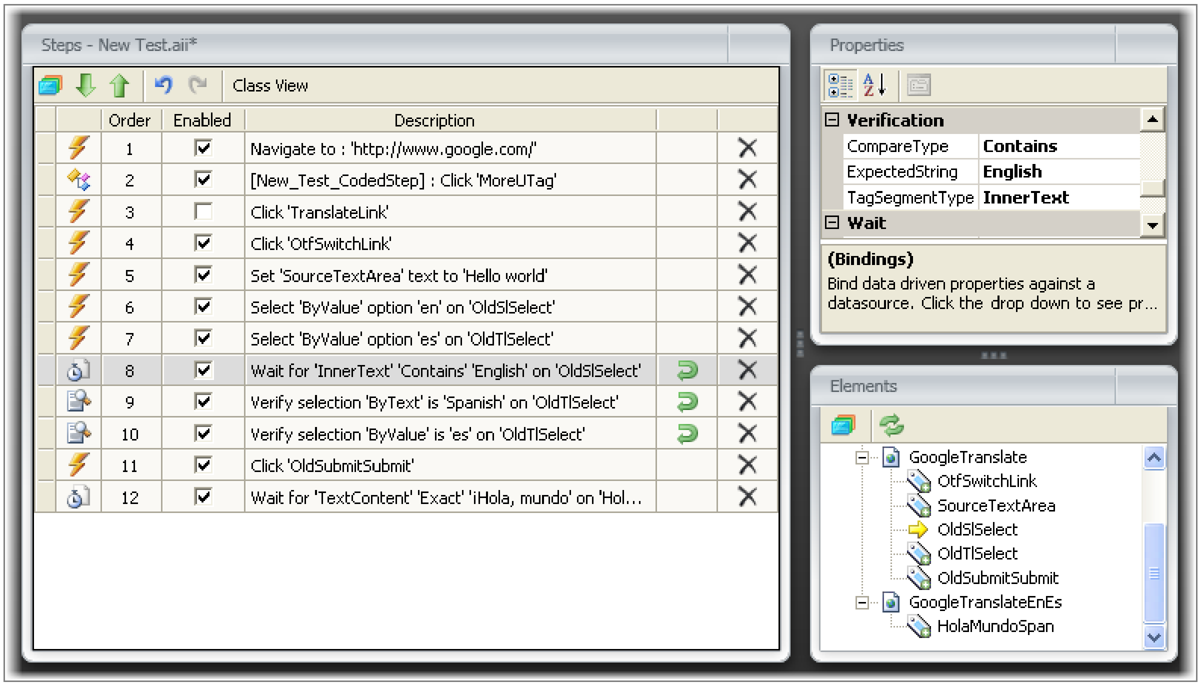
Show or give focus to the DOM Explorer Check 'Dock Explorer' to dock it to IE

Re-attach to all frames or Silverlight apps on current page

Record Tab Views

Steps

The **Steps view** holds three related panes for **Steps Elements** and **Properties**. As test steps are selected in the Steps Pane, the Properties Pane reflects the attributes for that step while the Elements Pane shows the web page element being tested. The screenshot below shows that "Verify 'InnerText' 'Contains' 'English'" is selected in the Steps Pane. The Properties Pane allows you to work with the properties that make up the verification, e.g. "CompareType", "ExpectedString" and "TagSegmentType". Likewise, the Elements Pane shows the element "OldSlSelect" is selected.



Order	Enabled	Description
1	☑	Navigate to : 'http://www.google.com/'
2	☑	[New_Test_CodedStep] : Click 'MoreUTag'
3	□	Click 'TranslateLink'
4	☑	Click 'OtfSwitchLink'
5	☑	Set 'SourceTextArea' text to 'Hello world'
6	☑	Select 'ByValue' option 'en' on 'OldSlSelect'
7	☑	Select 'ByValue' option 'es' on 'OldTlSelect'
8	☑	Wait for 'InnerText' 'Contains' 'English' on 'OldSlSelect'
9	☑	Verify selection 'ByText' is 'Spanish' on 'OldTlSelect'
10	☑	Verify selection 'ByValue' is 'es' on 'OldTlSelect'
11	☑	Click 'OldSubmitSubmit'
12	☑	Wait for 'TextContent' 'Exact' 'iHola, mundo' on 'Hol...

⊟ **Verification**

CompareType	**Contains**
ExpectedString	**English**
TagSegmentType	**InnerText**

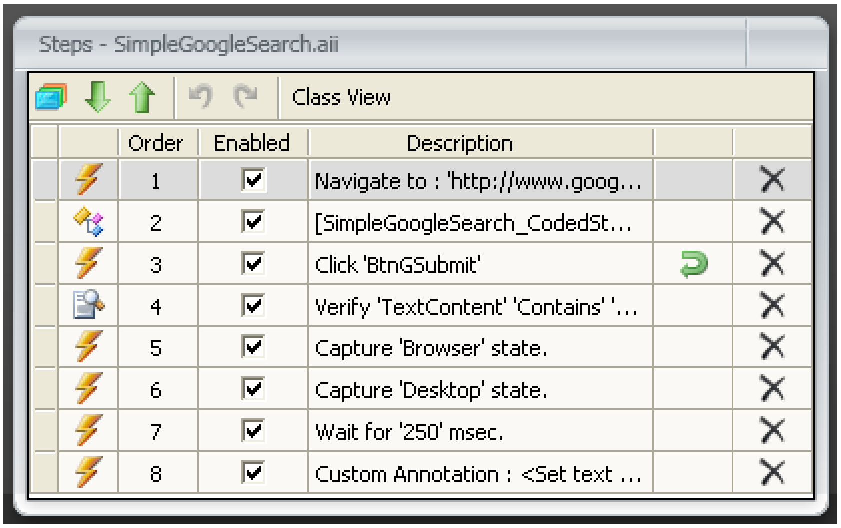
⊟ **Wait**

(Bindings)
Bind data driven properties against a datasource. Click the drop down to see pr...

- GoogleTranslate
  - OtfSwitchLink
  - SourceTextArea
  - OldSlSelect
  - OldTlSelect
  - OldSubmitSubmit
- GoogleTranslateEnEs
  - HolaMundoSpan

Steps Pane

The **Steps Pane** lists the test steps in the order they will run. The buttons across the top of the Steps pane allow you to reorder selected test steps, highlight corresponding screen elements, undo/redo actions taken in the grid and display a code view for the entire test. Buttons within the grid allow you to disable or delete steps.

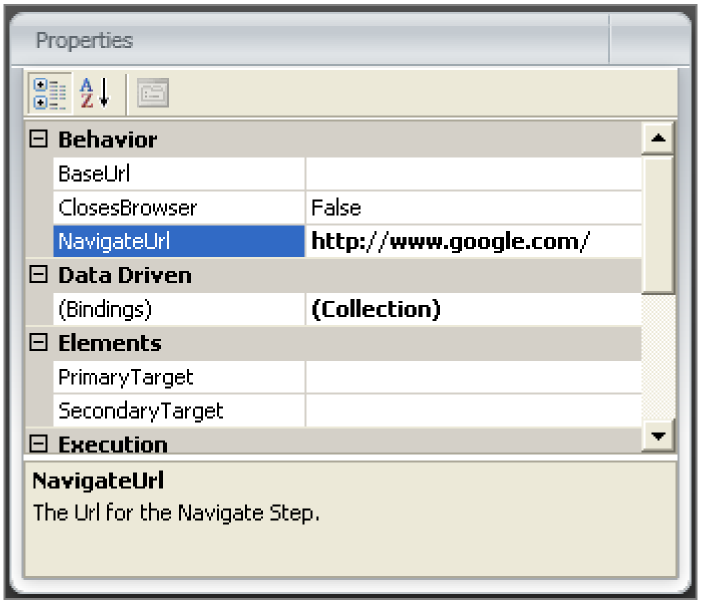


Steps - SimpleGoogleSearch.aii

	Order	Enabled	Description		
	1	☑	Navigate to : 'http://www.goog...		✕
	2	☑	[SimpleGoogleSearch_CodedSt...		✕
	3	☑	Click 'BtnGSubmit'	↩	✕
	4	☑	Verify 'TextContent' 'Contains' '...		✕
	5	☑	Capture 'Browser' state.		✕
	6	☑	Capture 'Desktop' state.		✕
	7	☑	Wait for '250' msec.		✕
	8	☑	Custom Annotation : <Set text ...		✕

Properties Pane

The **Properties Pane** lets you interact with a selected test step to change how that step functions. The screenshot below shows how a "Navigate" step looks in the Properties Pane. For example, you could change the "NavigateUrl" property to use a different URL, or have the test use the "Pause" property before or after the test step. Notice the property description at the bottom of the pane that explains the meaning of a selected property.



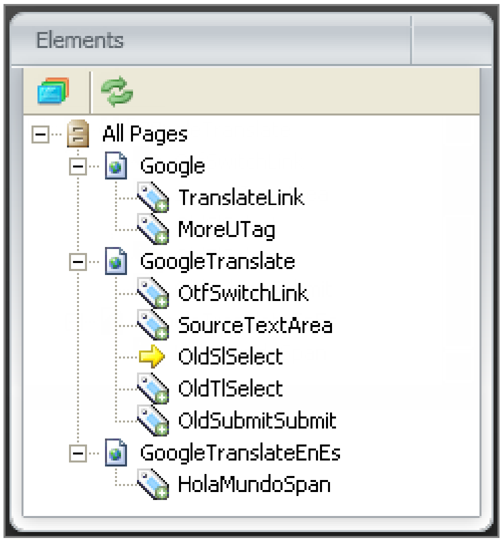
Properties

⊟ **Behavior**	
BaseUrl	
ClosesBrowser	False
NavigateUrl	**http://www.google.com/**
⊟ **Data Driven**	
(Bindings)	**(Collection)**
⊟ **Elements**	
PrimaryTarget	
SecondaryTarget	
⊟ **Execution**	

NavigateUrl
The Url for the Navigate Step.

Elements Pane

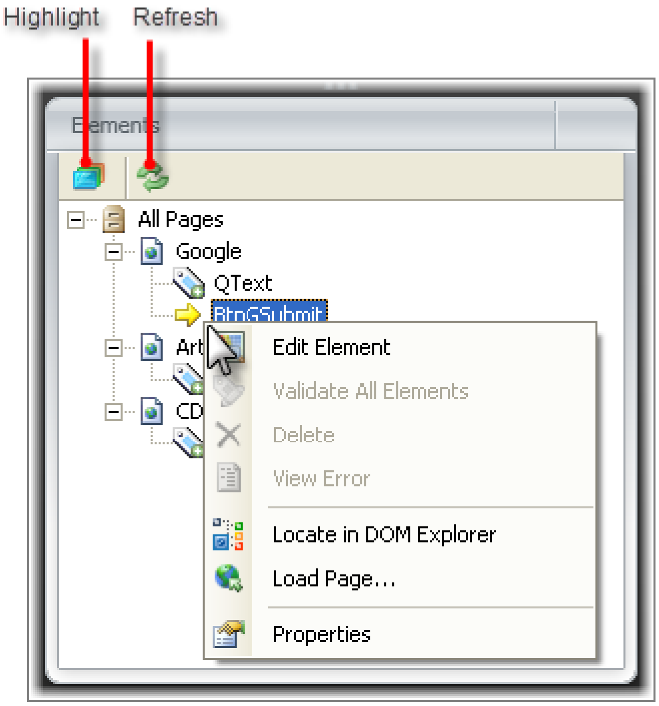
The **Elements Pane** displays a tree view of elements that are being recorded against in the Recording Surface. This pane only contains elements you want to use in your tests, unlike the DOM Explorer that encompasses everything on the web page. Although elements may be used in several tests and test steps, each element is shown only once in the Elements Pane, allowing you to centralize configuration in one place. A context menu for the tree view allows you to work with elements and the web page, e.g. configure and validate elements, reload the page, locate an element in the DOM, etc.



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.



The screenshot above shows the context menu for items in the Elements Pane 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.

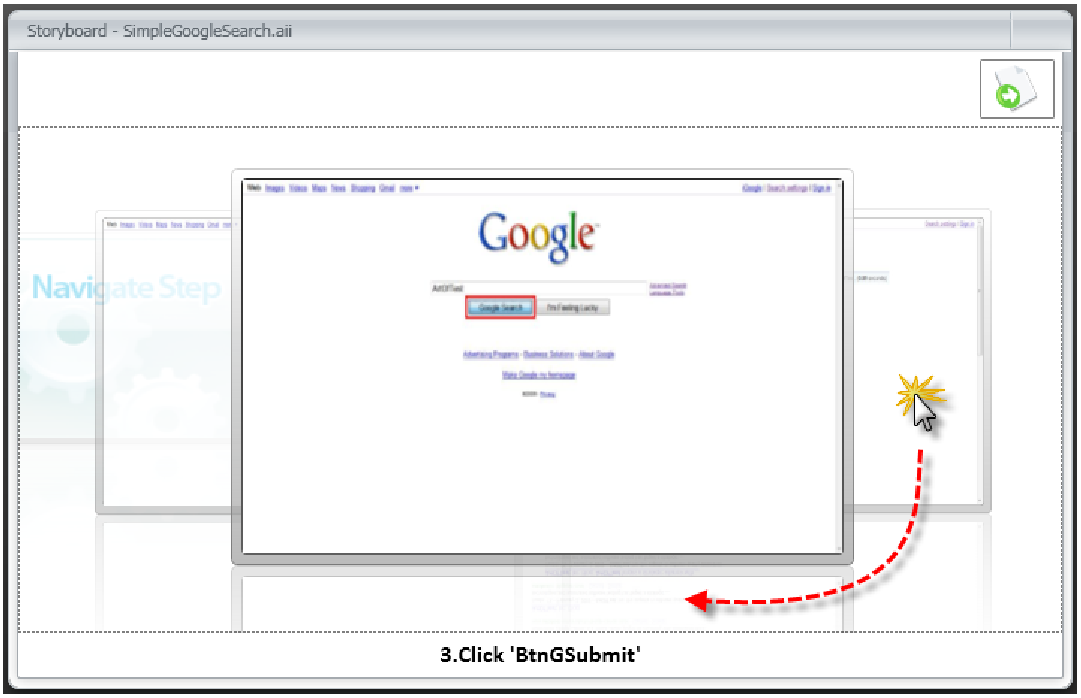
- **Locate in DOM Explorer** navigates to the DOM Explorer and selects the element being acted on in the selected test step.

- **Load Page...** loads or re-loads the page that contains the element. Note that in order to have the "Locate in DOM Explorer" option available, you need to load the page first.

- **Properties** navigates to the Properties Pane and displays properties for the selected element.

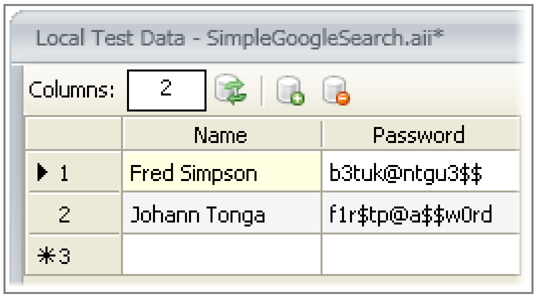
Storyboard

The **Storyboard** is a three-dimensional, visual representation of test steps. WebUI 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 is synchronized with the selected item of the Steps Pane.



Local Data

The **Local Data** view 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 for such a simple task. In this situation you could define "name" and "password" columns in a Local Data table, add several sample rows and immediately run your test. The screenshot below shows Local Data with "Name" and "Password" columns and two rows of data defined.

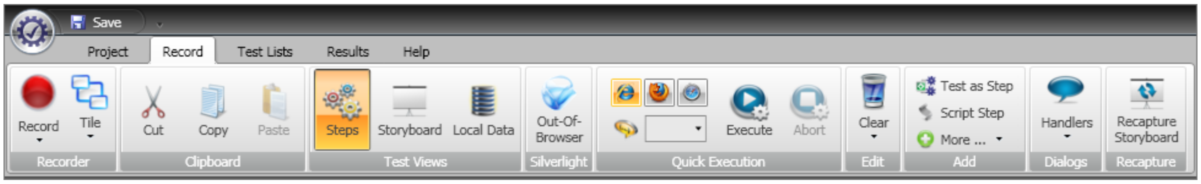


Columns: 2

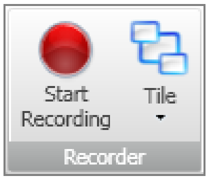
	Name	Password
▶ 1	Fred Simpson	b3tuk@ntgu3$$
2	Johann Tonga	f1r$tp@a$$w0rd
✱3		

Record Tab Tools

The ribbon bar for **Record** has sections of tools for **Recorder**, **Test Views**, **Quick Execution**, **Edit**, **Add**, **Dialogs** and **Recapture**.

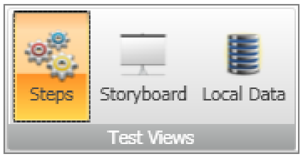


Recorder



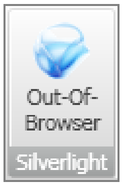
The **Recorder** section of the ribbon bar contains a **Start Recording** button that launches the Recording Surface side-by-side with WebUI. The **Tile** button can place the Recording Surface on the left or right of the WebUI window.

Test Views



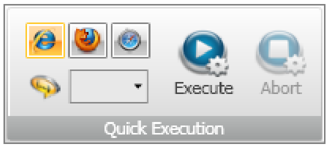
The **Test Views** section of the ribbon bar has buttons that switch between the **Steps**, **Storyboard** and **Local Data** views.

Silverlight



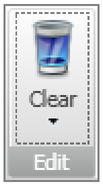
The Silverlight option lets you configure your test to use Silverlight in a browser or as an out-of-browser application.

Quick Execution



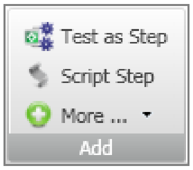
The **Quick Execution** section of the ribbon bar allows you to "audition" your tests in several different browsers, i.e. **Internet Explorer**, **Firefox** or **Safari**. Other Quick Execution tools configure the test execution to slow the test run down and have the browser annotate each step with a brief message.

Edit



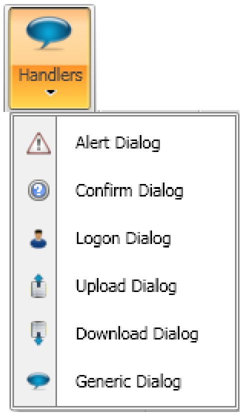
The **Edit** portion of the ribbon bar has a single **Clear** button with options for clearing all steps in the current test and a second option to remove the script file.

Add



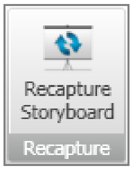
The **Add** section of the ribbon bar allows you to add steps to your test that can't be triggered interactively in the Recording Surface including screen captures, delays, annotations, coded steps and other tests performed as a single step.

Dialogs



The **Dialogs** section of the ribbon bar has a single **Handlers** button with multiple options that respond to dialogs that pop up in the web page including alerts, confirmations and other web browser instances.

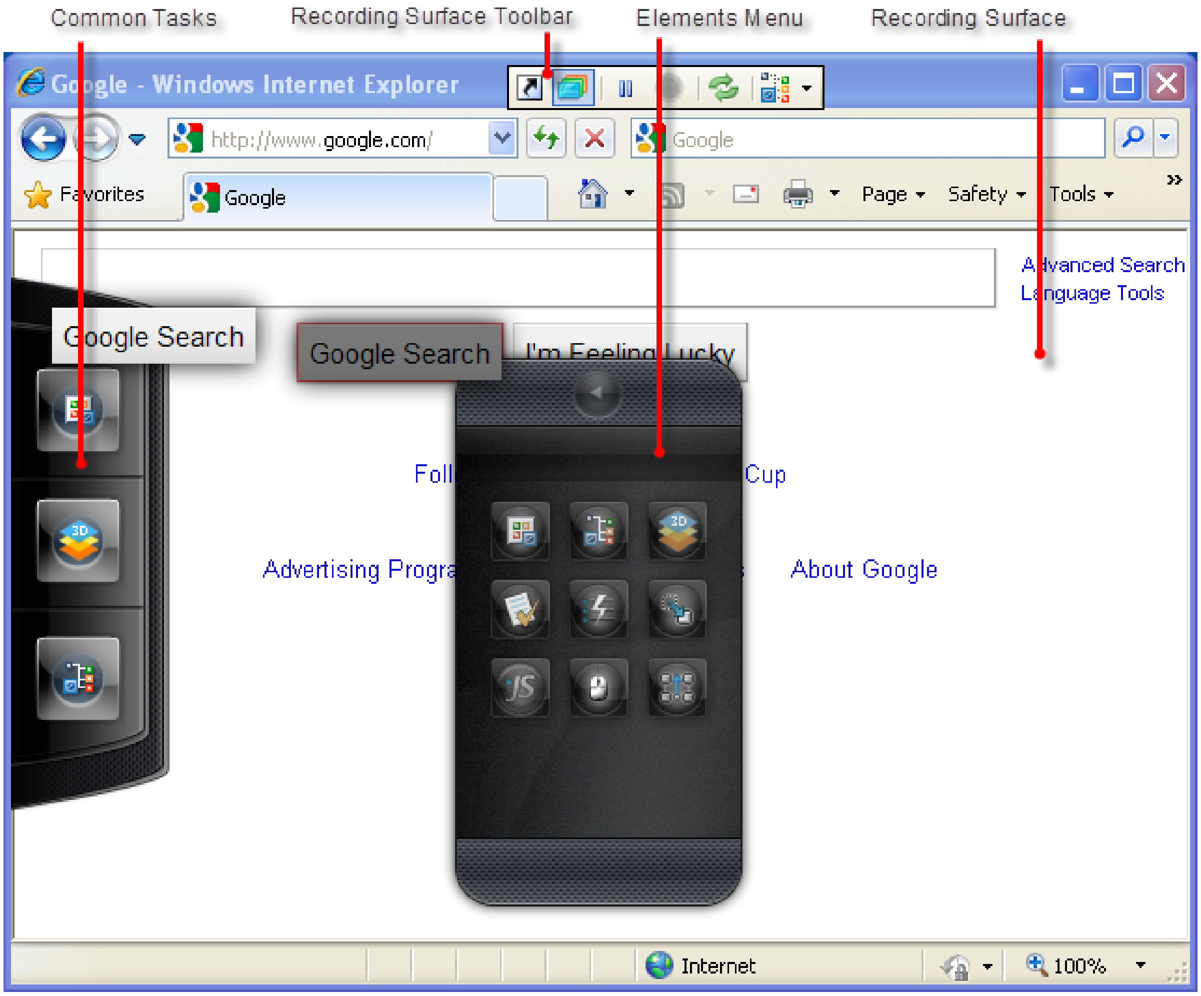
Recapture



The **Recapture Storyboard** button runs your test and stores new screenshots of each test step to the storyboard. This allows you to refresh your baseline of screenshots without having to rebuild each screenshot by hand.

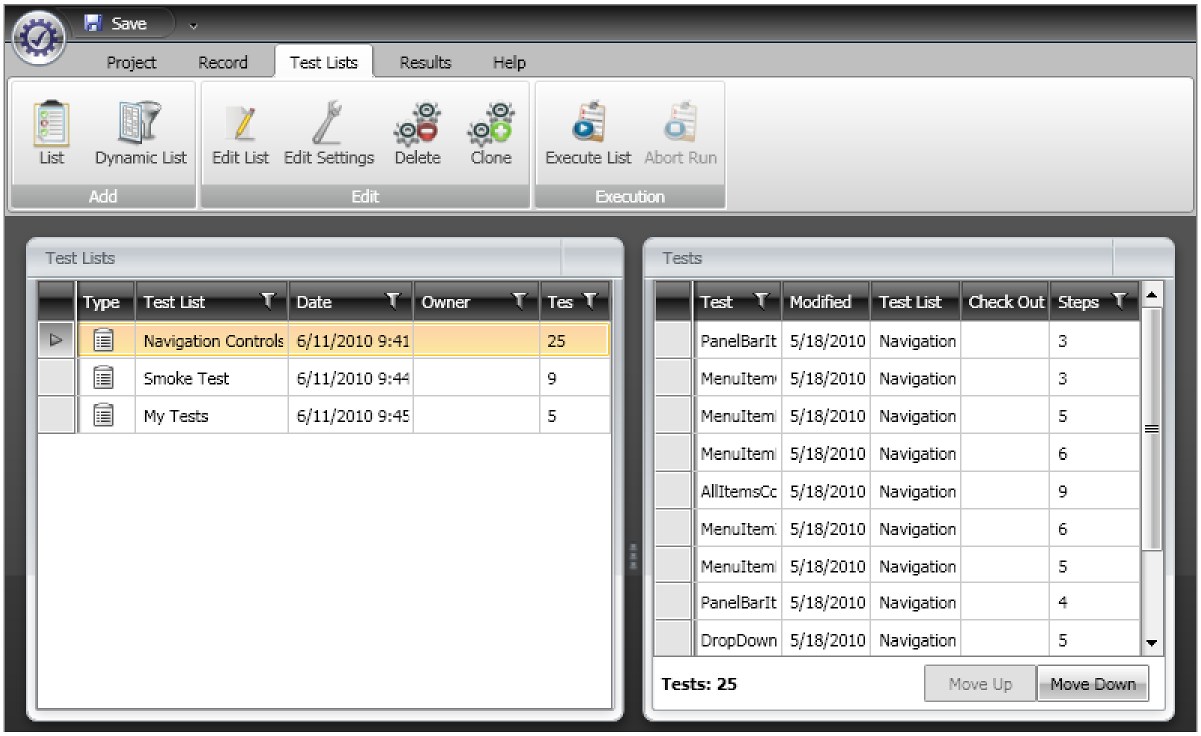
Recording Surface

The Recording Surface contains all the tools you need to interactively record tests, directly within a browser instance. The Recording Surface Toolbar provides overall control of the recording while the Elements Menu allows you to drill down to specific tasks, such as adding verifications or handling drag-and-drop operations. The Common Tasks menu provides quick access to element related tasks. The "Recording Tests" chapter will expand on the rich functionality of the Recording Surface.



Test Lists Tab

The Test Lists Tab allow you to arrange existing tests into lists that can be configured, executed and evaluated as a single entity. Using the tools in the Test Lists Tab you can manage static lists of tests or create dynamic test lists on-the-fly that respond to rules at the time of execution. See the "Test Organization" chapter for details on creating, configuring, executing and analyzing test list results.



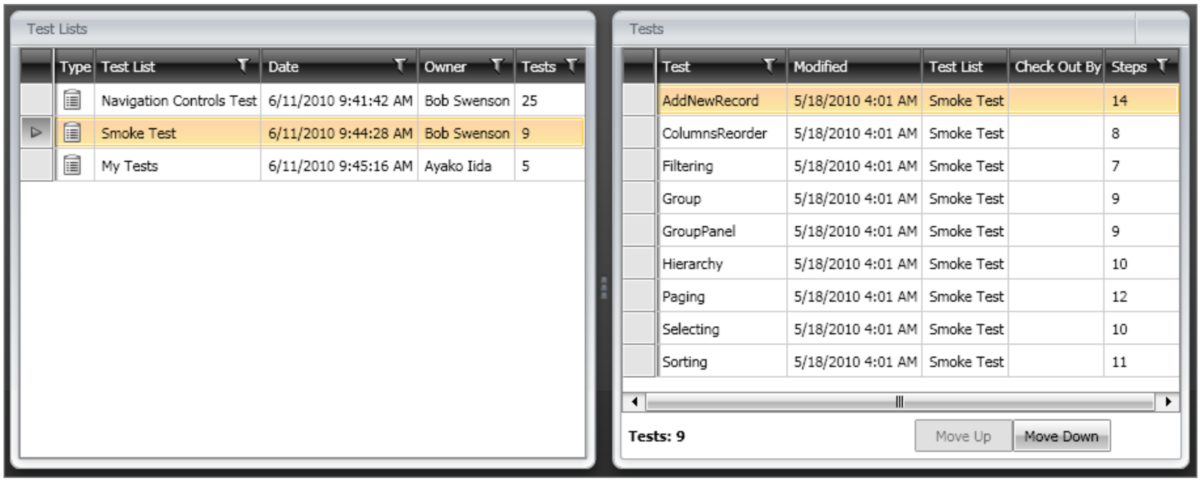
Type	Test List	Date	Owner	Tes
	Navigation Controls	6/11/2010 9:41		25
	Smoke Test	6/11/2010 9:44		9
	My Tests	6/11/2010 9:45		5

Test	Modified	Test List	Check Out	Steps
PanelBarIt	5/18/2010	Navigation		3
MenuItem	5/18/2010	Navigation		3
MenuItem	5/18/2010	Navigation		5
MenuItem	5/18/2010	Navigation		6
AllItemsCc	5/18/2010	Navigation		9
MenuItem	5/18/2010	Navigation		6
MenuItem	5/18/2010	Navigation		5
PanelBarIt	5/18/2010	Navigation		4
DropDown	5/18/2010	Navigation		5

Tests: 25

Test Lists Tab Views

The Test Lists Tab has only a single view with two panels **Test Lists** and **Tests**, where each Test List contains multiple tests.



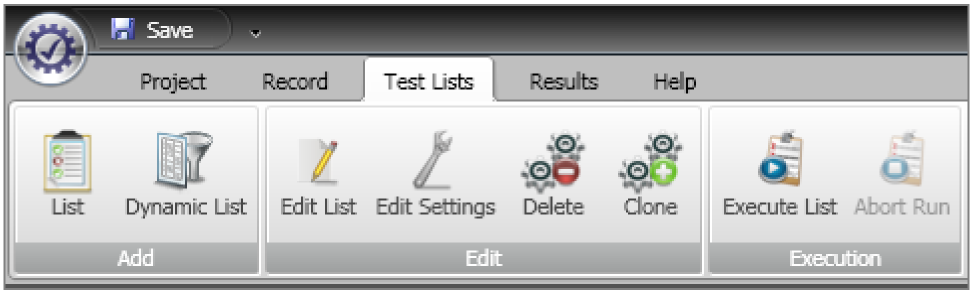
Type	Test List	Date	Owner	Tests
	Navigation Controls Test	6/11/2010 9:41:42 AM	Bob Swenson	25
	Smoke Test	6/11/2010 9:44:28 AM	Bob Swenson	9
	My Tests	6/11/2010 9:45:16 AM	Ayako Iida	5

Test	Modified	Test List	Check Out By	Steps
AddNewRecord	5/18/2010 4:01 AM	Smoke Test		14
ColumnsReorder	5/18/2010 4:01 AM	Smoke Test		8
Filtering	5/18/2010 4:01 AM	Smoke Test		7
Group	5/18/2010 4:01 AM	Smoke Test		9
GroupPanel	5/18/2010 4:01 AM	Smoke Test		9
Hierarchy	5/18/2010 4:01 AM	Smoke Test		10
Paging	5/18/2010 4:01 AM	Smoke Test		12
Selecting	5/18/2010 4:01 AM	Smoke Test		10
Sorting	5/18/2010 4:01 AM	Smoke Test		11

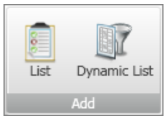
Tests: 9

Test Lists Tab Tools

The Test Lists Tab has tools for **Add**, **Edit** and **Execution** of Test Lists.



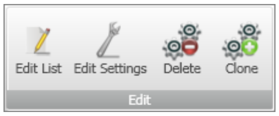
Add



Clicking the **List** button creates a static list of tests that does not change when run. The button displays the Add New Test List dialog that allows you to name the Test List and select tests from the project.

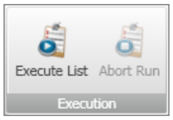
The **Dynamic List** button creates a list of tests at the time of test execution based on rules.

Edit



The buttons in the **Edit** group of the ribbon bar are used to edit existing lists and their settings. The **Edit List** button displays a dialog that allows you to add, remove or reorder tests in the list. If the test is a dynamic test, you can change the rules that shape the test execution. The **Edit Settings** button allows you to apply settings to all the tests in the list at one time. For example, you could change the log file location or enable Silverlight for all tests. You can also remove a Test List permanently using the **Delete** button or make a copy of an existing Test List using the **Clone** button.

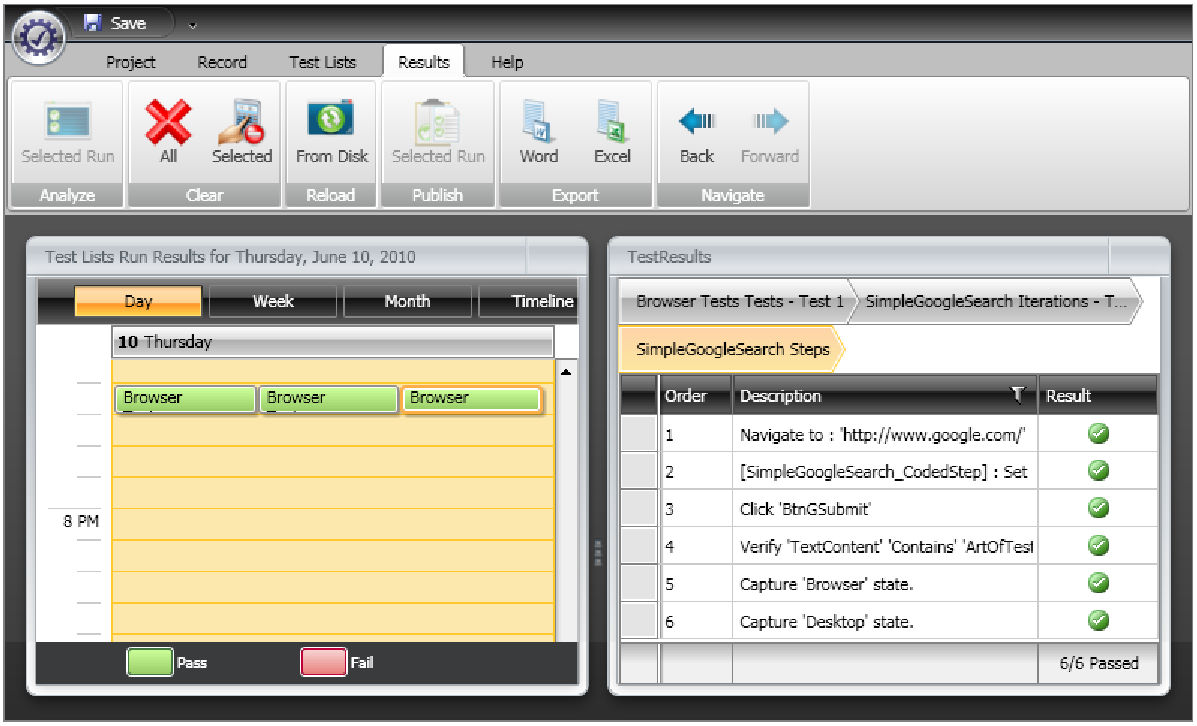
Execution



The **Execute** button executes the currently selected test list. At the conclusion of the execution, WebUI will navigate automatically to the Results Tab. The **Abort** button stops the currently running test list.

Results Tab

The **Results Tab** allows you to overview the outcome of test execution over time, to analyze the results, to navigate between levels of detail and to communicate the results by publishing to TFS or exporting to Word or Excel.



Test Lists Run Results for Thursday, June 10, 2010

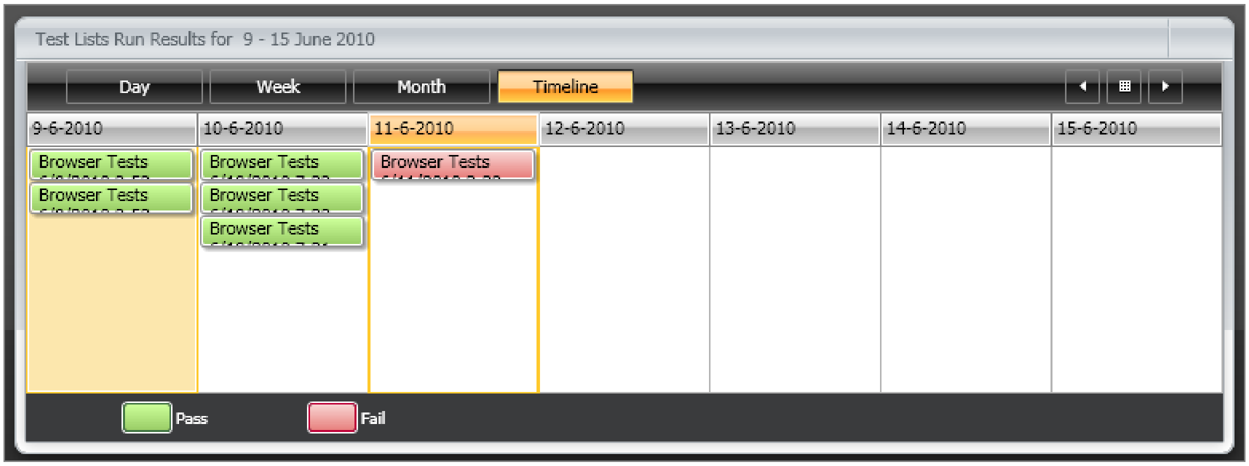
Order	Description	Result
1	Navigate to : 'http://www.google.com/'	✓
2	[SimpleGoogleSearch_CodedStep] : Set	✓
3	Click 'BtnGSubmit'	✓
4	Verify 'TextContent' 'Contains' 'ArtOfTest	✓
5	Capture 'Browser' state.	✓
6	Capture 'Desktop' state.	✓
		6/6 Passed

Results Tab Views

The Results Tab has two main views. The first displays results in a calendar for a high-level look at what tests are passing or failing at a given time. The second allows you to drill down and analyze the results for a test and individual test steps.

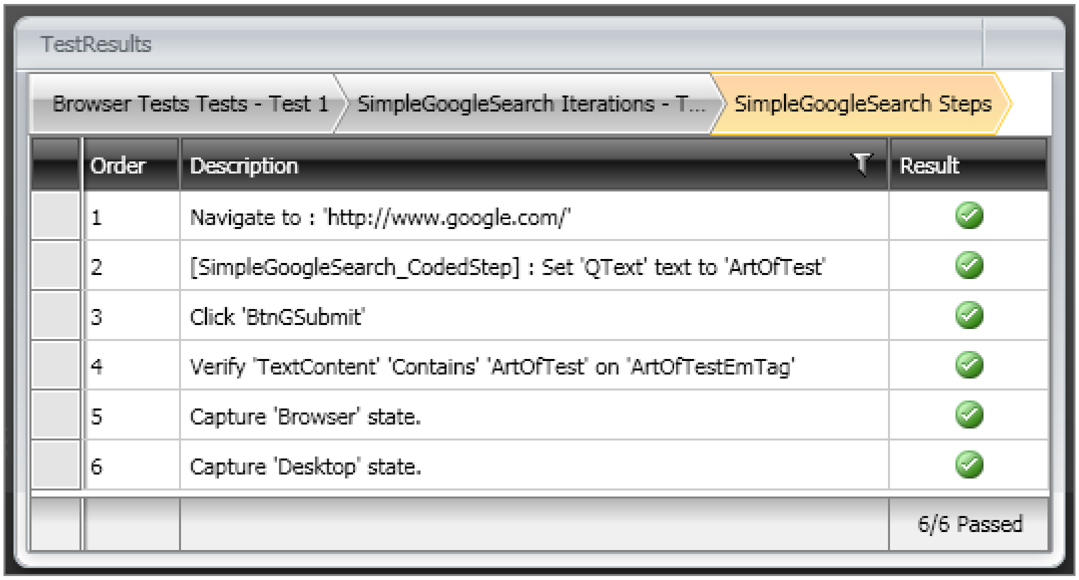
Test Results Calendar

Results of test execution display in the **Test Results Calendar** in a **Timeline** format by default, but also can be viewed in **Day**, **Week** or **Month** views. These views provide an overview of testing activity over a range of time. Double-clicking results shown in the calendar opens that set of results in a detail view used for analysis.



Test Results Analysis

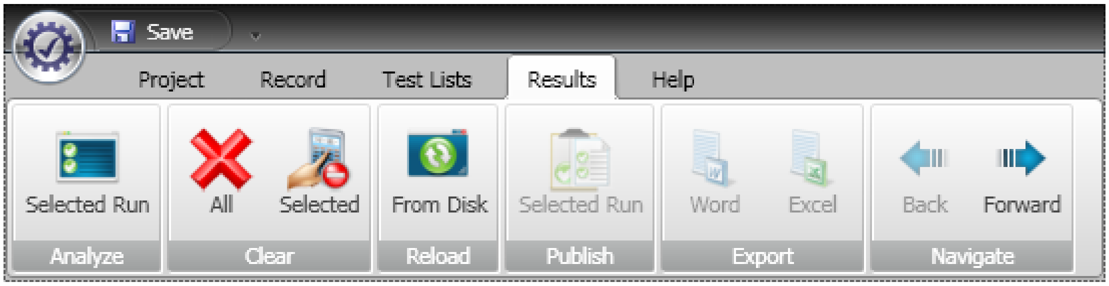
The **TestResults** pane lets you drill down through the test to the results for the individual test steps. In the screenshot below, notice the bread crumb trail at the top of the pane that leads from the test to iterations of the test to the individual test steps. You can click any of the breadcrumbs to navigate between levels.


TestResults
Browser Tests Tests - Test 1
SimpleGoogleSearch Iterations - T...
SimpleGoogleSearch Steps

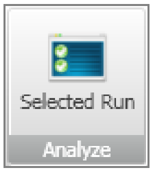
Order	Description	Result
1	Navigate to : 'http://www.google.com/'	✓
2	[SimpleGoogleSearch_CodedStep] : Set 'QText' text to 'ArtOfTest'	✓
3	Click 'BtnGSubmit'	✓
4	Verify 'TextContent' 'Contains' 'ArtOfTest' on 'ArtOfTestEmTag'	✓
5	Capture 'Browser' state.	✓
6	Capture 'Desktop' state.	✓
		6/6 Passed

Results Tab Tools

The **Results Tab** has tools to Analyze a **Selected Run**, Clear the results for a **Selected** test or for **All** tests in the project, Publish the **Selected Run**, Export to **Word** or **Excel** and to Navigate **Back** or **Forward** over your previous path in the Test Results pane.



Analyze



The **Selected Run** button displays the Test Results Analysis pane for the selected test list. You can also get the same effect by double-clicking a test list.

Clear

Use the **Clear All** button to remove all test results permanently. Use the **Selected** button to remove the currently selected test results, or you can use the Delete key.

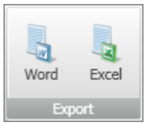
Reload

The **From Disk** button displays reloads test results from your hard drive or network.

Publish

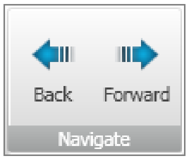
If you are connected to TFS, use the **Publish Selected Run** option to send the results to TFS.

Export



The **Export** buttons are enabled when you have test results displayed for analysis. Clicking either the **Word** or **Excel** brings up a "Save As" dialog.

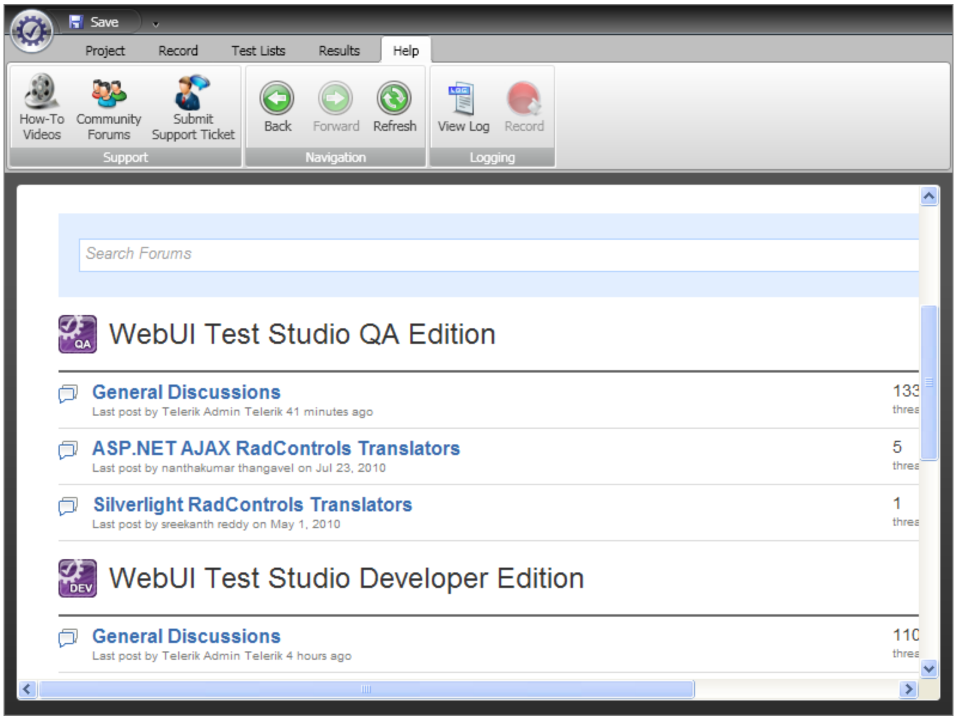
Navigate



The **Navigate** buttons simply move you **Forward** and **Back** through all your activity in the test results, with similar behavior to a web browser.

Help Tab

The **Help Tab** gives you direct access to **How To Videos**, **Community Forums** and **Support Tickets** without having to leave WebUI. The **Navigation** buttons work in the same manner as the buttons for external browsers. The **View Log** button displays the history of the WebUI itself and can be used when communicating with Telerik support engineers.



WebUI Test Studio QA Edition

General Discussions 133 threa
Last post by Telerik Admin Telerik 41 minutes ago

ASP.NET AJAX RadControls Translators 5 threa
Last post by nanthakumar thangavel on Jul 23, 2010

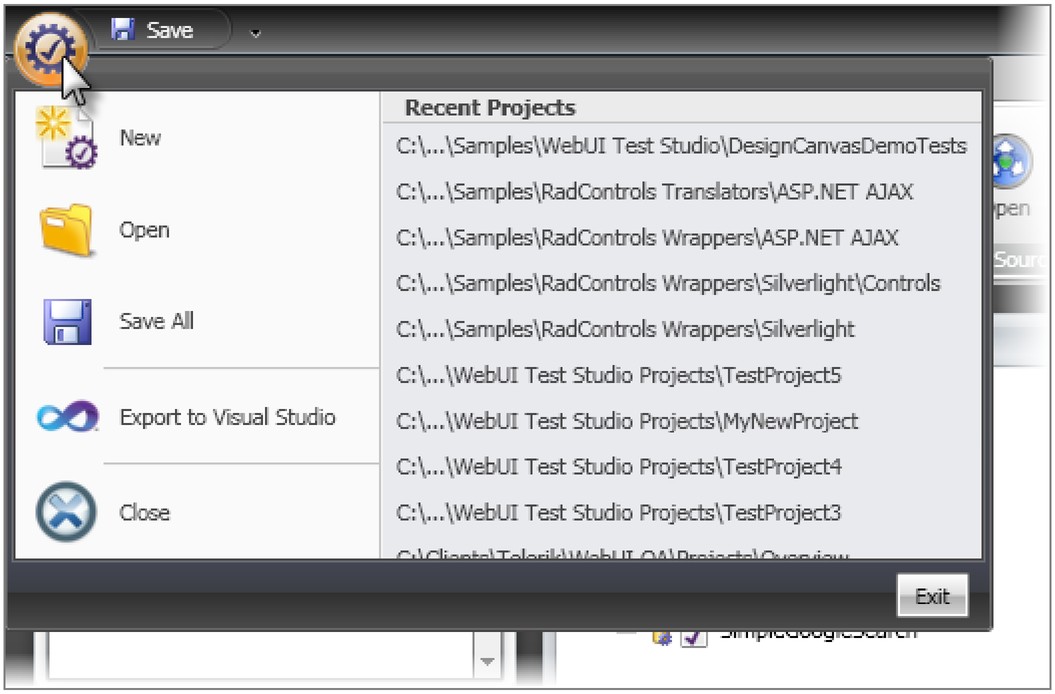
Silverlight RadControls Translators 1 threa
Last post by sreekanth reddy on May 1, 2010

WebUI Test Studio Developer Edition

General Discussions 110 threa
Last post by Telerik Admin Telerik 4 hours ago

Ribbon Bar Menu

The ribbon bar menu works at the project level and allows you to create a brand new project, open or close an existing project, save everything in the project or export the entire project to Visual Studio 2008 or 2010.


Save
New
Open
Save All
Export to Visual Studio
Close
Recent Projects
C:\...\Samples\WebUI Test Studio\DesignCanvasDemoTests
C:\...\Samples\RadControls Translators\ASP.NET AJAX
C:\...\Samples\RadControls Wrappers\ASP.NET AJAX
C:\...\Samples\RadControls Wrappers\Silverlight\Controls
C:\...\Samples\RadControls Wrappers\Silverlight
C:\...\WebUI Test Studio Projects\TestProject5
C:\...\WebUI Test Studio Projects\MyNewProject
C:\...\WebUI Test Studio Projects\TestProject4
C:\...\WebUI Test Studio Projects\TestProject3
Exit
pen
Sourc

3.4 Summary

In this chapter you took a guided tour of the product. You became familiar with major areas of the product including Projects, Record, Tests lists and Results.

Part IV

Getting Started

4 Getting Started

4.1 Objectives

In this chapter you will use WebUI to create a simple test, execute the test and view the results. Along the way you'll see the major areas of WebUI Test Studio QA Edition.

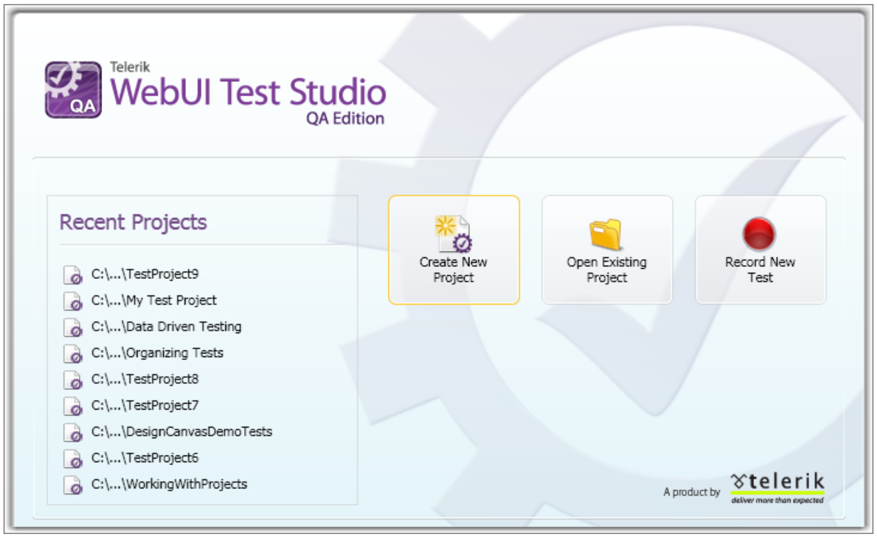
4.2 Walk Through

This tutorial will walk you through creating a simple test, executing the test and viewing the results. Once you have a working test and have briefly touched the major functionality, we can dive deeper into the rich functionality of WebUI.

Find the materials for this Topic at...

\Projects\GettingStarted

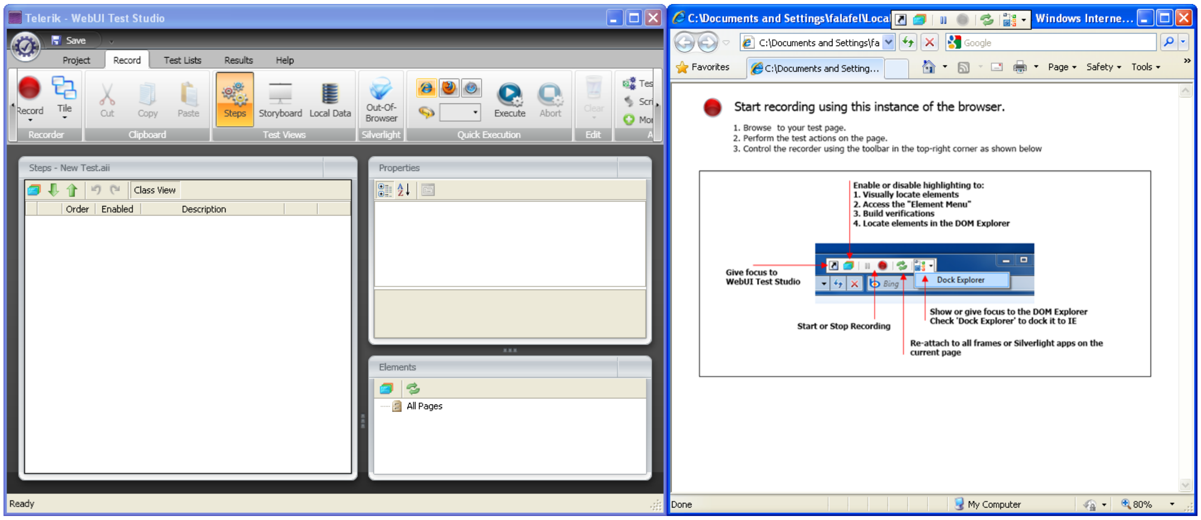
1. Start WebUI. This will display the Welcome Screen.



Recent Projects

C:\...\TestProject9
C:\...\My Test Project
C:\...\Data Driven Testing
C:\...\Organizing Tests
C:\...\TestProject8
C:\...\TestProject7
C:\...\DesignCanvasDemoTests
C:\...\TestProject6
C:\...\WorkingWithProjects

2. Click the **Record New Test** button. This will create a new test and project automatically. WebUI will navigate to the Record Tab and the Recording Surface will display alongside WebUI.

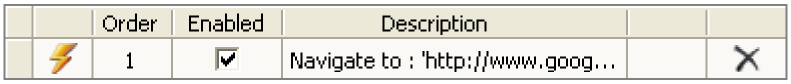


Start recording using this instance of the browser.

1. Browse to your test page.
2. Perform the test actions on the page.
3. Control the recorder using the toolbar in the top-right corner as shown below

3. In the Recording Surface, type the URL "www.google.com" into the address line and press the **ENTER** key.

4. Notice that a new "Navigate" step has been created in the Test Pane.

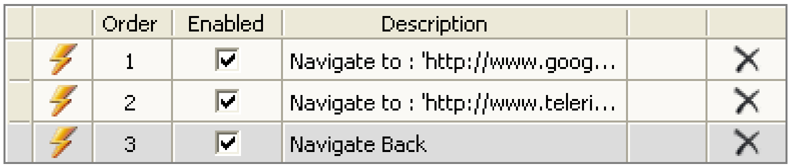


		Order	Enabled	Description		
	⚡	1	☑	Navigate to : 'http://www.goog...		✕

5. In the Recording Surface, type the URL "www.telerik.com" into the address line and press the **ENTER** key.

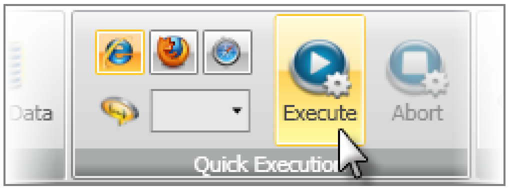
6. Click the Recording Surface **Back** button.

7. The Test Pane should now look like the figure below.



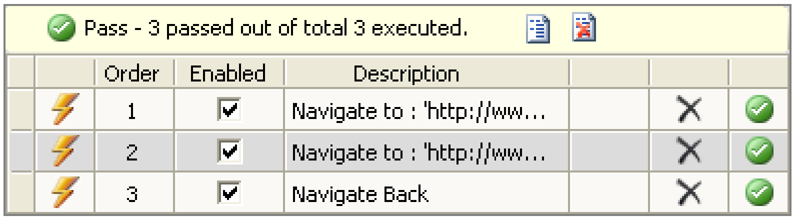
		Order	Enabled	Description		
	⚡	1	☑	Navigate to : 'http://www.goog...		✕
	⚡	2	☑	Navigate to : 'http://www.teleri...		✕
	⚡	3	☑	Navigate Back		✕

8. Click the **Execute** button.



9. When prompted to save the test, leave the default values and click the **Save** button.

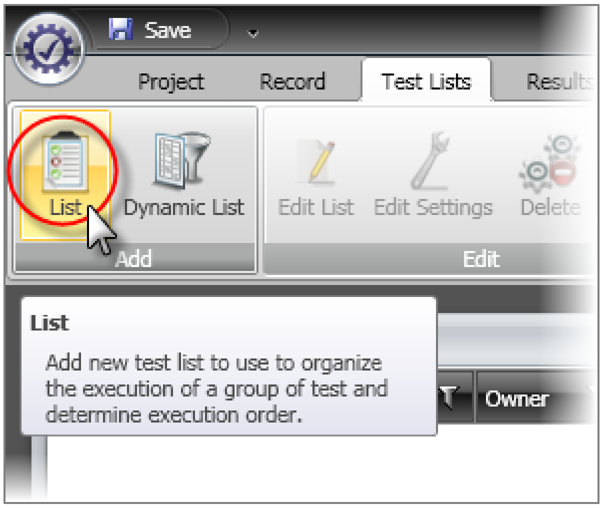
10. The test will run, first displaying a browser, navigating to Google, then to the Telerik site and back again. The test steps should all display green check marks as shown in the screenshot below.



Pass - 3 passed out of total 3 executed.

Order	Enabled	Description		
1	☑	Navigate to : 'http://ww...	✕	✓
2	☑	Navigate to : 'http://ww...	✕	✓
3	☑	Navigate Back	✕	✓

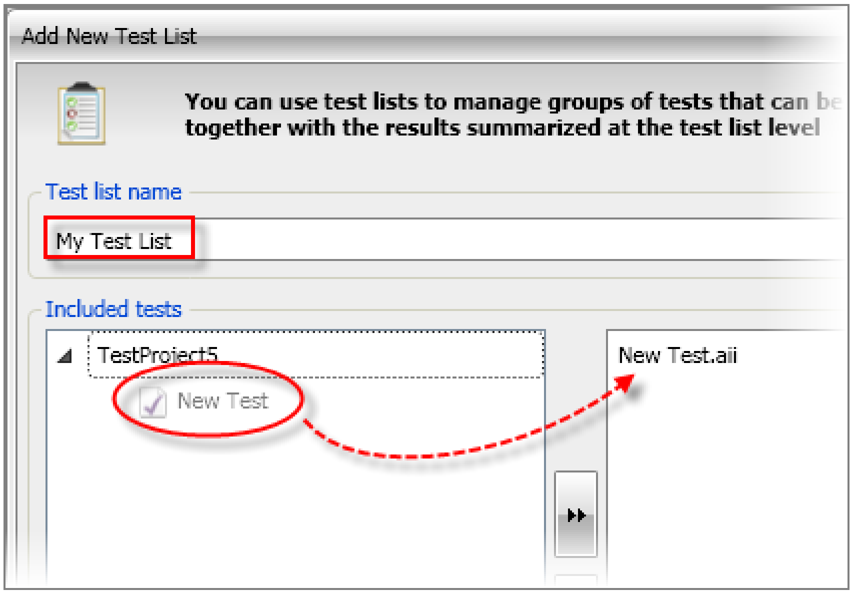
11. Navigate to the Test Lists Tab and click the **List** button. This will display the "Add New Test List" dialog.



List

Add new test list to use to organize the execution of a group of test and determine execution order.

12. In the Add New Test List dialog, enter "My Test List" as the **Test List Name**. Double-click the test titled "New Test" in the **Included Tests** list. This will move "New Test" into a second list on the right.



Add New Test List

You can use test lists to manage groups of tests that can be together with the results summarized at the test list level

Test list name

My Test List

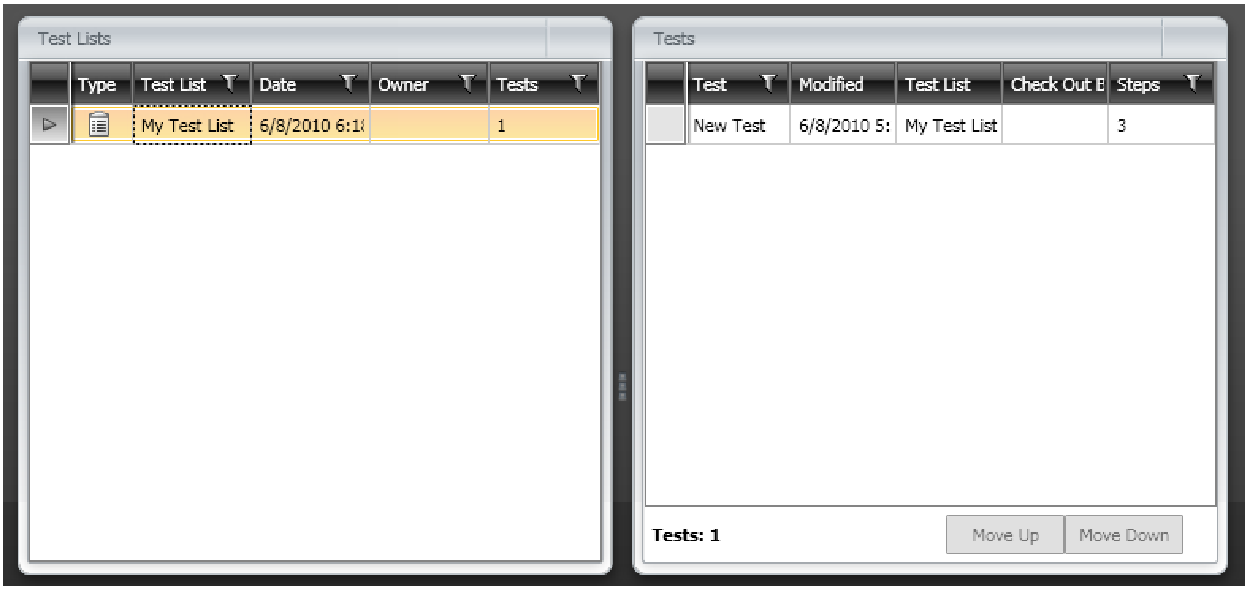
Included tests

- TestProject5
  - New Test

New Test.aii

13. Click the **OK** button.

14. Notice that "My Test List" has been added to the **Test Lists** Pane and "New Test" displays in the detail **Tests** Pane on the right.



Test Lists

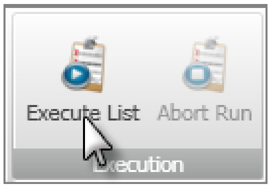
	Type	Test List	Date	Owner	Tests
▷		My Test List	6/8/2010 6:1		1

Tests

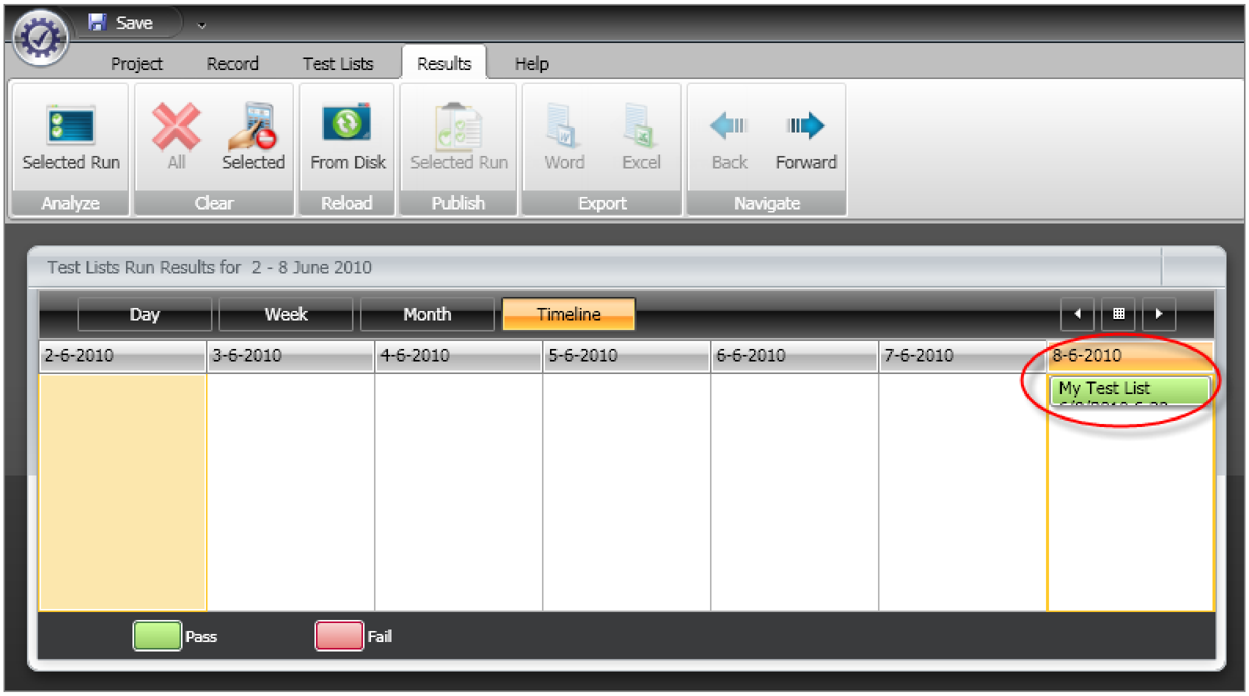
	Test	Modified	Test List	Check Out B	Steps
	New Test	6/8/2010 5:	My Test List		3

Tests: 1

15. Click the **Execute List** button. The test will execute and WebUI will automatically navigate to the Results Tab.



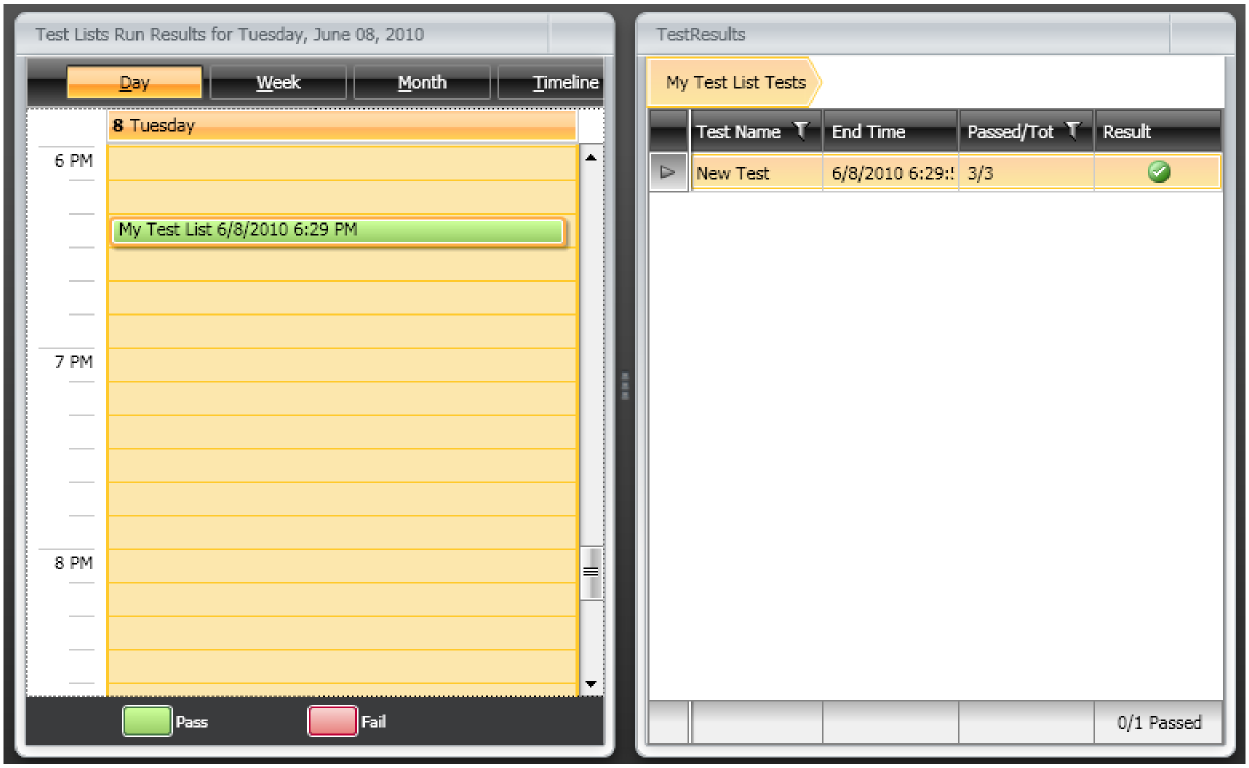
16. Notice that the Results Tab displays "My Test List" in the Results Calendar.



Test Lists Run Results for 2 - 8 June 2010

2-6-2010	3-6-2010	4-6-2010	5-6-2010	6-6-2010	7-6-2010	8-6-2010
						My Test List

17. Double-click "My Test List" in the calendar. This will display the calendar alongside the TestResults pane.



Test Lists Run Results for Tuesday, June 08, 2010

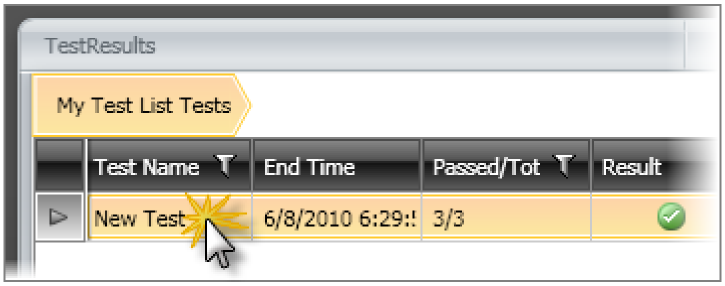
8 Tuesday

My Test List 6/8/2010 6:29 PM

Test Name	End Time	Passed/Tot	Result
New Test	6/8/2010 6:29:!	3/3	✓

0/1 Passed

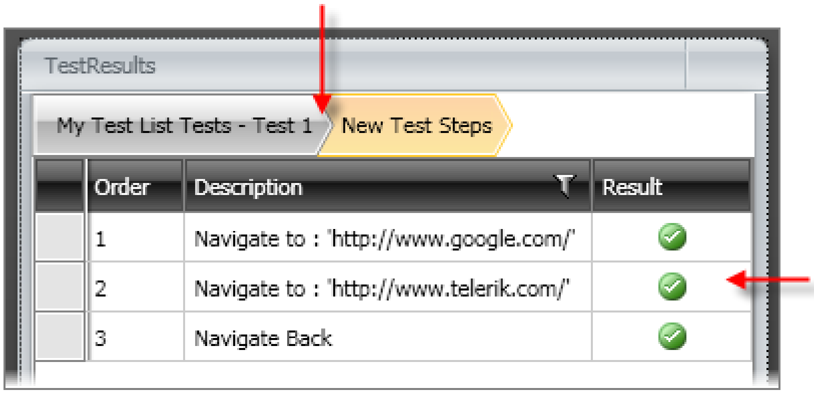
18. Double-click the "New Test" entry of the TestResults Pane.



	Test Name	End Time	Passed/Tot	Result
▷	New Test	6/8/2010 6:29:!	3/3	✓

19. Notice the "breadcrumb trail" that allows you to navigate back and forth from the test, to the individual test steps and back again.

Breadcrumb Trail



	Order	Description	Result
	1	Navigate to : 'http://www.google.com/'	✓
	2	Navigate to : 'http://www.telerik.com/'	✓
	3	Navigate Back	✓

Test Steps

4.3 Summary

In this chapter you used WebUI to create a simple test, execute the test and view the results. Along the way you saw the major areas of WebUI Test Studio QA Edition.

Part V

Working with Projects

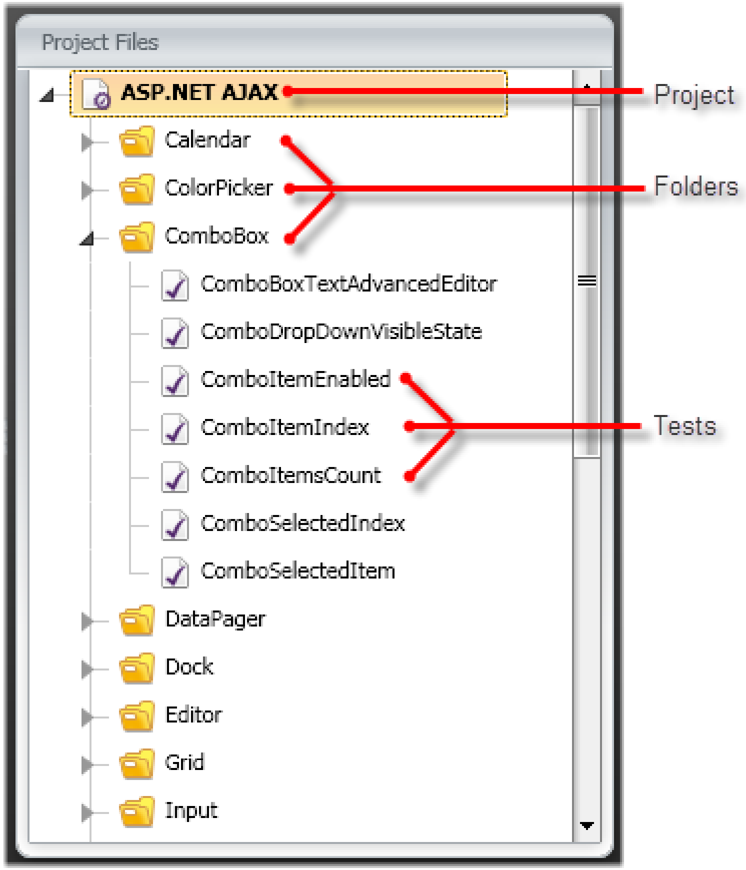
5 Working with Projects

5.1 Objectives

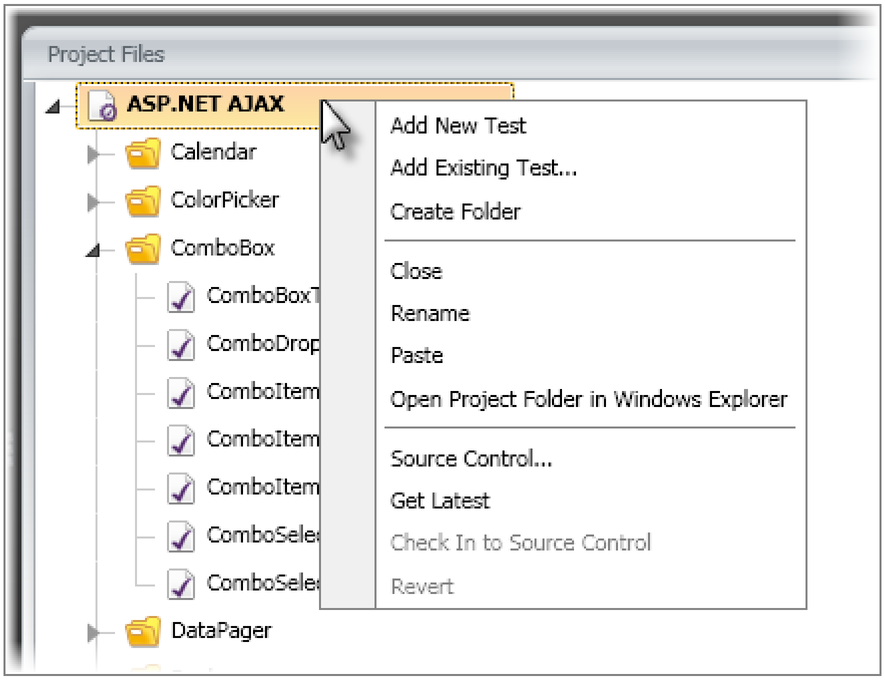
In this chapter you will learn how tests, folders and other files are organized into projects. You will see how the context menu interacts with the Project Files tree view to build the project structure and to work with individual tests. You will create a simple project and open a test for recording.

5.2 Project Files

The **Project Files** tree view holds the current WebUI project, folders and individual tests. The screenshot below shows a project called "ASP.NET AJAX", with folders for "Calendar", "ColorPicker", "ComboBox", etc. underneath. Under the ComboBox folder are a number of tests.

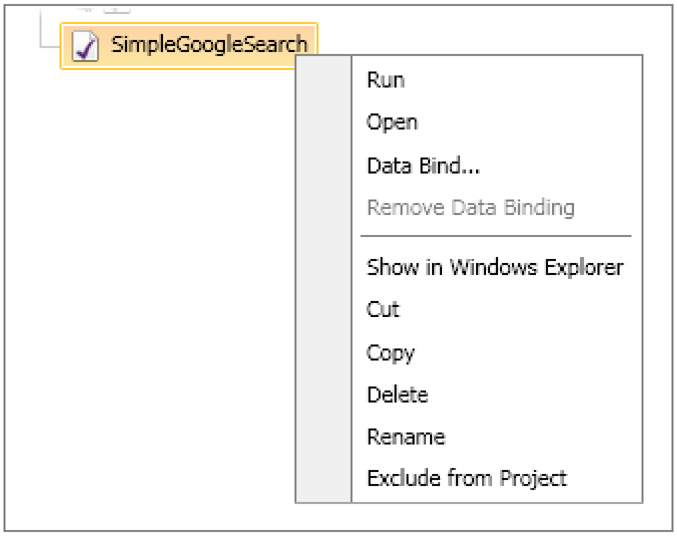


Right-clicking any node brings up a context menu. The project level context menu allows you to add tests and create folders. As the test suite grows to cover more of an application, you can use folders to organize tests by functional group or any other scheme that suits your requirements. Folders can be nested. The second group of context menu options allows you to **Close** the project, taking you back to the Welcome Screen. You can **Rename** the project which will cause the rename of the project on disk. The **Paste** option adds any tests that you have copied to your clipboard. This is a handy feature when you want to use an existing test as a starting point. The last set of options allows you to check projects and tests in and out of TFS.

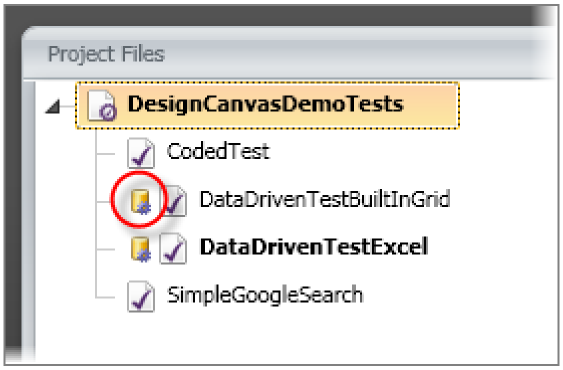


The folder level context menu is similar but also allows you to **Delete** a folder or **Exclude from Project** which eliminates the folder from the project but still leaves the folder on disk. Use the **Create Folder** option to nest folders.

The context menu for individual tests includes option to **Open** or **Run** a test. If you have data sources defined in your test you can click the option to **Data Bind...** or **Remove Data Binding**. Data binding is covered in the "Data Driven Tests" chapter.

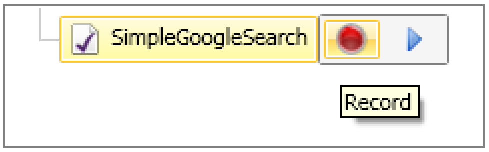


The Project Files Pane uses visual cues to indicate the current status of projects, folders and tests. Tests that are data bound display the database icon as shown in the screenshot below. Also notice that "DesignCanvasDemoTests" and "DataDrivenTestExcel" are shown in bold indicating that the test and project contain changes that haven't been saved.



Tip!

If you hover your mouse over any test, Record and Play buttons will appear so that you can append steps to your test or execute the test directly from the Project Files Pane.



Tip!

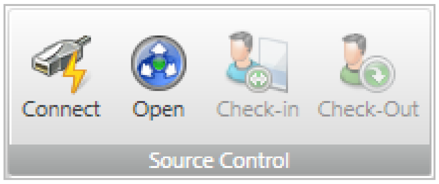
The projects in WebUI Test Studio QA Edition are the same as those used in WebUI Test Studio Developer Edition. You can open a WebUI Test Studio Developer Edition project directly in WebUI Test Studio QA Edition. You can also save your WebUI Test Studio QA Edition projects and send them to developers that are using WebUI Test Studio Developer Edition.

5.3 Team Foundation Server Integration

"Version Control" allows multiple people to access the same documents. Documents are "checked out" from a source control database "repository" that tracks changes made to a document and handles any collisions with other users that may occur when the documents are checked back in to the repository.

Team Foundation Server (TFS) is a prominent source control system offered by Microsoft that integrates with WebUI to source control your test projects, tests and other files. Developers using Visual Studio with TFS will also be able to check out your test projects.

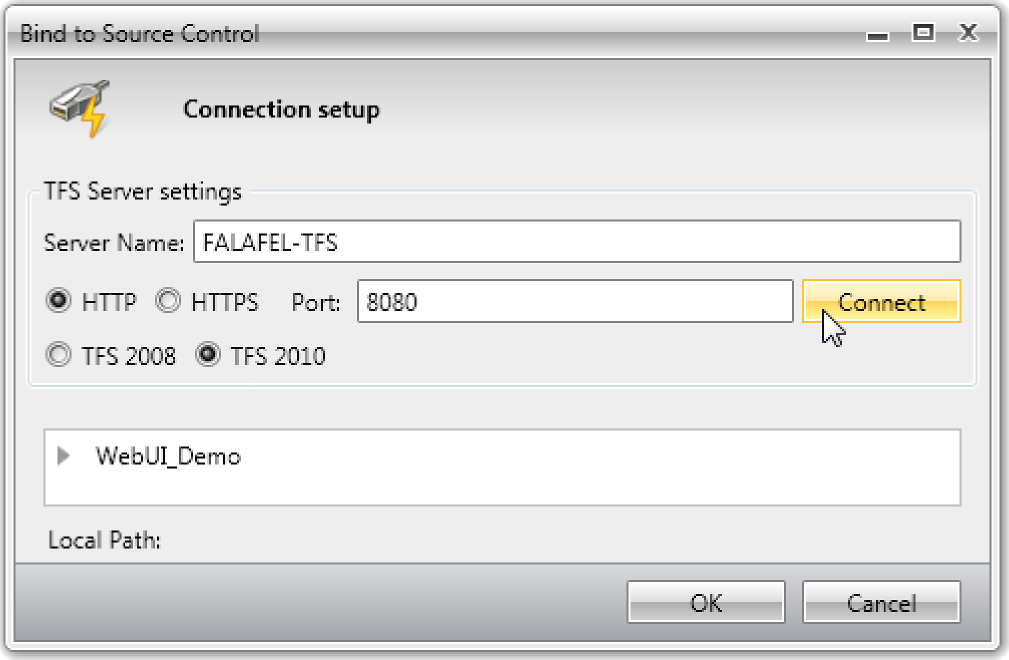
You can **Connect** to TFS and add an existing project to source control, or **Open** a project that's already in the source control repository. Once you are connected to TFS you can check in and check out your projects, folders and tests.



Connecting to Source Control

When you click the **Connect** or **Open** button, the **Bind to Source Control** dialog appears. To connect to TFS you will need some information from your TFS administrator, namely:

- The Server Name.
- The Protocol and port. By default, the protocol is HTTP over port 8080.
- The version of the TFS server that you're using.
- User name and password.



Bind to Source Control

Connection setup

TFS Server settings

Server Name: FALAFEL-TFS

● HTTP ○ HTTPS Port: 8080

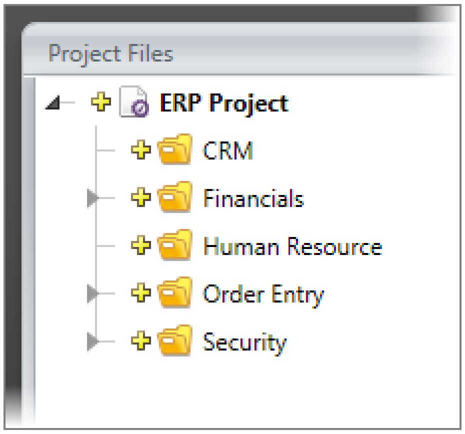
○ TFS 2008 ● TFS 2010

WebUI_Demo

Local Path:

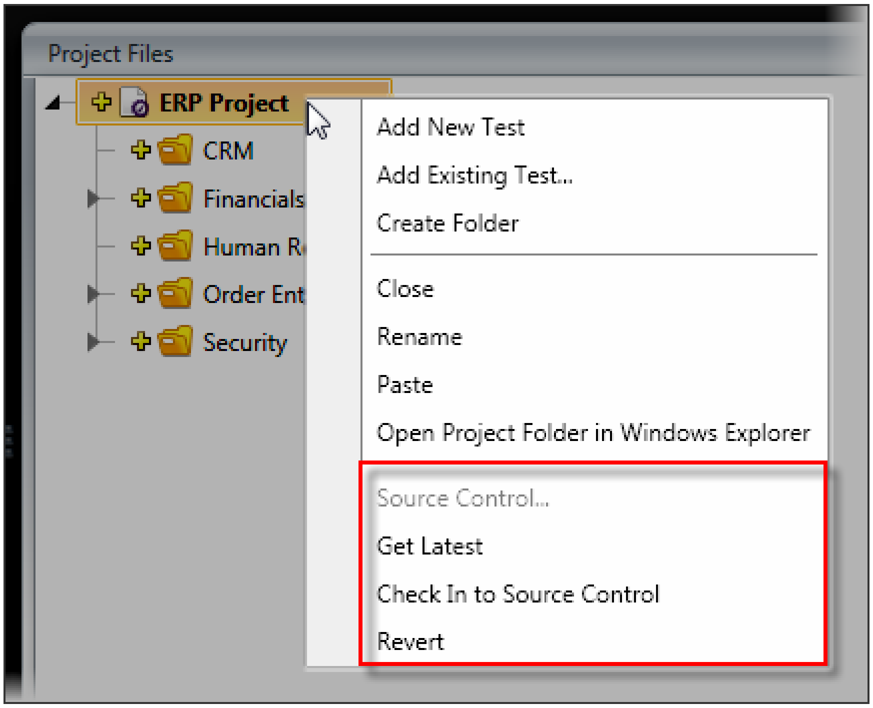
Once you've filled in the information, click the **Connect** button. You will have to enter a name and password to an authentication dialog and then the projects that you have access to will display in the tree view at the bottom of the **Bind to Source Control** dialog. Select the project you want to bind to and click the **OK** button.

If you already have a test project open in WebUI, the project will now have "plus" icons indicating that the project has been bound to source control.



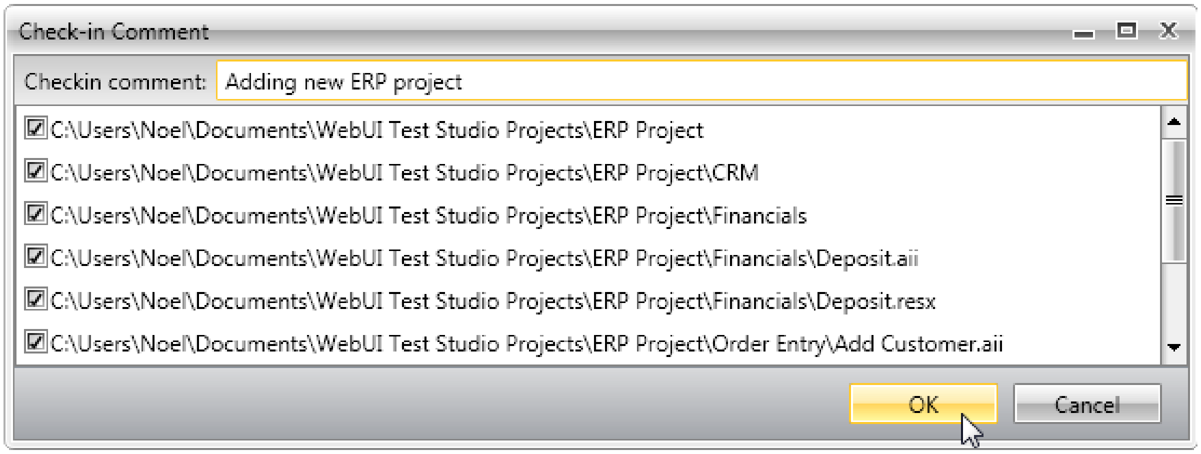
Source Control Options

The Project Files context menu allows you to **Get Latest** from source control, **Check In** the selected item to source control or **Revert** the selected item to whatever is stored in source control.



Checking In

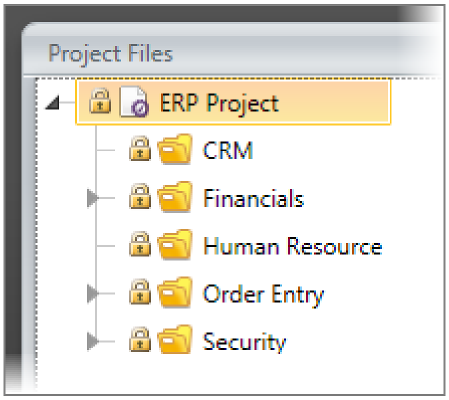
If you Check In your project, the **Check-in Comment** dialog lists the files that will be committed to source control. Enter a "Checkin comment" so that others on the team can easily see what has changed, un-select any files that should not be sent to source control and click the **OK** button to finish.



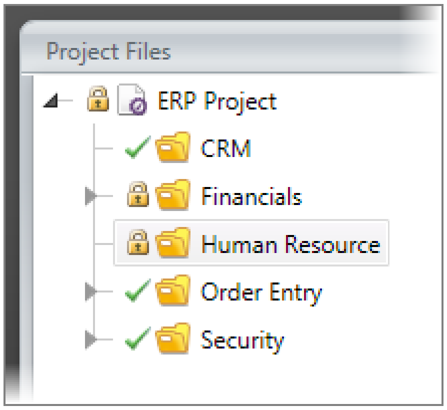
Checkin comment: Adding new ERP project

☑ C:\Users\Noel\Documents\WebUI Test Studio Projects\ERP Project
☑ C:\Users\Noel\Documents\WebUI Test Studio Projects\ERP Project\CRM
☑ C:\Users\Noel\Documents\WebUI Test Studio Projects\ERP Project\Financials
☑ C:\Users\Noel\Documents\WebUI Test Studio Projects\ERP Project\Financials\Deposit.aii
☑ C:\Users\Noel\Documents\WebUI Test Studio Projects\ERP Project\Financials\Deposit.resx
☑ C:\Users\Noel\Documents\WebUI Test Studio Projects\ERP Project\Order Entry\Add Customer.aii

Now the icons for the project show that these files are checked in.

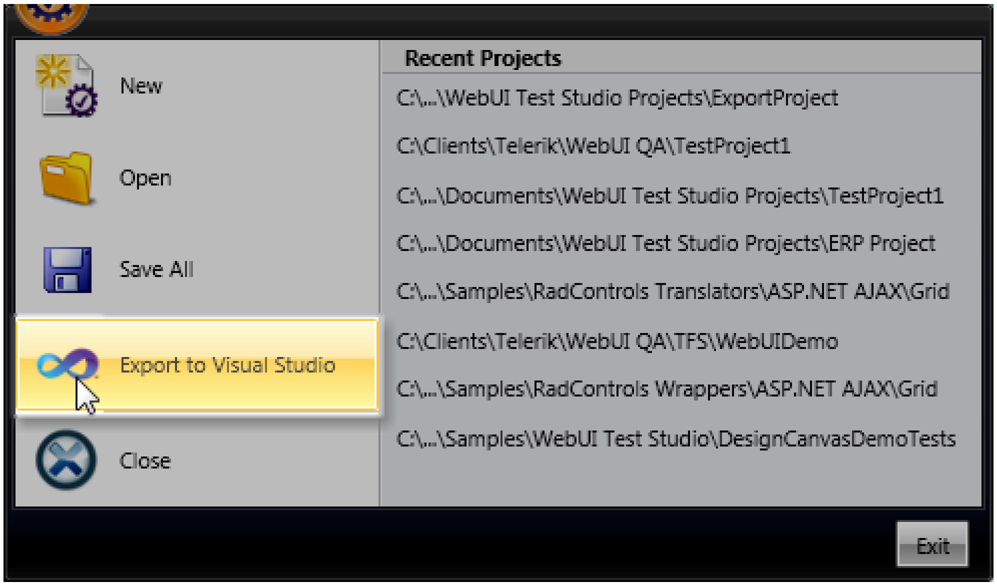


By selecting a file and clicking the **Check-out** button, or simply by changing a file, the file is "checked out" for editing and will show a check mark icon. The screenshot below shows the "CRM", "Order Entry" and "Security" folders have been checked out.



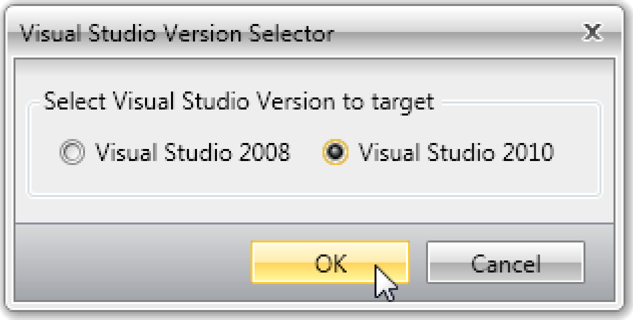
5.4 Exporting Projects to Visual Studio

Even without TFS you can export your test project in a form that can be consumed by developers using Visual Studio. The **Export to Visual Studio** Ribbon Bar menu item will create a Visual Studio 2008 or 2010 project.



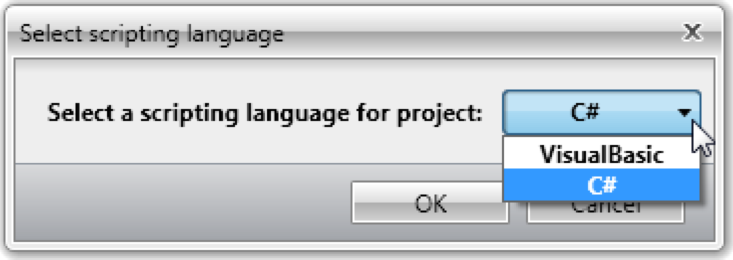
Recent Projects
C:\...\WebUI Test Studio Projects\ExportProject
C:\Clients\Telerik\WebUI QA\TestProject1
C:\...\Documents\WebUI Test Studio Projects\TestProject1
C:\...\Documents\WebUI Test Studio Projects\ERP Project
C:\...\Samples\RadControls Translators\ASP.NET AJAX\Grid
C:\Clients\Telerik\WebUI QA\TFS\WebUIDemo
C:\...\Samples\RadControls Wrappers\ASP.NET AJAX\Grid
C:\...\Samples\WebUI Test Studio\DesignCanvasDemoTests

After clicking the Export to Visual Studio option, the **Visual Studio Version Selector** dialog appears.



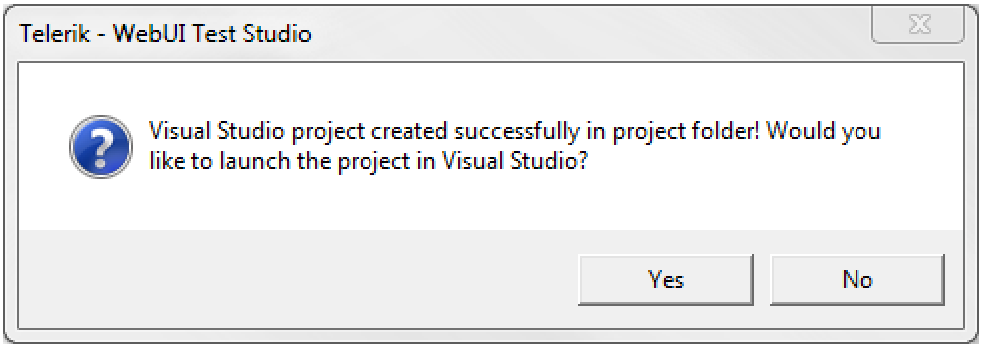
Select Visual Studio Version to target
○ Visual Studio 2008 ● Visual Studio 2010

After selecting a Visual Studio version and clicking the **OK** button, a second dialog will allow you to select the scripting language used by the project.



Select a scripting language for project: C#

The Visual Studio project is created and a final dialog allows you to open the project directly in Visual Studio.



Telerik - WebUI Test Studio

Visual Studio project created successfully in project folder! Would you like to launch the project in Visual Studio?

5.5 Walk Through

The following walk through shows you how to create a WebUI project, folders and tests.

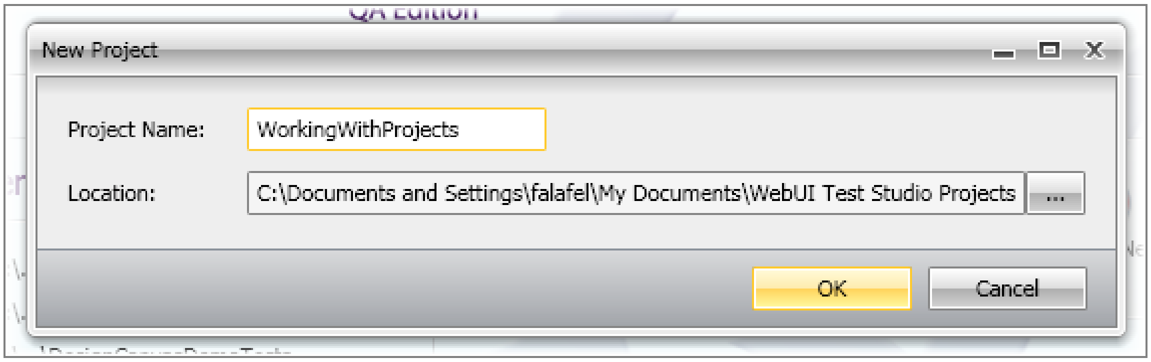
Find the materials for this Topic at...

\Projects\WorkingWithProjects

1. Open WebUI. You should see the Welcome Screen.

2. Click the **Create New Project** button. This will display the **New Project** dialog.

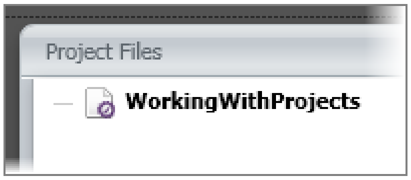
3. Enter "WorkingWithProjects" to the **Project Name** text box. leave the **Location** text box with the default value. Click the **OK** button.



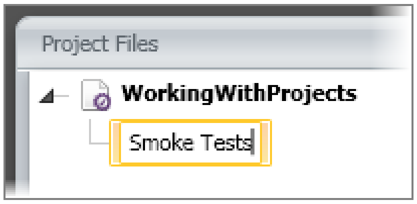
Project Name: WorkingWithProjects

Location: C:\Documents and Settings\falafel\My Documents\WebUI Test Studio Projects

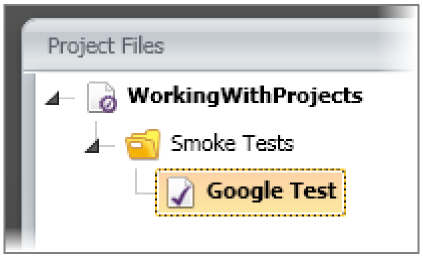
4. The **Data Sources**, **Properties** and **Project Files** panes should appear. Notice that the Project Files pane has a single node "WorkingWithProjects" that represents the project.



5. Right-click the "WorkingWithProjects" node. Select **Create Folder** from the context menu. Enter "Smoke Tests" as the name of the folder.



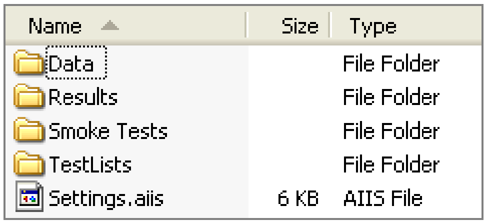
6. Right-click the "Smoke Tests" folder and select **Add New Test** from the context menu. Enter "Google Test" as the name of the new test.



7. Right-click the "Google Test" node and select **Open** from the context menu. This action will navigate to the Record Tab where "Google Test" is ready for recording.

8. Click back to the Project Tab.

9. Right-click the "WorkingWithProjects" node and select **Open Project with Windows Explorer**. This will display the Windows Explorer with the project folders listed. The screenshot below shows a folder for the "Smoke Tests" project that contains our test files and folders for Data, Results and TestLists.



Name	Size	Type
Data		File Folder
Results		File Folder
Smoke Tests		File Folder
TestLists		File Folder
Settings.aiis	6 KB	AIIS File

10. Navigate back to WebUI, right-click the "WorkingWithProjects" node and select **Close** from the context menu. Click **OK** for prompts to save the test and project. This will return you back to the Welcome Screen.

5.6 Summary

In this chapter you learned how tests, folders and other files are organized into projects. You saw how the context menu interacts with the Project Files tree view to build the project structure and to work with individual tests. You created a simple project and opened a test for recording.

Part VI

Recording Tests

6 Recording Tests

6.1 Objectives

In this chapter you will take a detailed look at how tests are recorded and the available tools used to handle specific web testing tasks. You will use the Recording Surface to save actions taken in the browser and to interact with individual web page elements. You will see how these recorded steps are managed using the Steps Pane and how to add steps manually. You will see how "Translators" are used to peer inside controls to glean information about the controls internals. You will also use WebUI to handle specialized web testing situations such as drag-and-drop and working with pop-up dialogs. Finally, you will learn how to "audition" tests in multiple browsers.

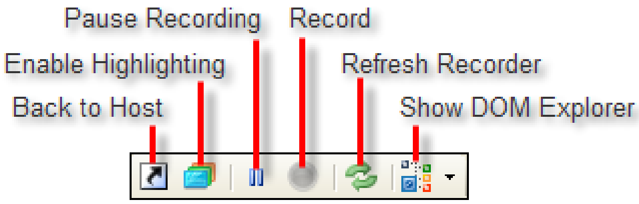
6.2 Building Tests

Recording Surface

Much of the time you spend with WebUI 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.

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 WebUI.

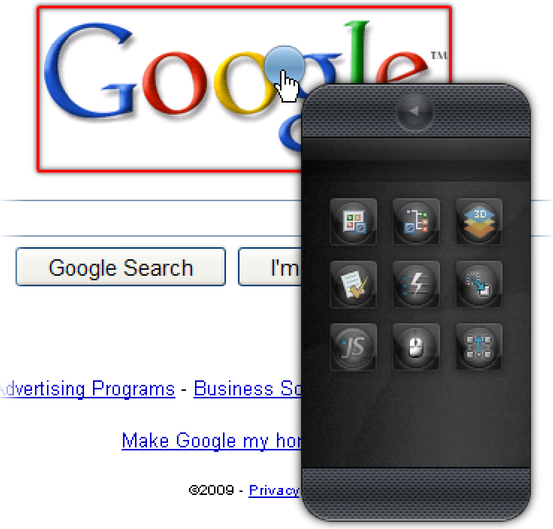


The parts of the toolbar are:

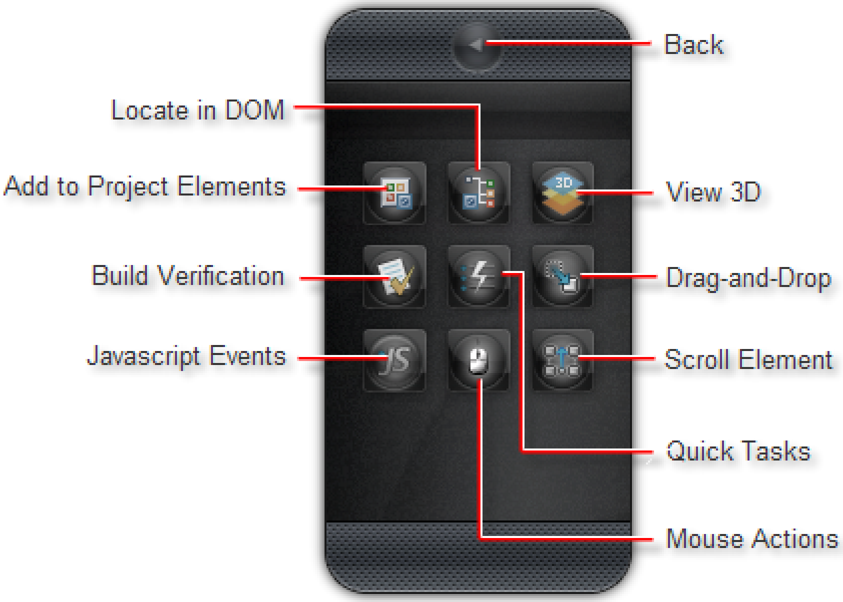
- **Back to Host**: Activates WebUI.
- **Enable Highlighting**: When pressed, elements in the page are highlighted as the mouse passes over. Highlighting allows you to add elements to the Elements Explorer and to display the Elements Menu for additional actions.
- **Pause** Recording: Use this button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

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 is open to user applications that can incorporated into the menu. Clicking the Nub displays the Elements Menu.

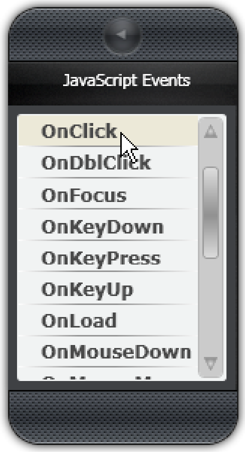


The Elements Menu provides quick access to relevant functions right in the page you are testing.

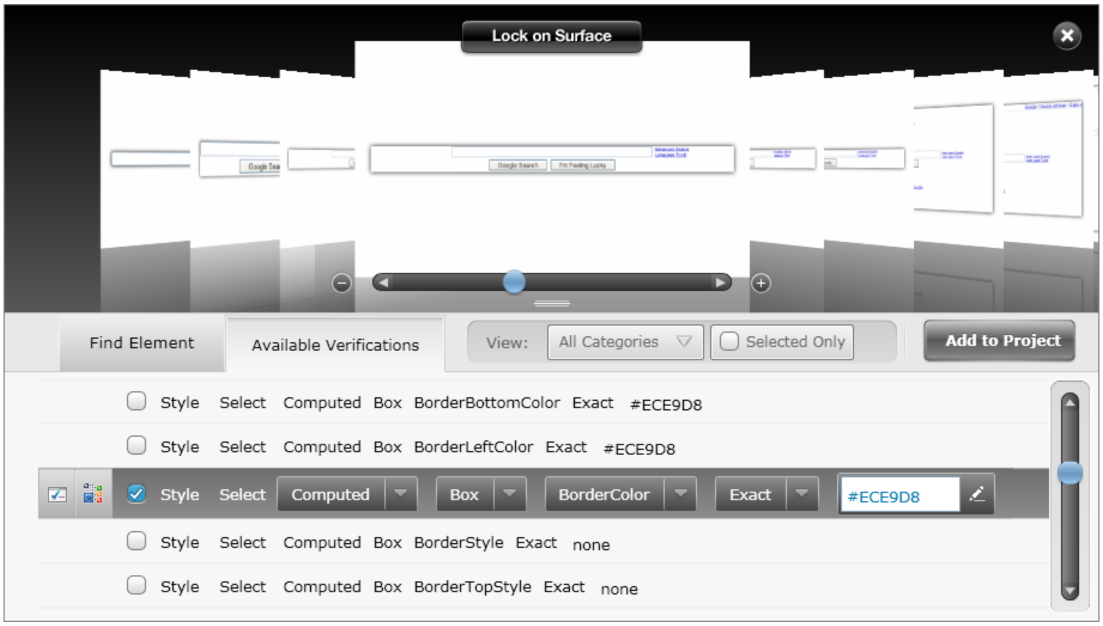


- **Locate in DOM** navigates to DOM Explorer and selects the corresponding element.
- **Add to Project Element** adds the highlighted element to the Elements Explorer.
- **Build Verification** navigates to the Sentence Verification Builder where you can interactively build verification criteria based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



□ Style Select Computed Box BorderBottomColor Exact #ECE9D8

□ Style Select Computed Box BorderLeftColor Exact #ECE9D8

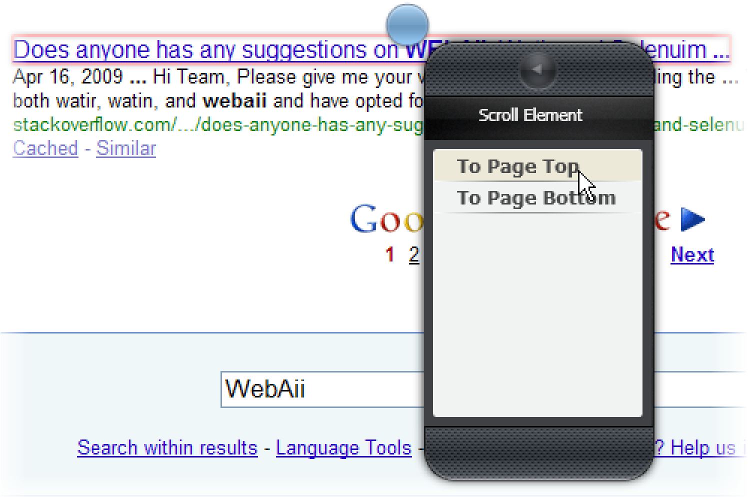
☑ Style Select Computed Box BorderColor Exact #ECE9D8

□ Style Select Computed Box BorderStyle Exact none

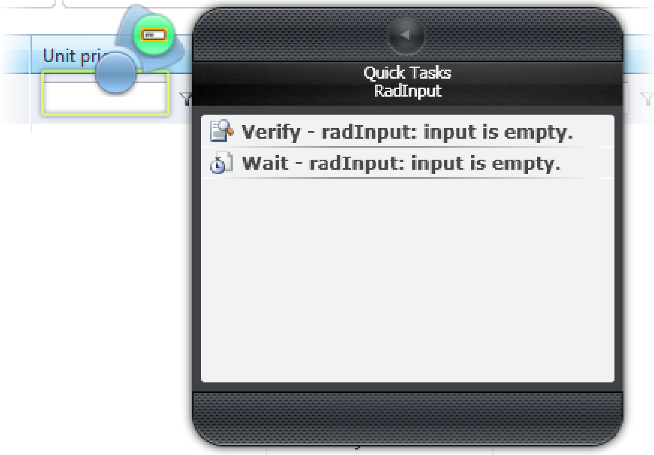
□ Style Select Computed Box BorderTopStyle Exact none

- **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" Topic for more information.

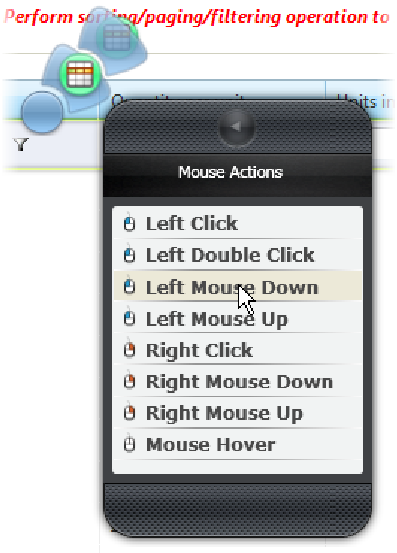
- **Scroll Element** scrolls the highlighted element to the top or the bottom of the page.


Does anyone has any suggestions on WE... ...lenuim ...
Apr 16, 2009 ... Hi Team, Please give me your v... ...ing the ...
both watir, watin, and webaii and have opted fo...
stackoverflow.com/.../does-anyone-has-any-sug... ...and-selenu
Cached - Similar
Scroll Element
To Page Top
To Page Bottom
1 2
Next
WebAii
Search within results - Language Tools - ... ? Help us

- **Quick Tasks** presents a context sensitive list of tasks that can be performed against the highlighted element. This is a very powerful feature that automatically presents the options you are most likely to need. The screenshot below shows Verify and a Wait tasks.

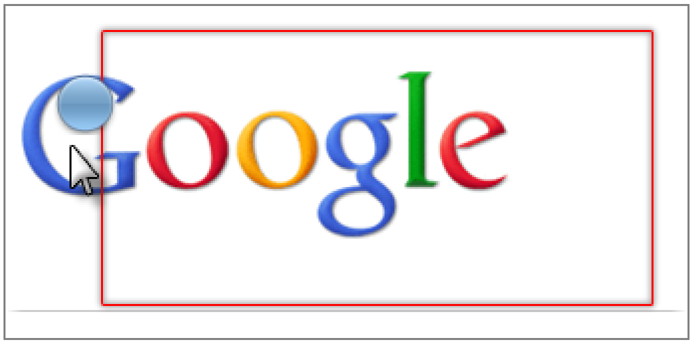

Unit pri...
Quick Tasks
RadInput
Verify - radInput: input is empty.
Wait - radInput: input is empty.

- **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.

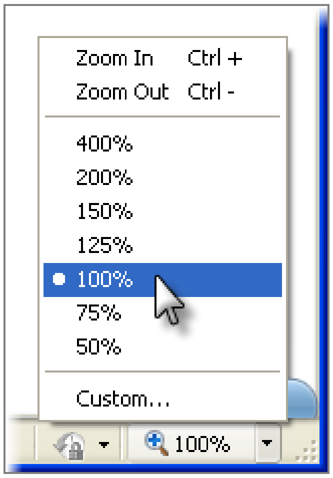


Browser Resolution

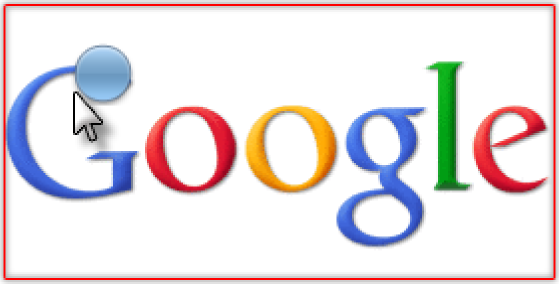
If the highlight appears offset from the element it should be surrounding, make sure that your browser zoom level is set to 100%. In the screenshot below, the zoom level is 90%.



The zoom level can be found in the lower right hand corner of the Recording Surface. Select 100% from the drop down list.

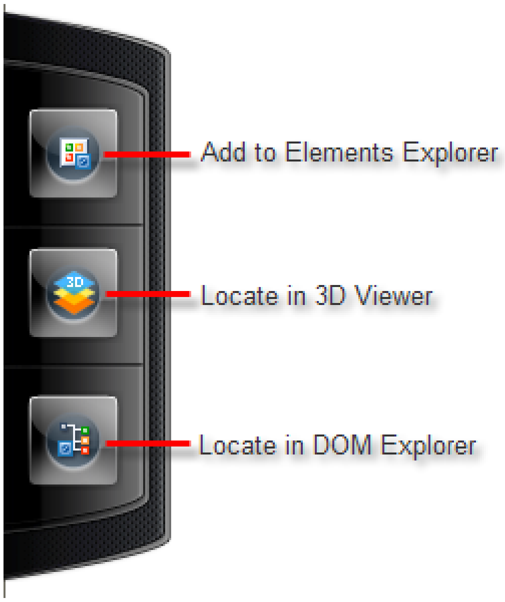


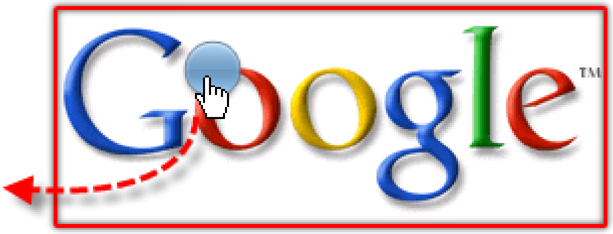
Now the highlighting will conform to the element exactly.



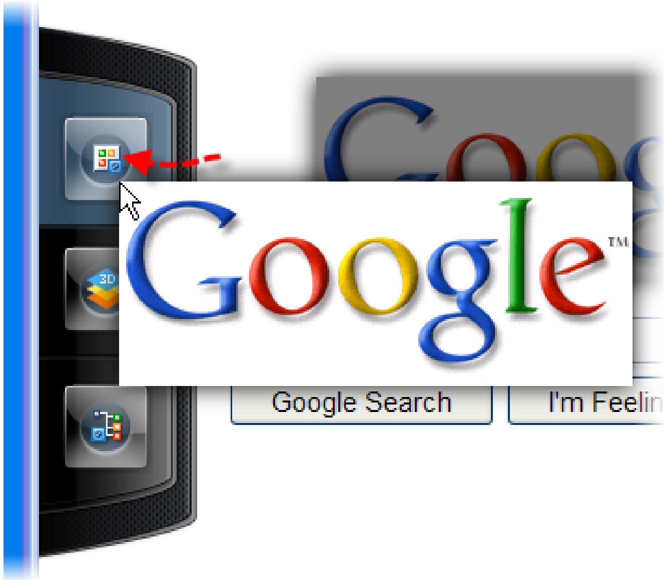
Common Tasks Menu

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.



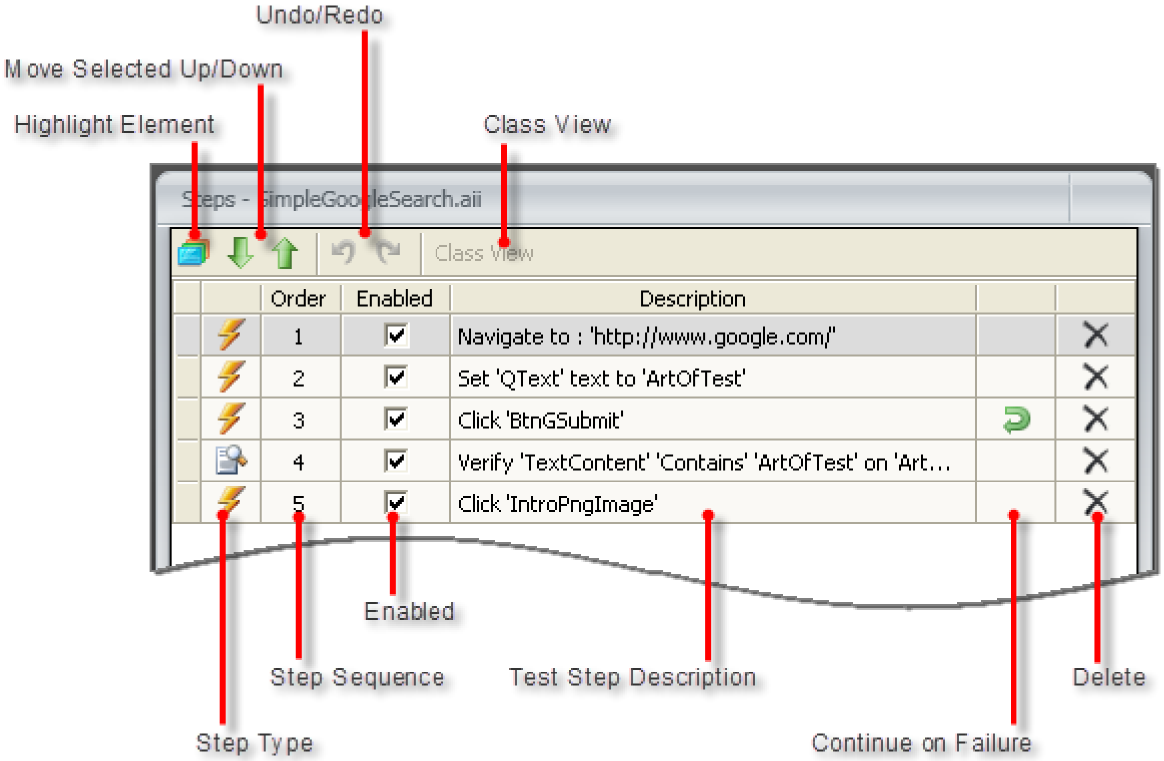
The Common Tasks Menu will pop out. Drop the element onto an icon to initiate a task you want performed against that element. The screenshot shows the Google logo image element being dragged to the Elements Explorer icon.

Test Steps

The Steps Pane allows you to work with individual steps in the test, the step's elements and any code for the test. The toolbar along the top of the Steps Pane lets you work with selected steps and the test as a whole:

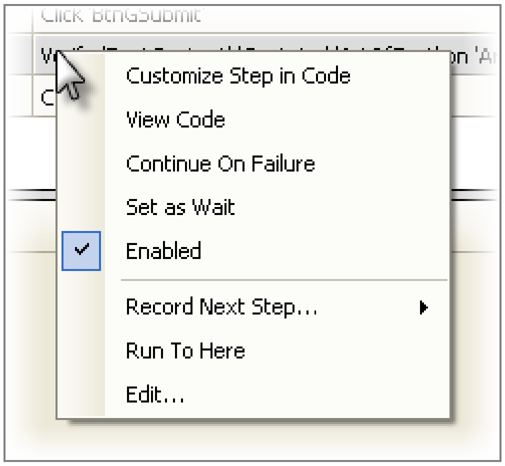
- **Highlight Element** puts visual emphasis on an element in the Recording Surface.
- **Move Selected Down/Up** moves selected test steps down or up in the list.
- **Undo/Redo** rewinds previous actions in the Steps Pane.
- **Class View** is used when writing coded steps when you want to see the entire test class at one time.

The "Enabled" column check box in the grid allows you to enable/disable a step. The "Delete" column allows you to remove 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.



	Order	Enabled	Description		
⚡	1	☑	Navigate to : 'http://www.google.com/'		✕
⚡	2	☑	Set 'QText' text to 'ArtOfTest'		✕
⚡	3	☑	Click 'BtnGSubmit'	↩	✕
🔍	4	☑	Verify 'TextContent' 'Contains' 'ArtOfTest' on 'Art...		✕
⚡	5	☑	Click 'IntroPngImage'		✕

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.

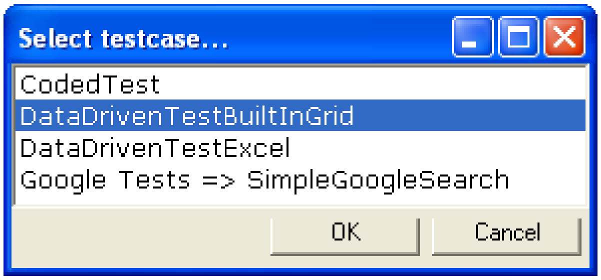


- **Customize Step in Code** creates a new test method in code. You'll also notice that the description and icon change to indicate the step is coded. Once you have converted to code, you cannot convert back.
- **View Code** simply navigates you to the code-behind for the test.
- **Continue On Failure** 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. 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.
- **Run To Here** runs the test up to the point of the selected test step. This is a nice feature that lets you run bits of the test without having to run the entire test.
- **Edit...** is enabled for Verification steps and displays the Sentence Verification Builder.

Adding Test Steps Manually

The Recording Surface can help build a wide range of automation and verification quickly and without having to resort to manual configuration. There are some steps that have to be added specifically and for these we use the **Add** section of the **Record** ribbon bar.

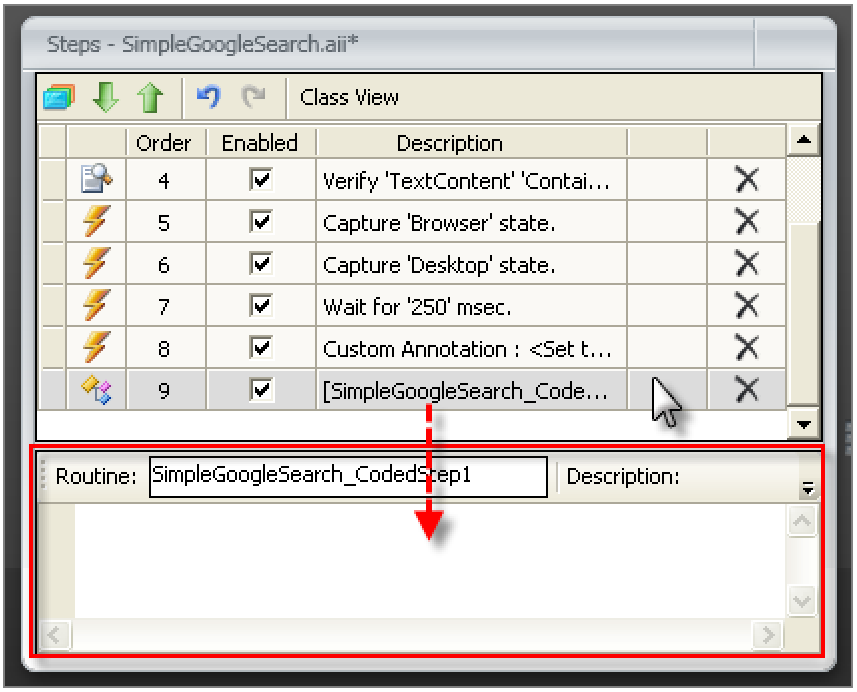
- **Test as Step** lets you run another test as a single step. Clicking this option displays the **Select Testcase...** dialog where you can select one of the other tests in the project and then click the **OK** button to add the test step.



Select testcase...

CodedTest
DataDrivenTestBuiltInGrid
DataDrivenTestExcel
Google Tests => SimpleGoogleSearch

- **Script Step** adds a coded step to the test and opens the code editor. The screenshot below shows a new coded step added to the test and the code editor open to the new code.



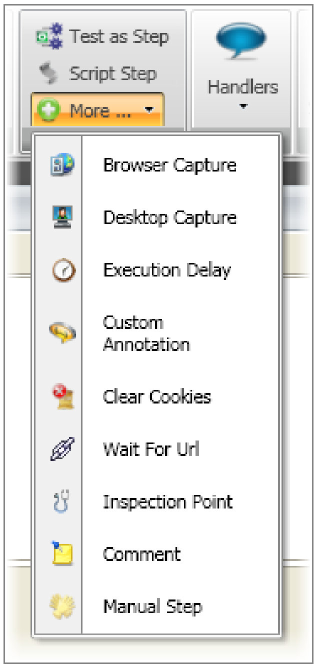
Steps - SimpleGoogleSearch.aii*

Order	Enabled	Description
4	☑	Verify 'TextContent' 'Contai...
5	☑	Capture 'Browser' state.
6	☑	Capture 'Desktop' state.
7	☑	Wait for '250' msec.
8	☑	Custom Annotation : <Set t...
9	☑	[SimpleGoogleSearch_Code...

Routine: SimpleGoogleSearch_CodedStep1

Description:

The **More...** button drops down to reveal additional steps to add to your test:



- **Browser Capture, Desktop Capture**: You can take screenshots of only the Browser or of the entire Desktop. Use the Properties Pane to change the **CaptureType** property between "Browser" and "Desktop" and the **FileNamePrefix** from the default "Snapshot" to any other prefix.

  Note: Capture steps won't take screenshots when you use the **Execute** button from the Quick Execution section of the ribbon bar. Instead you must execute a "Test List" as described in the "Test Organization" chapter of this courseware.

- **Execution Delay** pauses the test. Use the Properties Pane to change the **WaitTime** property of this step to any number of milliseconds

- **Custom Annotations** are notes that display in the browser when using the **Quick Execution | Execute** button. 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.

- **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.

- **Comment** displays a dialog box where you can enter a text comment. The comment is represented as a single test step and the comment is shown in the test log.

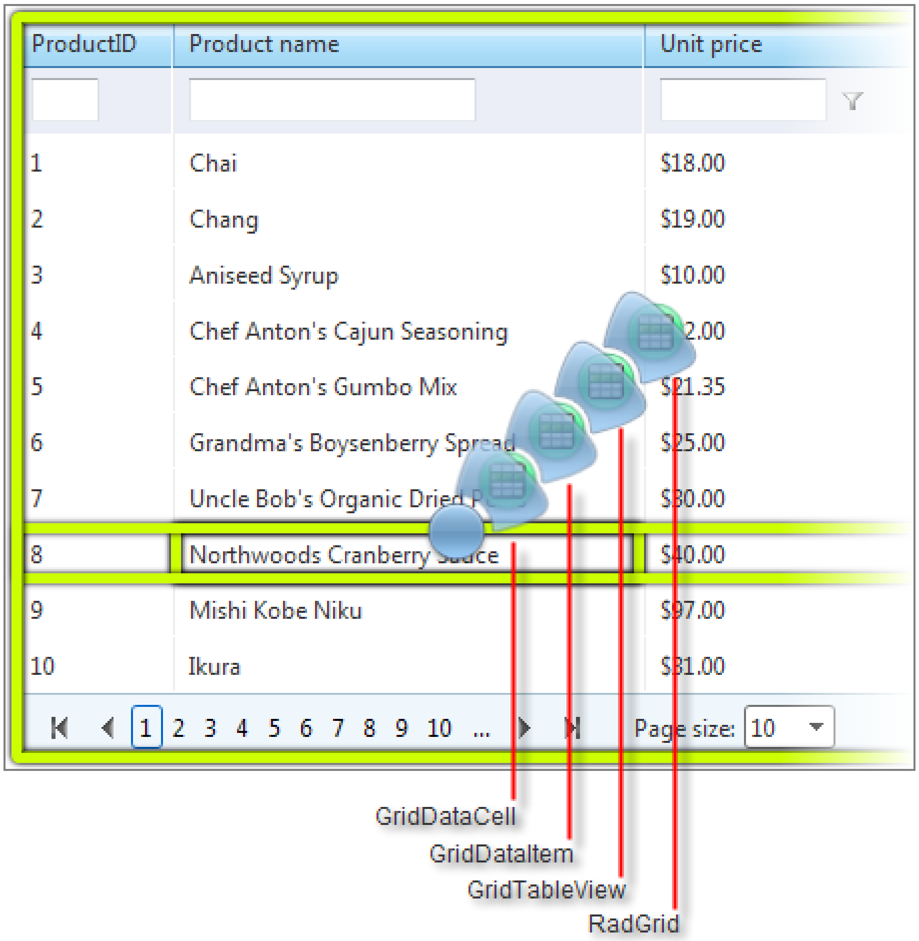
- **Manual Step** displays a dialog box where you can enter directions for some manual step. This description is placed in the **ManualDescription** property. The ExecutionTimeout is the number of minutes to wait before logging a timeout failure for the test.

Translators

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

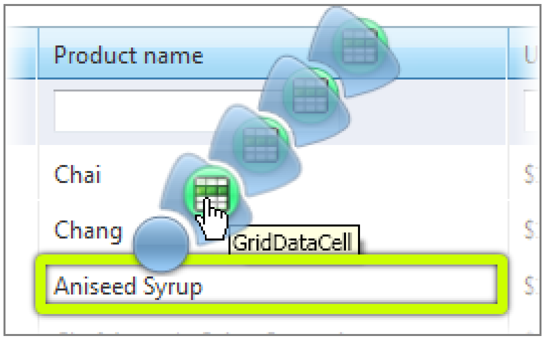
Translators are extensions that open up an element to work with WebUI. A translator describes the actions of an element that can be automated and verifications that can be performed. Translators allow interaction with the WebUI user interface including the Elements Menu and Elements Explorer. WebUI ships with basic translators for HTML and Silverlight, and translators built specifically for AJAX and Silverlight RadControls. WebUI 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 colored borders around a "translated" element that indicate how elements are contained within each other. In the screenshot of the RadGrid below we see the enhanced highlighting and fan of translator nubs where the innermost leaf is a GridDataCell and the outermost leaf is a RadGrid.

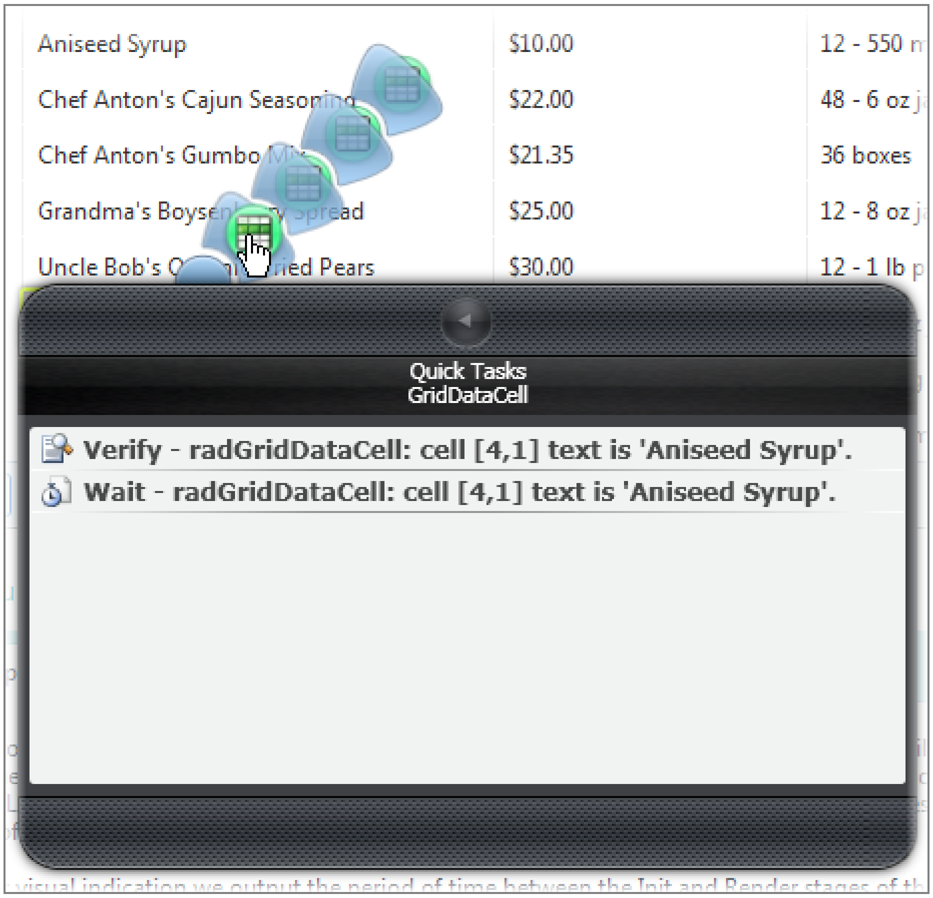


ProductID	Product name	Unit price
1	Chai	$18.00
2	Chang	$19.00
3	Aniseed Syrup	$10.00
4	Chef Anton's Cajun Seasoning	2.00
5	Chef Anton's Gumbo Mix	$21.35
6	Grandma's Boysenberry Spread	$25.00
7	Uncle Bob's Organic Dried P	$30.00
8	Northwoods Cranberry Sauce	$40.00
9	Mishi Kobe Niku	$97.00
10	Ikura	$31.00

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.



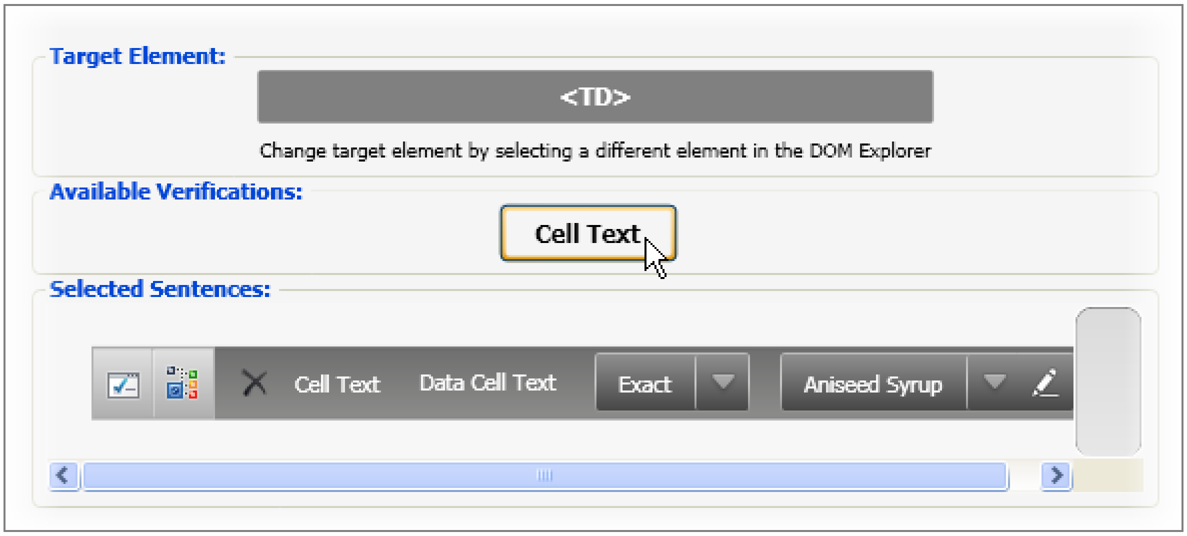
When you click on one of the nub leaves, the Elements Menu, Quick Tasks button displays tasks for the specific leaf. The screenshot below shows verification and wait tasks for a particular grid cell. Without the translator you couldn't get to this level of detail easily.



Verify - radGridDataCell: cell [4,1] text is 'Aniseed Syrup'.

Wait - radGridDataCell: cell [4,1] text is 'Aniseed Syrup'.

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.



Target Element:

<TD>

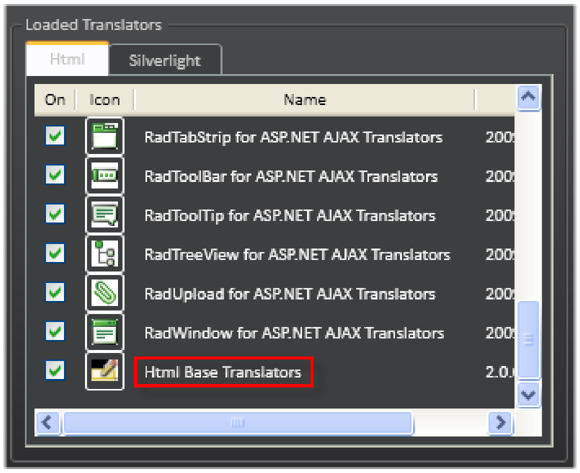
Change target element by selecting a different element in the DOM Explorer

Available Verifications:

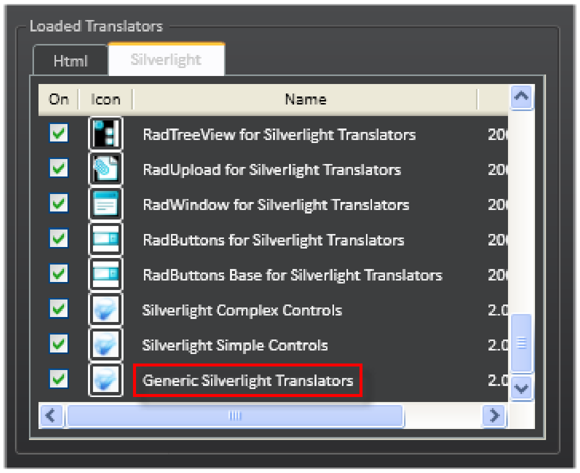
Selected Sentences:

Cell Text Data Cell Text Exact Aniseed Syrup

WebUI comes with translators for both HTML and Silverlight. These translators have a "base" or "generic" group of intrinsic translators that are used whenever a more specific translator is not available. These translators are listed in the User Settings dialog.



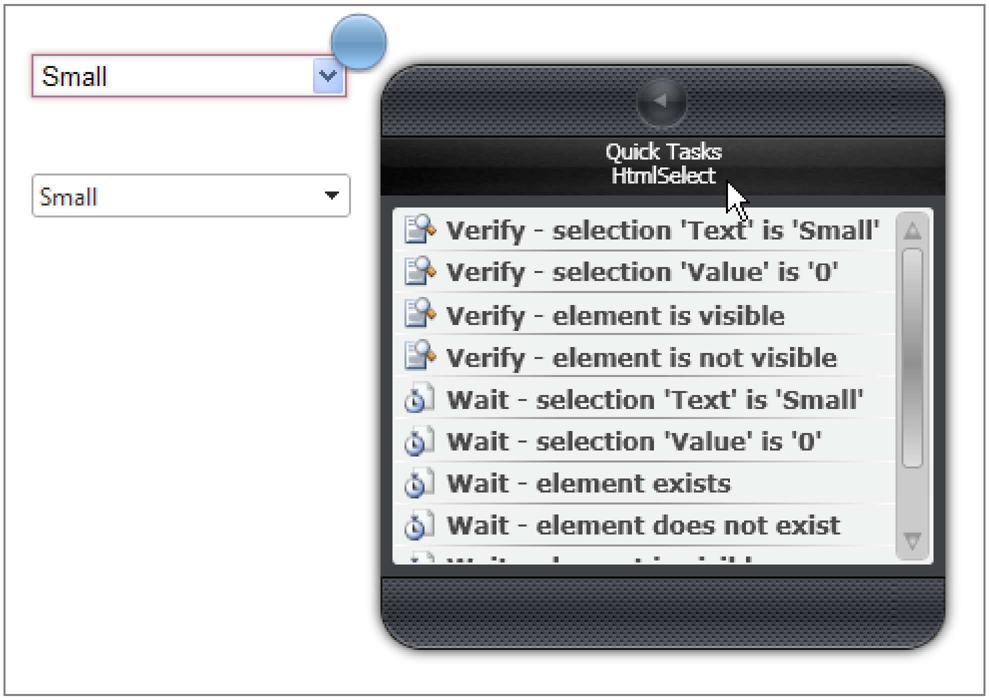
On	Icon	Name	
☑		RadTabStrip for ASP.NET AJAX Translators	200
☑		RadToolBar for ASP.NET AJAX Translators	200
☑		RadToolTip for ASP.NET AJAX Translators	200
☑		RadTreeView for ASP.NET AJAX Translators	200
☑		RadUpload for ASP.NET AJAX Translators	200
☑		RadWindow for ASP.NET AJAX Translators	200
☑		Html Base Translators	2.0.



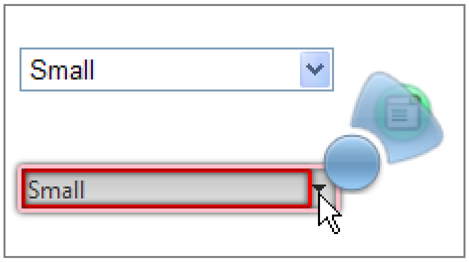
On	Icon	Name	
☑		RadTreeView for Silverlight Translators	20
☑		RadUpload for Silverlight Translators	20
☑		RadWindow for Silverlight Translators	20
☑		RadButtons for Silverlight Translators	20
☑		RadButtons Base for Silverlight Translators	20
☑		Silverlight Complex Controls	2.0
☑		Silverlight Simple Controls	2.0
☑		Generic Silverlight Translators	2.0

Standard vs Translated Comparison

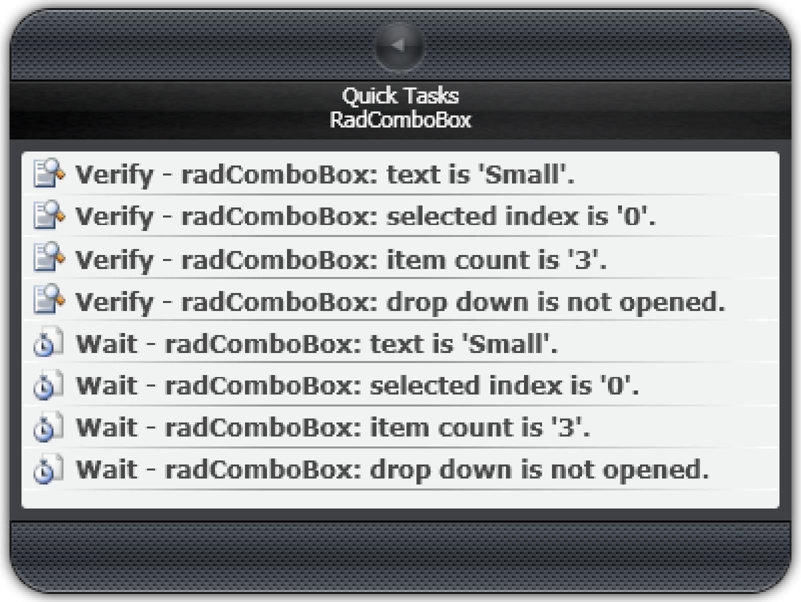
Depending on the complexity of the control, translators can surface volumes of detail about a control's inner workings. To get a feel for the differences between generic translation using the intrinsic translators and translators used for specific controls, let's compare a DropDownList control (standard 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.


Small
Small
Quick Tasks
HtmlSelect
Verify - selection 'Text' is 'Small'
Verify - selection 'Value' is '0'
Verify - element is visible
Verify - element is not visible
Wait - selection 'Text' is 'Small'
Wait - selection 'Value' is '0'
Wait - element exists
Wait - element does not exist

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.



Quick Tasks
RadComboBox

- Verify - radComboBox: text is 'Small'.
- Verify - radComboBox: selected index is '0'.
- Verify - radComboBox: item count is '3'.
- Verify - radComboBox: drop down is not opened.
- Wait - radComboBox: text is 'Small'.
- Wait - radComboBox: selected index is '0'.
- Wait - radComboBox: item count is '3'.
- Wait - radComboBox: drop down is not opened.

Drag and Drop

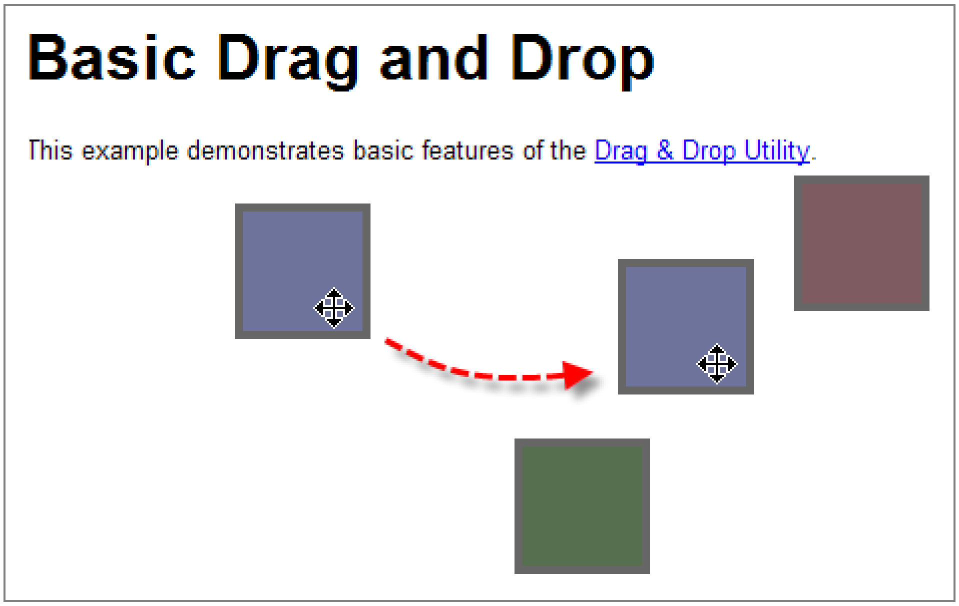
You can automate the drag and drop of any element in a web page including simple HTML elements, RadControls for ASP.NET AJAX and Silverlight elements. Drag and Drop operations have been abstracted by WebUI so that the general automation process is the same no matter if we're dragging an HTML element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

There are two ways to automate a drag and 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 and 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 and drop action as a test step.

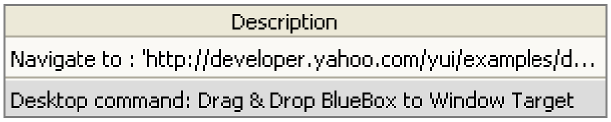
The destination of the drag and drop operation is called the "drop target" and can be the entire browser window or another element. The drag and drop mechanism is flexible enough to handle drop targets that are resized or that move multiple times. WebUI allows you to fine tune the exact placement of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

Drag and 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.

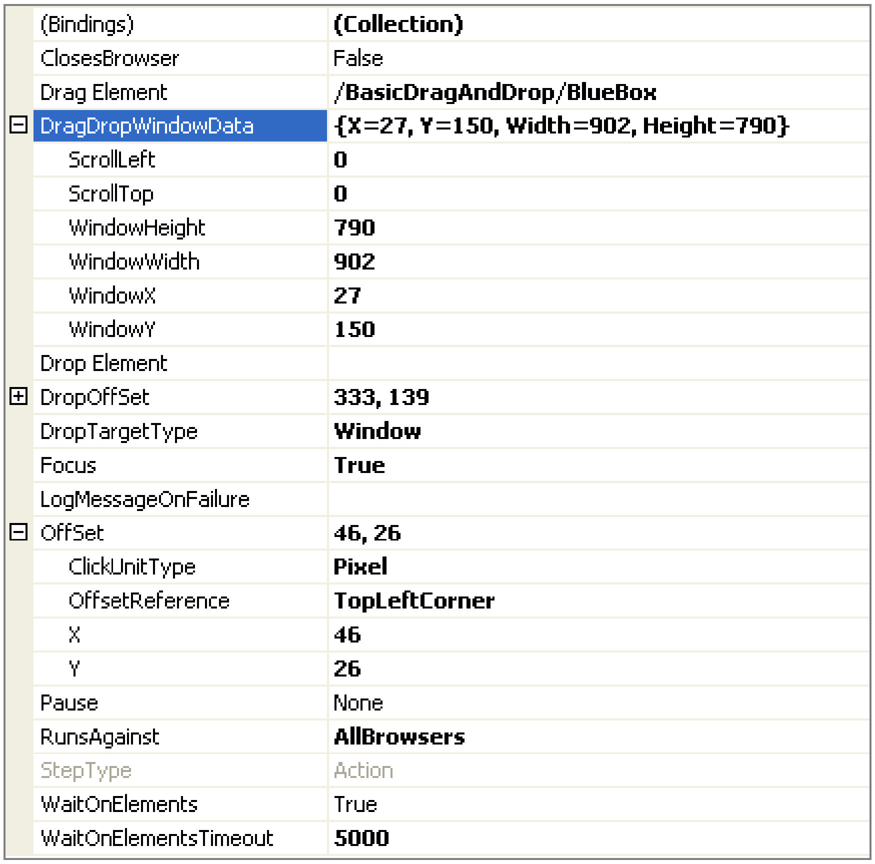

Basic Drag and Drop
This example demonstrates basic features of the Drag & Drop Utility.

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



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

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



(Bindings)	**(Collection)**
ClosesBrowser	False
Drag Element	**/BasicDragAndDrop/BlueBox**
⊟ DragDropWindowData	**{X=27, Y=150, Width=902, Height=790}**
ScrollLeft	**0**
ScrollTop	**0**
WindowHeight	**790**
WindowWidth	**902**
WindowX	**27**
WindowY	**150**
Drop Element	
⊞ DropOffSet	**333, 139**
DropTargetType	**Window**
Focus	**True**
LogMessageOnFailure	
⊟ OffSet	**46, 26**
ClickUnitType	**Pixel**
OffsetReference	**TopLeftCorner**
X	**46**
Y	**26**
Pause	None
RunsAgainst	**AllBrowsers**
StepType	Action
WaitOnElements	True
WaitOnElementsTimeout	**5000**

- 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 is 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.

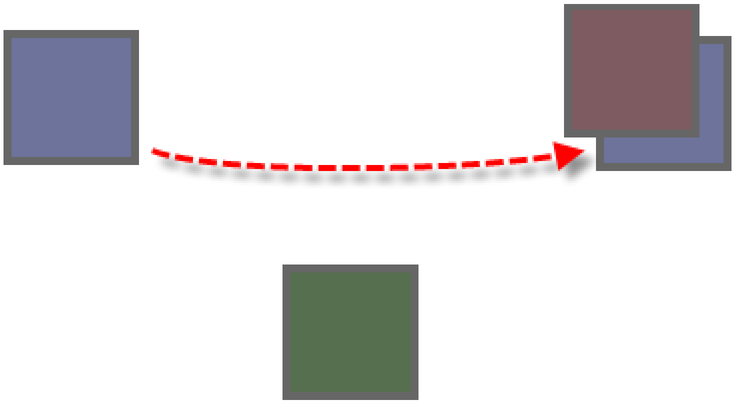
Tip!

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.

Dragging to an Element

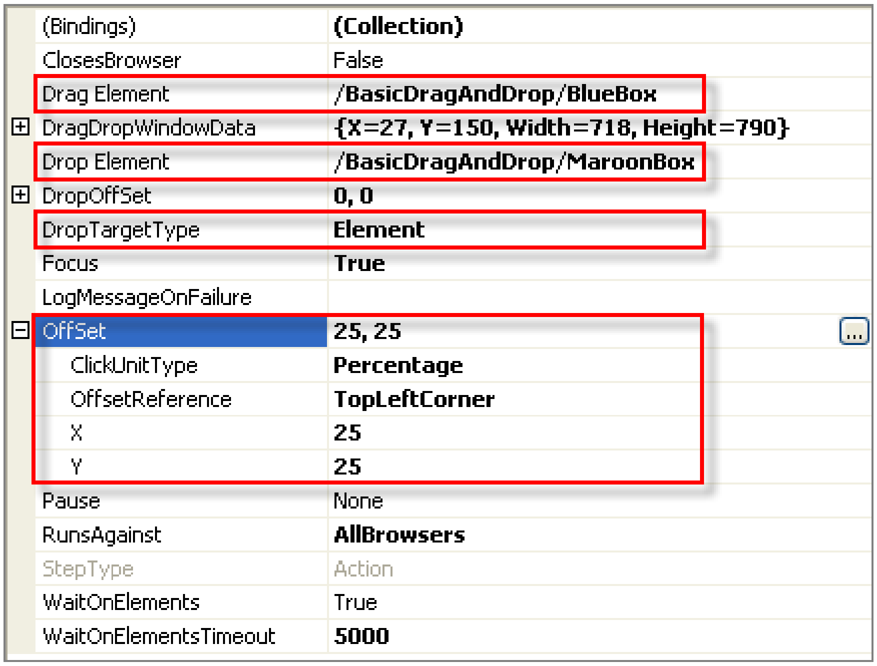
For the sake of comparison, here's another drag and drop automation sample where the "Blue" element is being dragged and the "Maroon" element is the drop target.



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.

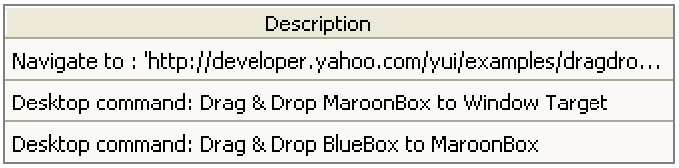
The properties of this drag operation in the screenshot below show the Drag Element is the "BlueBox", the Drop Element is the "MaroonBox", the DropTargetType is "Element", and finally the Offset of the dragged element is 25% from the top left corner.



(Bindings)	(Collection)
ClosesBrowser	False
Drag Element	/BasicDragAndDrop/BlueBox
⊞ DragDropWindowData	{X=27, Y=150, Width=718, Height=790}
Drop Element	/BasicDragAndDrop/MaroonBox
⊞ DropOffSet	0, 0
DropTargetType	Element
Focus	True
LogMessageOnFailure	
⊟ OffSet	25, 25
ClickUnitType	Percentage
OffsetReference	TopLeftCorner
X	25
Y	25
Pause	None
RunsAgainst	AllBrowsers
StepType	Action
WaitOnElements	True
WaitOnElementsTimeout	5000

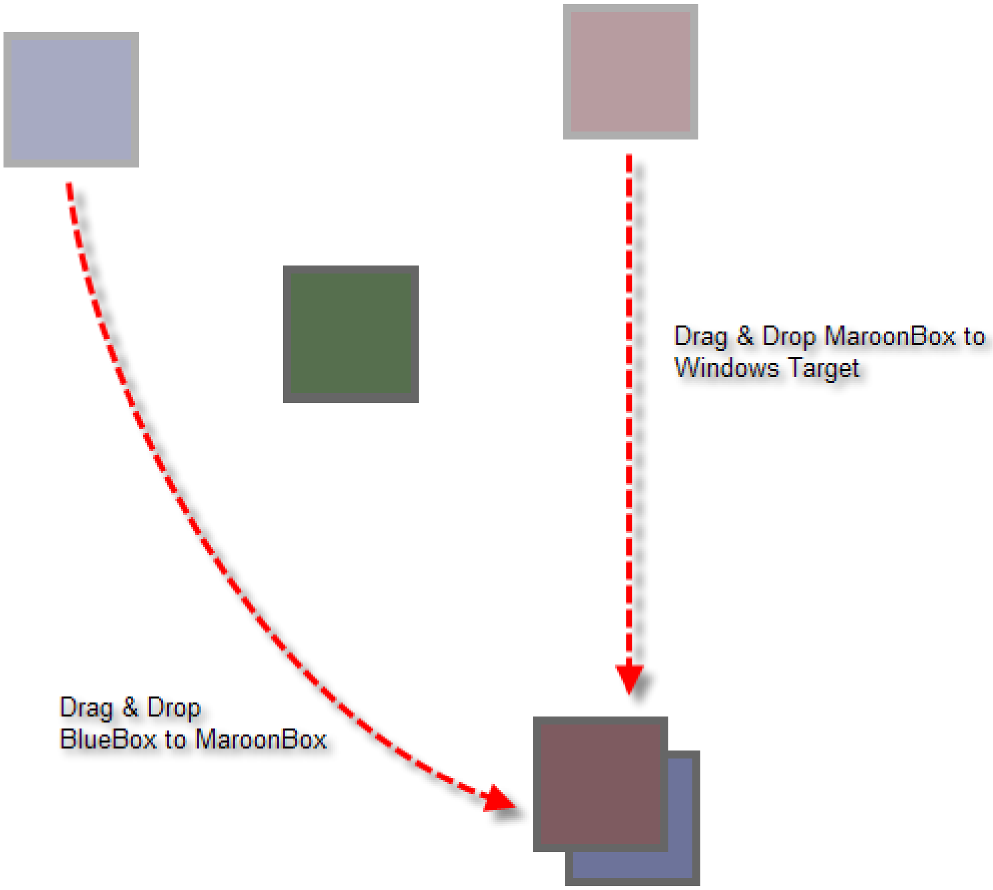
Hitting a Moving Target

One of the nice drag and drop features in WebUI 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

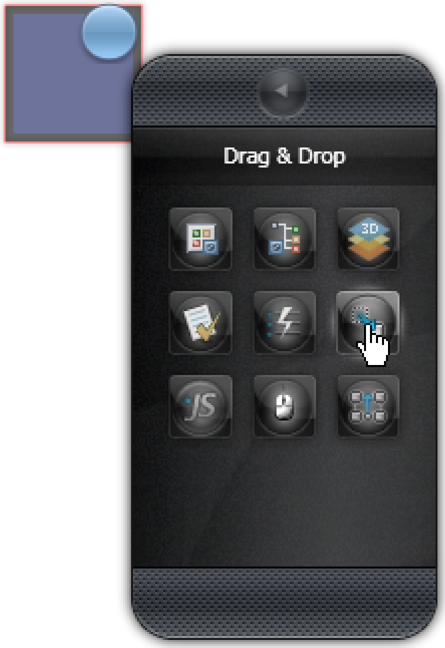
The "Drag & Drop MaroonBox to Windows Target" step drags the maroon box to a point below its initial position. The next step uses the exact same definition as the previous example where the blue box element is dragged to the maroon box. Nothing has changed in this step, yet the blue box "follows" the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


Drag & Drop MaroonBox to Windows Target
Drag & Drop BlueBox to MaroonBox

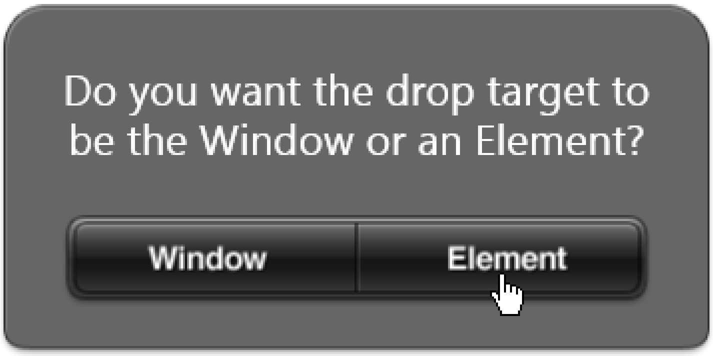
Walk Through

1) 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.

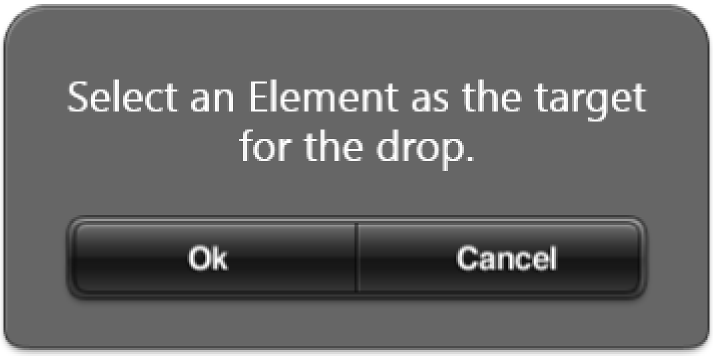
2) To use the enhanced UI, click the Highlighting button and hover the mouse above the element to be dragged until the Nub appears. Click the Nub, then click the Drag and Drop button.



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

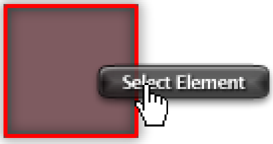

Do you want the drop target to be the Window or an Element?

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

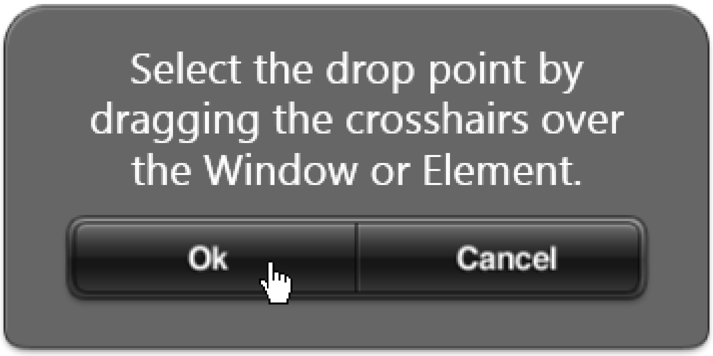


Select an Element as the target for the drop.

5) Hover the mouse above the maroon element and click the "Select Element" button.

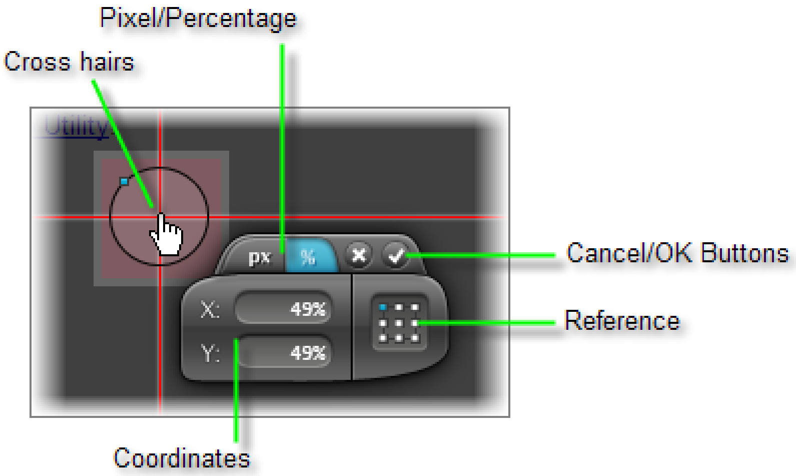


6) The next prompt asks you to select a drop point. Click **OK** to continue.

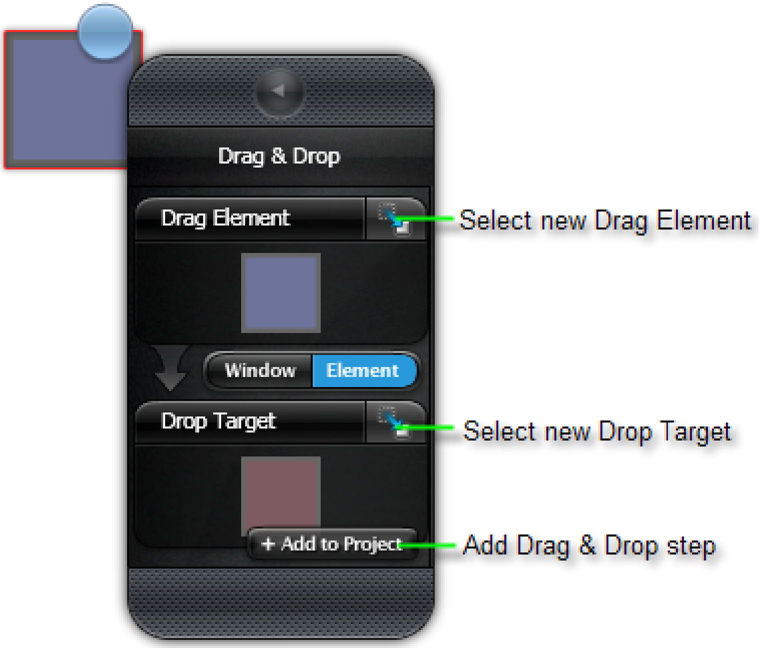


Select the drop point by dragging the crosshairs over the Window or Element.

7) 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.



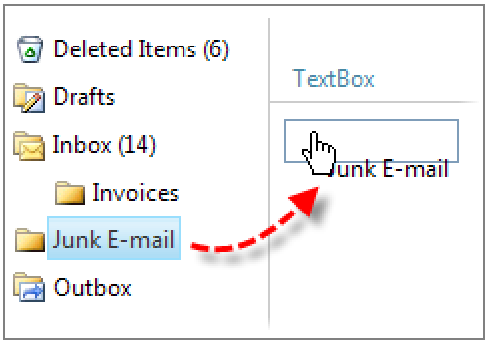
8) A summary of the Drag and Drop operation will appear in the Elements Menu. Click the "Add to Project" button to include the new drag and drop action as a test step.



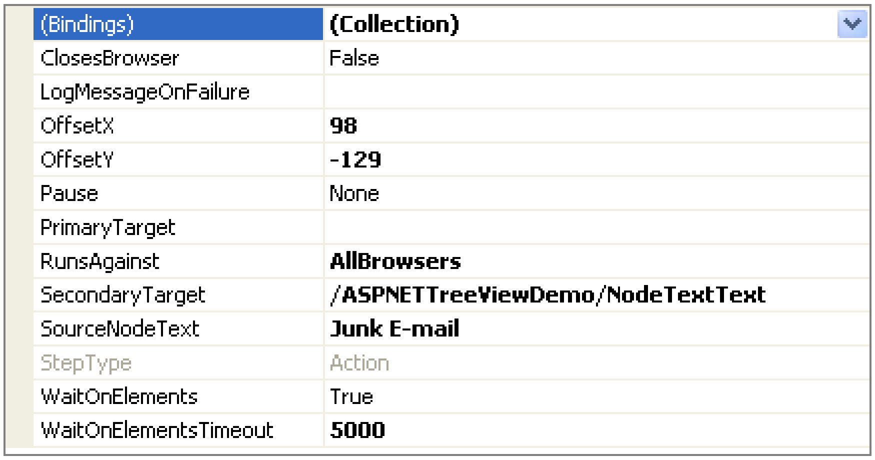
9) Run the test. The blue element should drag over to the maroon element. All test steps should pass.

Translators

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

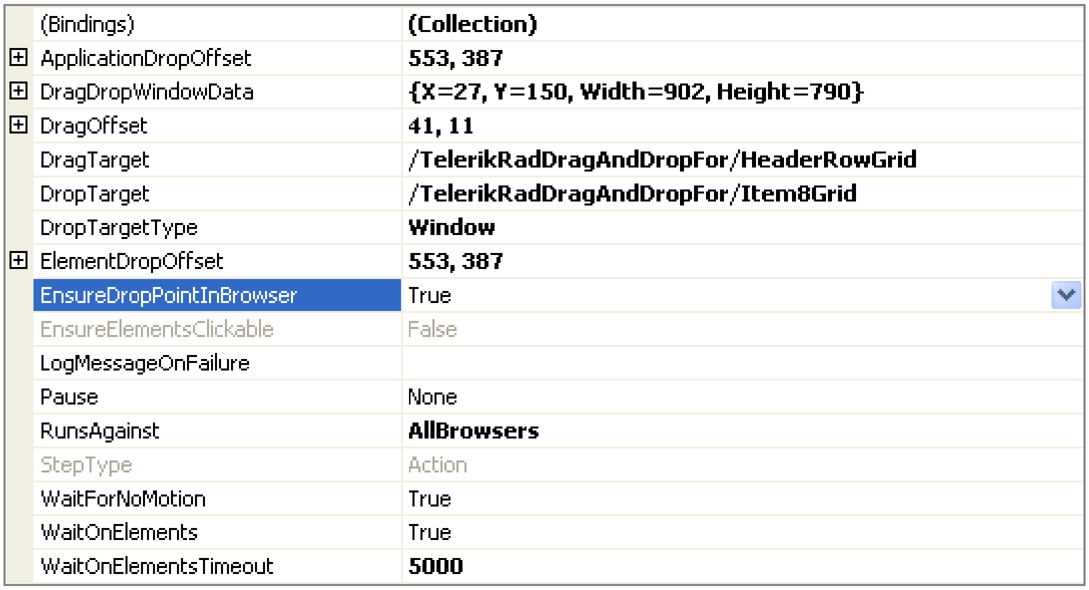


A TreeView DragAndDropAction action is created automatically with properties that contain the identity of the tree view node being dragged and the location of the drop target. The drop target can be an offset in relation to the source node using the **OffsetX** and **OffsetY** properties or can be another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



(Bindings)	**(Collection)**
ClosesBrowser	False
LogMessageOnFailure	
OffsetX	**98**
OffsetY	**-129**
Pause	None
PrimaryTarget	
RunsAgainst	**AllBrowsers**
SecondaryTarget	**/ASPNETTreeViewDemo/NodeTextText**
SourceNodeText	**Junk E-mail**
StepType	Action
WaitOnElements	True
WaitOnElementsTimeout	**5000**

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.



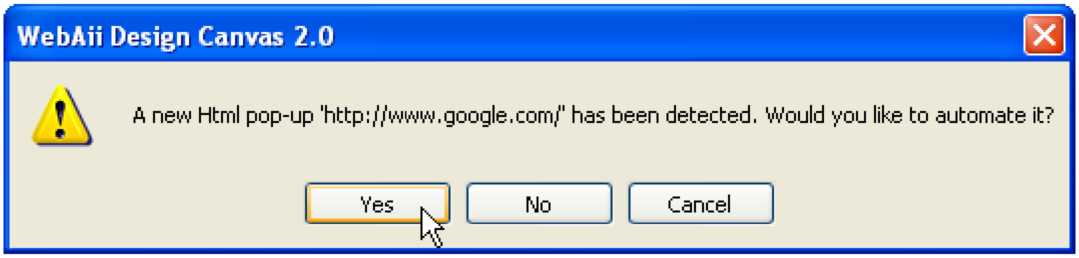
	(Bindings)	**(Collection)**
⊞	ApplicationDropOffset	**553, 387**
⊞	DragDropWindowData	**{X=27, Y=150, Width=902, Height=790}**
⊞	DragOffset	**41, 11**
	DragTarget	**/TelerikRadDragAndDropFor/HeaderRowGrid**
	DropTarget	**/TelerikRadDragAndDropFor/Item8Grid**
	DropTargetType	**Window**
⊞	ElementDropOffset	**553, 387**
	EnsureDropPointInBrowser	True
	EnsureElementsClickable	False
	LogMessageOnFailure	
	Pause	None
	RunsAgainst	**AllBrowsers**
	StepType	Action
	WaitForNoMotion	True
	WaitOnElements	True
	WaitOnElementsTimeout	**5000**

Handling Dialogs

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 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 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.

HTML Popups

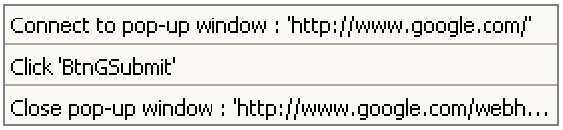
HTML popups are detected by WebUI automatically. When an HTML popup is about to appear, WebUI allows you to automate the popup.



WebAii Design Canvas 2.0

A new Html pop-up 'http://www.google.com/' has been detected. Would you like to automate it?

If you click the "Yes" button in the prompt, WebUI will display the popup window along with a toolbar that includes buttons for turning on highlighting, recording and the DOM Explorer. WebUI 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.

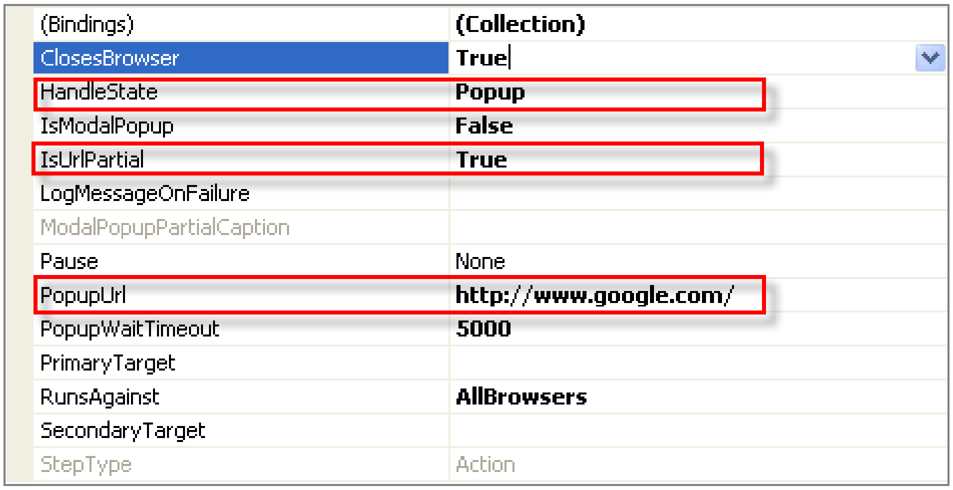


Connect to pop-up window : 'http://www.google.com/'

Click 'BtnGSubmit'

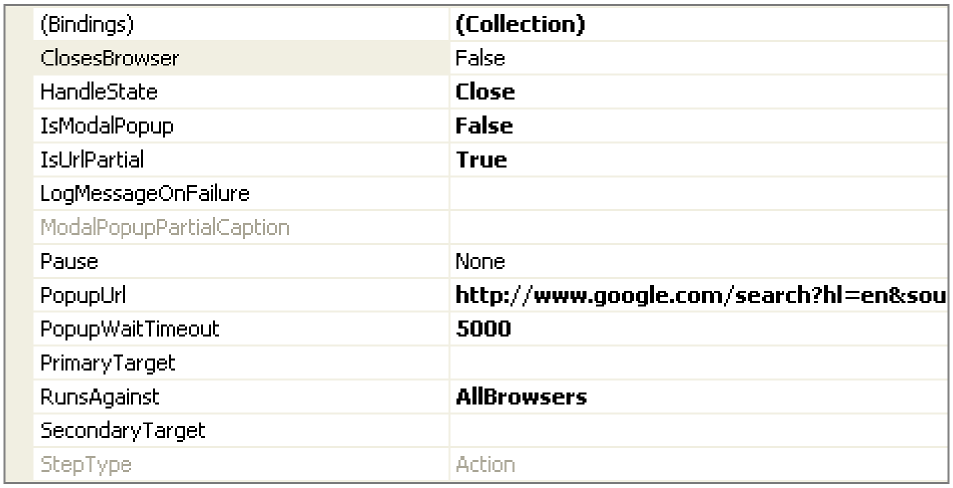
Close pop-up window : 'http://www.google.com/webh...

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.



(Bindings)	**(Collection)**
ClosesBrowser	**True**
HandleState	**Popup**
IsModalPopup	**False**
IsUrlPartial	**True**
LogMessageOnFailure	
ModalPopupPartialCaption	
Pause	None
PopupUrl	**http://www.google.com/**
PopupWaitTimeout	**5000**
PrimaryTarget	
RunsAgainst	**AllBrowsers**
SecondaryTarget	
StepType	Action

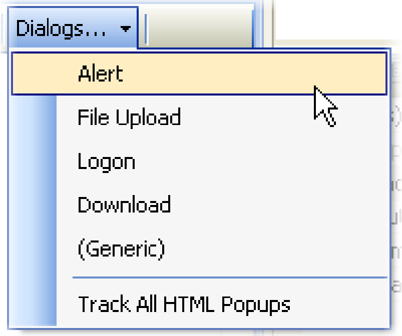
Once the pop-up is open, you can highlight, add verifications and record test steps. When you close the pop-up, a last test step that handles the "Close" state is added.



(Bindings)	**(Collection)**
ClosesBrowser	False
HandleState	**Close**
IsModalPopup	**False**
IsUrlPartial	**True**
LogMessageOnFailure	
ModalPopupPartialCaption	
Pause	None
PopupUrl	**http://www.google.com/search?hl=en&sou**
PopupWaitTimeout	**5000**
PrimaryTarget	
RunsAgainst	**AllBrowsers**
SecondaryTarget	
StepType	Action

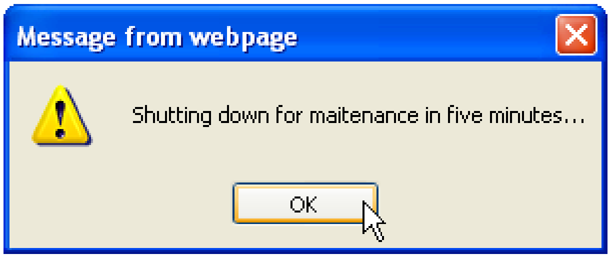
Win32 Dialogs

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 be handled as test steps.



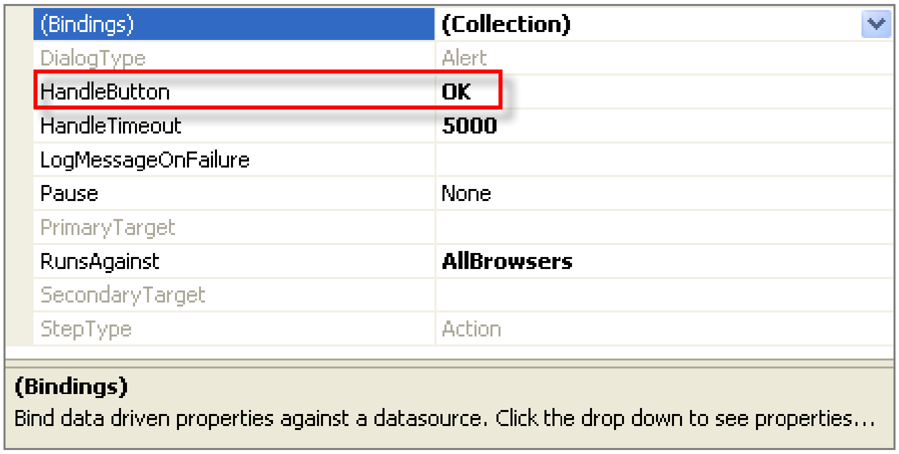
Alert

An "Alert" dialog displays a message and a single "OK" button.



Shutting down for maitenance in five minutes...

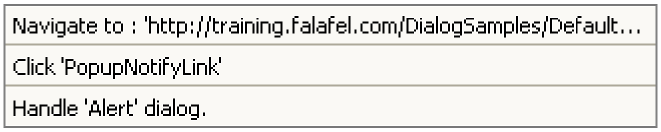
When you create a test step to handle the Alert dialog, the Properties pane includes a **HandleButton** property that should be set to "OK" or "CANCEL".



(Bindings)	(Collection)
DialogType	Alert
HandleButton	OK
HandleTimeout	5000
LogMessageOnFailure	
Pause	None
PrimaryTarget	
RunsAgainst	AllBrowsers
SecondaryTarget	
StepType	Action

(Bindings)
Bind data driven properties against a datasource. Click the drop down to see properties...

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.



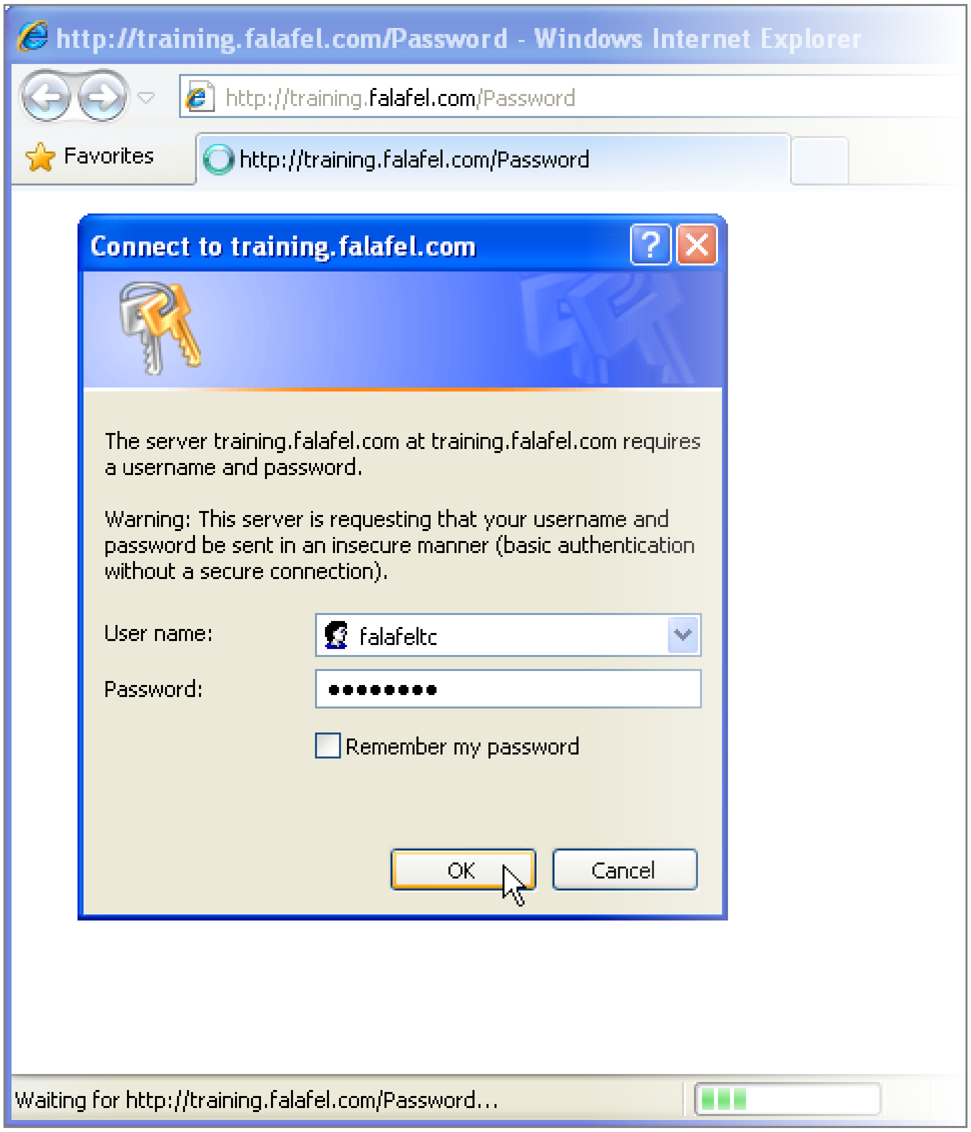
Navigate to : 'http://training.falafel.com/DialogSamples/Default...

Click 'PopupNotifyLink'

Handle 'Alert' dialog.

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.



http://training.falafel.com/Password - Windows Internet Explorer
http://training.falafel.com/Password
Favorites
http://training.falafel.com/Password
Connect to training.falafel.com

The server training.falafel.com at training.falafel.com requires a username and password.

Warning: This server is requesting that your username and password be sent in an insecure manner (basic authentication without a secure connection).

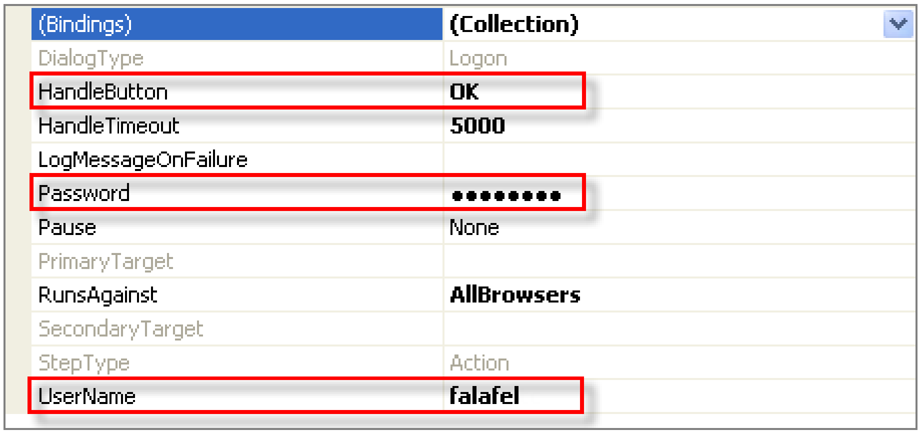
User name: falafeltc

Password: ●●●●●●●●●

□ Remember my password

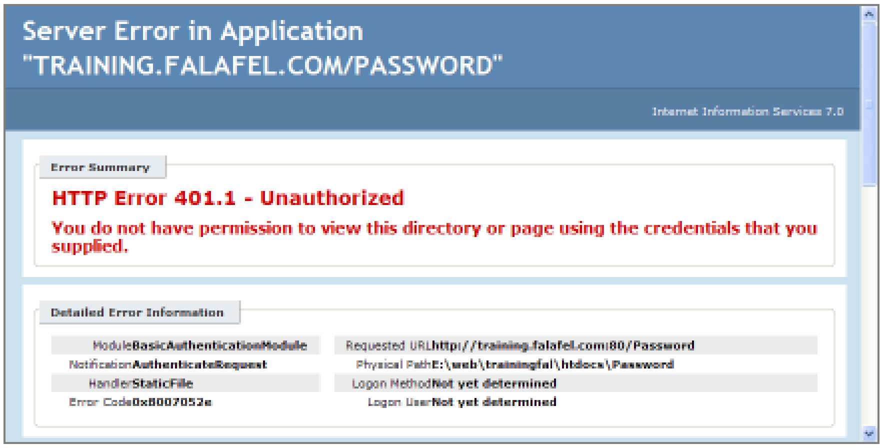
Waiting for http://training.falafel.com/Password...

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.



(Bindings)	(Collection)
DialogType	Logon
HandleButton	OK
HandleTimeout	5000
LogMessageOnFailure	
Password	●●●●●●●●●
Pause	None
PrimaryTarget	
RunsAgainst	AllBrowsers
SecondaryTarget	
StepType	Action
UserName	falafel

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:



Server Error in Application "TRAINING.FALAFEL.COM/PASSWORD"

Internet Information Services 7.0

Error Summary

HTTP Error 401.1 - Unauthorized

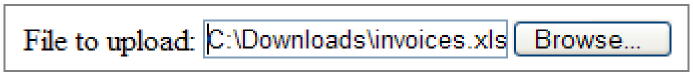
You do not have permission to view this directory or page using the credentials that you supplied.

Detailed Error Information

Module	BasicAuthenticationModule	Requested URL	http://training.falafel.com:80/Password
Notification	AuthenticateRequest	Physical Path	E:\web\trainingfal\htdocs\Password
Handler	StaticFile	Logon Method	Not yet determined
Error Code	0x8007052e	Logon User	Not yet determined

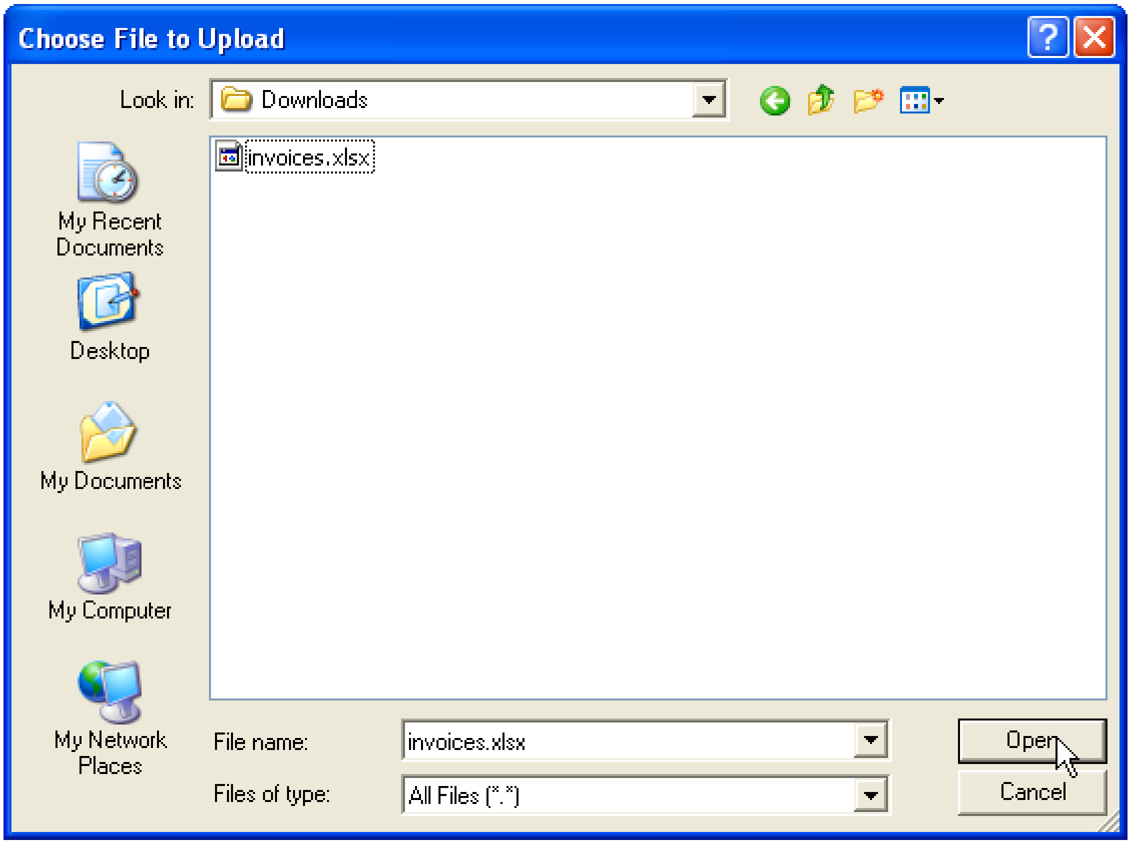
File Upload

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.



File to upload: C:\Downloads\invoices.xls

When the user clicks the "Browse" button, a File Upload dialog displays.

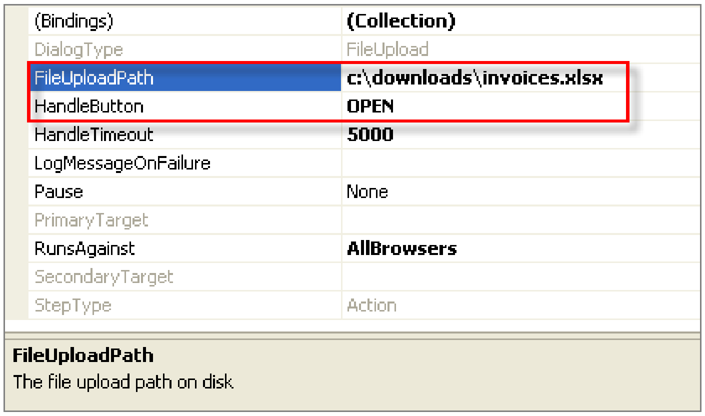


Choose File to Upload

Look in: Downloads

File name: invoices.xlsx

Files of type: All Files (*.*)



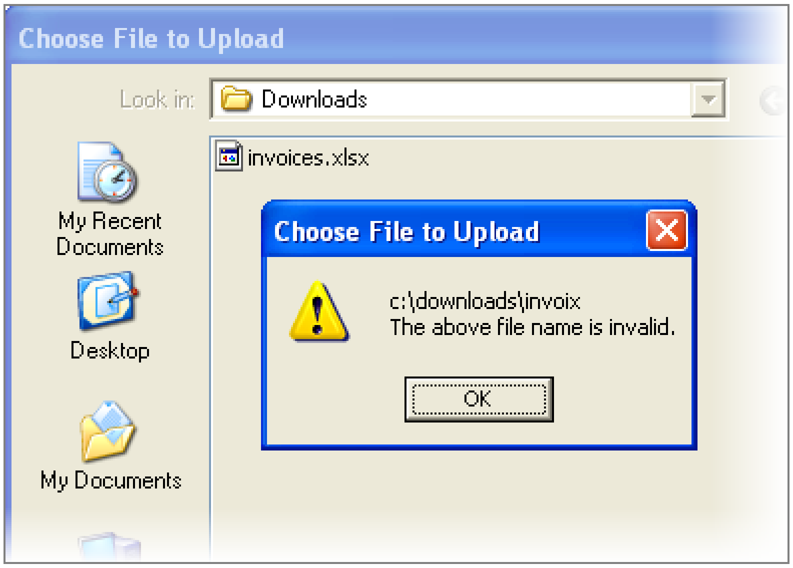
(Bindings)	(Collection)
DialogType	FileUpload
FileUploadPath	c:\downloads\invoices.xlsx
HandleButton	OPEN
HandleTimeout	5000
LogMessageOnFailure	
Pause	None
PrimaryTarget	
RunsAgainst	AllBrowsers
SecondaryTarget	
StepType	Action

FileUploadPath
The file upload path on disk

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.

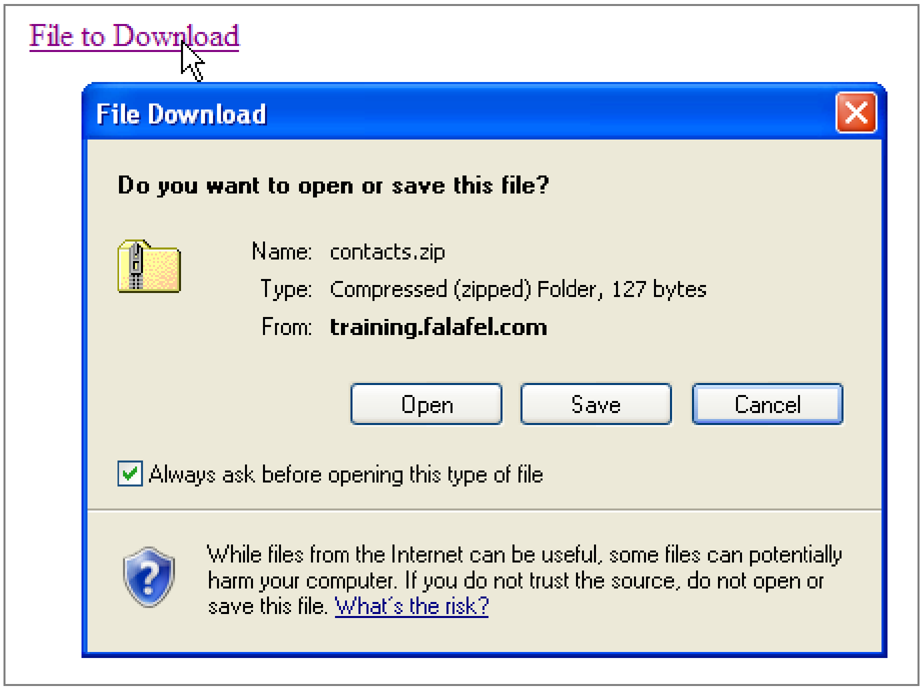


Choose File to Upload

c:\downloads\invoix
The above file name is invalid.

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.


File to Download

File Download

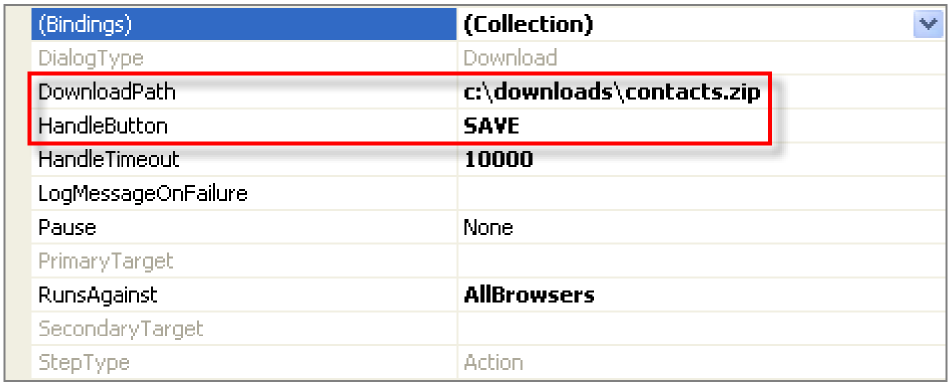
Do you want to open or save this file?

Name: contacts.zip
Type: Compressed (zipped) Folder, 127 bytes
From: training.falafel.com

☑ Always ask before opening this type of file

While files from the Internet can be useful, some files can potentially harm your computer. If you do not trust the source, do not open or save this file. What's the risk?

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.



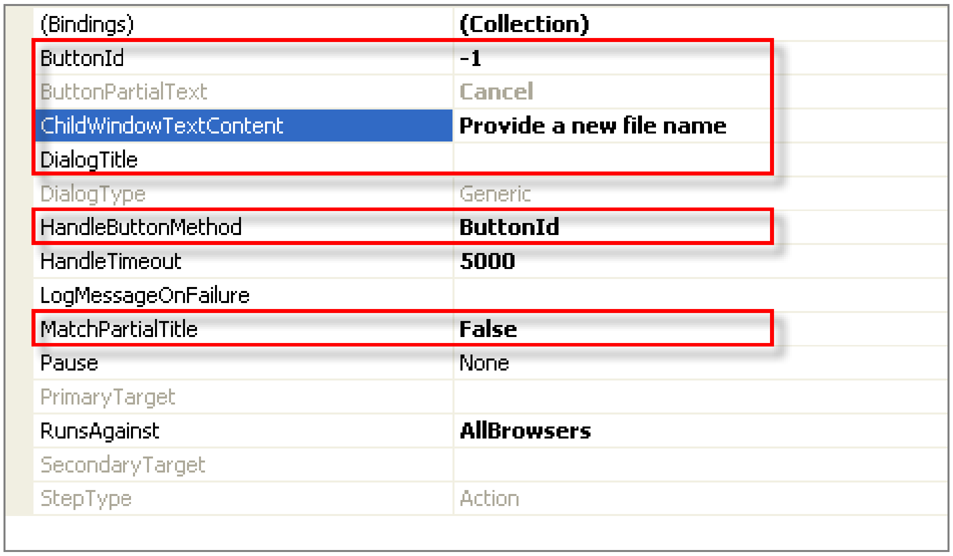
(Bindings)	**(Collection)**
DialogType	Download
DownloadPath	**c:\downloads\contacts.zip**
HandleButton	**SAVE**
HandleTimeout	**10000**
LogMessageOnFailure	
Pause	None
PrimaryTarget	
RunsAgainst	**AllBrowsers**
SecondaryTarget	
StepType	Action

Generic

"Generic" is a customizable dialog handler that deals with any "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 set to "ButtonPartialText", then the **ButtonPartialText** property is used 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".

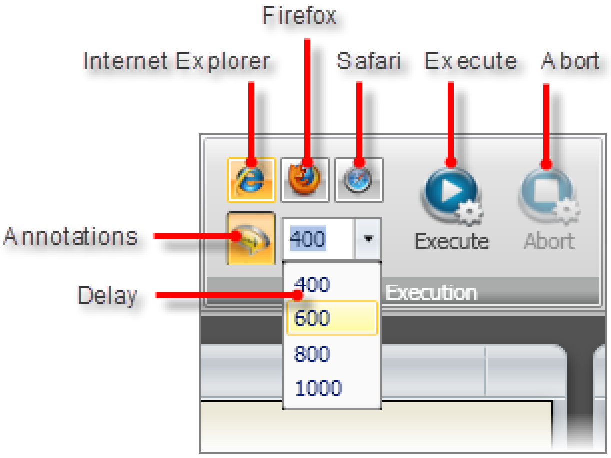


(Bindings)	**(Collection)**
ButtonId	**-1**
ButtonPartialText	**Cancel**
ChildWindowTextContent	**Provide a new file name**
DialogTitle	
DialogType	Generic
HandleButtonMethod	**ButtonId**
HandleTimeout	**5000**
LogMessageOnFailure	
MatchPartialTitle	**False**
Pause	None
PrimaryTarget	
RunsAgainst	**AllBrowsers**
SecondaryTarget	
StepType	Action

6.3 Running Tests

Before organizing tests into larger lists and executing the tests formally, you will want to perform a "dry run" to make sure the test performs as expected in various browsers.

Pressing the **Execute** button will run the test in the browser specified by the **Internet Explorer**, **Firefox** or **Safari** button. During test execution, the **Abort** button will become enabled, allowing you to stop the test at any time. To slow the test run down and have the browser annotate each step with a brief message, enable the **Annotations** button. Set a **Delay** in milliseconds between each test step from the drop down menu.



6.4 Walk Through

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

Find the materials for this Topic at...

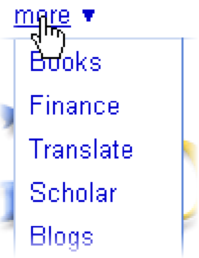
\Projects\RecordingTests

1) From the Welcome Screen, Click the **Record New Test** button. This step will navigate to the Recording Surface, with recording turned on.

2) In the browser address bar, enter "www.google.com".

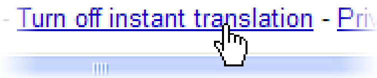
3) Press the **Enter** key. This step will display the Google home page.

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.

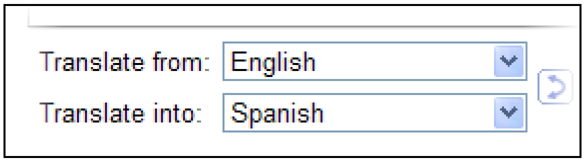
6) If "instant translation" is on (the default), click the link that toggles instant translation off.



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

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

9) Select "Spanish" in the "Translate Into" drop down list.



Translate from: English

Translate into: Spanish

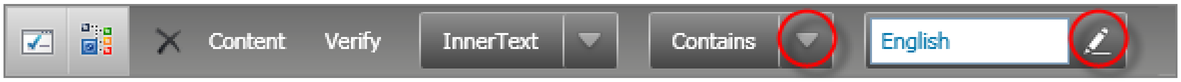
10)Click the Highlighting button from the Recording Surface toolbar.

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

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

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

14)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.



Content Verify InnerText Contains English

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

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

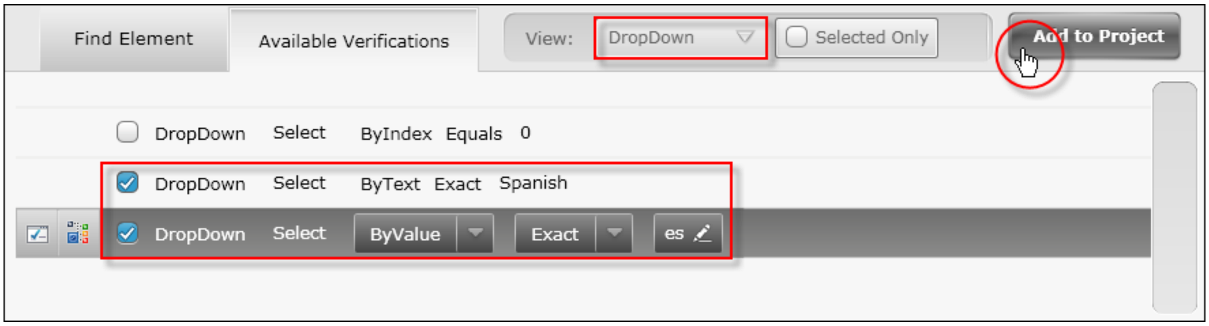
17)Click the Elements Menu **View 3D** icon.

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

19)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.

20)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.

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



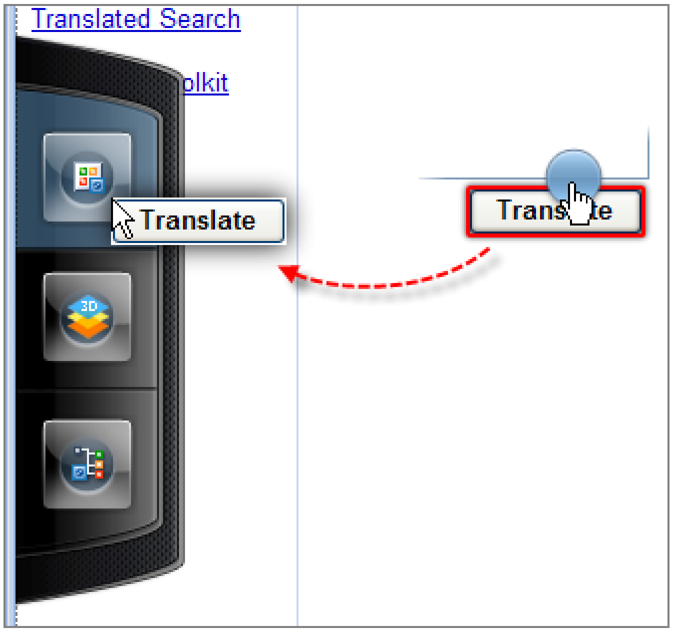
□ DropDown Select ByIndex Equals 0

☑ DropDown Select ByText Exact Spanish

☑ DropDown Select ByValue Exact es

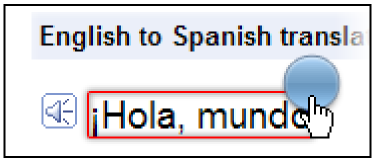
22)Click the **Close** button located in the upper right of the 3D Viewer.

23)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.

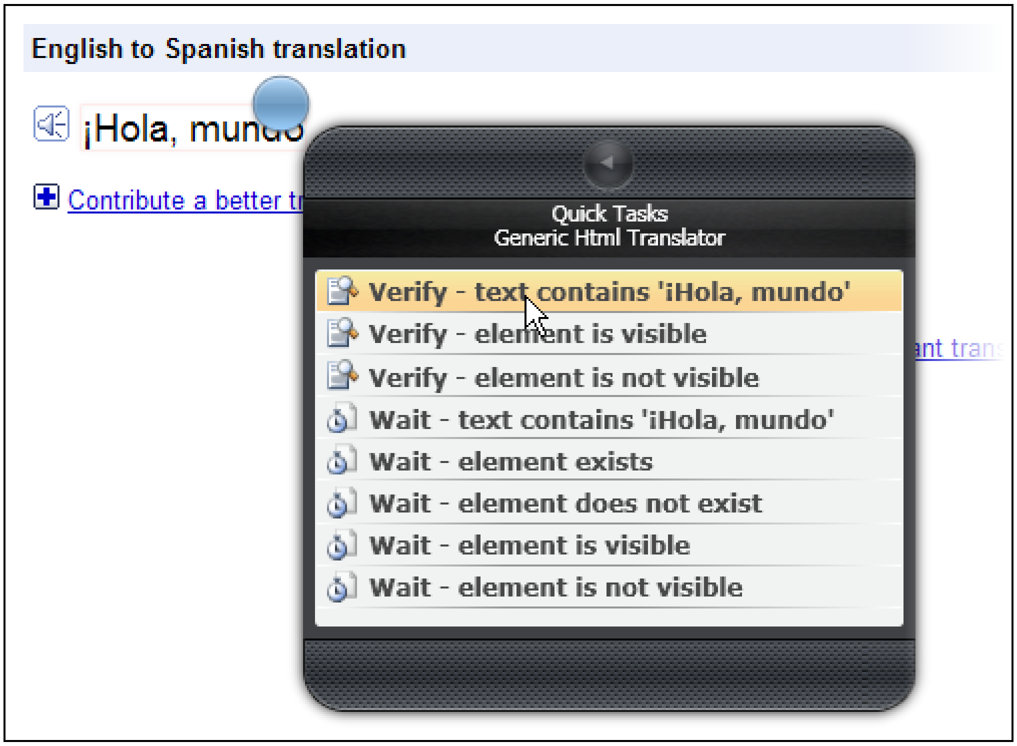


24)Click the "Translate" button.

25)Pass the mouse over the translated text and wait for the Nub to display under the mouse. Click the Nub to display the Elements Menu.



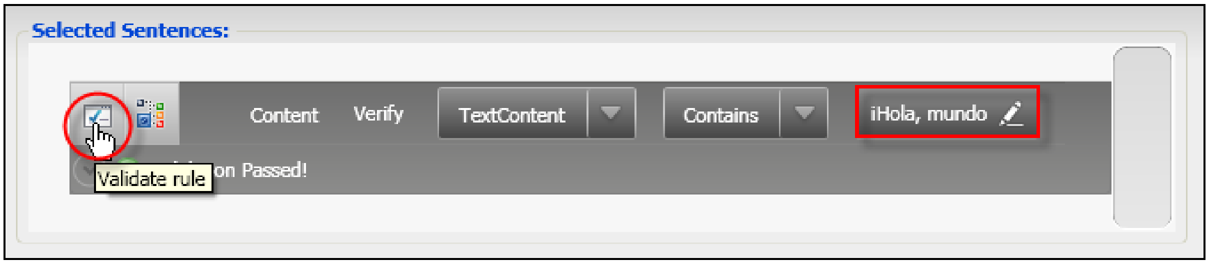
26)Select Quick Tasks from the Elements Menu. Select the "Verify - text contains" task from the list and double click to create the test step and close the dialog.



English to Spanish translation
¡Hola, mundo
Contribute a better tr
Quick Tasks
Generic Html Translator
Verify - text contains 'iHola, mundo'
Verify - element is visible
Verify - element is not visible
Wait - text contains 'iHola, mundo'
Wait - element exists
Wait - element does not exist
Wait - element is visible
Wait - element is not visible

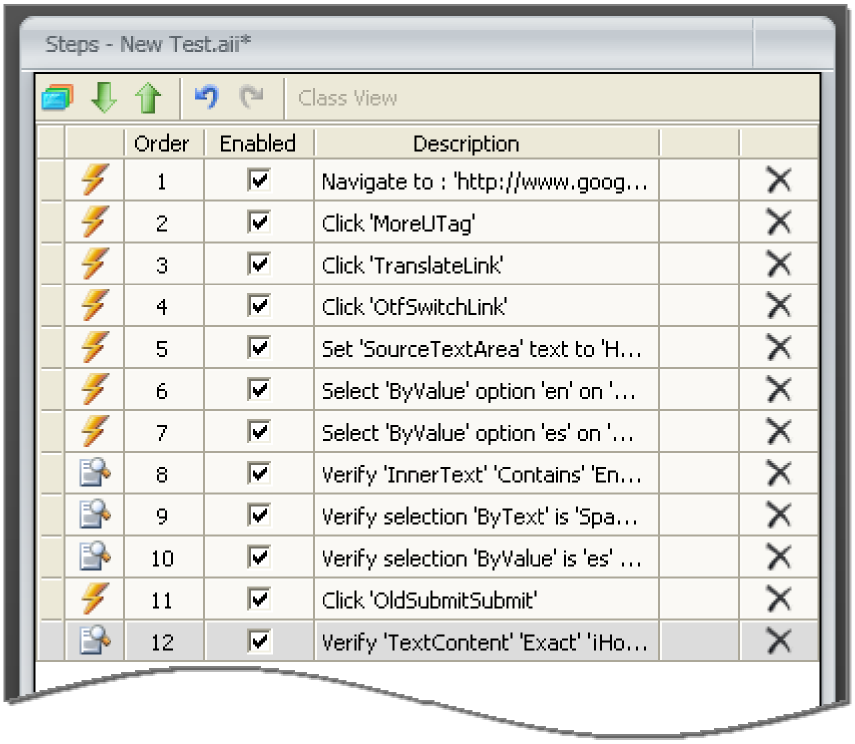
27)In the Steps Pane, double-click the last test step (this should be the "Verify - text contains" step just added from Quick Tasks). Double-click the test step. This will display the Sentence Verification Builder.

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



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

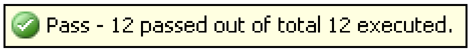
30)The steps in the Steps Pane should now look something like the list in the screenshot below.



	Order	Enabled	Description	
⚡	1	☑	Navigate to : 'http://www.goog...	✕
⚡	2	☑	Click 'MoreUTag'	✕
⚡	3	☑	Click 'TranslateLink'	✕
⚡	4	☑	Click 'OtfSwitchLink'	✕
⚡	5	☑	Set 'SourceTextArea' text to 'H...	✕
⚡	6	☑	Select 'ByValue' option 'en' on '...	✕
⚡	7	☑	Select 'ByValue' option 'es' on '...	✕
📝	8	☑	Verify 'InnerText' 'Contains' 'En...	✕
📝	9	☑	Verify selection 'ByText' is 'Spa...	✕
📝	10	☑	Verify selection 'ByValue' is 'es' ...	✕
⚡	11	☑	Click 'OldSubmitSubmit'	✕
📝	12	☑	Verify 'TextContent' 'Exact' 'iHo...	✕

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

32)The test steps should run through to completion and all test steps should pass:


✓ Pass - 12 passed out of total 12 executed.

6.5 Summary

In this chapter you took a detailed look at how tests are recorded and the available tools used to handle specific web testing tasks. You used the Recording Surface to save actions taken in the browser and to interact with individual web page elements. You saw how these recorded steps are managed using the Steps Pane and how to add steps manually. You saw how "Translators" are used to peer inside controls to glean information about the controls internals. You also used WebUI to handle specialized web testing situations such as drag-and-drop and working with pop-up dialogs. Finally, you learned how to "audition" tests in multiple browsers.

Part VII

Verification

7 Verification

7.1 Objectives

In this chapter you will learn how to test, i.e. assure that certain conditions exist in the browser, using verifications. You will learn how to access verifications from more than one tool in WebUI. You will use the Sentence Verification Builder to interactively build verification rules and validate them against live web documents. You will also explore the structure verification sentences and look at how individual verification types are used. Finally, you will use the 3D Viewer to handle verifications for multiple elements at one time.

7.2 Overview

Automation is only the starting point of testing where the browser is manipulated 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.

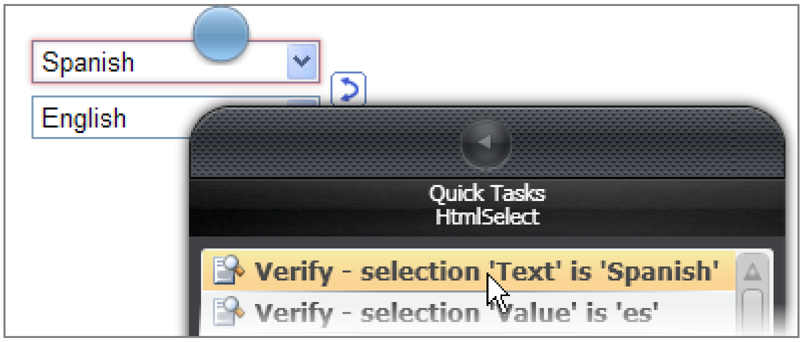
That's where verification comes in. WebUI verification allows you to measure against multiple criteria at one time and to build these measurements in an interactive manner without code. With verifications you detect if elements are in a particular state (e.g. visible, exist) and that attributes and properties compare with specific values. WebUI can verify content, attributes, styles, visibility, drop-down list selections, checkboxes, radio buttons, tables and Silverlight property values. WebUI 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, allow for rich verifications against the exposed portions of the translated control. See the "Translators" Topic for more information.

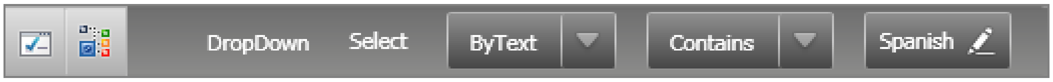
7.3 Verification Access

You can reach sentence verification through a number of avenues:

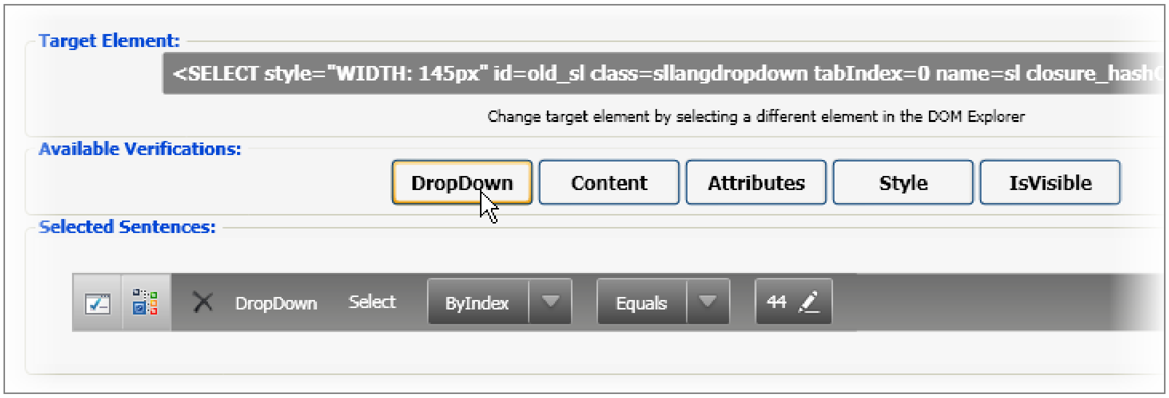
- Through the Elements Menu Quick Tasks button. The screenshot below shows a "Verify" quick task selected for a highlighted drop down list.


Spanish
English
Quick Tasks
HtmlSelect
Verify - selection 'Text' is 'Spanish'
Verify - selection 'Value' is 'es'

  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".



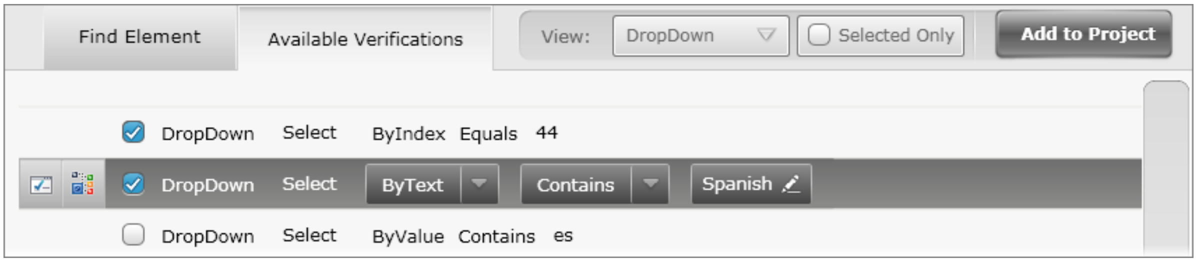
- Through the Elements Menu Sentence Verification Builder you can access all available verification types and create multiple verifications for a single element. The element can be changed on-the-fly by selecting a new target item in the DOM Explorer. The screenshot below shows a "DropDown" type verification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



Target Element: <SELECT style="WIDTH: 145px" id=old_sl class=sllangdropdown tabIndex=0 name=sl closure_hash

Change target element by selecting a different element in the DOM Explorer

- 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.



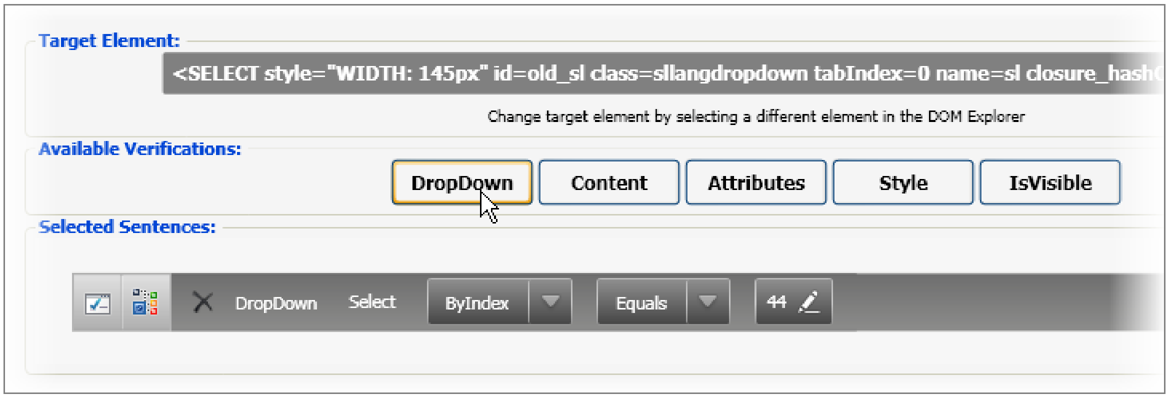
View: DropDown ☐ Selected Only

☑	DropDown	Select	ByIndex	Equals	44
☑	DropDown	Select	ByText	Contains	Spanish
☐	DropDown	Select	ByValue	Contains	es

7.4 Sentence Verification Builder

The Sentence Verification Builder allows you to interactively build verification rules and validate them against a live web document.

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" Topic for more information). Click the buttons to add verification sentences to the **Selected Sentences** area.



Target Element:

<SELECT style="WIDTH: 145px" id=old_sl class=sllangdropdown tabIndex=0 name=sl closure_hash(

Change target element by selecting a different element in the DOM Explorer

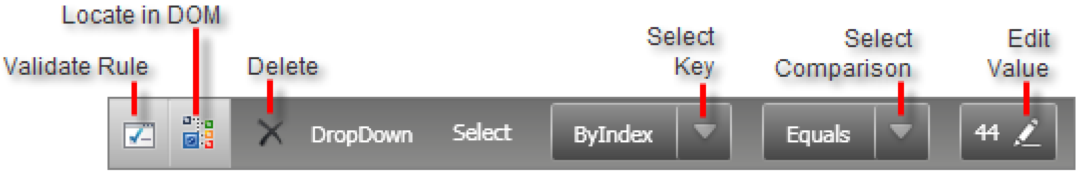
Available Verifications:

Selected Sentences:

DropDown Select ByIndex Equals 44

The **Selected Sentences** is used to build the individual verification rules. You can validate the rule or 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.

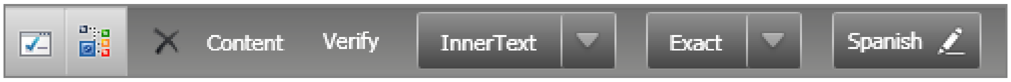
Once you have added a sentence using one of the Available Verifications buttons, use the drop down lists to select a key and comparison, then use the Edit "Pencil" icon to open the value for editing. After editing the value, click the Edit button a second time to close it. Then click the Validate Rule 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 Test Pane or **Cancel** to abandon your edits.

7.5 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.

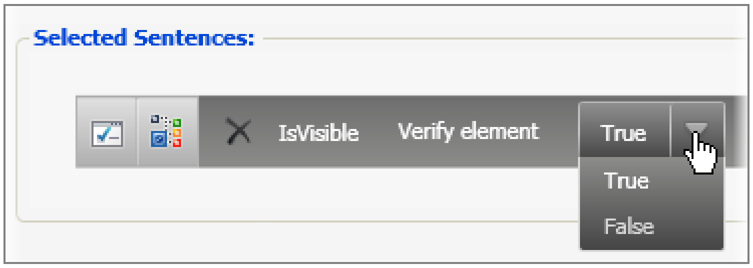
Verification Type	Structure
Content	Content Type-Compare-Value
Attribute	Attribute Name-Compare-Value
IsVisible	Value
Style	Inline/Computed-Category-Attribute-Compare-Value
DropDown	Attribute Name-Compare-Value

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.

7.6 Verification Types

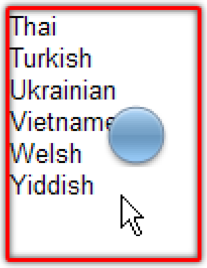
IsVisible

The most basic verifications you can perform are against an element's visibility. WebUI determines visibility by following the CSS (Cascading Style Sheets) 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.



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?



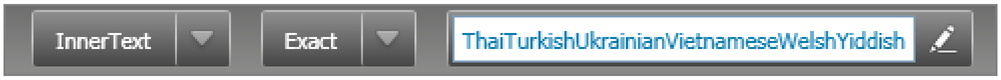
Thai
Turkish
Ukrainian
Vietname
Welsh
Yiddish

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>
```

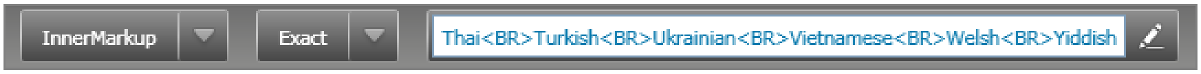
Content Type

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.



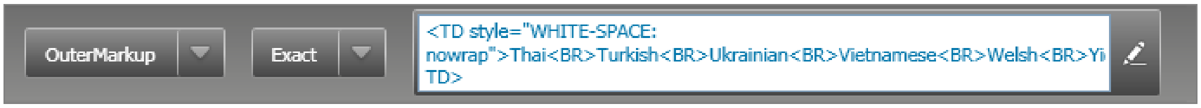
InnerText | Exact | ThaiTurkishUkrainianVietnameseWelshYiddish

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



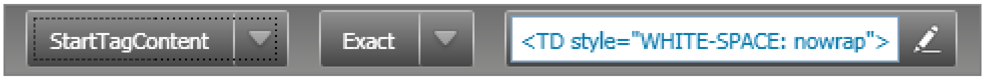
InnerMarkup | Exact | Thai<BR>Turkish<BR>Ukrainian
Vietnamese<BR>Welsh<BR>Yiddish

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



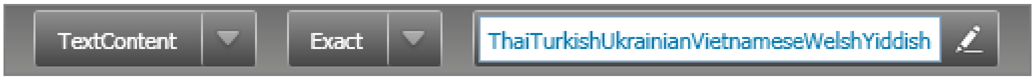
OuterMarkup | Exact | <TD style="WHITE-SPACE: nowrap">Thai<BR>Turkish<BR>Ukrainian
Vietnamese<BR>Welsh<BR>Yi TD>

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



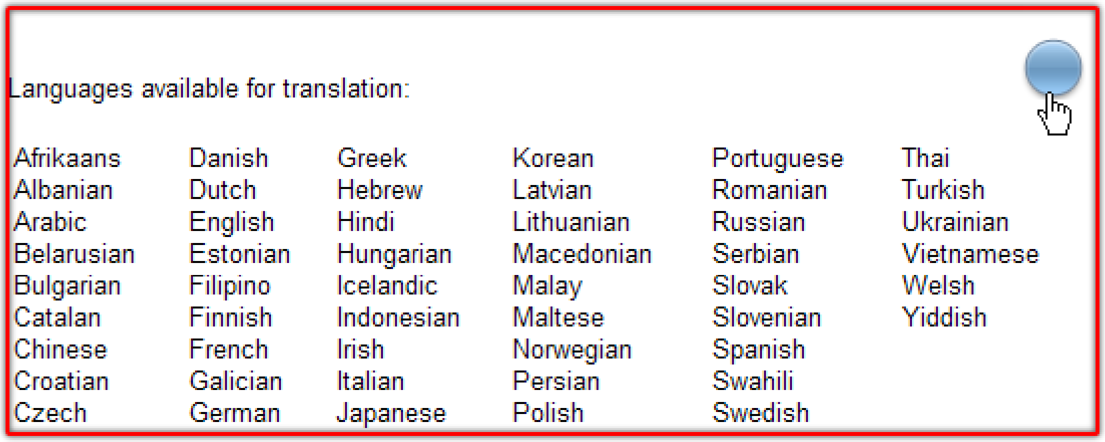
StartTagContent | Exact | <TD style="WHITE-SPACE: nowrap">

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



TextContent | Exact | ThaiTurkishUkrainianVietnameseWelshYiddish

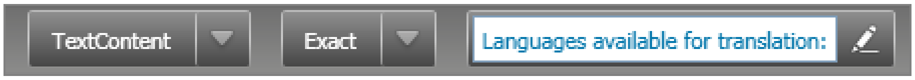
TextContent only looks at the content of the immediate element while InnerText is "recursive" and looks at all the text content in elements contained by the current element. Let's look at the HTML table that contains the "<td>" cell we've been working with:



Languages available for translation:

Afrikaans
Albanian
Arabic
Belarusian
Bulgarian
Catalan
Chinese
Croatian
Czech
Danish
Dutch
English
Estonian
Filipino
Finnish
French
Galician
German
Greek
Hebrew
Hindi
Hungarian
Icelandic
Indonesian
Irish
Italian
Japanese
Korean
Latvian
Lithuanian
Macedonian
Malay
Maltese
Norwegian
Persian
Polish
Portuguese
Romanian
Russian
Serbian
Slovak
Slovenian
Spanish
Swahili
Swedish
Thai
Turkish
Ukrainian
Vietnamese
Welsh
Yiddish

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.

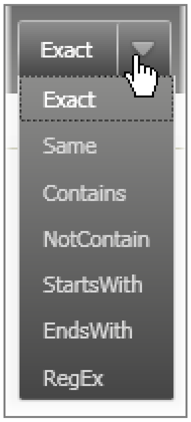


TextContent ▾ Exact ▾ Languages available for translation:

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

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

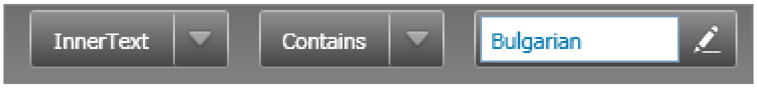
Comparison



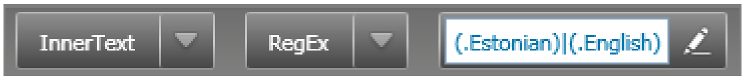
Content is equated against a value using a comparison. The possible comparisons depend on the content type. For strings of alpha numeric characters, the possible comparisons are shown in this screenshot.

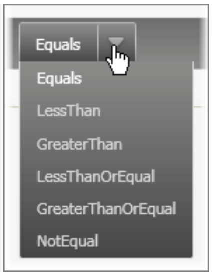
Exact performs a strict match against the value and is case sensitive. **Same** also performs a precise match but the value can be upper or lower case. Consider the previous table example where the content is *"Languages available for translation:".* If we change the content to "languages available for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





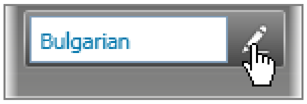
Numeric comparisons have their own set of operators as shown in this screenshot.

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!

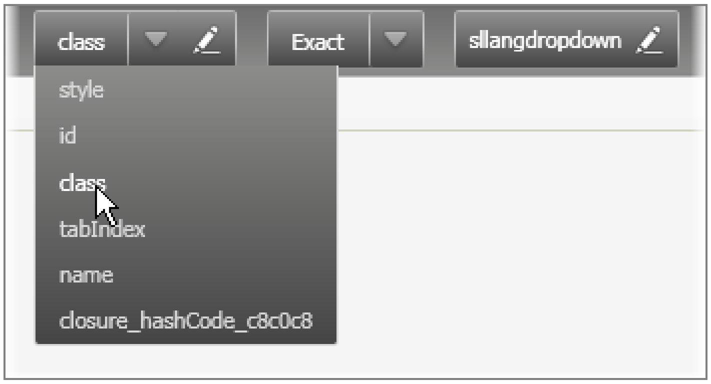
After editing a value, be sure to close the edit box.



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.

Attribute

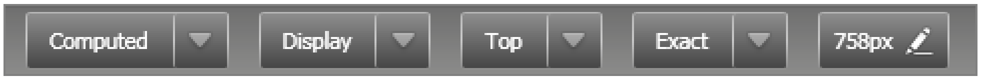
The Attribute verification allows a great deal of flexibility against any attribute in an element. The attribute name drop down lists all the attributes in the element. When you select an attribute, the Value will automatically populate with appropriate matching text.



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

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 entries list values for the comparison and allow entry for a 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".

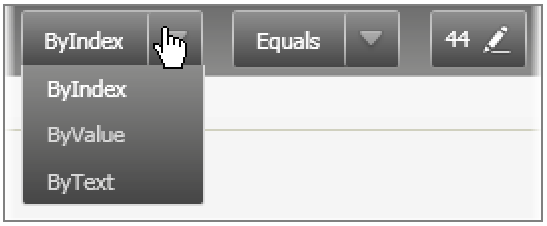


Drop down values were populated automatically against the HTML element shown below.

```
<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; " />
```

DropDown

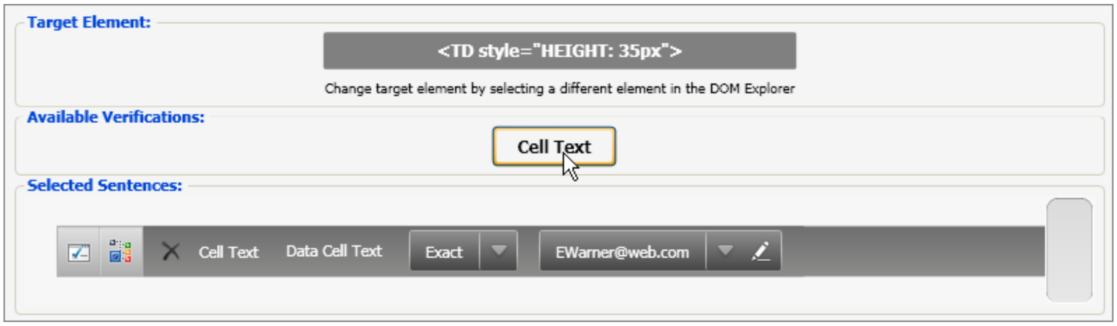
The DropDown verification has built-in attribute types "ByIndex", "ByValue" and "ByText".



AJAX and Silverlight

AJAX and Silverlight web applications up to now have been difficult to test against, but WebUI 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.

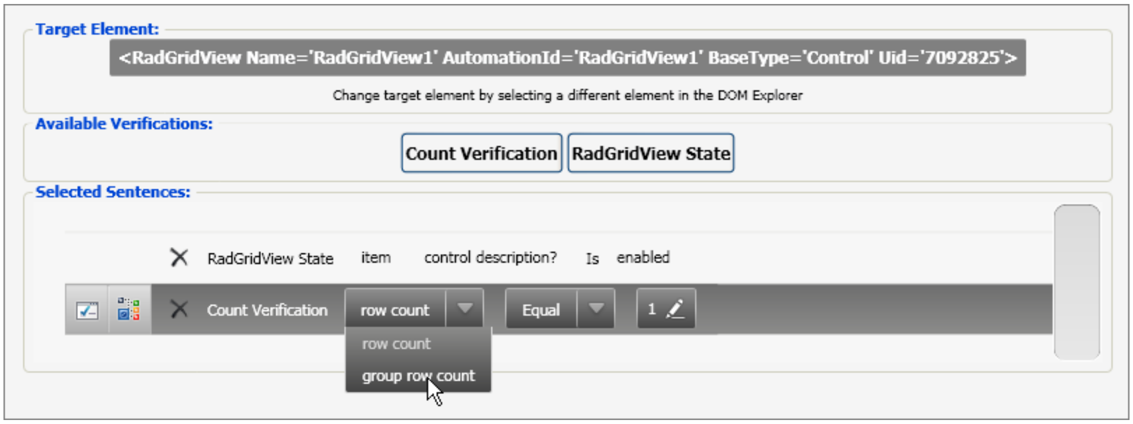
In later chapters when we test AJAX and Silverlight controls and work with "Translators", i.e. extensions that open up RadControls for more complete examination, the number of verifications expands. For example, the screenshot below shows a verification against an ASP.NET AJAX grid. Using the built-in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



Target Element:
<TD style="HEIGHT: 35px">
Change target element by selecting a different element in the DOM Explorer
Available Verifications:

Selected Sentences:
Cell Text Data Cell Text Exact EWarner@web.com

WebUI 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.



Target Element:
<RadGridView Name='RadGridView1' AutomationId='RadGridView1' BaseType='Control' Uid='7092825'>
Change target element by selecting a different element in the DOM Explorer
Available Verifications:

Selected Sentences:
RadGridView State item control description? Is enabled
Count Verification row count Equal 1

7.7 Create Verifications Walk Through

This walk through will demonstrate creating multiple verifications against a single element using the Sentence Verification Builder.

Find the materials for this Topic at...

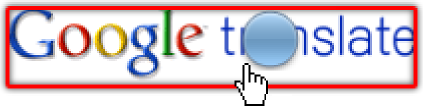
\Projects\SentenceVerficationBuilder

1) From the Welcome Screen, Click the **Record New Test** button. This step will navigate to the Recording Surface, with recording turned on.

2) In the Recording Surface, enter "translate.google.com" to the browser address bar and then press the **ENTER** key.. 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.

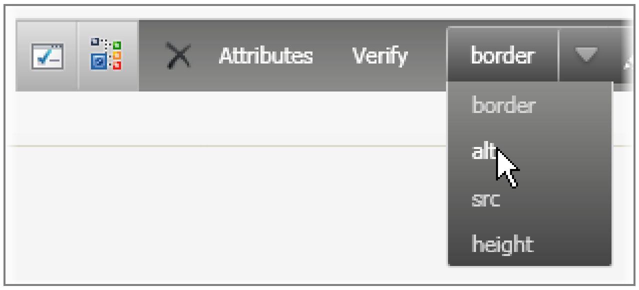


5) Click the Nub to display the Elements Menu.

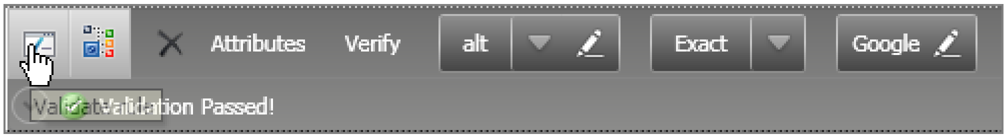
6) Click the Build Verification icon

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.



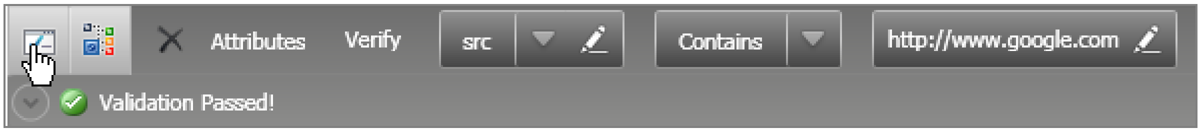
9) Click the Validate Rule button. The message should read "Validation Passed".



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 "StartsWith" and the value to "http://www.google.com". Don't forget to click the edit "pencil" button a second time to save your edits. Click the Validate Rule button. The message should read "Validation Passed".



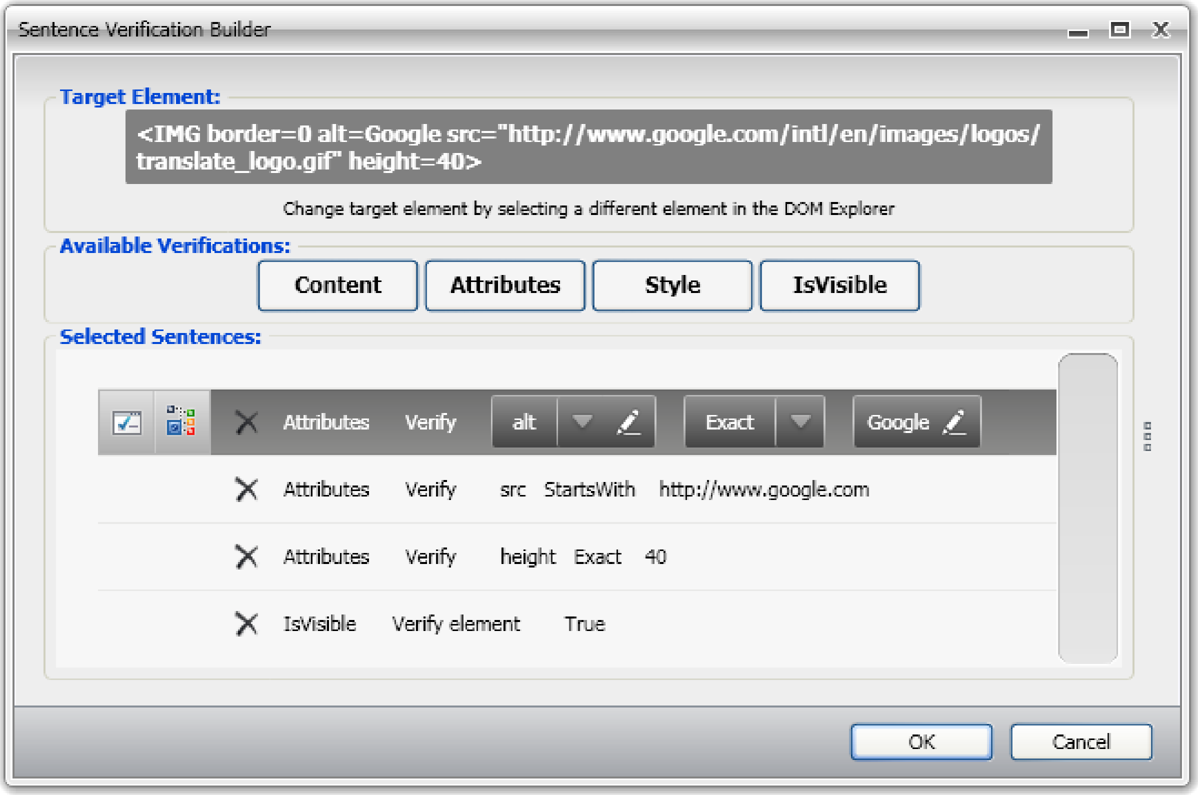
Attributes Verify src Contains http://www.google.com

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:



Sentence Verification Builder

Target Element:

<IMG border=0 alt=Google src="http://www.google.com/intl/en/images/logos/translate_logo.gif" height=40>

Change target element by selecting a different element in the DOM Explorer

Available Verifications:

Selected Sentences:

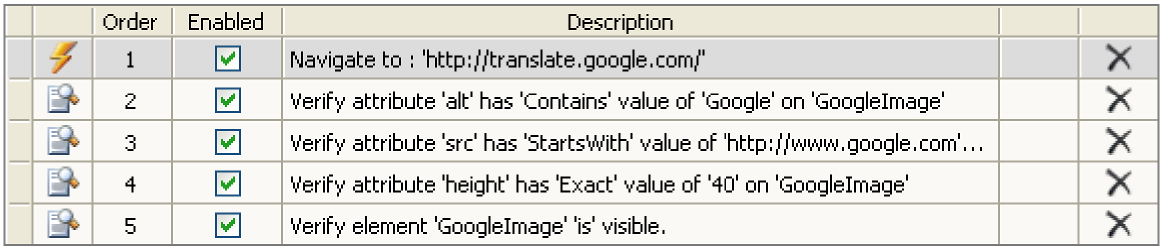
Attributes Verify alt Exact Google

Attributes Verify src StartsWith http://www.google.com

Attributes Verify height Exact 40

IsVisible Verify element True

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

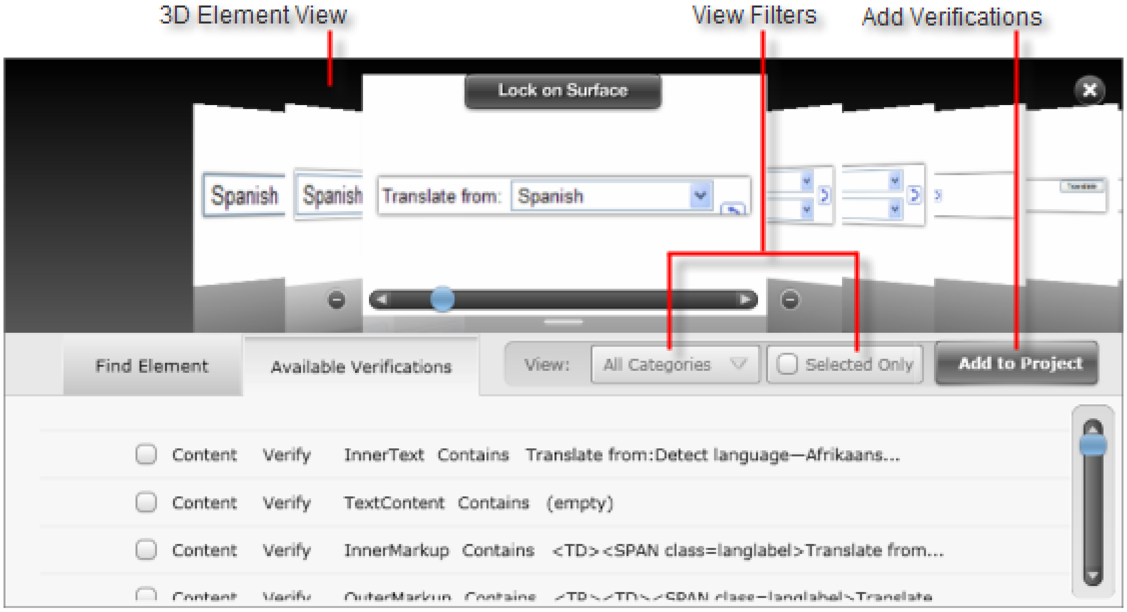


		Order	Enabled	Description		
	⚡	1	☑	Navigate to : 'http://translate.google.com/'		✕
		2	☑	Verify attribute 'alt' has 'Contains' value of 'Google' on 'GoogleImage'		✕
		3	☑	Verify attribute 'src' has 'StartsWith' value of 'http://www.google.com'...		✕
		4	☑	Verify attribute 'height' has 'Exact' value of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

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

7.8 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.



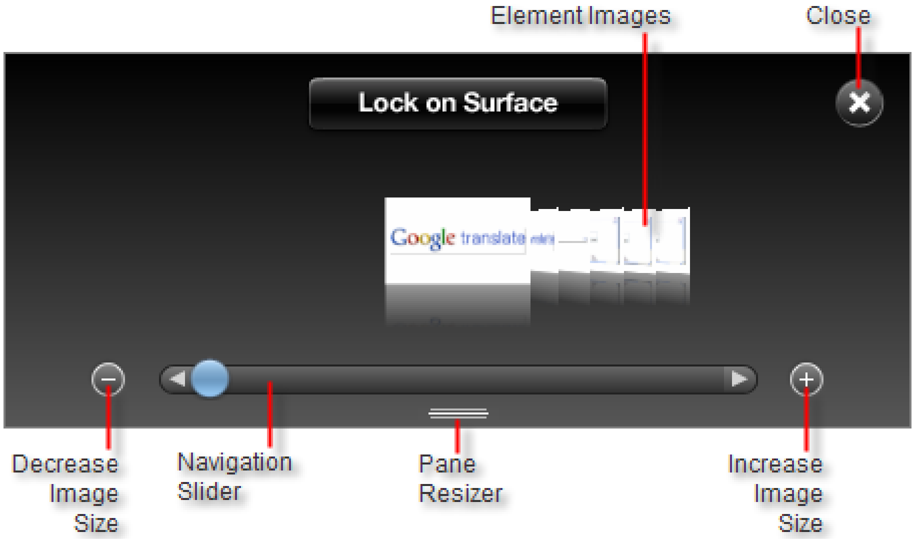
□ Content Verify InnerText Contains Translate from:Detect language—Afrikaans...

□ Content Verify TextContent Contains (empty)

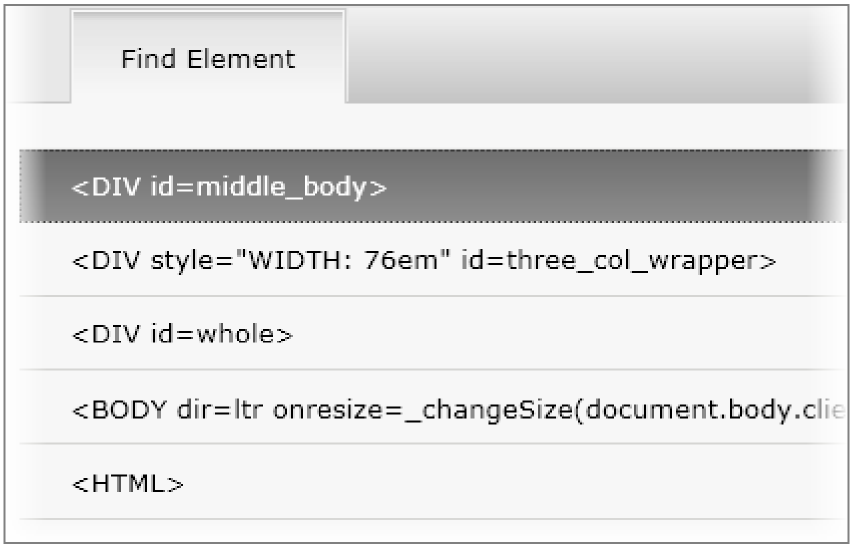
□ Content Verify InnerMarkup Contains <TD><SPAN class=langlabel>Translate from...

□ Content Verify OuterMarkup Contains <TD><TD><SPAN class=langlabel>Translate

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.



<DIV id=middle_body>

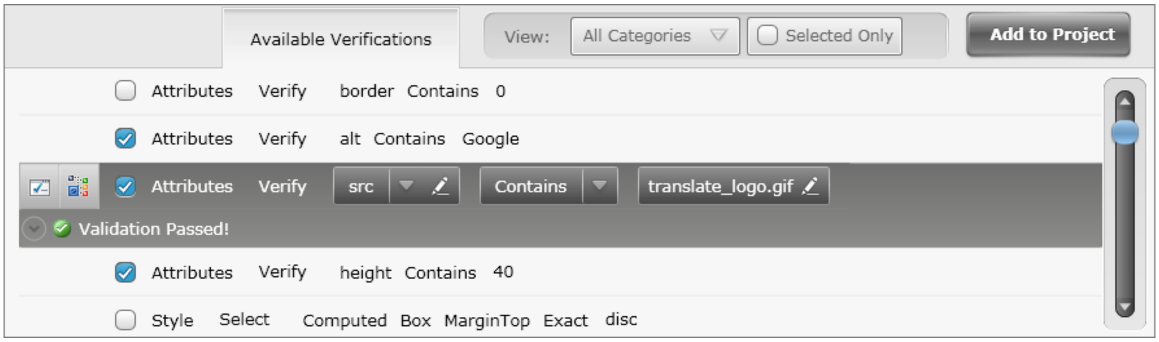
<DIV style="WIDTH: 76em" id=three_col_wrapper>

<DIV id=whole>

<BODY dir=ltr onresize=_changeSize(document.body.clie

<HTML>

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 for each verification sentence.



☐ Attributes Verify border Contains 0

☑ Attributes Verify alt Contains Google

☑ Attributes Verify src Contains translate_logo.gif

Validation Passed!

☑ Attributes Verify height Contains 40

☐ Style Select Computed Box MarginTop Exact disc

7.9 3D Viewer Walk Through

This walk through demonstrates using the 3D Viewer to create multiple verifications.

Find the materials for this Topic at...

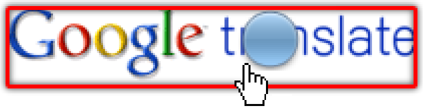
\Projects\3DViewer

1) From the Welcome Screen, Click the **Record New Test** button. This step will navigate to the Recording Surface, with recording turned on.

2) In the Recording Surface, enter "translate.google.com", then press the **ENTER** key. This will load the "Google" translation web page.

3) Press the Highlighting button to enable it.

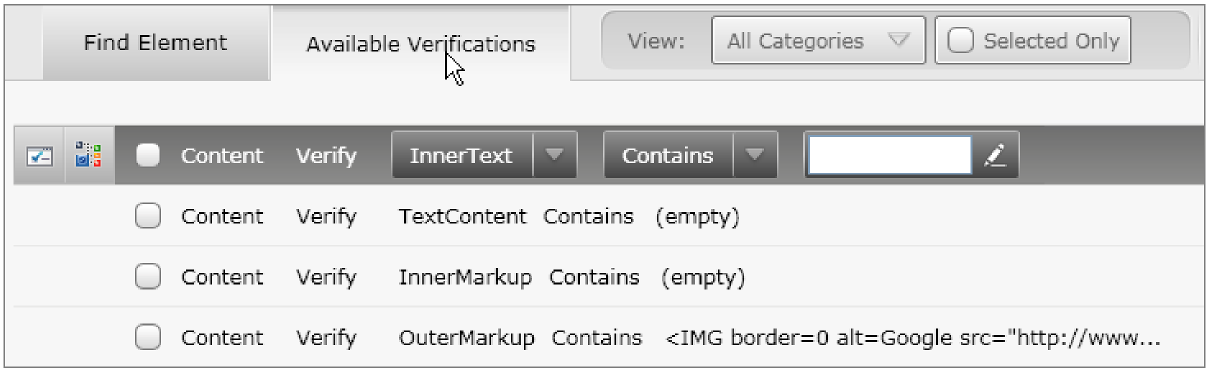
4) Move the mouse over the "Google Translate" image to highlight it. Wait for the Nub to appear.



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.

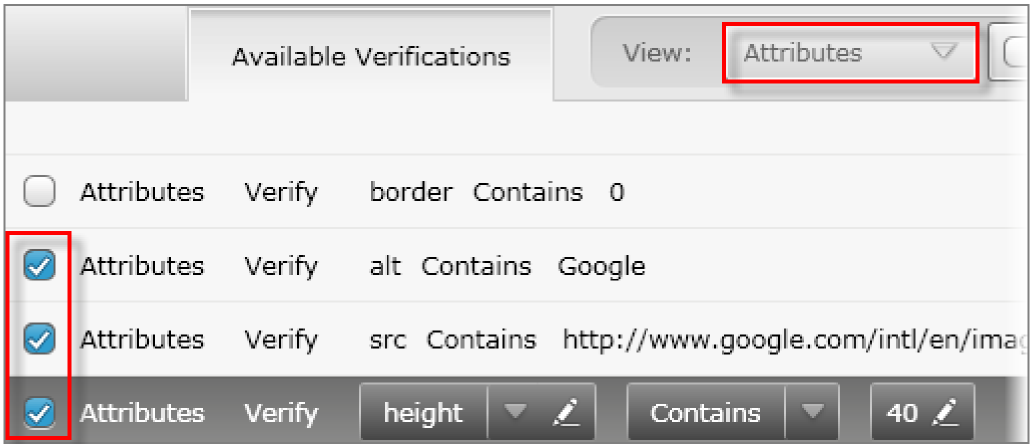


☐	Content	Verify	InnerText	Contains	
☐	Content	Verify	TextContent	Contains	(empty)
☐	Content	Verify	InnerMarkup	Contains	(empty)
☐	Content	Verify	OuterMarkup	Contains	<IMG border=0 alt=Google src="http://www...

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".



Available Verifications

View: Attributes

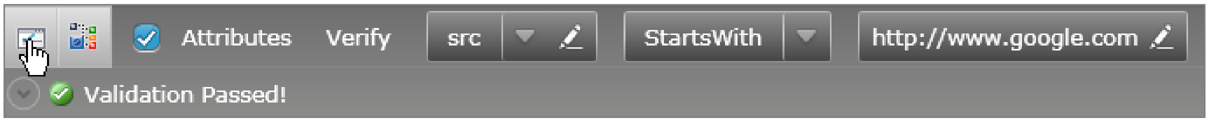
□ Attributes Verify border Contains 0

☑ Attributes Verify alt Contains Google

☑ Attributes Verify src Contains http://www.google.com/intl/en/imag

☑ Attributes Verify height Contains 40

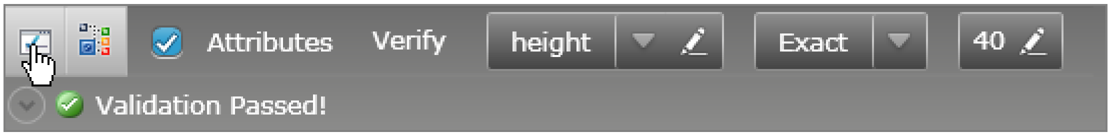
10) Select the "src Contains http://www.google.com..." sentence. Change the comparison to "StartsWith" and the value to "http://www.google.com". Click the Validate Rule button. A message should display below the sentence that reads "Validation Passed".



☑ Attributes Verify src StartsWith http://www.google.com

Validation Passed!

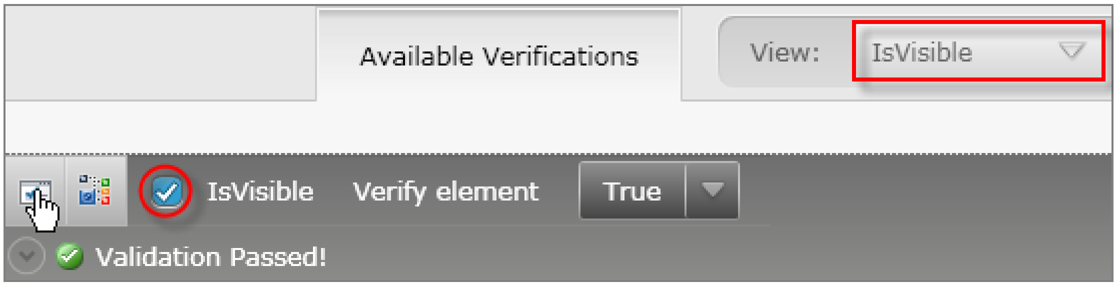
11) Select the "height Contains 40" sentence. Change the comparison to "Exact". Click the Validate Rule button. A message should display below the sentence that reads "Validation Passed".



☑ Attributes Verify height Exact 40

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".



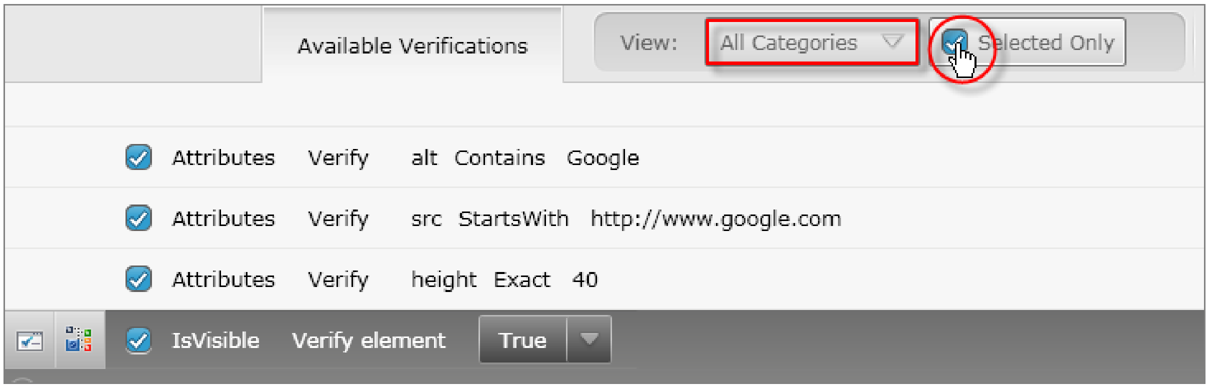
Available Verifications

View: IsVisible

☑ IsVisible Verify element True

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.



Available Verifications

View: All Categories ☑ Selected Only

☑ Attributes Verify alt Contains Google

☑ Attributes Verify src StartsWith http://www.google.com

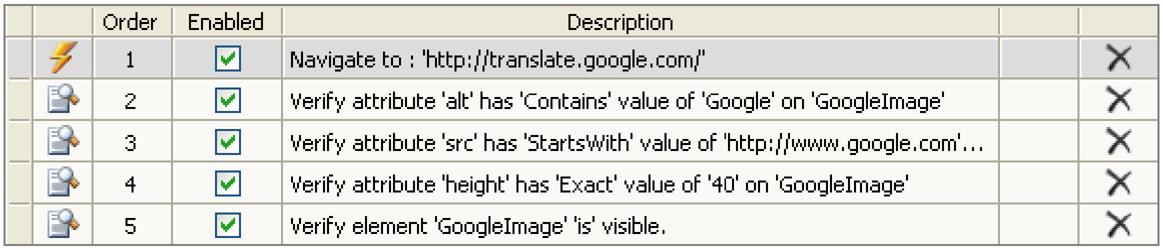
☑ Attributes Verify height Exact 40

☑ IsVisible Verify element True

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

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

16)The test steps 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.

7.10 Summary

In this chapter you learned how to test, i.e. assure that certain conditions exist in the browser, using verifications. You learned how to access verifications from more than one tool in WebUI. You used the Sentence Verification Builder to interactively build verification rules and validate them against live web documents. You also explored the structure verification sentences and looked at how individual verification types are used. Finally, you used the 3D Viewer to handle verifications for multiple elements at one time.

Part VIII

Test Organization

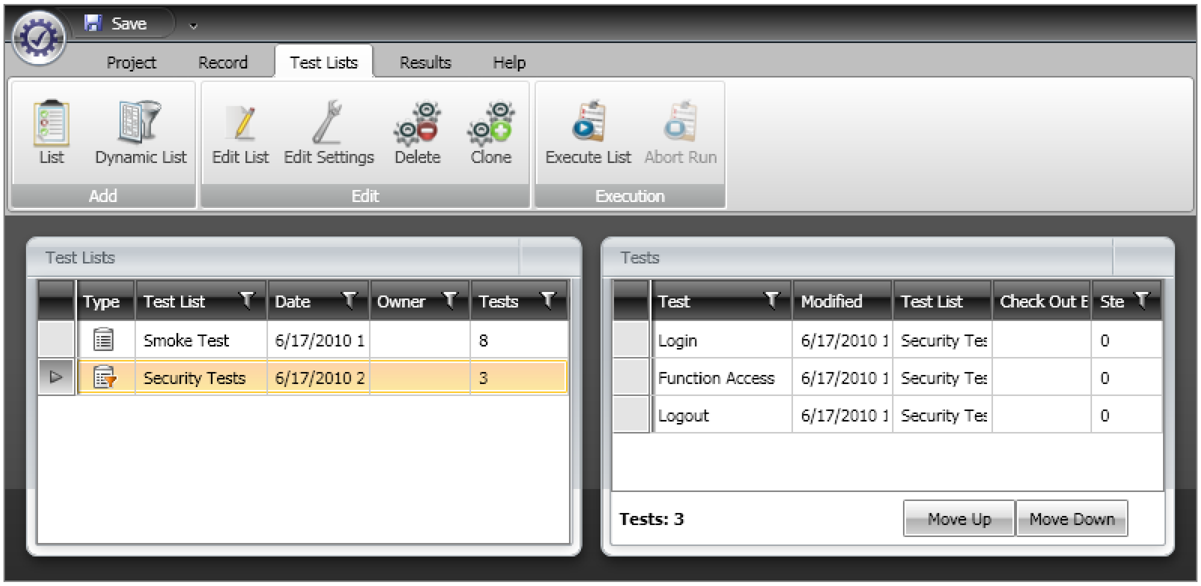
8 Test Organization

8.1 Objectives

In this chapter you will learn how Test Lists help get the best use of recorded tests by allowing reuse in more than one configuration. You will learn to build static test lists from existing tests and to manually set the order that the tests run. You will also learn how dynamic test lists automatically select tests from your project at the time of execution using rules about properties of the individual tests. You will see how to configure settings for all the tests within a test list.

8.2 Overview

Test Lists help you get the best use of recorded tests by allowing you to reuse tests in more than one configuration. For example, you may want only a small selection of tests to use as a "Smoke Test". You could also collect tests that a particular QA engineer is responsible for, or list all the tests for a category of functionality. You can use the flexibility of **Test Lists** to best suit your organization's goals.



	Type	Test List	Date	Owner	Tests
		Smoke Test	6/17/2010 1		8
▷		Security Tests	6/17/2010 2		3

	Test	Modified	Test List	Check Out E	Ste
	Login	6/17/2010 1	Security Tes		0
	Function Access	6/17/2010 1	Security Tes		0
	Logout	6/17/2010 1	Security Tes		0

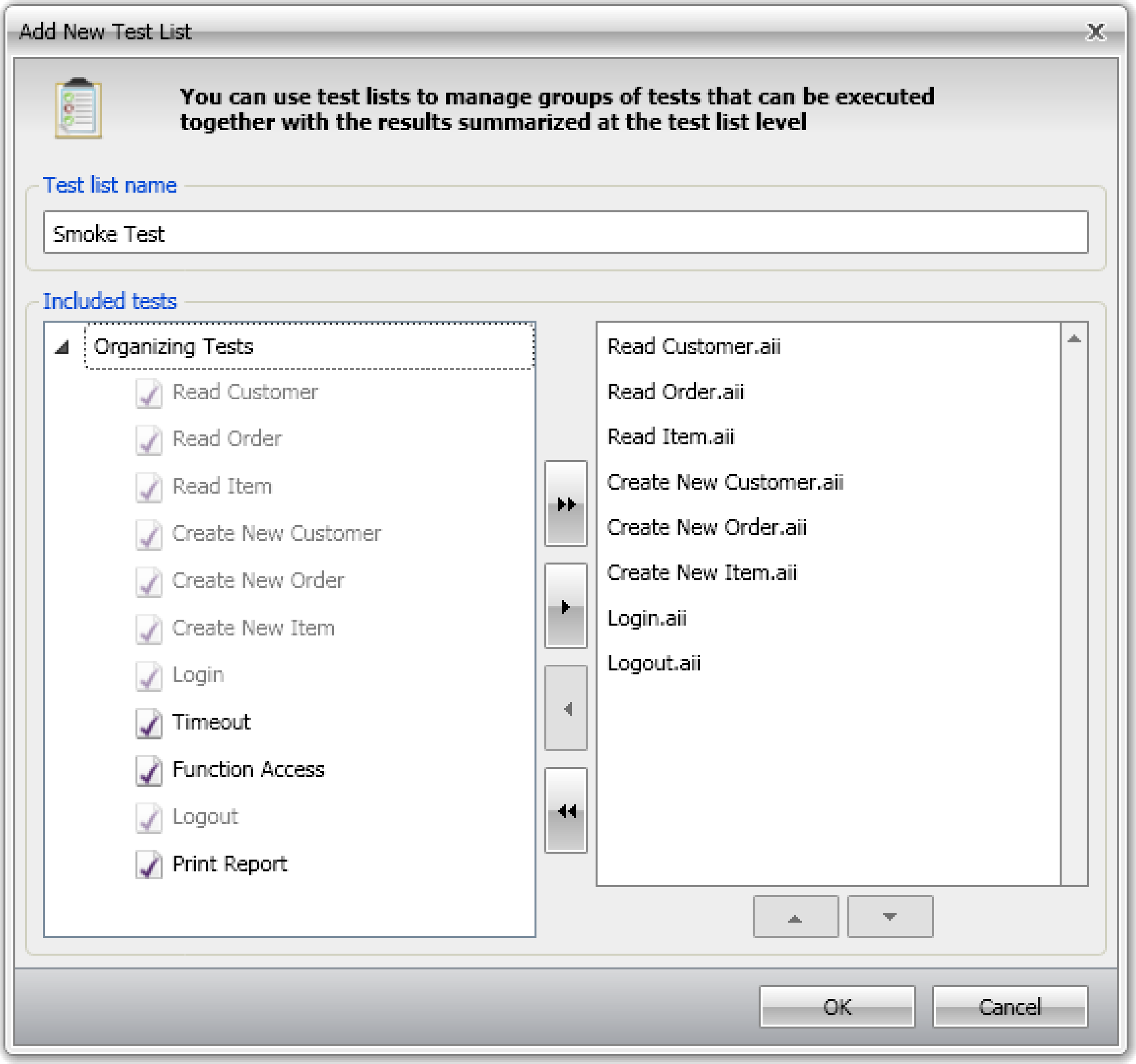
Tests: 3

8.3 Creating New Test Lists

Static Lists

You can manually build Static Lists from existing tests and manually set the order that the tests should run. When the Test List executes, each test runs in order, exactly as you have it configured.

To create a static test list, click the **List** button to display the **Add New Test List** dialog. Enter a **Test List Name**. Move items from the **Included Tests** list on the left hand of the dialog to the list on the right hand side. To move items from one list to another you can double click an item or you can select items and use the arrow buttons. The double-arrow buttons move all items from one list to another. The screenshot below shows a new Test List that contains only very basic tests that are appropriate for a smoke test.



Add New Test List

You can use test lists to manage groups of tests that can be executed together with the results summarized at the test list level

Test list name

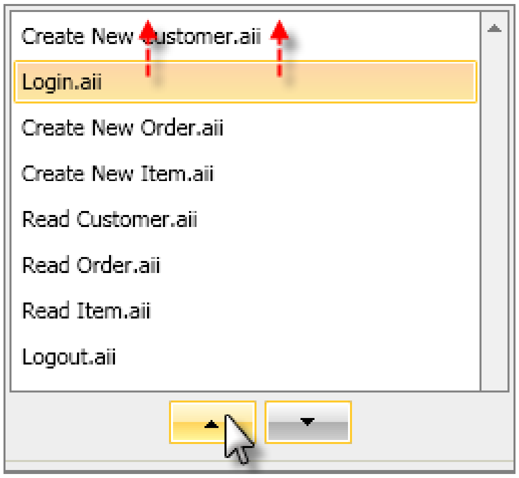
Smoke Test

Included tests

- Organizing Tests
  - ☑ Read Customer
  - ☑ Read Order
  - ☑ Read Item
  - ☑ Create New Customer
  - ☑ Create New Order
  - ☑ Create New Item
  - ☑ Login
  - ☑ Timeout
  - ☑ Function Access
  - ☑ Logout
  - ☑ Print Report

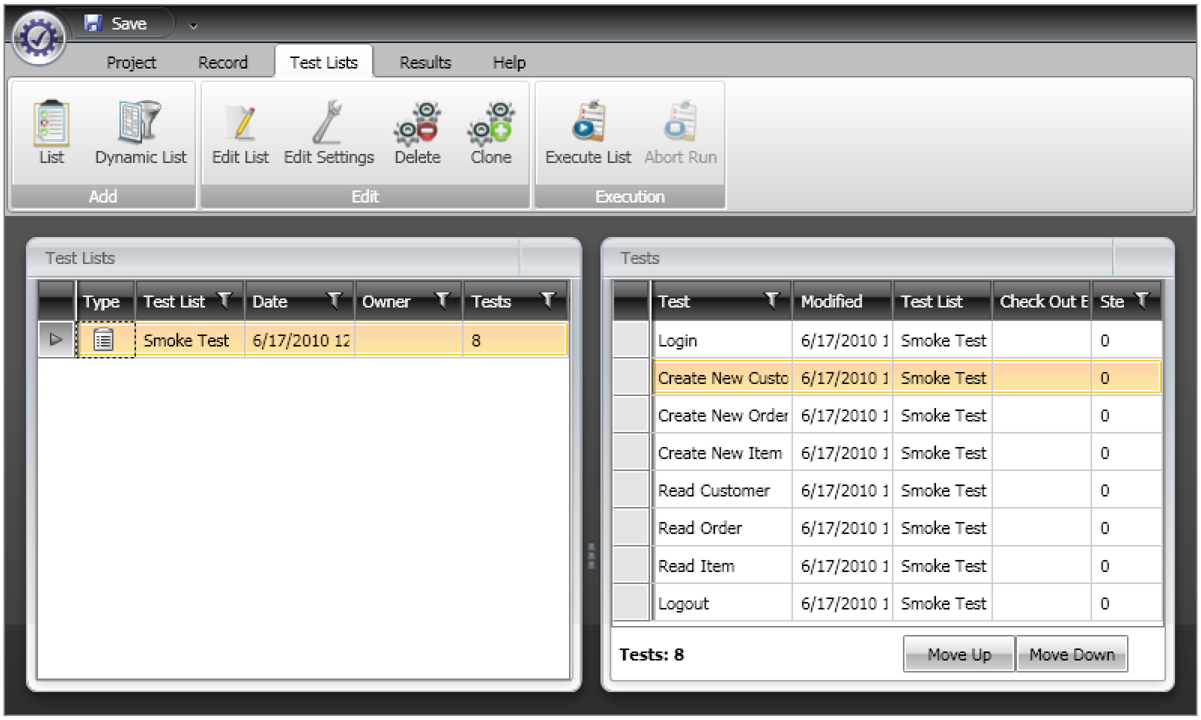
Read Customer.aii
Read Order.aii
Read Item.aii
Create New Customer.aii
Create New Order.aii
Create New Item.aii
Login.aii
Logout.aii

Items in the right-hand list can be reordered using the buttons on the lower right-hand side of the dialog. In the Screen below, the "Login" test is moved up to first in the list.



Create New Customer.aii
Login.aii
Create New Order.aii
Create New Item.aii
Read Customer.aii
Read Order.aii
Read Item.aii
Logout.aii

Click the **OK** button to create the Test List. The screenshot below shows the new "Smoke Test" in the left hand Test Lists pane and the individual tests in the right hand Tests pane.



Test Lists

Type	Test List	Date	Owner	Tests
▤	Smoke Test	6/17/2010 12		8

Tests

Test	Modified	Test List	Check Out E	Ste
Login	6/17/2010 1	Smoke Test		0
Create New Custo	6/17/2010 1	Smoke Test		0
Create New Order	6/17/2010 1	Smoke Test		0
Create New Item	6/17/2010 1	Smoke Test		0
Read Customer	6/17/2010 1	Smoke Test		0
Read Order	6/17/2010 1	Smoke Test		0
Read Item	6/17/2010 1	Smoke Test		0
Logout	6/17/2010 1	Smoke Test		0

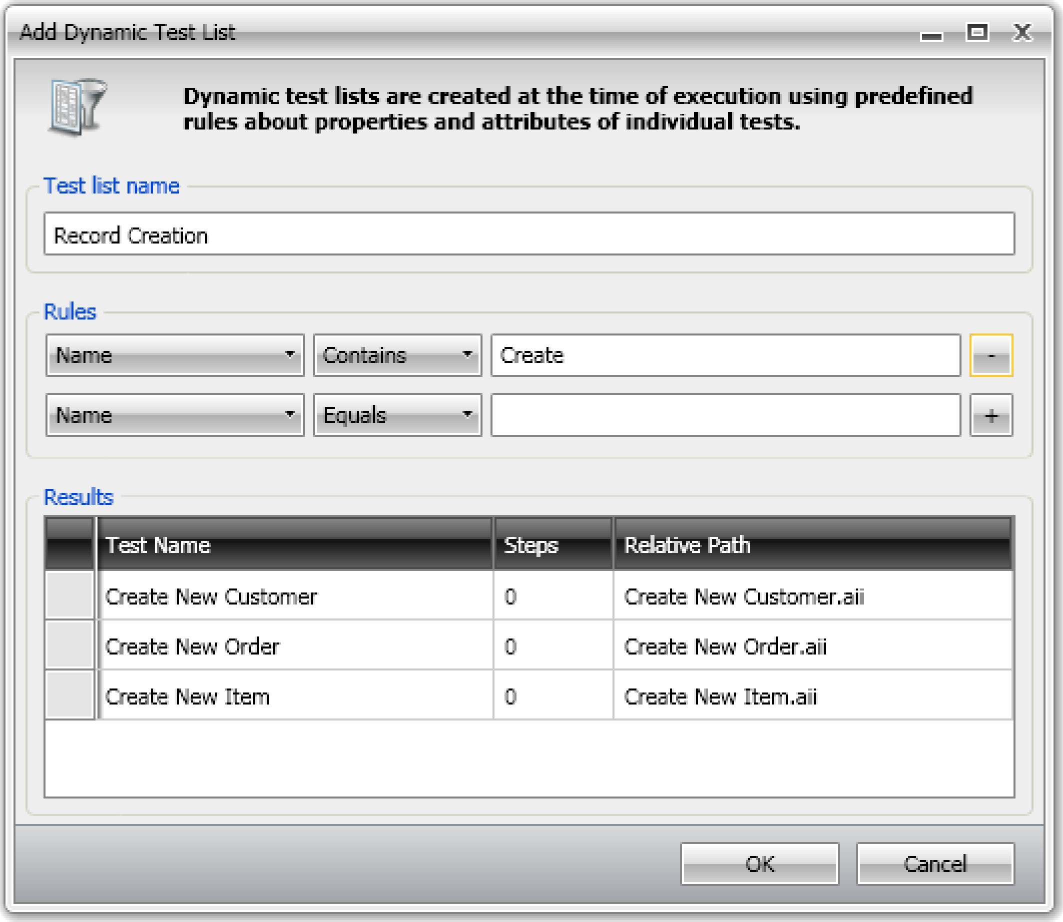
Tests: 8

Dynamic Lists

Dynamic Test Lists automatically select tests from your project at the time of execution using rules about properties of the individual tests. To create a dynamic test list, click the **Dynamic List** button to display the **Add Dynamic Test List** dialog. Enter a **Test List Name**. Instead of selecting tests by hand, define one or more **Rules**. Each rule is made of three parts, a property, a comparison operation and a value:

- In the first drop down list on the left of the Rules section, select a test property to compare against. Test properties are defined in the Properties Pane of the Project Tab. In the screenshot below, the "Name" of the test is selected.
- Select a compare operation from the middle drop down list. This list will vary depending on the type of property you are comparing against. For example, if you select the "Priority" property, you will only see numeric comparisons, e.g. "Greater than", "Equals", etc., in the drop down list. In the screenshot below, the "Contains" operator is selected.
- Add a value in the text box to the right of the comparison. In the screenshot below, the value is set to "Create".

Click the "+" button to add the rule and see the filtered list of tests. The screenshot below shows tests where the test "Name" property contains "Create".



Dynamic test lists are created at the time of execution using predefined rules about properties and attributes of individual tests.

Record Creation

Name | Contains | Create | -

Name | Equals | | +

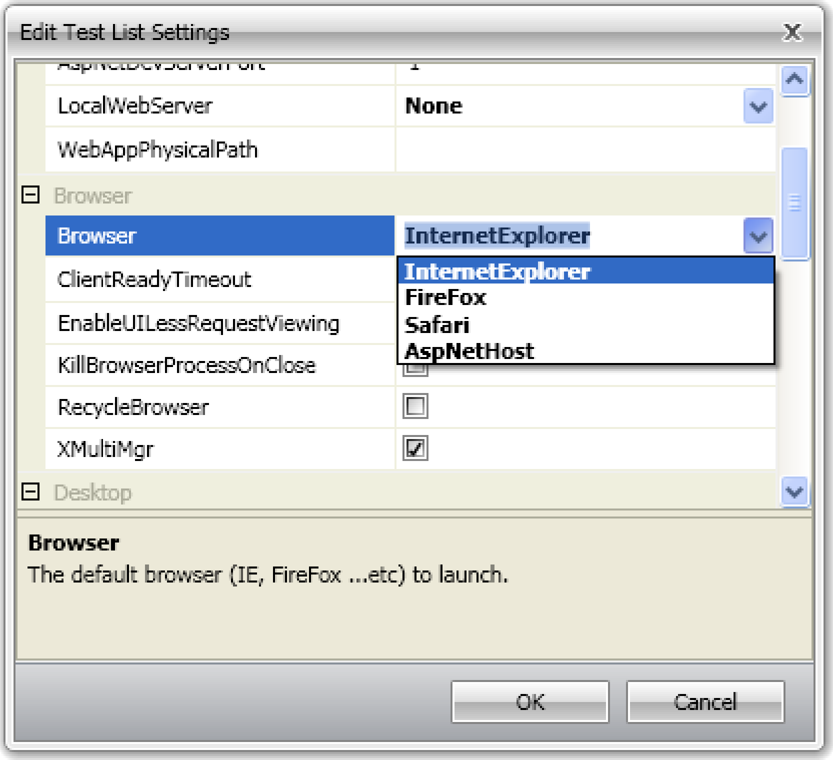
Test Name	Steps	Relative Path
Create New Customer	0	Create New Customer.aii
Create New Order	0	Create New Order.aii
Create New Item	0	Create New Item.aii

Tip!

Each test has several properties called "CustomProperty1", "CustomProperty2" and "CustomProperty3". You can place whatever you like in these properties and use them in your rules. For example, you could set CustomProperty1 to "Security" on all tests that impact security testing, such as the "Login", "Logout" and "Function Access". The rule "CustomProperty1 Equals 'Security'" would return these three tests, but the real benefit would come into play when you add a new security test, e.g. "invalid password", and set its "CustomProperty1" to "Security". The new test would automatically be picked up during the next dynamic test list execution.

8.4 Editing Test List Settings

The **Edit Settings** button displays the **Edit Test List Settings** dialog. This dialog allows you to configure all tests in the list at one time.



LocalWebServer None

WebAppPhysicalPath

⊟ Browser

Browser InternetExplorer

ClientReadyTimeout

EnableUILessRequestViewing

KillBrowserProcessOnClose

RecycleBrowser □

XMultiMgr ☑

⊟ Desktop

Browser
The default browser (IE, FireFox ...etc) to launch.

The properties in the dialog are divided into several categories that are described briefly here:

- **Annotation**: These properties control if elements in the test are highlighted and what annotations are shown. The test list run can display custom annotations added by the test author, native WebAii framework actions or both.

- **ASP.NET**: The server, path and port used to serve pages that will be tested.

- **Browser**: Configuration details of the browser process, including the default browser to use and timeout settings.

- **Desktop**: Controls the simulated mouse movement speed.

- **Execution**: Delay, timeout and wait times.

- **HttpProxy**: Whether to use the built-in http proxy during automation.

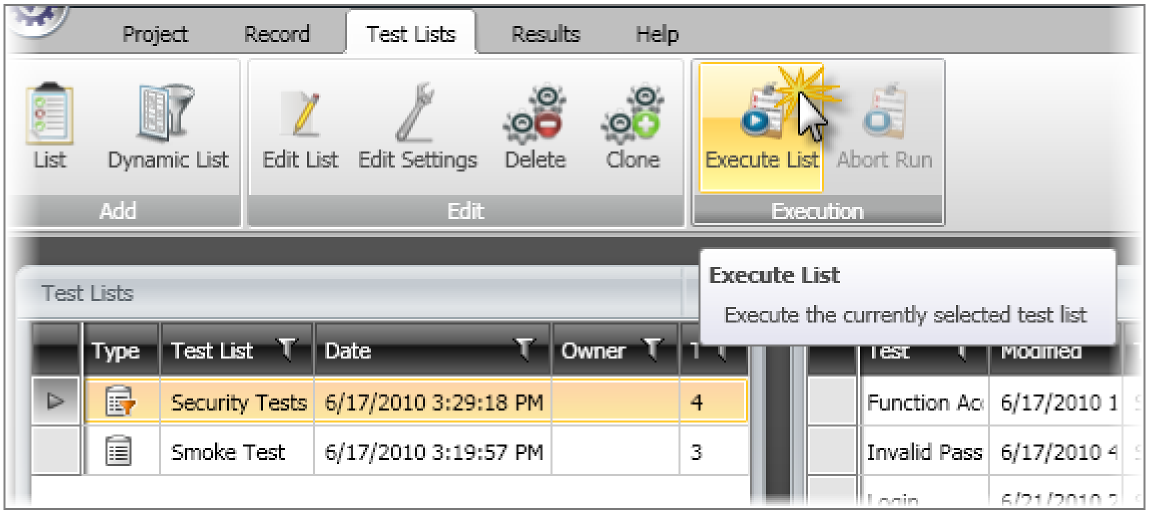
- **Log**: Controls the existence, content, location and other behaviors of log files.

- **Navigation**: The base URL used for any Navigate commands.

- **Silverlight**: Enables Silverlight automation and configures the timeout interval.

8.5 Executing Test Lists

To execute a list, select a Test List and click the Execute List button. All the tests defined in the list will execute using the current Test List Settings. When the test concludes, WebUI will automatically navigate to the Results pane where you can see if the Test List passed as a whole or drill down to see the results of individual tests in the list. The screenshot below shows the currently selected "Security Tests" list being executed.



Execute List

Execute the currently selected test list

Type	Test List	Date	Owner	T
	Security Tests	6/17/2010 3:29:18 PM		4
	Smoke Test	6/17/2010 3:19:57 PM		3

Test	Modified
Function Ac	6/17/2010 1
Invalid Pass	6/17/2010 4
Login	6/21/2010 2

8.6 Walk Through

The following walk through shows you how to create static and dynamic lists and execute them.

Find the materials for this Topic at...

\Projects\OrganizingTests

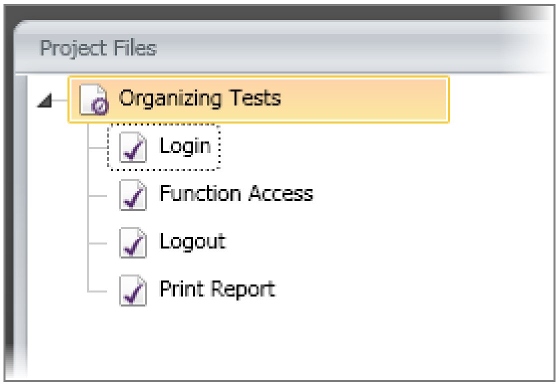
1. Open WebUI. You should see the Welcome Screen.

2. Click the **Create New Project** button. This will display the **New Project** dialog.

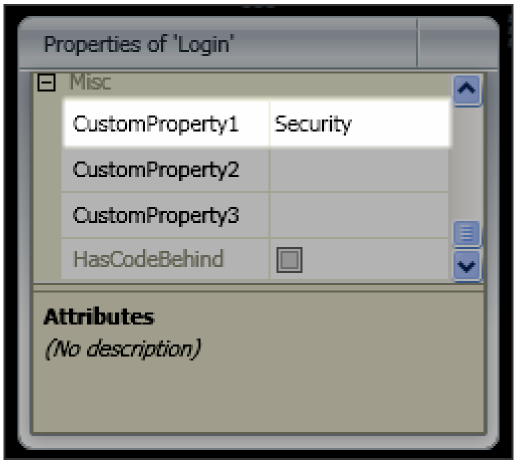
3. Enter "Organizing Tests" to the **Project Name** text box. Leave the **Location** text box with the default value. Click the **OK** button.

4. In the Project Files Pane, right-click the "Organizing Tests" node. Select **Add New Test** from the context menu. Enter "Login" as the name of the test.

5. Repeat this step for the following tests: "Function Access", "Logout", and "Print Report". The Project Files Pane should look like the screenshot below.



6. Select the "Login" test. In the Properties Pane, locate the "CustomProperty1" property. Set the value to "Security".



Properties of 'Login'

Misc

CustomProperty1	Security
CustomProperty2	
CustomProperty3	
HasCodeBehind	□

Attributes

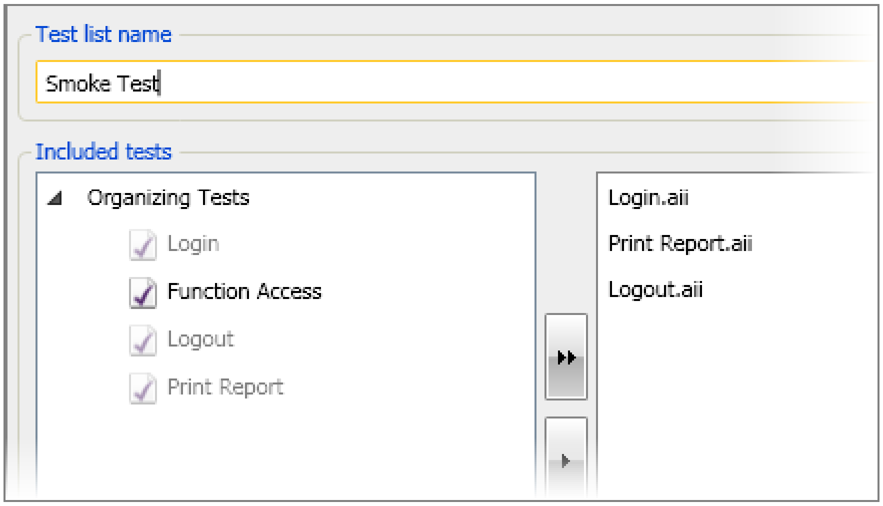
(No description)

7. Repeat this step for "Function Access" and "Logout" tests, setting the CustomProperty1 property to a value of "Security". Do not change the "Print Report" test.

8. Navigate to the Test Lists Tab.

9. Click the **List** button. This will display the **Add New Test List** dialog.

10. Enter the **Test Name** as "Smoke Test". Double click the "Login", "Print Report" and "Logout" reports, in that order. The test list should look like the screenshot below.



Test list name

Smoke Test

Included tests

Organizing Tests

☑ Login

☑ Function Access

☑ Logout

☑ Print Report

Login.aii

Print Report.aii

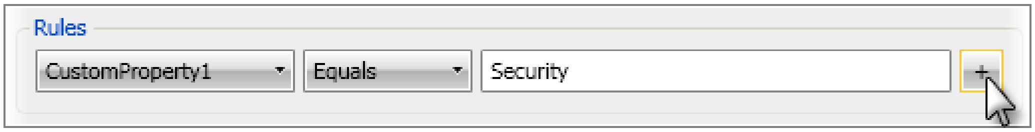
Logout.aii

11. Click the **OK** button to create the Test List.

12. Click the **Dynamic List** button to display the Add Dynamic Test List dialog.

13. Enter the Test List Name as "Security Tests".

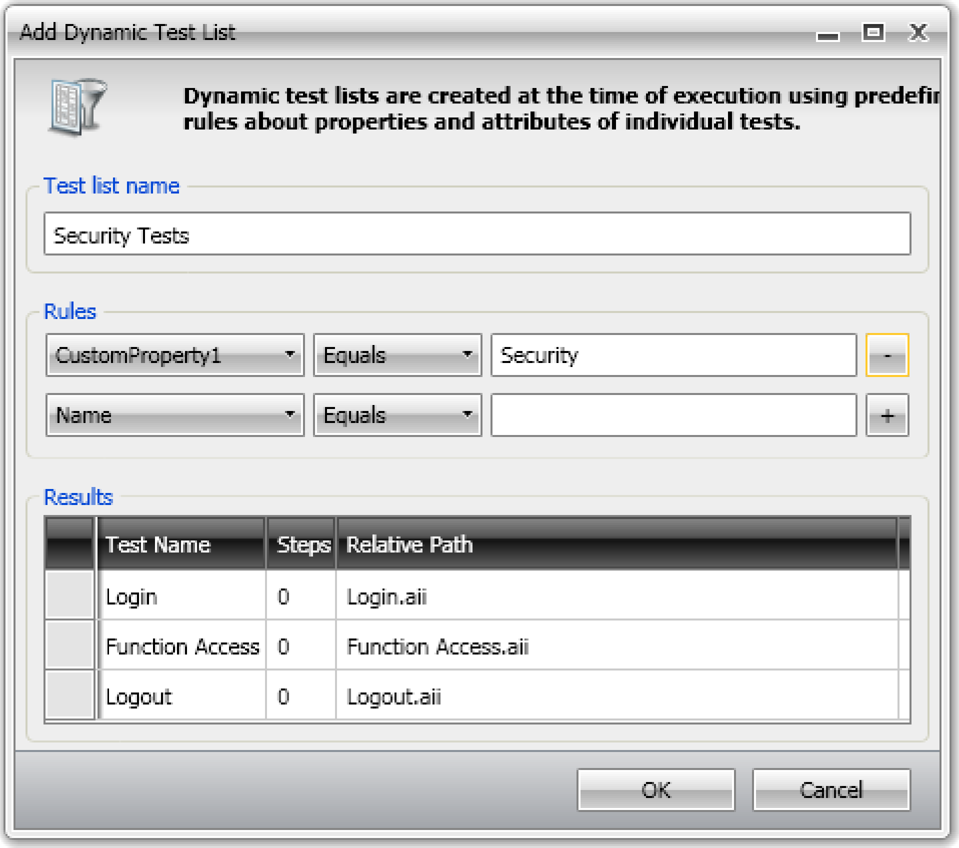
14. In the Rules section of the dialog, select "CustomProperty1" from the property drop down list. Set the comparison drop down to "Equals". Enter "Security" as the value. The rule should look like the screenshot below. Click the "+" button to add the rule and see the filtered list of tests.



Rules

CustomProperty1 | Equals | Security

15. The dialog should now look like the screenshot below. Click the **OK** button to create the dynamic list.



Add Dynamic Test List

Dynamic test lists are created at the time of execution using predefi rules about properties and attributes of individual tests.

Test list name

Security Tests

Rules

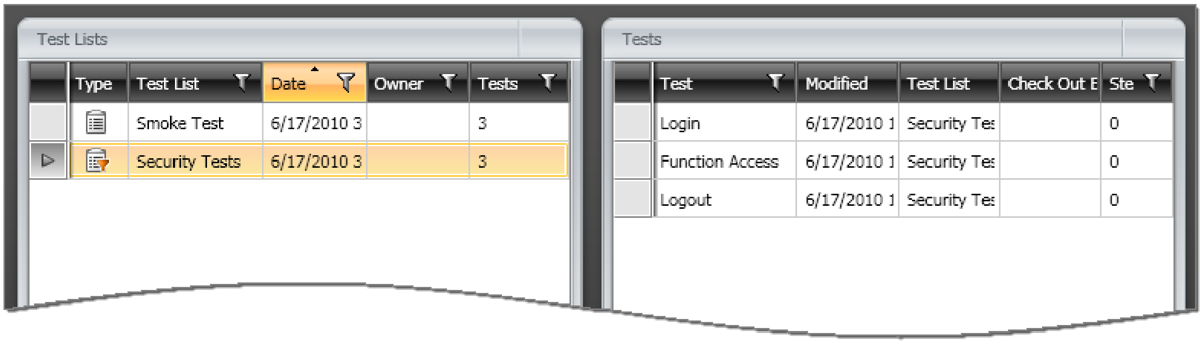
CustomProperty1 | Equals | Security

Name | Equals |

Results

Test Name	Steps	Relative Path
Login	0	Login.aii
Function Access	0	Function Access.aii
Logout	0	Logout.aii

16. The Test Lists Pane will list the new "Smoke Test" and "Security Tests" entries.



Test Lists

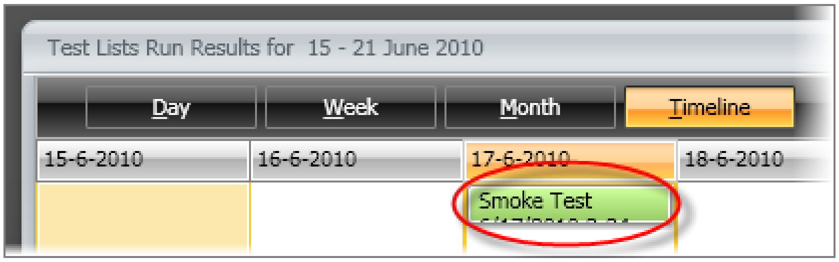
Type	Test List	Date	Owner	Tests
	Smoke Test	6/17/2010 3		3
	Security Tests	6/17/2010 3		3

Tests

Test	Modified	Test List	Check Out E	Ste
Login	6/17/2010 1	Security Tes		0
Function Access	6/17/2010 1	Security Tes		0
Logout	6/17/2010 1	Security Tes		0

17.Select the "Smoke Test" list and then click the **Execute List** button.

18.When the test completes, WebUI should navigate automatically to the Results Tab. The "Smoke Test" should show up in the Timeline in green to indicate a successful test.

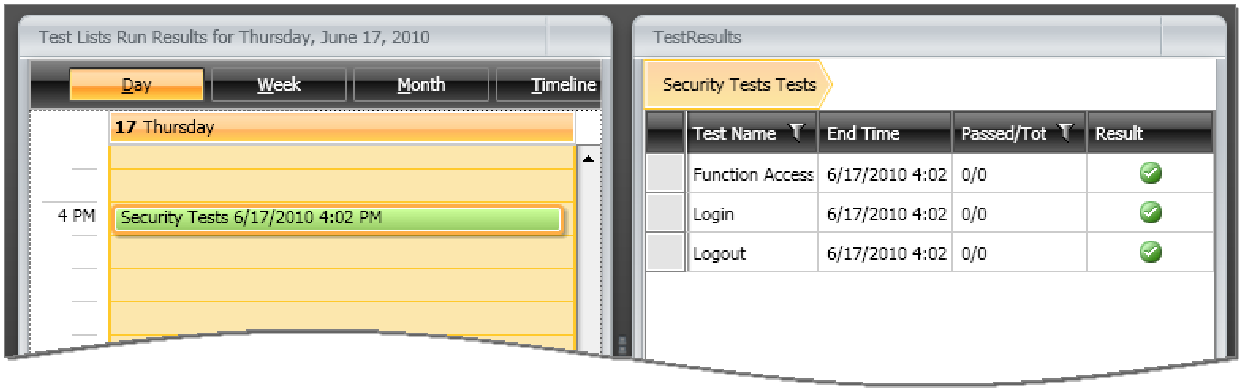


19.Navigate back to the Test Lists Tab.

20.Select the "Security Tests" list and then click the **Execute List** button.

21.When the test completes, WebUI should navigate automatically to the Results Tab. The "Security Tests" should show up in the Timeline in green to indicate a successful test.

22.Double-click the "Security Tests" entry in the timeline. Notice that the three tests with "CustomProperty1" equal to "Security" were run.


Test Lists Run Results for Thursday, June 17, 2010
Day
Week
Month
Timeline
17 Thursday
4 PM
Security Tests 6/17/2010 4:02 PM
TestResults
Security Tests Tests

Test Name	End Time	Passed/Tot	Result
Function Access	6/17/2010 4:02	0/0	✓
Login	6/17/2010 4:02	0/0	✓
Logout	6/17/2010 4:02	0/0	✓

Gotcha!

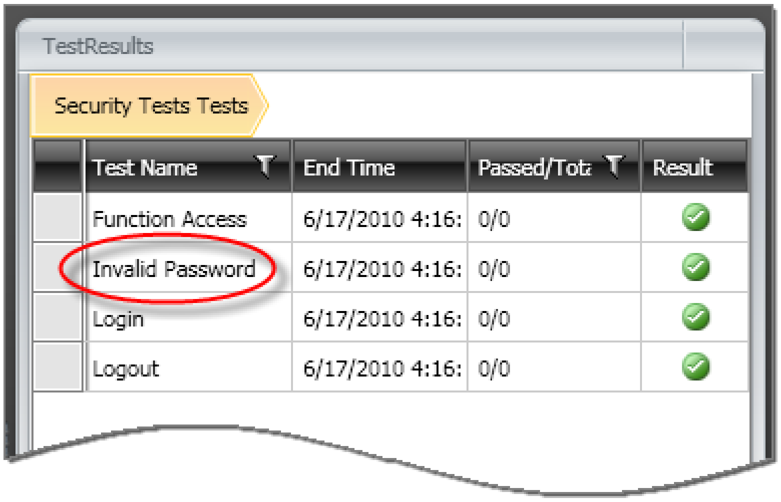
Notice that there is no control over the order of tests in the dynamic list. Each test should be self-sufficient and not require any particular state, especially from another test in the list. You may have dependencies that require a specific initialization, cleanup or a specific ordering of tests. In these cases, use the "Add Test as Step" feature. For example, if your "Function Access" test requires a "Login" test to run first and a "Logout" to run just afterward, these can be added using "Add Test as Step" to the "Function Access" test itself.

23.Navigate back to the Project Tab and add a new test "Invalid Password". Set the "CustomProperty1" property value to "Security".

24.Save the project.

25.Re-run the "Security Tests" dynamic test list.

26. Again, open the results from the Timeline and notice that the new test was automatically included based on the rules of the dynamic test list.



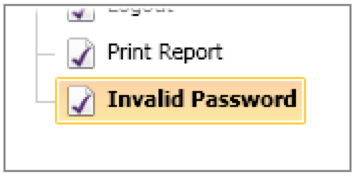
TestResults

Security Tests Tests

Test Name	End Time	Passed/Tot:	Result
Function Access	6/17/2010 4:16:	0/0	✓
Invalid Password	6/17/2010 4:16:	0/0	✓
Login	6/17/2010 4:16:	0/0	✓
Logout	6/17/2010 4:16:	0/0	✓

Gotcha!

Be sure to save the project before running the test or the new test will not show up in the dynamic list, regardless of its property settings. Remember that the test will show in bold text before it is saved.



8.7 Summary

In this chapter you learned how Test Lists help get the best use of recorded tests by allowing reuse in more than one configuration. You learned to build static test lists from existing tests and manually set the order that the tests run. You also learned how dynamic test lists automatically select tests from your project at the time of execution using rules about properties of the individual tests. You saw how to configure settings for all the tests within a test list.

Part IX

Working with Test Results

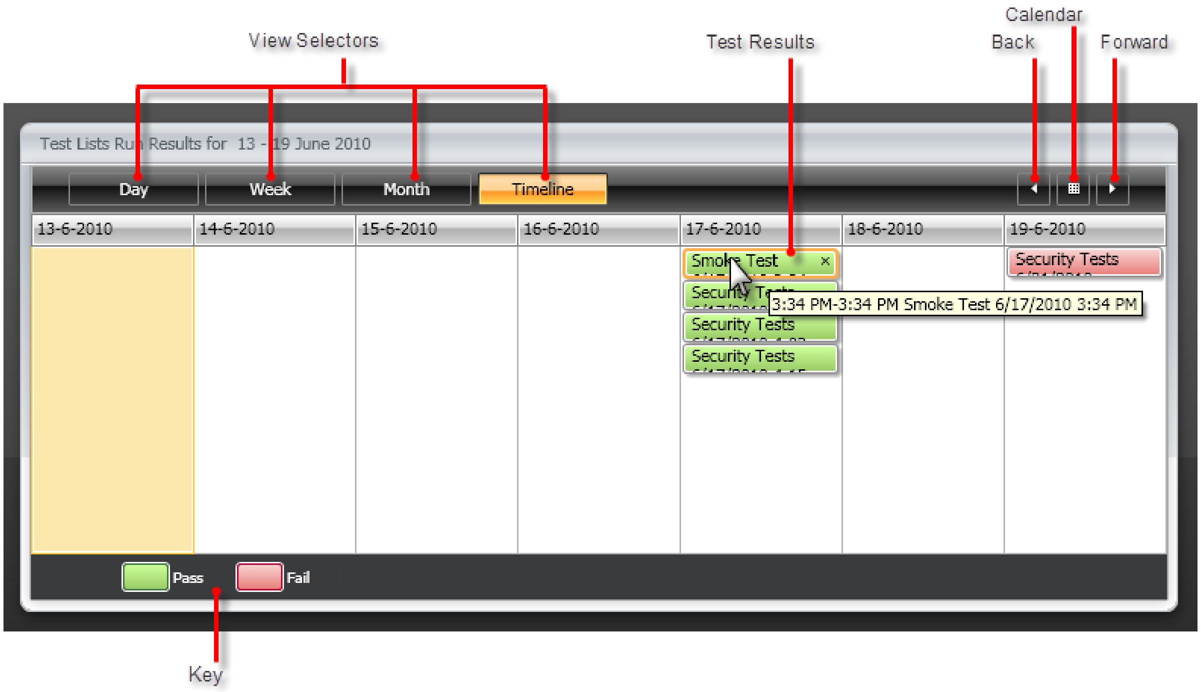
9 Working with Test Results

9.1 Objectives

In this chapter you will learn how to analyze and share the results of your tests. First you will see how the calendar view of test results makes it easy to see the tests that passed or failed in a given period of time. Next you will see how the TestResults panel allows you to traverse test execution results, drilling down to individual test steps and back up again. Finally, you will learn how the Step Failure Details dialog is used to work with a single failed test step.

9.2 Test Results Calendar

The calendar view of test results makes it easy to see the tests that passed or failed in a month, week or day. The buttons along the top of the calendar area let you jump between Day, Week, Month and Timeline views. The test results are shown as colored blocks in the calendar. Results cannot be moved to another location in the calendar, but you can click the "x" button to remove a result. The navigation buttons on the upper right allow you to move back and forward through some chunk of time, based on the calendar view. The Calendar button displays a popup calendar and allows you to jump to a particular day. Test results are displayed in the colors indicated by the Key at the bottom of the view.

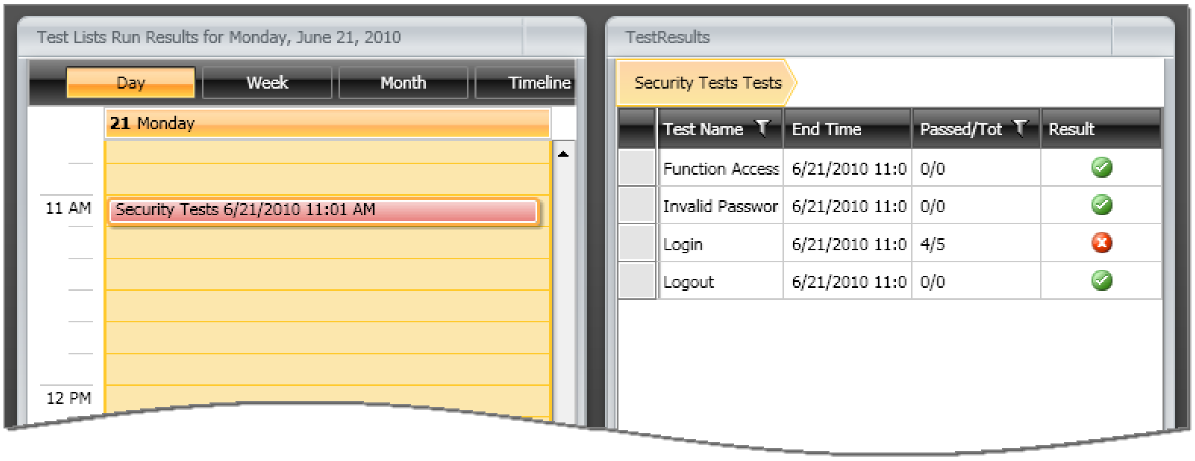


3:34 PM-3:34 PM Smoke Test 6/17/2010 3:34 PM

9.3 Analyzing Test Results

The TestResults panel allows you to traverse test execution results, drilling down to the individual test step and back up again to the test list level.

To see the results for a test, either double-click the test result in the calendar or select the test result and click the toolbar **Analyze** button. The view splits to show the calendar on the left and the "Test Results" panel on the right. The test results panel in the screenshot below show results for the entire "Security Tests" test list. There are four tests in the list and the "Login" test has an icon that indicates it has failed. The "Passed/Total" column shows that four of the five steps in the test passed. Double-clicking a test in this list drills down to show the steps of that test.

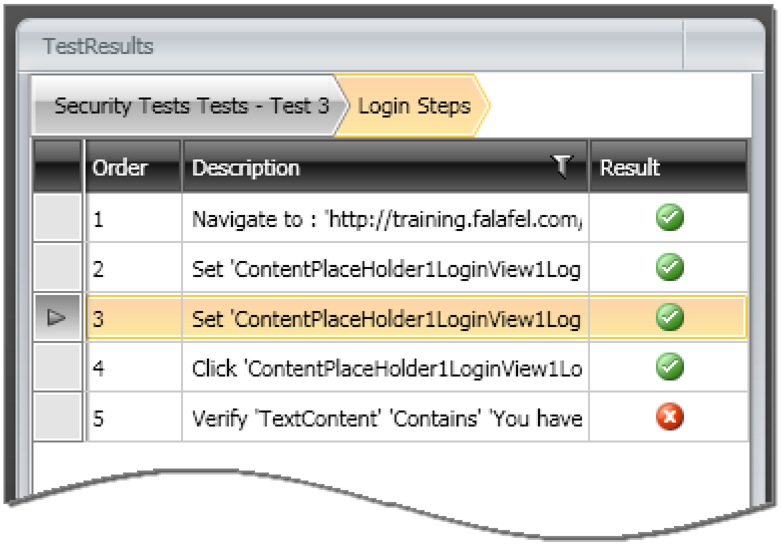


Security Tests 6/21/2010 11:01 AM

Test Name	End Time	Passed/Tot	Result
Function Access	6/21/2010 11:0	0/0	✓
Invalid Passwor	6/21/2010 11:0	0/0	✓
Login	6/21/2010 11:0	4/5	✗
Logout	6/21/2010 11:0	0/0	✓

By double-clicking the "Login" test results from the screenshot above, we can see that the fifth step of the test has failed. Also notice that the "bread crumb trail" at the top of the panel now shows the Test List followed by the test name. The items in this current look at the results show the individual test steps.

You can click the top level in the bread crumb trail to jump directly back to the test list results as a whole. You can also click the toolbar **Navigate | Back** to backtrack up through the hierarchy. Double clicking a failed test step displays the Step Failure Details dialog.



Order	Description	Result
1	Navigate to : 'http://training.falafel.com,	✓
2	Set 'ContentPlaceHolder1LoginView1Log	✓
3	Set 'ContentPlaceHolder1LoginView1Log	✓
4	Click 'ContentPlaceHolder1LoginView1Lo	✓
5	Verify 'TextContent' 'Contains' 'You have	✗

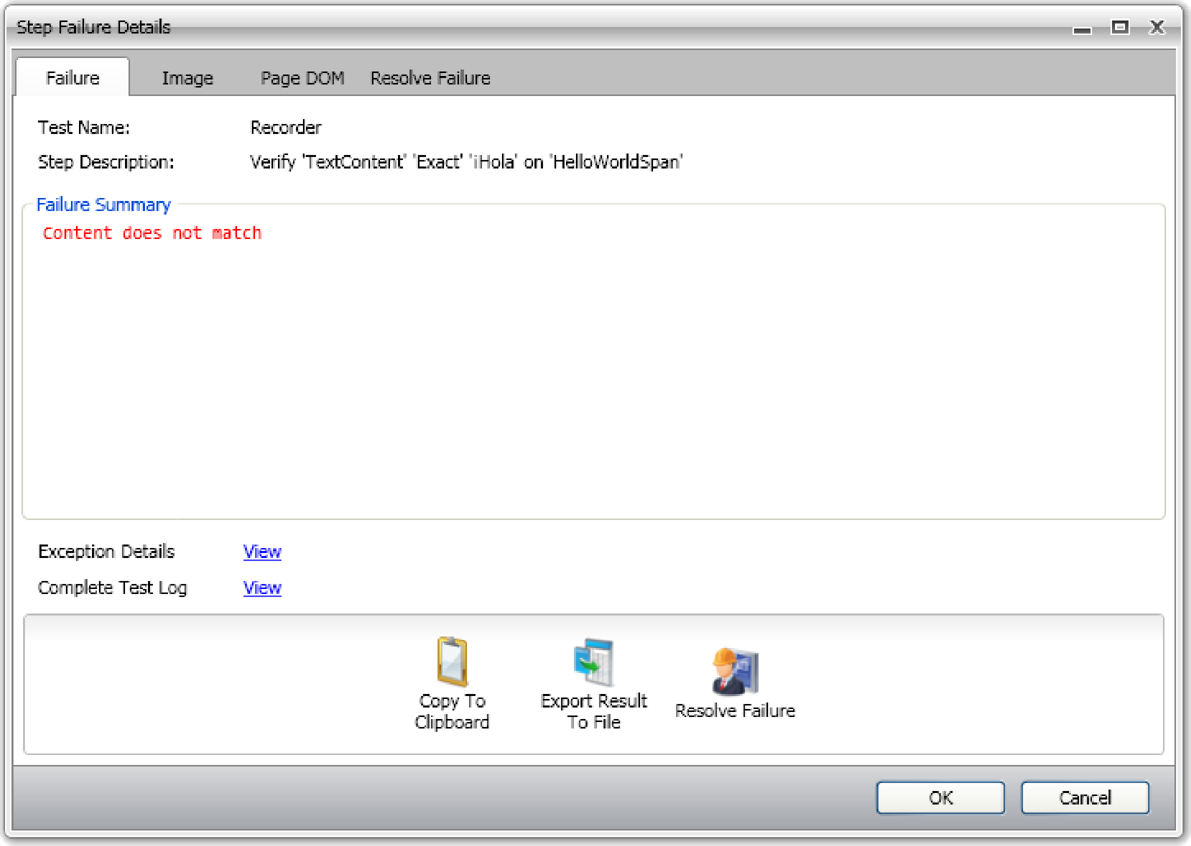
Step Failure Details Dialog

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, all in this one window.

Failure Tab

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.

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 where you can fix element location errors.



Test Name: Recorder

Step Description: Verify 'TextContent' 'Exact' 'iHola' on 'HelloWorldSpan'

Failure Summary

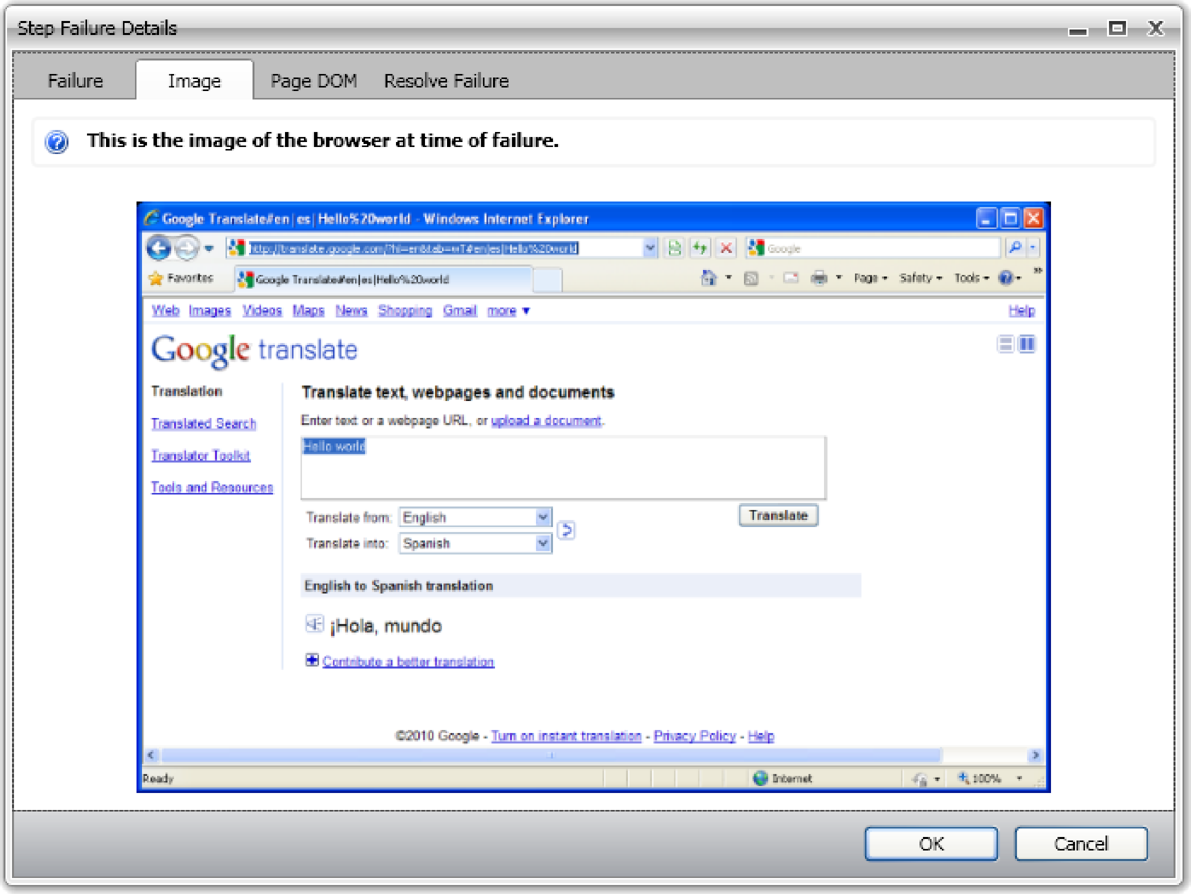
```
Content does not match
```

Exception Details View

Complete Test Log View

Image Tab

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



Step Failure Details
Failure
Image
Page DOM
Resolve Failure
Google Translate#en|es|Hello%20world - Windows Internet Explorer
http://translate.google.com/?hl=en&tab=wT#en|es|Hello%20world
Favorites
Google Translate#en|es|Hello%20world
Page
Safety
Tools
Web Images Videos Maps News Shopping Gmail more
Help
Google translate
Translation
Translated Search
Translator Toolkit
Tools and Resources
Translate
Contribute a better translation
Ready
Internet
100%
OK
Cancel

This is the image of the browser at time of failure.

Translate text, webpages and documents

Enter text or a webpage URL, or upload a document.

Hello world

Translate from: English

Translate into: Spanish

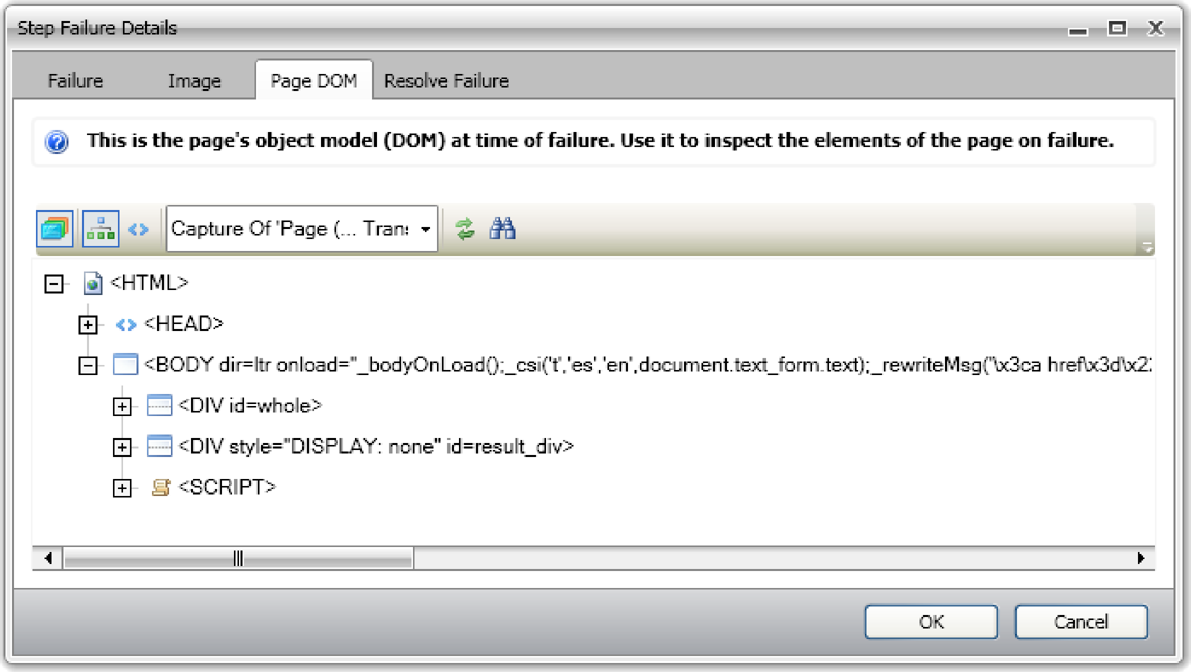
English to Spanish translation

¡Hola, mundo

©2010 Google - Turn on instant translation - Privacy Policy - Help

Page DOM Tab

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.



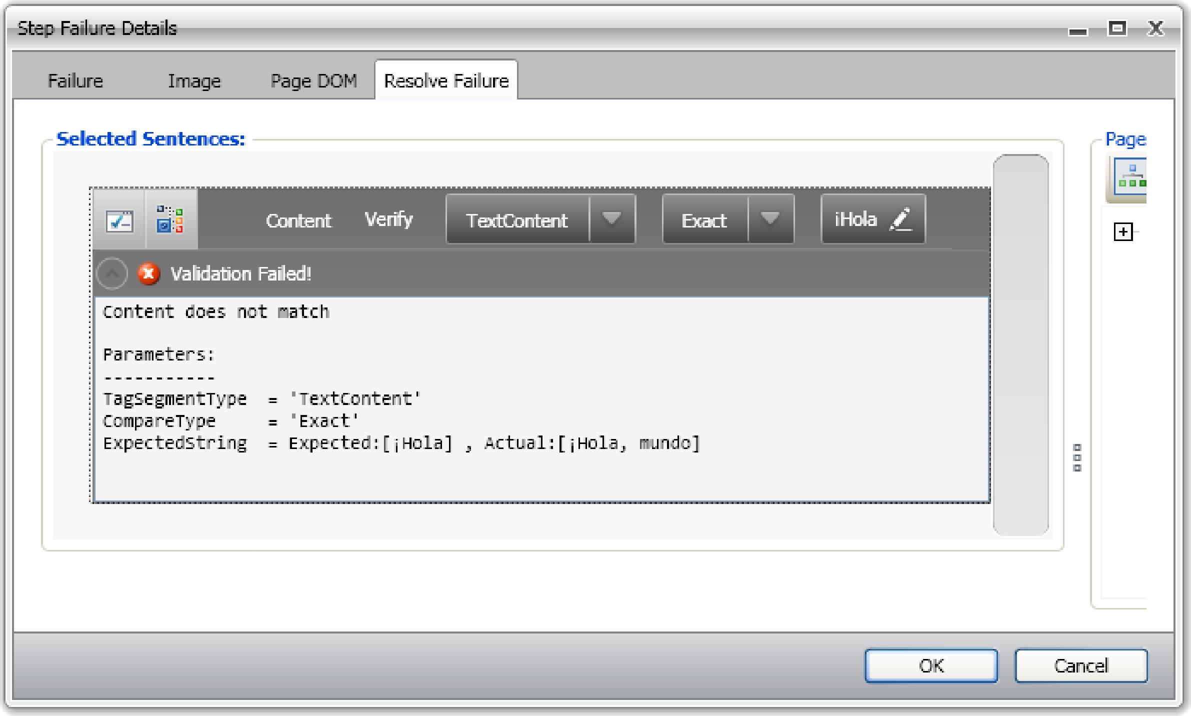
This is the page's object model (DOM) at time of failure. Use it to inspect the elements of the page on failure.

```
<HTML>
  <HEAD>
  <BODY dir=ltr onload="_bodyOnLoad();_csi('t','es','en',document.text_form.text);_rewriteMsg('\x3ca href\x3d\x2
    <DIV id=whole>
    <DIV style="DISPLAY: none" id=result_div>
    <SCRIPT>
```

Resolve Failure Tab

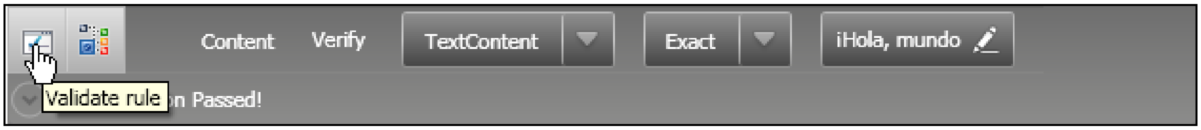
The **Resolve Failure** tab of the dialog provides the opportunity to identify and correct the issue that caused the failure. For validation steps, for example, the Sentence Verification Builder allows you to reload the page, make changes to the verification sentence and re-run the verification until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



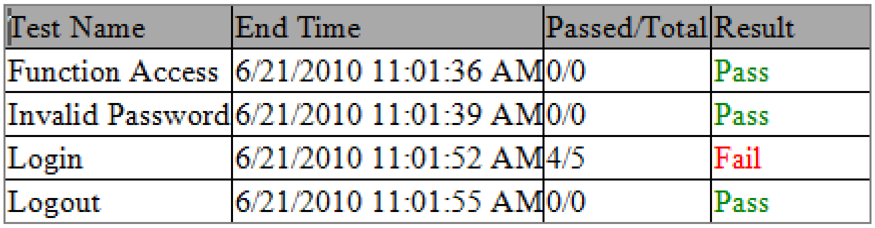
```
Content does not match

Parameters:
-----------
TagSegmentType  = 'TextContent'
CompareType     = 'Exact'
ExpectedString  = Expected:[¡Hola] , Actual:[¡Hola, mundo]
```

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



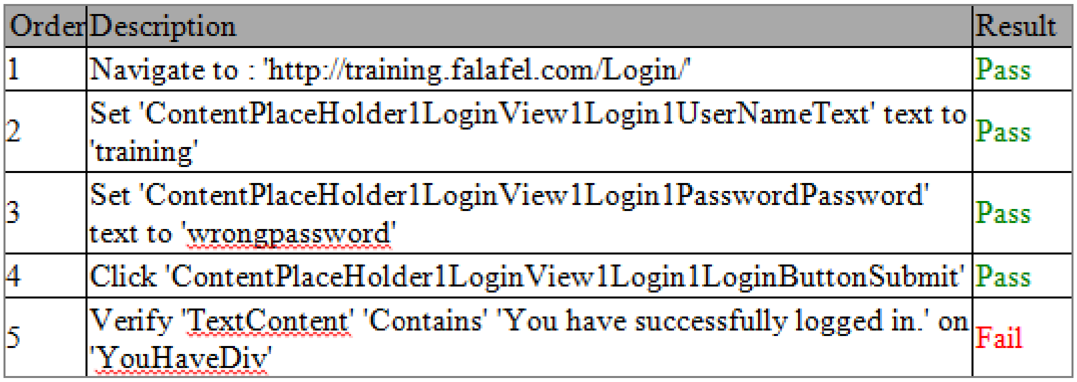
9.4 Exporting Test Results

Test results become truly useful when they are pushed out to the members of your organization. The Export tools allow you to create Word or Excel versions of the entire Test List. The screenshot below shows four tests in the "Security Tests" list example. The "Login" test is flagged as having failed.



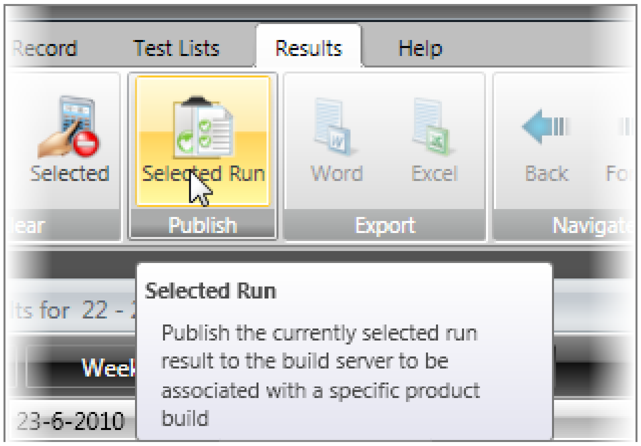
Test Name	End Time	Passed/Total	Result
Function Access	6/21/2010 11:01:36 AM	0/0	Pass
Invalid Password	6/21/2010 11:01:39 AM	0/0	Pass
Login	6/21/2010 11:01:52 AM	4/5	Fail
Logout	6/21/2010 11:01:55 AM	0/0	Pass

At the test level, the screenshot below shows that the fifth test step was attempting to verify the contents of an element and failed.



Order	Description	Result
1	Navigate to : 'http://training.falafel.com/Login/'	Pass
2	Set 'ContentPlaceHolder1LoginView1Login1UserNameText' text to 'training'	Pass
3	Set 'ContentPlaceHolder1LoginView1Login1PasswordPassword' text to 'wrongpassword'	Pass
4	Click 'ContentPlaceHolder1LoginView1Login1LoginButtonSubmit'	Pass
5	Verify 'TextContent' 'Contains' 'You have successfully logged in.' on 'YouHaveDiv'	Fail

If you have access to Team Foundation Server (TFS) you can make a selected run result easily accessible to others by pushing out the results to TFS.



Selected Run

Publish the currently selected run result to the build server to be associated with a specific product build

9.5 Walk Through

The following walk through shows you how to navigate through your results. The first part of this exercise will create a test that generates an error. Once the test list is executed, we will examine the test results.

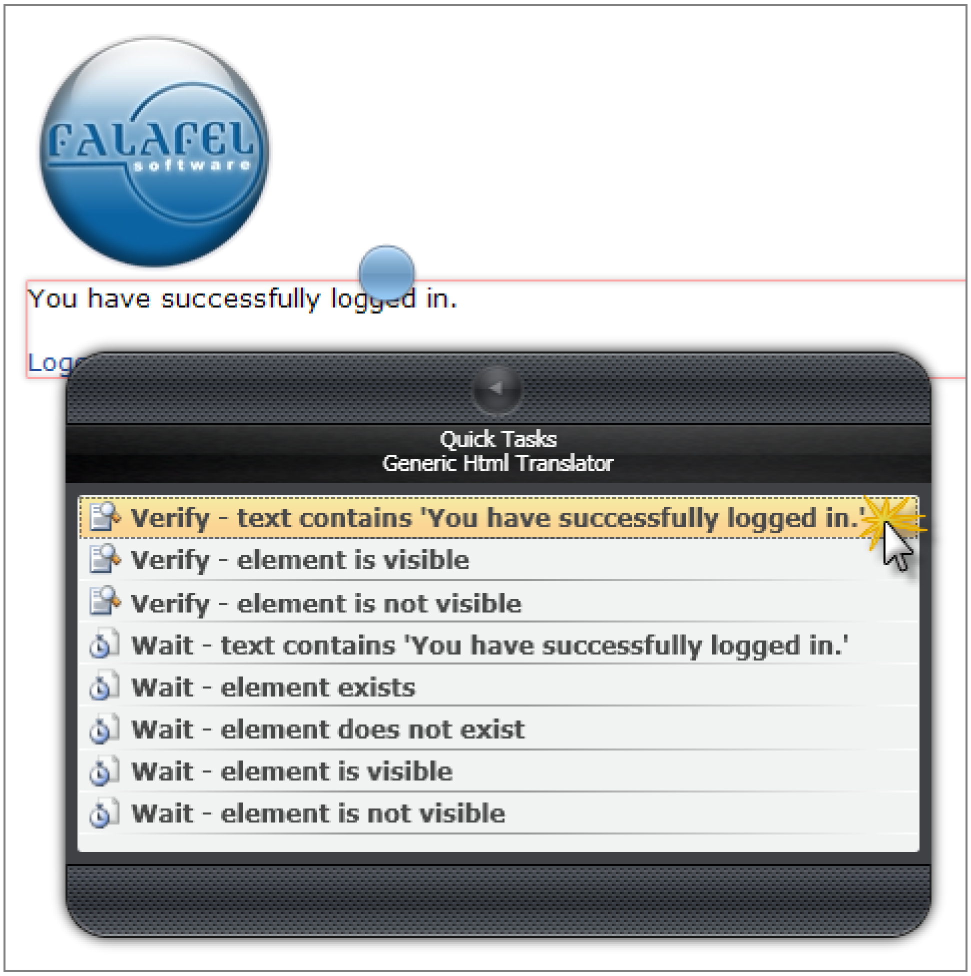
Find the materials for this Topic at...

\Projects\Results

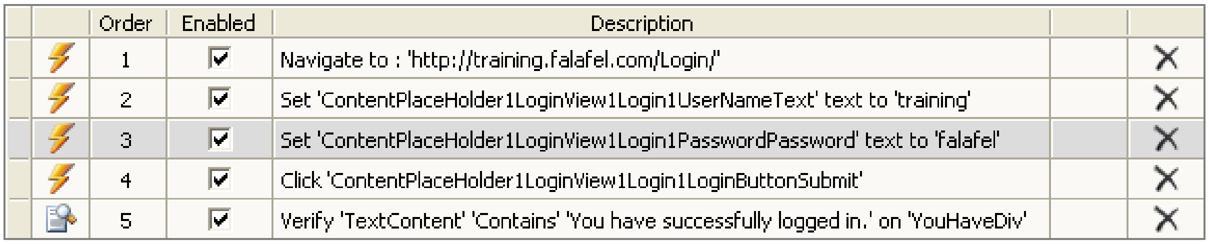
Prepare the Test

1. Make a copy of the walk through for the "Test Organization" chapter. Name the copy "WorkingWithTestResults".
2. Open WebUI. This will display the Welcome Screen.
3. Click the **Open Existing Project** button, navigate to the "WorkingWithTestResuts" folder an open the project.
4. Navigate to the Test Lists Tab and select the "Smoke Test" test list.
5. In the Tests panel, double-click the "Login" test.
6. From the Record Tab, click the **Start Recording** button.
7. In the Recording Surface enter **"http://training.falafel.com/login/"** to the browser address bar and click the Go to Url button.
8. In the "User Name" edit box, enter "training".
9. In the "Password" edit box, enter "falafel".
10. Click the **Log In** button.
11. Click the Highlighting button .

12. Move the mouse over the "You have successfully logged in." message until the Nub appears. Click the Nub to display the Elements Menu and select "Verify - text contains 'You have successfully logged in.'" from the Quick Tasks menu.


FALAFEL software
You have successfully logged in.
Log
Quick Tasks
Generic Html Translator
Verify - text contains 'You have successfully logged in.'
Verify - element is visible
Verify - element is not visible
Wait - text contains 'You have successfully logged in.'
Wait - element exists
Wait - element does not exist
Wait - element is visible
Wait - element is not visible

13. The test steps should look like those in the screenshot below.

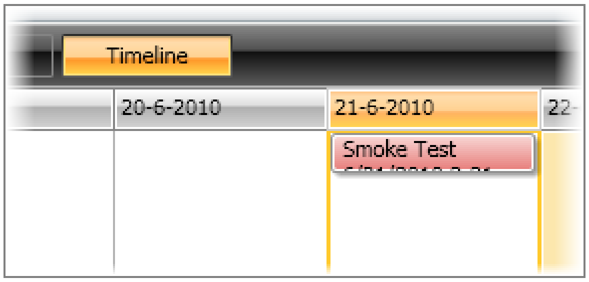


		Order	Enabled	Description		
	⚡	1	☑	Navigate to : 'http://training.falafel.com/Login/'		✕
	⚡	2	☑	Set 'ContentPlaceHolder1LoginView1Login1UserNameText' text to 'training'		✕
	⚡	3	☑	Set 'ContentPlaceHolder1LoginView1Login1PasswordPassword' text to 'falafel'		✕
	⚡	4	☑	Click 'ContentPlaceHolder1LoginView1Login1LoginButtonSubmit'		✕
		5	☑	Verify 'TextContent' 'Contains' 'You have successfully logged in.' on 'YouHaveDiv'		✕

14. Select the last step that performs the verification. In the Properties Pane, locate the **ExpectedString** property and change it to "wrong value".

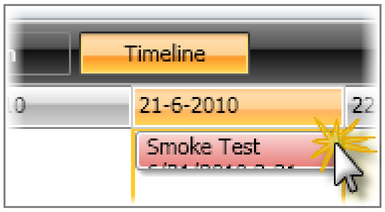
15. Navigate to the Test Lists Tab.

16. Select the "Smoke Test" test, then click the **Execute List** button. Once the test has executed, WebUI will navigate automatically to the Results Tab. Because we have engineered the last step to fail, the test results will be shaded red to indicate a failed test run.

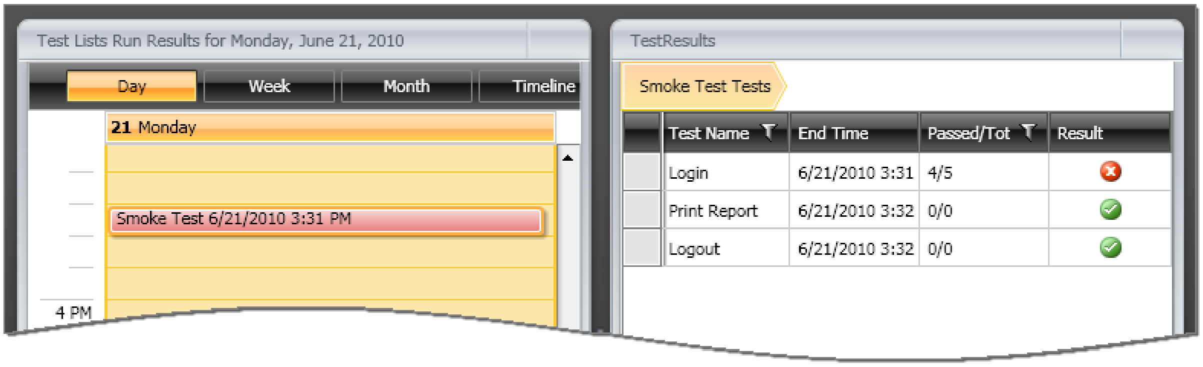


Examine Test Results

1. Double-click the test result.

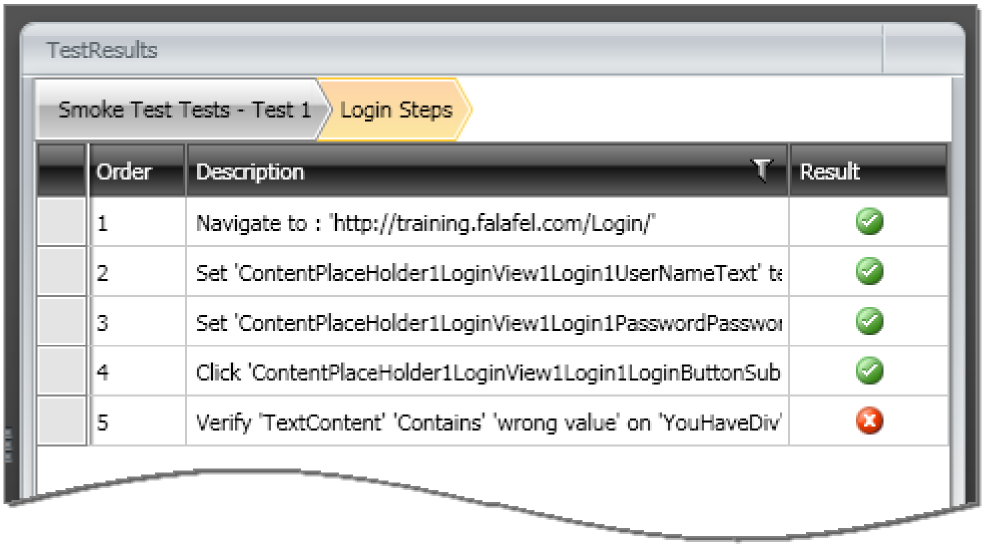


2. The TestResults window shows the results for the "Smoke Test" test list. There are three tests in this test list, "Login", "Print Report" and "Logout". The "Login" test has failed. Double-click the "Login" test.


Test Lists Run Results for Monday, June 21, 2010
Day
Week
Month
Timeline
21 Monday
Smoke Test 6/21/2010 3:31 PM
4 PM
TestResults
Smoke Test Tests

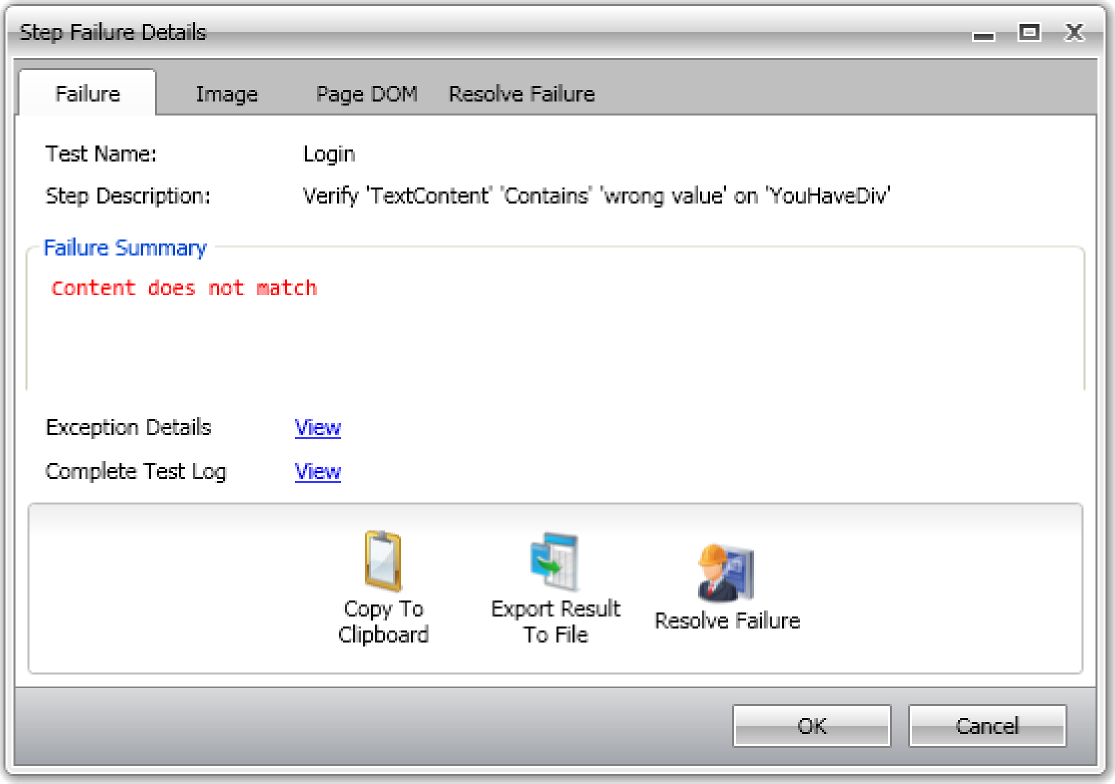
Test Name	End Time	Passed/Tot	Result
Login	6/21/2010 3:31	4/5	✗
Print Report	6/21/2010 3:32	0/0	✓
Logout	6/21/2010 3:32	0/0	✓

3. The TestResults window now lists the steps for the "Login" test. Notice that the last step has failed and has a red "x" icon in the "Result" column.



Order	Description	Result
1	Navigate to : 'http://training.falafel.com/Login/'	✓
2	Set 'ContentPlaceHolder1LoginView1Login1UserNameText' te	✓
3	Set 'ContentPlaceHolder1LoginView1Login1PasswordPasswoı	✓
4	Click 'ContentPlaceHolder1LoginView1Login1LoginButtonSub	✓
5	Verify 'TextContent' 'Contains' 'wrong value' on 'YouHaveDiv'	✗

4. Double-click the failed last step to display the **Step Failure Details** dialog. From the Failure Summary we can see that the "Content does not match".



Test Name: Login

Step Description: Verify 'TextContent' 'Contains' 'wrong value' on 'YouHaveDiv'

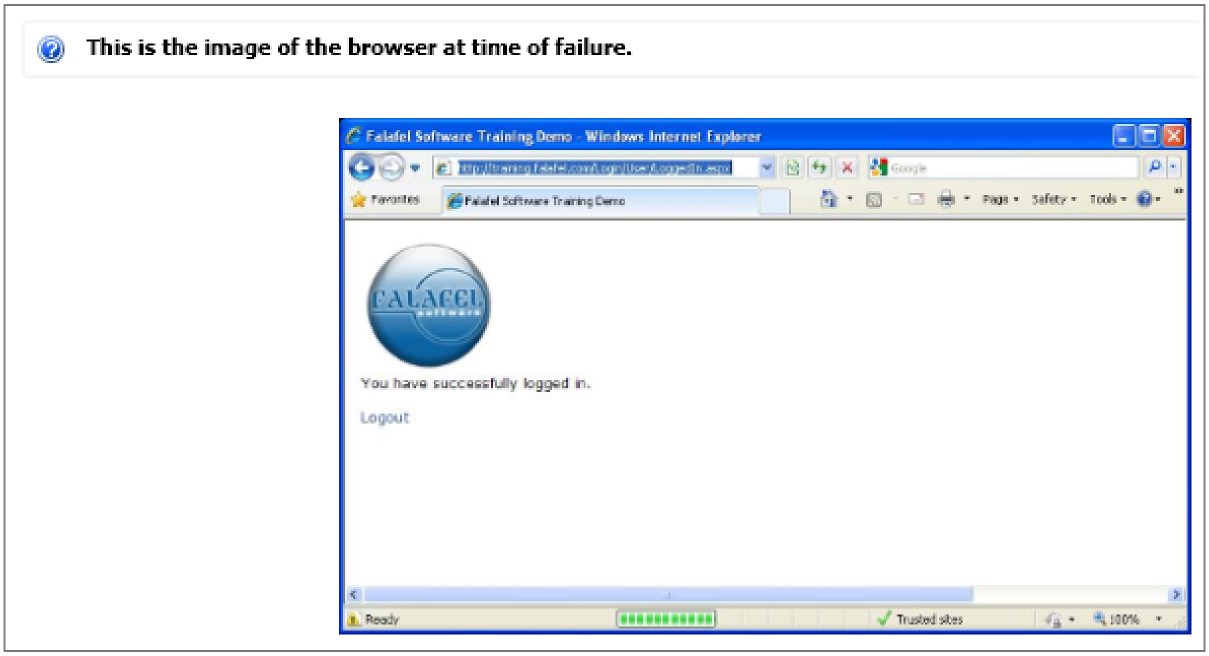
Failure Summary

```
Content does not match
```

Exception Details View

Complete Test Log View

5. Click the **Image** tab. The Image tab shows us what the browser looked like at the time the test failed. We can see from the image that we reached the "successfully logged in" page. So the steps leading to this point are correct, but the verification itself is probably at fault.



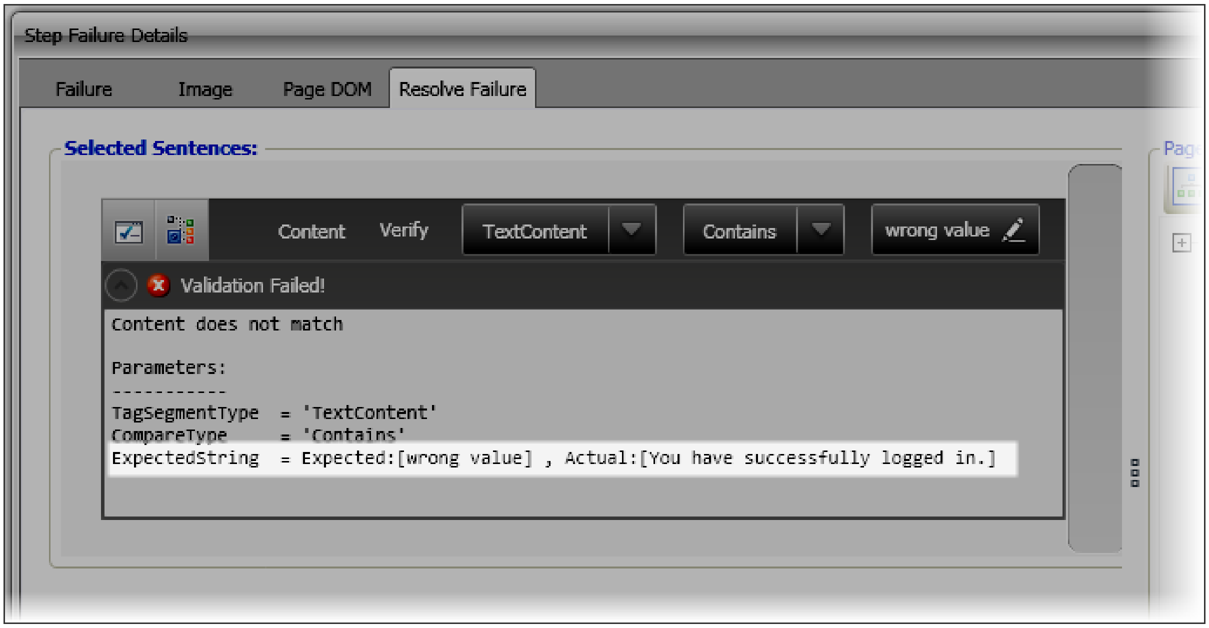
This is the image of the browser at time of failure.

You have successfully logged in.

Logout

Falafel Software Training Demo - Windows Internet Explorer
http://training.falafel.com/Login/UserLoggedIn.aspx
Google
Favorites
Falafel Software Training Demo
Page
Safety
Tools
FALAFEL software
Ready
Trusted sites
100%

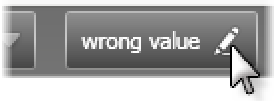
6. Click the **Resolve Failure** tab. Here we see that the verification expects the string "wrong value" and the actual value in the browser was "You have successfully logged in".



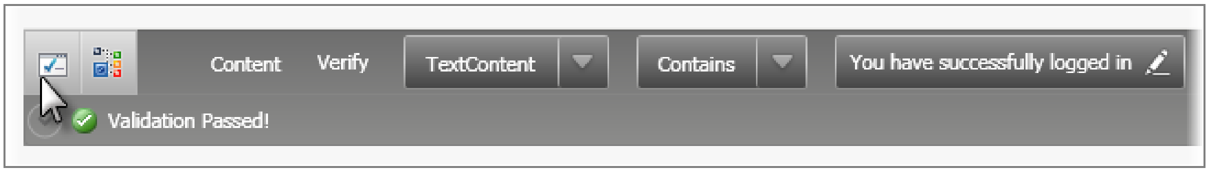
```
Content does not match

Parameters:
-----------
TagSegmentType  = 'TextContent'
CompareType     = 'Contains'
ExpectedString  = Expected:[wrong value] , Actual:[You have successfully logged in.]
```

Once you have used the Step Failure Details dialog to pinpoint the problem, your choices are either to change the verification to match the conditions of the page, or exit the Step Failure Details dialog and change the steps leading up to the failure. In this case the verification itself is the problem, so we can correct it right within the Resolve Failure tab.

7. Click the value edit button.

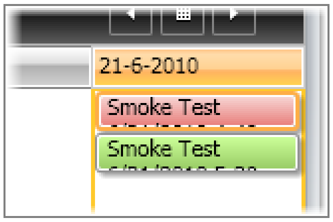


8. Enter "You have successfully logged in", then click the edit button again to commit the edit. Click the validation button ☑ to make sure that the comparison has been corrected.



9. Click the **OK** button to close the Step Failure Details dialog.

10. Navigate back to the Test Lists Tab and re-execute the test list. This second execution of the test list should succeed.



9.6 Summary

In this chapter you learned how to analyze and share the results of your tests. First you saw how the calendar view of test results makes it easy to see the tests that passed or failed in a given period of time. Next you saw how the TestResults panel allows you to traverse test execution results, drilling down to individual test steps and back up again. Finally, you learned how the Step Failure Details dialog is used to work with a single failed test step

Part X

Web Application Tests

10 Web Application Tests

10.1 Objectives

In this chapter you will learn about some of the technologies that make web applications so easy to work with as a user, but also make testing more difficult as a Quality Assurance engineer. You'll learn how WebUI handles asynchronous updates in the browser, timing issues and Silverlight animation. In particular, you will learn how WebUI can efficiently wait for a set of conditions to occur.

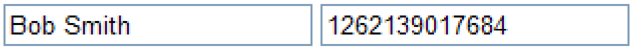
10.2 Web Testing Issues

Testing a web application is quite a bit more difficult than testing the same functionality in a Windows Desktop application. You can expect to encounter timing difficulties, differences between browsers, and a stack of new technologies that make web browsing a rich, but difficult-to-test, experience. The following topics explain how these web technologies cause problems with testing and how WebUI handles these issues.

10.3 JavaScript

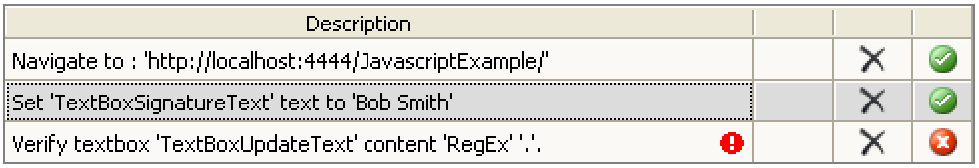
Prior to the advent of JavaScript in 1995 by Netscape, the browser experience was completely server based. That is, you pressed a button or clicked a link, the page was sent to the server and a new batch of HTML was constructed and sent back to the browser. The entire page was refreshed each time. With 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.



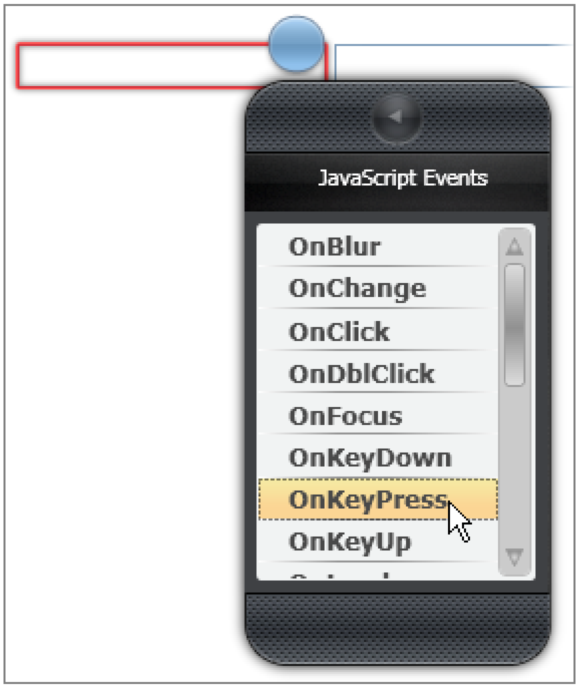
Bob Smith 1262139017684

By default, WebUI 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 below shows a failing test step where a regular expression is verifying that the second text box is non-blank.

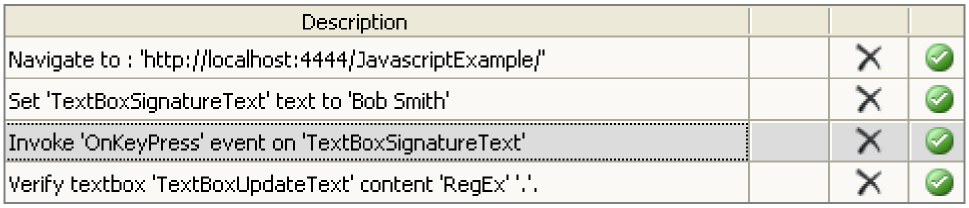


Description			
Navigate to : 'http://localhost:4444/JavascriptExample/'		✕	✓
Set 'TextBoxSignatureText' text to 'Bob Smith'		✕	✓
Verify textbox 'TextBoxUpdateText' content 'RegEx' '.'. ❗		✕	✗

Fortunately, WebUI 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 test step, the text for the second text box is updated properly and the test runs successfully.



Description			
Navigate to : 'http://localhost:4444/JavascriptExample/'		✕	✓
Set 'TextBoxSignatureText' text to 'Bob Smith'		✕	✓
Invoke 'OnKeyPress' event on 'TextBoxSignatureText'		✕	✓
Verify textbox 'TextBoxUpdateText' content 'RegEx' '.'.		✕	✓

10.4 AJAX

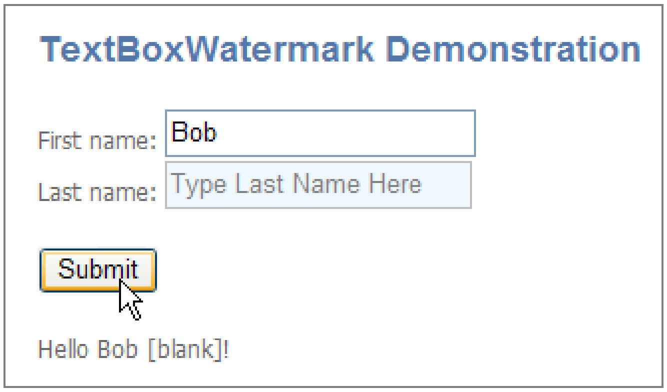
AJAX stands for **A**synchronous **J**avaScript **A**nd **X**ML and is a melding of browser, "client side" functionality with traditional server communication. JavaScript is capable of making calls to the server and updating selected portions of the page. From the testing perspective, AJAX may add a pause while information is retrieved from the server. AJAX is also "asynchronous" where not all parts of the page are updated at one time. For example, if you have prices from multiple locations, these prices can be displayed as they are received. 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 is by waiting for a particular element to reach some state, e.g. "text content = '1234'".

10.5 ASP.NET AJAX

ASP.NET AJAX is a Microsoft framework for AJAX web development. From a testing perspective, ASP.NET AJAX can be thought of as similar to manually programmed AJAX. ASP.NET AJAX comes with a set of components that make AJAX applications easier to develop. The 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 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.



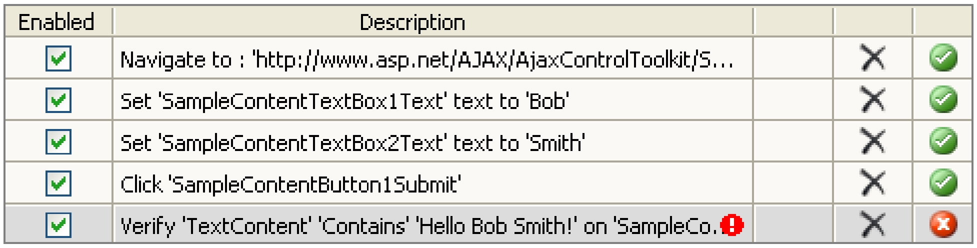
TextBoxWatermark Demonstration

First name: Bob

Last name: Type Last Name Here

Hello Bob [blank]!

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, the last test step fails. Why? And just as important, how do we find out what is wrong with the construction of the test?



Enabled	Description			
☑	Navigate to : 'http://www.asp.net/AJAX/AjaxControlToolkit/S...		✕	✓
☑	Set 'SampleContentTextBox1Text' text to 'Bob'		✕	✓
☑	Set 'SampleContentTextBox2Text' text to 'Smith'		✕	✓
☑	Click 'SampleContentButton1Submit'		✕	✓
☑	Verify 'TextContent' 'Contains' 'Hello Bob Smith!' on 'SampleCo. !		✕	✗

Notes

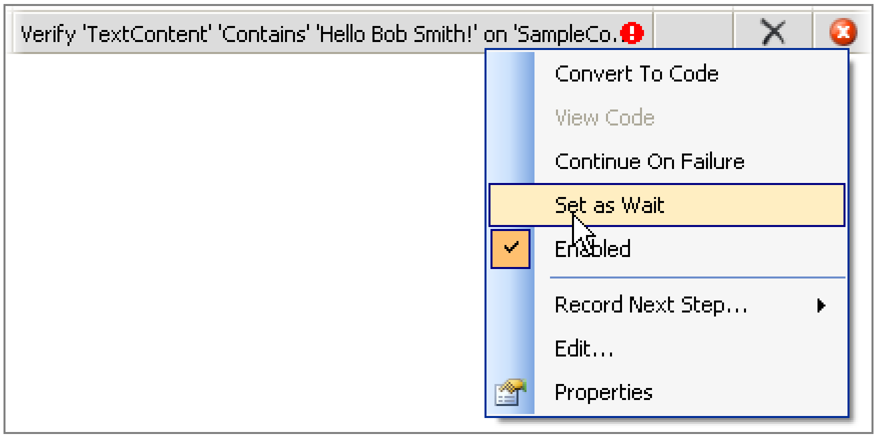
WebUI is a tool that measures behavior of a software product under certain conditions. As you become proficient with WebUI you can turn your attention from "what's wrong with the construction of my test?" to "what's wrong with the product I'm testing?"

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, we can slow down the action a bit during execution. 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.

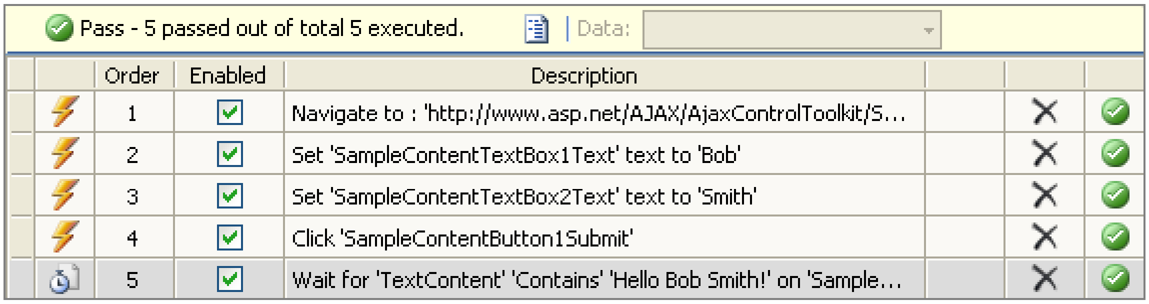
What if we turn off annotations and run the test at full speed? The verification test step fails because the text content of the label is checked *before* the server has responded. What we need is a way to perform the verification *after* the server has responded.

WebUI 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.



Verify 'TextContent' 'Contains' 'Hello Bob Smith!' on 'SampleCo.

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.



Pass - 5 passed out of total 5 executed.

Data:

	Order	Enabled	Description		
⚡	1	☑	Navigate to : 'http://www.asp.net/AJAX/AjaxControlToolkit/S...	✕	✓
⚡	2	☑	Set 'SampleContentTextBox1Text' text to 'Bob'	✕	✓
⚡	3	☑	Set 'SampleContentTextBox2Text' text to 'Smith'	✕	✓
⚡	4	☑	Click 'SampleContentButton1Submit'	✕	✓
⏱	5	☑	Wait for 'TextContent' 'Contains' 'Hello Bob Smith!' on 'Sample...	✕	✓

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.

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.

Find the materials for this Topic at...

\Projects\AspNetAjax

1) From the Welcome Screen, Click the **Record New Test** button. This step will navigate to the Recording Surface, with recording turned on.

2) In the browser address bar, enter "http://www.asp.net/AJAX/AjaxControlToolkit/Samples/ValidatorCallout/ValidatorCallout.aspx".

3) Press the **Enter** key. This will load the "ValidatorCallout" demonstration web page.

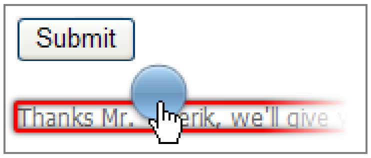
4) Enter the content "Mr. Telerik" into the "Name" text box.

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

6) Click the "Submit" button.

7) Press the Highlighting button to enable it.

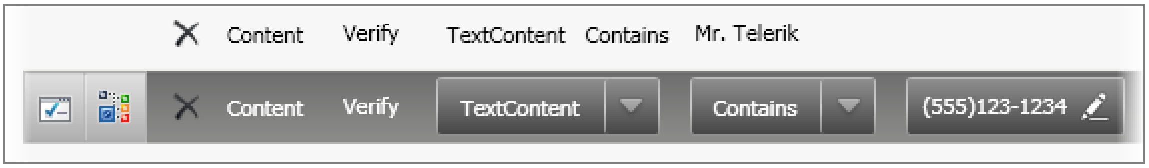
8) Move the mouse over the confirmation label, just below the "Submit" button, to highlight it. Wait for the Nub to appear.



9) Click the Nub to display the Elements Menu.

10)Click the Build Verification icon

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

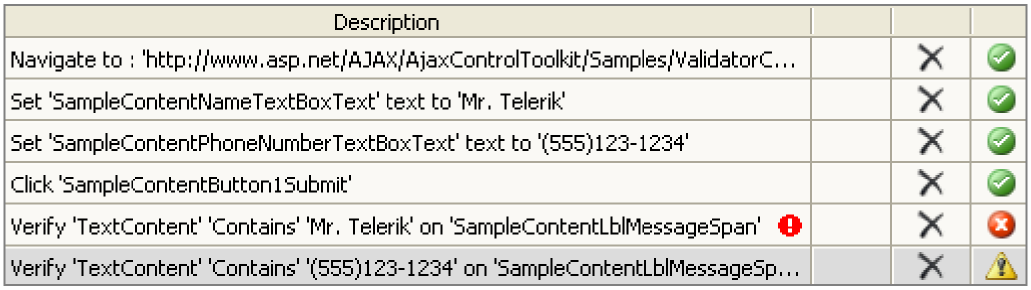


✕ Content Verify TextContent Contains Mr. Telerik

✕ Content Verify TextContent Contains (555)123-1234

12)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.

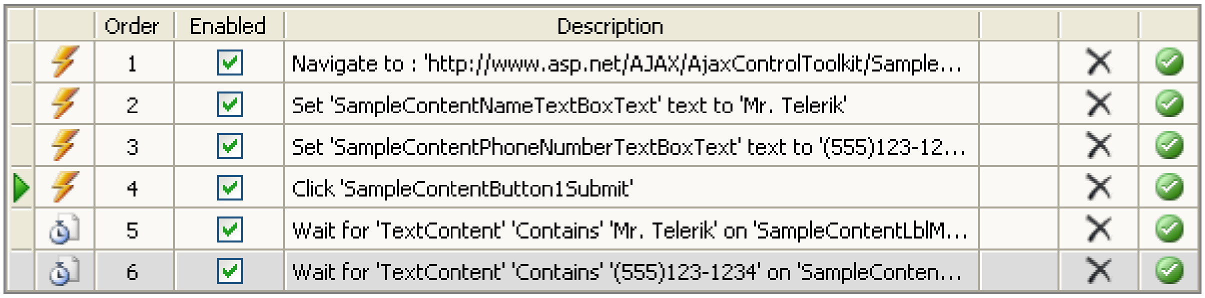


Description			
Navigate to : 'http://www.asp.net/AJAX/AjaxControlToolkit/Samples/ValidatorC...		✕	✓
Set 'SampleContentNameTextBoxText' text to 'Mr. Telerik'		✕	✓
Set 'SampleContentPhoneNumberTextBoxText' text to '(555)123-1234'		✕	✓
Click 'SampleContentButton1Submit'		✕	✓
Verify 'TextContent' 'Contains'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

13)Convert the last two test steps to "wait" steps. To do this, right-click both steps and select "Set as Wait" from the context menu.

14)Save the project.

15)Click the Quick Execute button to run the test. Now all test steps will complete successfully.

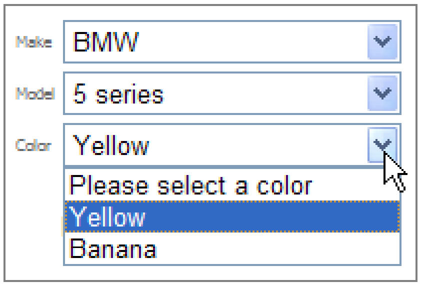


		Order	Enabled	Description			
	⚡	1	☑	Navigate to : 'http://www.asp.net/AJAX/AjaxControlToolkit/Sample...		✕	✓
	⚡	2	☑	Set 'SampleContentNameTextBoxText' text to 'Mr. Telerik'		✕	✓
	⚡	3	☑	Set 'SampleContentPhoneNumberTextBoxText' text to '(555)123-12...		✕	✓
▶	⚡	4	☑	Click 'SampleContentButton1Submit'		✕	✓
	⏱	5	☑	Wait for 'TextContent' 'Contains' 'Mr. Telerik' on 'SampleContentLblM...		✕	✓
	⏱	6	☑	Wait for 'TextContent' 'Contains' '(555)123-1234' on 'SampleConten...		✕	✓

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."



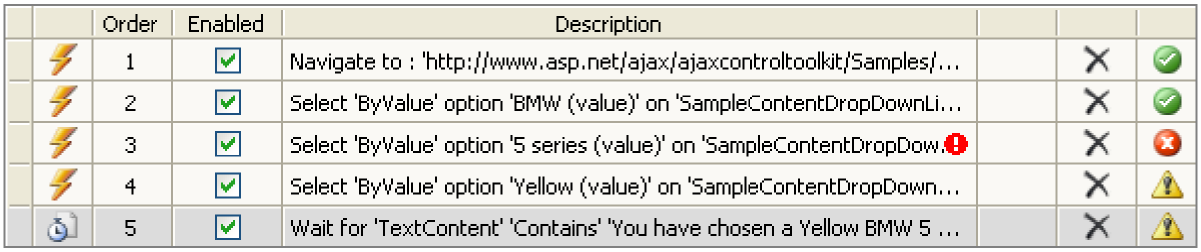
Make: BMW

Model: 5 series

Color: Yellow

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.



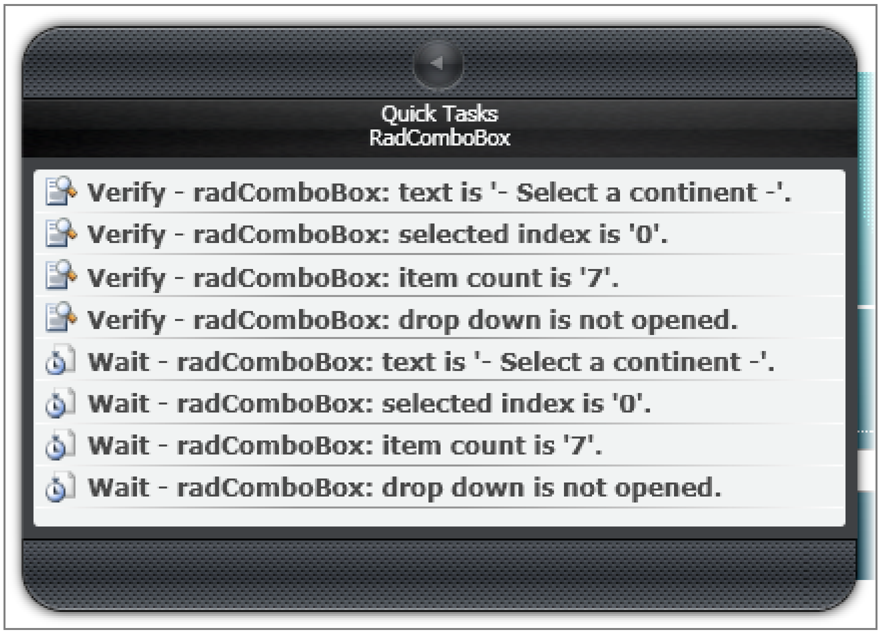
	Order	Enabled	Description			
⚡	1	☑	Navigate to : 'http://www.asp.net/ajax/ajaxcontroltoolkit/Samples/...		✕	✓
⚡	2	☑	Select 'ByValue' option 'BMW (value)' on 'SampleContentDropDownLi...		✕	✓
⚡	3	☑	Select 'ByValue' option '5 series (value)' on 'SampleContentDropDow❗		✕	✖
⚡	4	☑	Select 'ByValue' option 'Yellow (value)' on 'SampleContentDropDown...		✕	⚠
⏱	5	☑	Wait for 'TextContent' 'Contains' 'You have chosen a Yellow BMW 5 ...		✕	⚠

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. The solution to this problem is to create test steps "defensively" and assume a slow trip to the server that will not complete in time for the following test step. You must check that a given option already exists in the list before trying to pick it.

10.6 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.

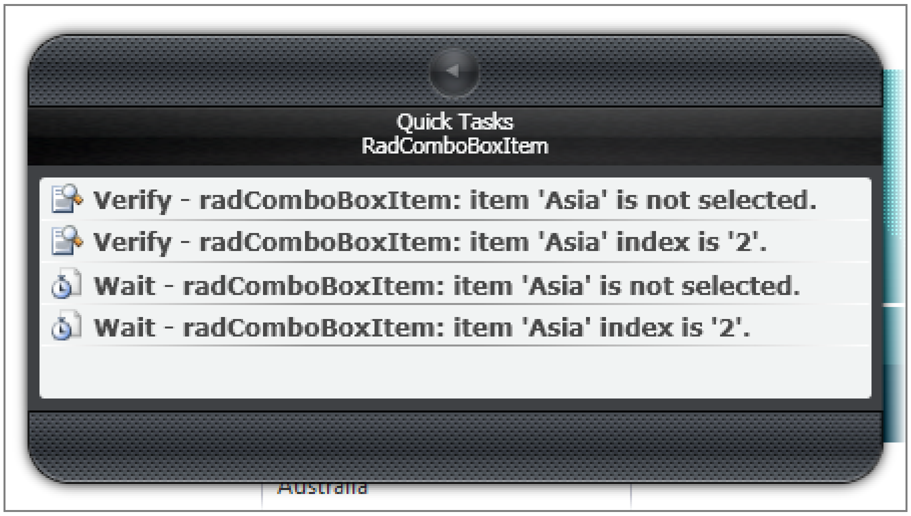
In the ASP.NET AJAX example, we looked at the outer markup of a "<SELECT>" (i.e., a drop down list) to see if options had been loaded. If we found an option we could assume the list was loaded from the server. The translator for the RadComboBox lets you simply check the count of items. The screenshot 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).



Quick Tasks
RadComboBox

Verify - radComboBox: text is '- Select a continent -'.
Verify - radComboBox: selected index is '0'.
Verify - radComboBox: item count is '7'.
Verify - radComboBox: drop down is not opened.
Wait - radComboBox: text is '- Select a continent -'.
Wait - radComboBox: selected index is '0'.
Wait - radComboBox: item count is '7'.
Wait - radComboBox: drop down is not opened.

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.



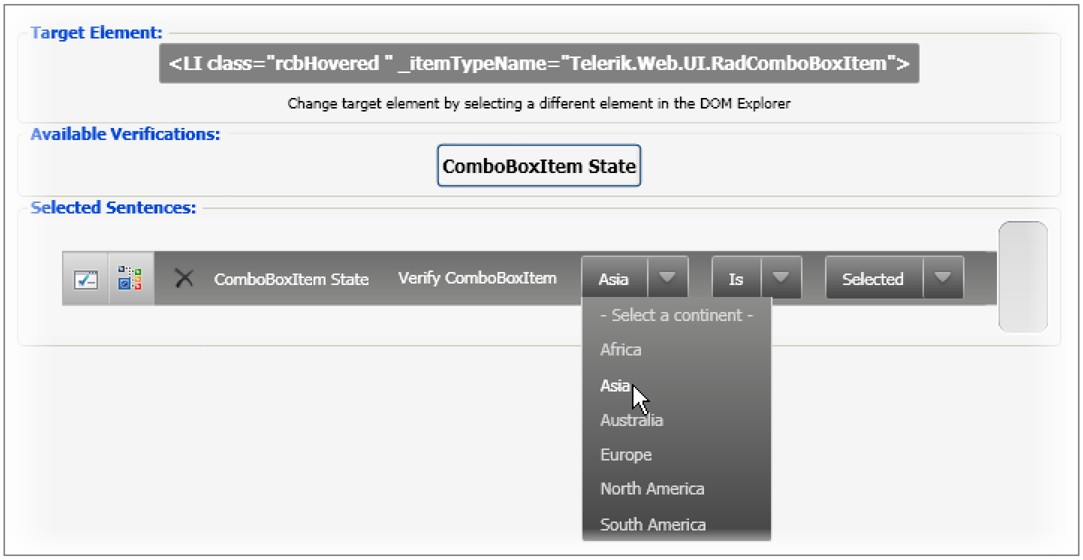
Verify - radComboBoxItem: item 'Asia' is not selected.

Verify - radComboBoxItem: item 'Asia' index is '2'.

Wait - radComboBoxItem: item 'Asia' is not selected.

Wait - radComboBoxItem: item 'Asia' index is '2'.

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.



Target Element:
<LI class="rcbHovered " _itemTypeName="Telerik.Web.UI.RadComboBoxItem">

Change target element by selecting a different element in the DOM Explorer

Available Verifications:

Selected Sentences:

ComboBoxItem State | Verify ComboBoxItem | Asia | Is | Selected

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.

10.7 Silverlight

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.

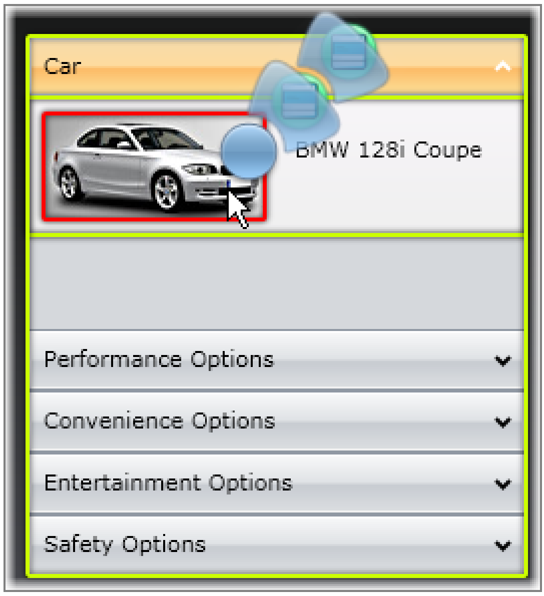
WebUI is the first scriptless record and playback solution for Silverlight. With WebUI you can build a single test case that interacts with both HTML and Silverlight elements, even on the same page. WebUI allows you to test applications that have heavy interaction between HTML, AJAX and Silverlight,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

WebUI features a consistent user interface that makes testing HTML and Silverlight elements substantially similar. Once the tester is familiar with the WebUI user interface, learning to test Silverlight elements is a short learning curve.

Silverlight Basics

Although you don't have to be thoroughly familiar with the inner workings of Silverlight to use WebUI 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"). 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.



.

WebUI and Silverlight

How Does WebUI 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 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 need to wait for an element to exist and be visible, but Silverlight elements may also fly-in, fly-out, expand, collapse or animate. WebUI 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.
- **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.

Walk Through

The Telerik Silverlight demo page is ideal for seeing how Silverlight testing differs from standard HTML web pages or AJAX. The Silverlight plug-in must load to the page, the page itself must load and a number of animations occur. In this example we will 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.

Find the materials for this Topic at...

\Projects\Silverlight

1) From the Welcome Screen, click the **Record Test** button.

a) In the Recording Surface, enter "http://demos.telerik.com/silverlight/#ComboBox/FirstLook" to the browser address bar and then press the **ENTER** key. This will load the RadControls for Silverlight demo application to the RadComboBox "First Look" example.

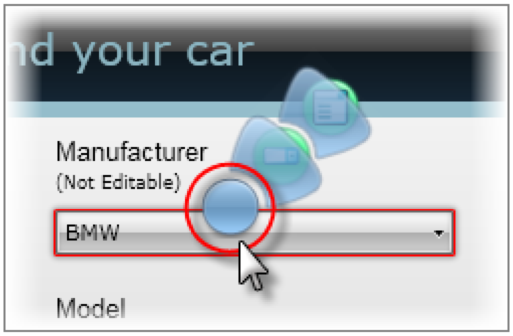
This step may take a minute as WebUI detects Silverlight and loads the page.

2) Click the Highlighting button to enable it.

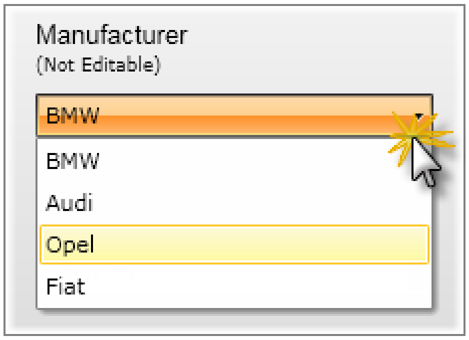
Notes

Enable the Highlighting button any time you need to use the Elements Menu and disable the Highlighting button whenever you feel it is in the way.

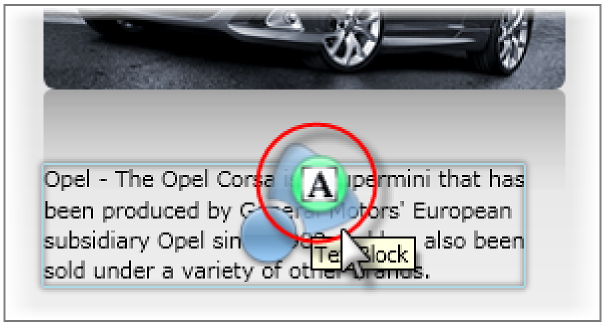
3) Hover the mouse above the car description until the Nub displays, click the Nub, then select the option for "Wait - Element Visibility is Visible".



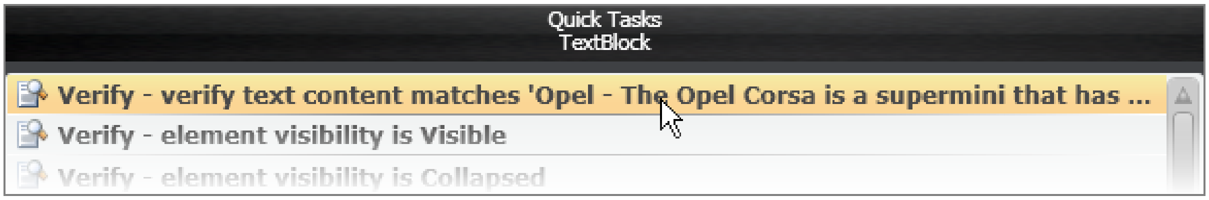
4) Click the drop down arrow of the Manufacturer combo box. Select the "Opel" item from the list.



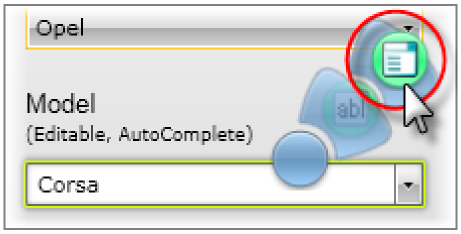
5) Hover the mouse above the car description until the Nub displays, then click the TextBlock leaf.


Opel - The Opel Corsa i permini that has
been produced by G eral Motors' European
subsidiary Opel sin also been
sold under a variety of othe s.
A
Te lock

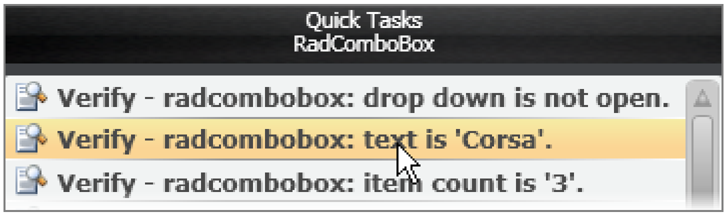
6) Double click the Quick Tasks item that starts "Verify - verify text content matches 'Opel - The Opel Corsa is a supermini that has...". This will add the verification as a test step.


Quick Tasks
TextBlock
Verify - verify text content matches 'Opel - The Opel Corsa is a supermini that has ...
Verify - element visibility is Visible
Verify - element visibility is Collapsed

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



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


Quick Tasks
RadComboBox

- Verify - radcombobox: drop down is not open.
- Verify - radcombobox: text is 'Corsa'.
- Verify - radcombobox: item count is '3'.

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

10)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.

11)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.

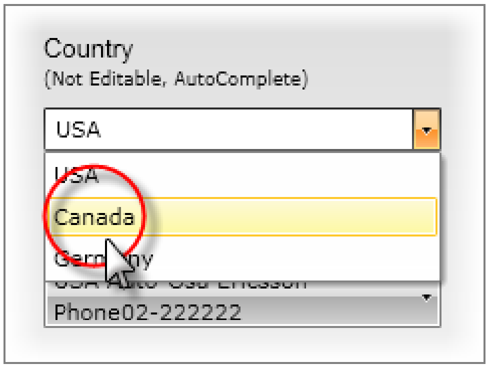
12)Hover the mouse above the "Model" drop down until the Nub displays and click the RadComboBox leaf.

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

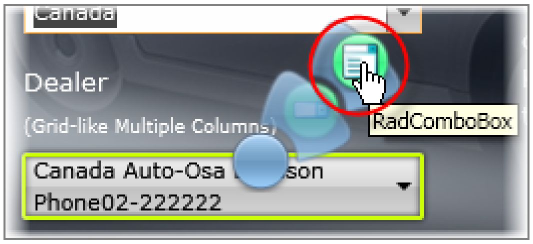
14)Hover the mouse above the car description until the Nub displays and click the TextBlock leaf.

15)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.

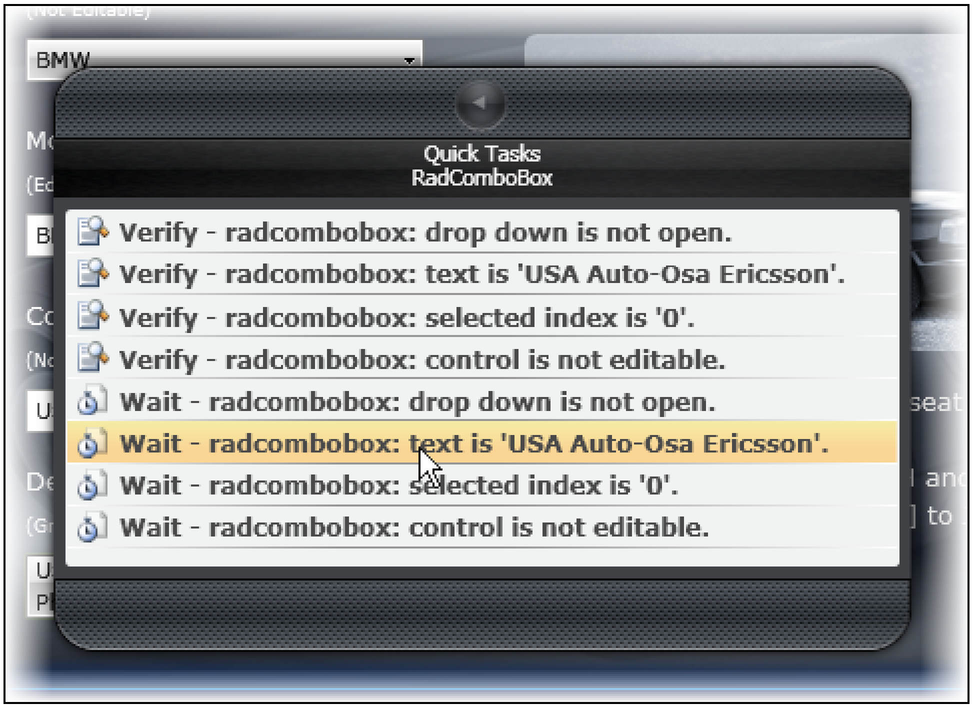
16)Click the "Country" drop down arrow to open the list. Select the "Canada" item.



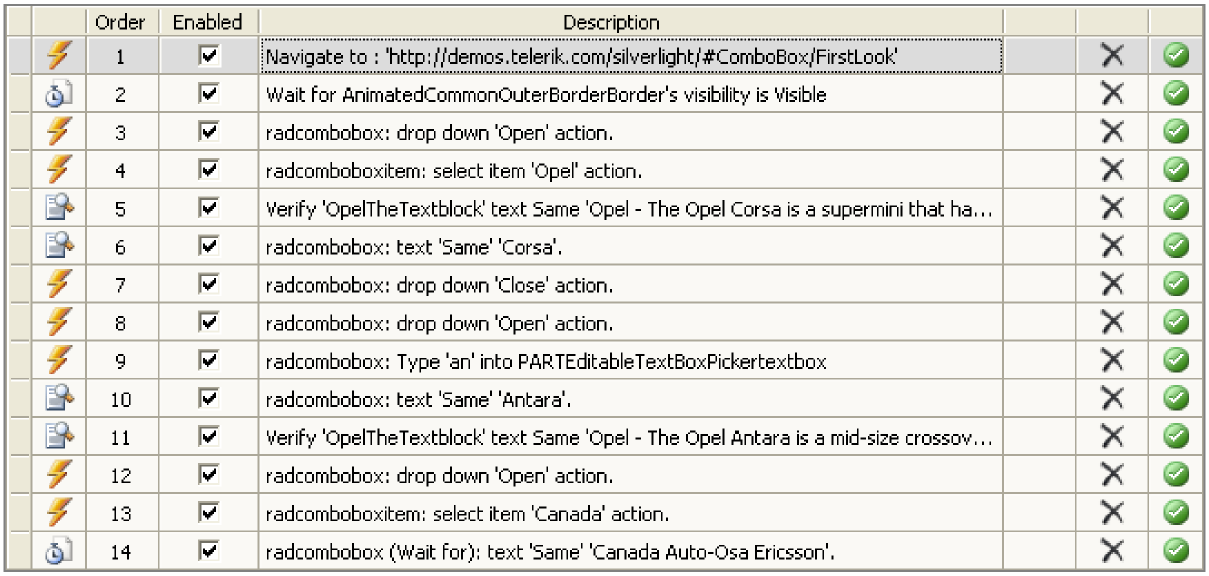
17)Hover the mouse above the "Dealer" drop down until the Nub displays and then click the RadComboBox leaf.



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


BMW
Quick Tasks
RadComboBox
Verify - radcombobox: drop down is not open.
Verify - radcombobox: text is 'USA Auto-Osa Ericsson'.
Verify - radcombobox: selected index is '0'.
Verify - radcombobox: control is not editable.
Wait - radcombobox: drop down is not open.
Wait - radcombobox: text is 'USA Auto-Osa Ericsson'.
Wait - radcombobox: selected index is '0'.
Wait - radcombobox: control is not editable.

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



	Order	Enabled	Description		
⚡	1	☑	Navigate to : 'http://demos.telerik.com/silverlight/#ComboBox/FirstLook'	✕	✓
⏱	2	☑	Wait for AnimatedCommonOuterBorderBorder's visibility is Visible	✕	✓
⚡	3	☑	radcombobox: drop down 'Open' action.	✕	✓
⚡	4	☑	radcomboboxitem: select item 'Opel' action.	✕	✓
📝	5	☑	Verify 'OpelTheTextblock' text Same 'Opel - The Opel Corsa is a supermini that ha...	✕	✓
📝	6	☑	radcombobox: text 'Same' 'Corsa'.	✕	✓
⚡	7	☑	radcombobox: drop down 'Close' action.	✕	✓
⚡	8	☑	radcombobox: drop down 'Open' action.	✕	✓
⚡	9	☑	radcombobox: Type 'an' into PARTEditableTextBoxPickertextbox	✕	✓
📝	10	☑	radcombobox: text 'Same' 'Antara'.	✕	✓
📝	11	☑	Verify 'OpelTheTextblock' text Same 'Opel - The Opel Antara is a mid-size crossov...	✕	✓
⚡	12	☑	radcombobox: drop down 'Open' action.	✕	✓
⚡	13	☑	radcomboboxitem: select item 'Canada' action.	✕	✓
⏱	14	☑	radcombobox (Wait for): text 'Same' 'Canada Auto-Osa Ericsson'.	✕	✓

Gotcha!

If a step fails because an element isn't found and you can see that the element is there on the page, here are some possibilities to help troubleshoot the problem:

- **The element doesn't exist**. Add a "Wait" to make sure that the element exists before you try to use it.
- **The element doesn't exist *yet***. Adjust the WaitOnElementsTimeout property upward to make sure that the element has time to load. The wait state is efficient, so if the element loads before the timeout, WebUI will continue immediately.
- **The element isn't visible**. Use the Elements Menu Quick Tasks to add a wait until the element is visible.
- **The element is still moving**. Again, use the Elements Menu Quick Tasks to add a wait until the element is not moving.

10.8 Summary

In this chapter you learned about technologies that make web applications easy to work with as a user, but also make testing more difficult as a Quality Assurance engineer. You learned how WebUI handles asynchronous updates in the browser, timing issues and Silverlight animation. In particular, you learned how WebUI can efficiently wait for a set of conditions to occur.

Part XI

Data Driven Tests

11 Data Driven Tests

11.1 Objectives

In this chapter you will learn how to drive your tests using data sources. You will learn how to add a data source, connect the data source to a test and how to bind data to properties in the test. First, you will use the built-in grid to build simple data-driven tests without needing an external data source. You will also learn how to drive tests with external data from spreadsheet files, comma delimited files, XML and database tables.

11.2 Overview

Using the WebUI 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 just about anything including Oracle, MS SQL, Access and ODBC.

Notes

Not all of these database types are right out of the box. For some database types, you may need to do some research and install a "data source provider".

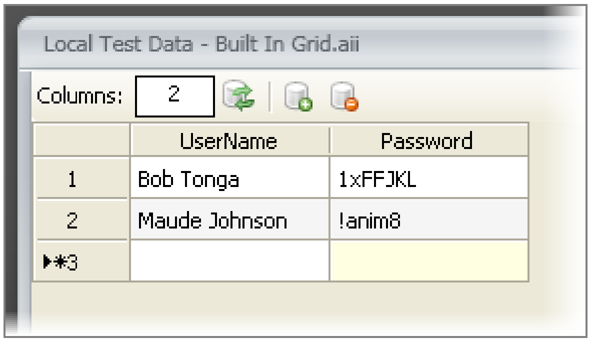
What we mean by "driving" a test with data is that we let the test know where a table of data exists and then use the data wherever required in the test. For example, if we have a test of a login dialog, the data will be a user name and password. If we are driving a test of several web searches, we could 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

11.3 The Built-In Grid

The built-in grid is available on the Record Tab and allows you to build simple, ad-hoc, data-driven tests without needing to connect to an external data source. The Screen below shows example data for a login test.



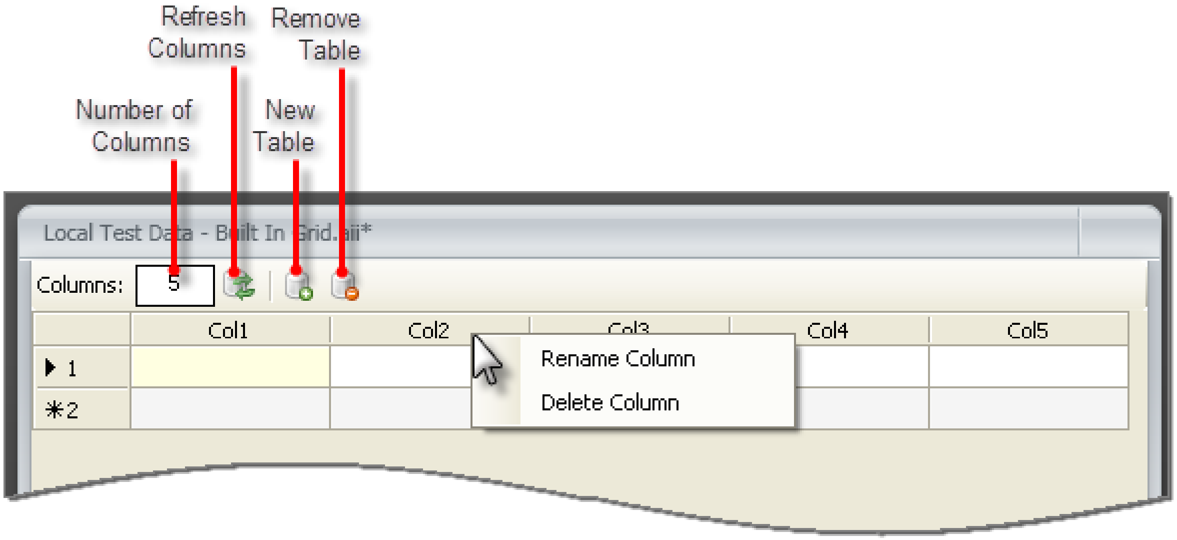
Local Test Data - Built In Grid.aii

Columns: 2

	UserName	Password
1	Bob Tonga	1xFFJKL
2	Maude Johnson	!anim8
▸*3		

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 built-in grid 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.



Local Test Data - Built In Grid.aii*

Columns: 5

	Col1	Col2	Col3	Col4	Col5
▸ 1					
*2					

Walk Through

This example tests against browser search results. The first part of this walk through compares against a predetermined "hard-coded" string. The second part of the walk through uses values from the built-in grid to use in the comparison.

Find the materials for this Topic at...

\Projects\DataDriven

The Test Without Data

This first part of the walk through creates a very simple test that enters a search in Google main page and verifies the result.

1. In the Project Tab, click the **Record New Test** button. This will create a new test and project automatically. WebUI will navigate to the Record Tab and the Recording Surface will display alongside WebUI.

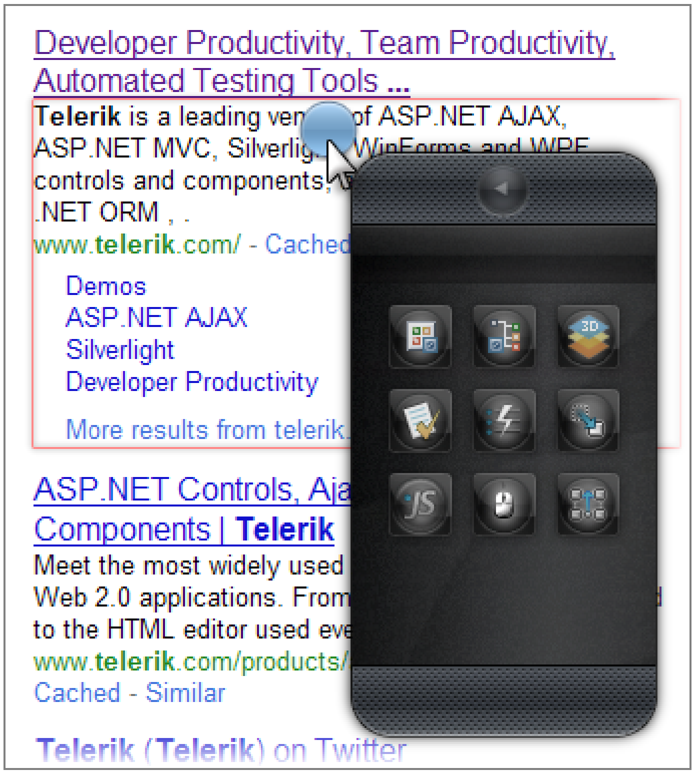
2. In the Recording Surface, type the url "www.google.com" into the address line and press the **ENTER** key.

3. In the search text box enter "Telerik".

4. Press the **ENTER** key

5. In the Recording Surface, press the Highlighting button .

6. Hover the mouse over the first returned search text until the Nub appears. Click the Nub to display the Elements Menu.



Developer Productivity, Team Productivity,
Automated Testing Tools ...
Telerik is a leading ven of ASP.NET AJAX,
ASP.NET MVC, Silverlig WinForms and WPF
controls and components,
.NET ORM , .
www.**telerik**.com/ - Cached

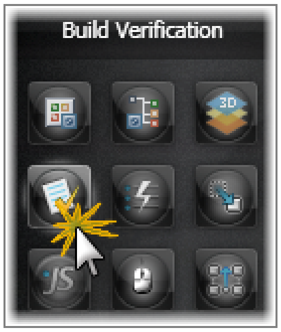
Demos
ASP.NET AJAX
Silverlight
Developer Productivity

More results from telerik.

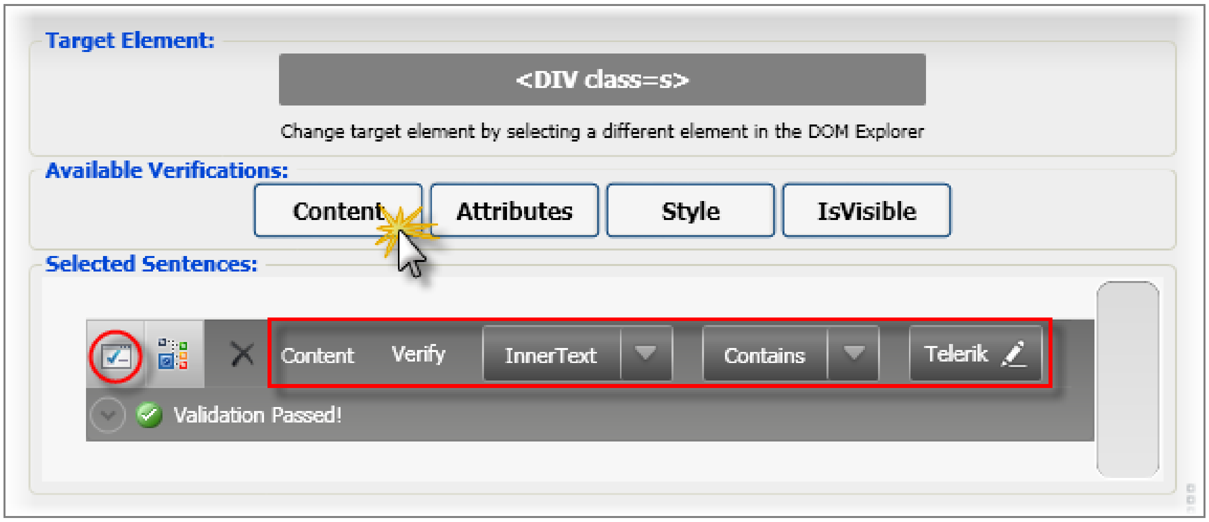
ASP.NET Controls, Aja
Components | **Telerik**
Meet the most widely used
Web 2.0 applications. From
to the HTML editor used eve
www.**telerik**.com/products/
Cached - Similar

Telerik (**Telerik**) on Twitter

7. Click the Elements Menu Build Verification button.



8. In the Sentence Verification Builder, click the **Content** button to add a verification sentence. Set the verification to "InnerText Contains Telerik". Click the check mark button on the left to validate the rule. Click the **OK** button to add the verification to the test steps.


Target Element:

<DIV class=s>

Change target element by selecting a different element in the DOM Explorer

Available Verifications:

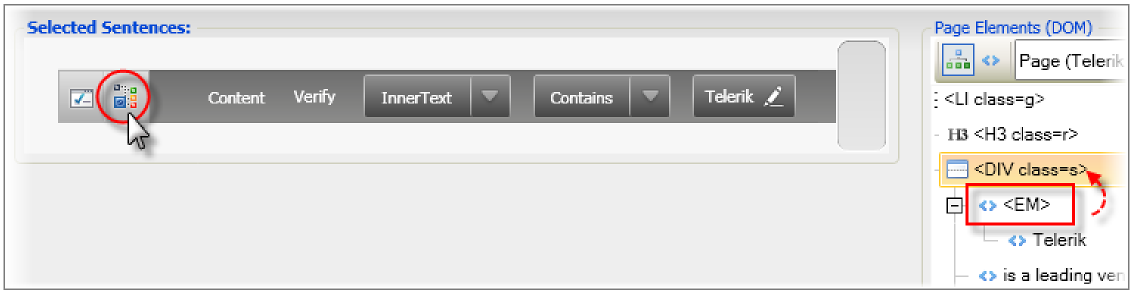
Selected Sentences:

Content Verify InnerText Contains Telerik

Validation Passed!

Tip!

Sometimes the element in the Sentence Verification Builder can be too specific. That is, the Recording Surface may have selected an element *within* a block of text you wanted to verify, not the entire block of text. To get around this, click the **DOM Explorer button** and select the element's parent element in the DOM tree. The screenshot below shows where we had "<EM>Telerik</EM>" selected and then select the "<DIV>" element that contains it.



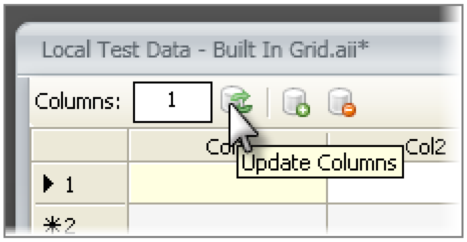
9. Back in the WebUI Record Tab, click the Execute button exercise the test. All steps should pass.

Driving the Test With Data

This second walk through extends the previous example. We will create a table of search comparisons using the built-in grid and put the data in place of the hard coded "Telerik".

1. Click the Record Tab **Local Data** button.

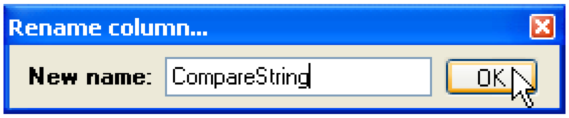
2. Enter "1" in the **Columns** edit box and click the **Update Columns** button. Click the **Yes** button to accept and close the confirmation dialog.



Local Test Data - Built In Grid.aii*

Columns: 1

3. 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.

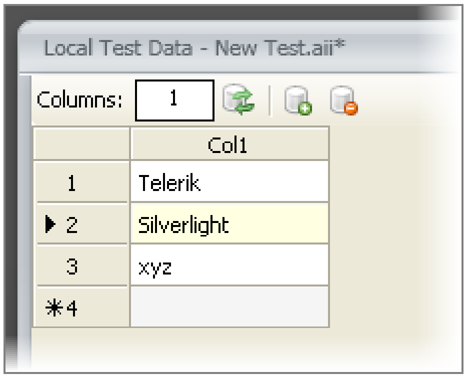


Rename column...

New name: CompareString

4. In the first row of the table, enter "Telerik" in the "CompareString" column and press **Enter**.

5. Add two more rows with text "Silverlight" and "xyz". The built in grid should look like the screenshot below.



Local Test Data - New Test.aii*

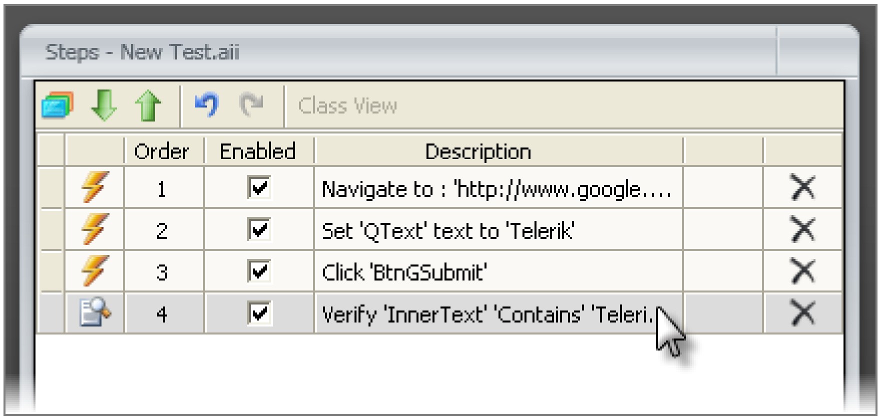
Columns: 1

	Col1
1	Telerik
2	Silverlight
3	xyz
*4	

Now that the data is defined, you need to bind it to some property in the test.

6. Click the **Steps** button.

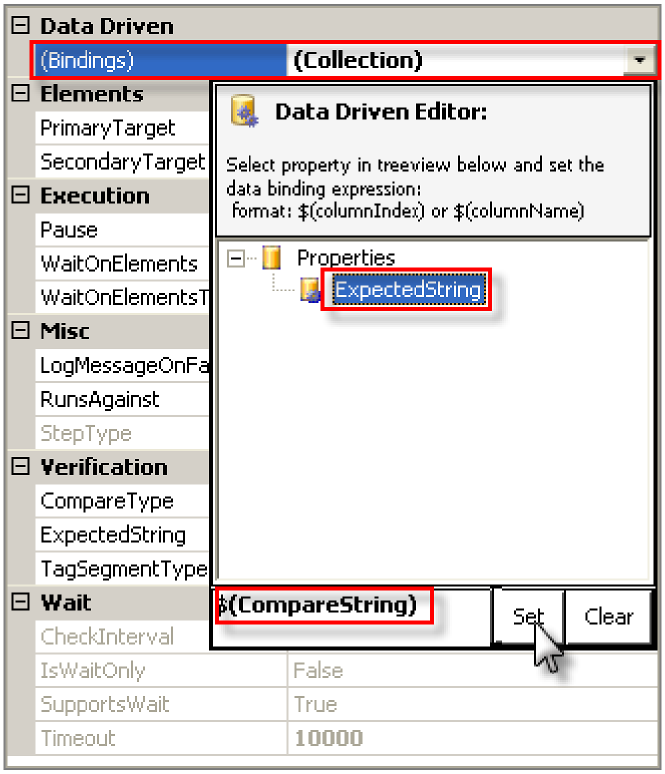
7. Select the verify step at the end of the test.



Steps - New Test.aii

Order	Enabled	Description	
1	☑	Navigate to : 'http://www.google....	✕
2	☑	Set 'QText' text to 'Telerik'	✕
3	☑	Click 'BtnGSubmit'	✕
4	☑	Verify 'InnerText' 'Contains' 'Teleri...	✕

8. In the Properties pane, locate the **Data Driven | Bindings** property and drop down the collection editor. Select the **ExpectedString** property from the tree view, enter "$(CompareString)" in the text box and finally, click the **Set** button.



Data Driven
(Bindings) (Collection)
Elements
PrimaryTarget
SecondaryTarget
Execution
Pause
WaitOnElements
WaitOnElementsT
Misc
LogMessageOnFa
RunsAgainst
StepType
Verification
CompareType
ExpectedString
TagSegmentType
Wait
CheckInterval
IsWaitOnly False
SupportsWait True
Timeout 10000

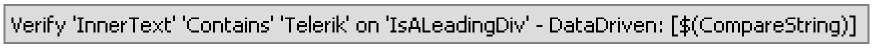
Data Driven Editor:

Select property in treeview below and set the data binding expression:
format: $(columnIndex) or $(columnName)

Properties
ExpectedString

$(CompareString)

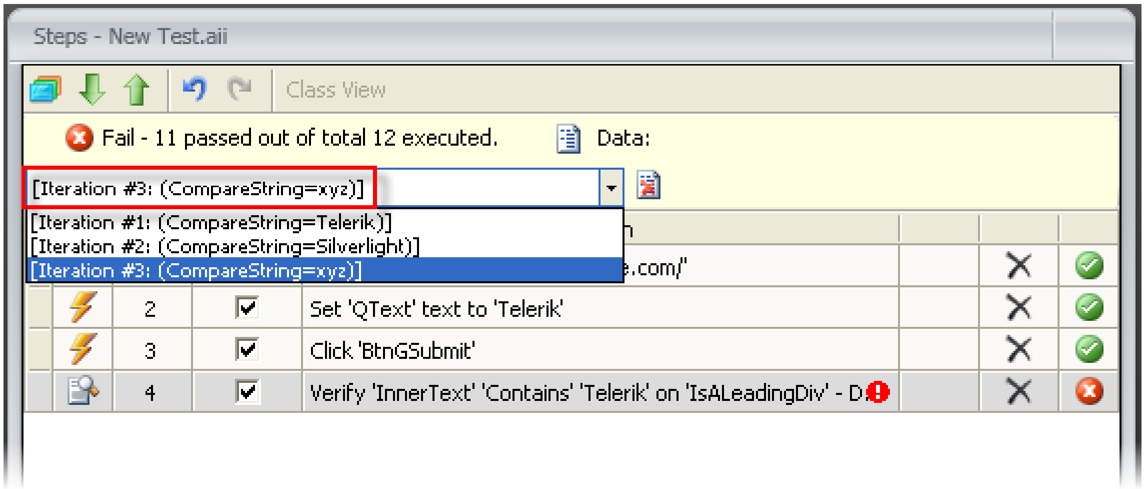
This step will feed data to the **ExpectedString** property of the test step (that originally had the hard-coded "Telerik" value) from the "CompareString" column of the built in grid. In the Steps Pane, the test step description is changed to add "DataDriven" and "$CompareString" as shown in the screenshot below.



Verify 'InnerText' 'Contains' 'Telerik' on 'IsALeadingDiv' - DataDriven: [$(CompareString)]

9. Save the project and then click the **Execute** button. The test steps should execute three times, one for each row in the built in grid.

10. View the summary results which should read "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 "Telerik" and "Silverlight", 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.



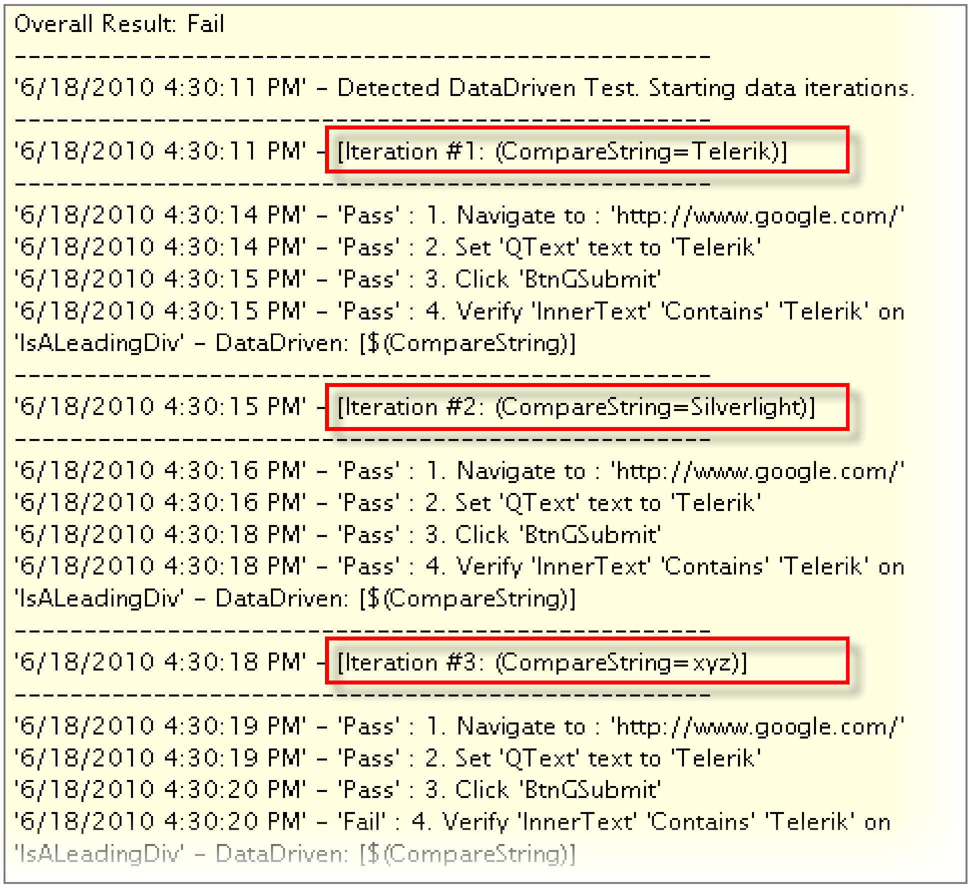
Fail - 11 passed out of total 12 executed. Data:

[Iteration #3: (CompareString=xyz)]

[Iteration #1: (CompareString=Telerik)]
[Iteration #2: (CompareString=Silverlight)]
[Iteration #3: (CompareString=xyz)]

			.com/'	✗	✓
2	☑	Set 'QText' text to 'Telerik'		✗	✓
3	☑	Click 'BtnGSubmit'		✗	✓
4	☑	Verify 'InnerText' 'Contains' 'Telerik' on 'IsALeadingDiv' - D		✗	✗

11. If you read the log you'll see the data used to drive each iteration:



```
Overall Result: Fail
------------------------------------------------------
'6/18/2010 4:30:11 PM' - Detected DataDriven Test. Starting data iterations.
------------------------------------------------------
'6/18/2010 4:30:11 PM' - [Iteration #1: (CompareString=Telerik)]
------------------------------------------------------
'6/18/2010 4:30:14 PM' - 'Pass' : 1. Navigate to : 'http://www.google.com/'
'6/18/2010 4:30:14 PM' - 'Pass' : 2. Set 'QText' text to 'Telerik'
'6/18/2010 4:30:15 PM' - 'Pass' : 3. Click 'BtnGSubmit'
'6/18/2010 4:30:15 PM' - 'Pass' : 4. Verify 'InnerText' 'Contains' 'Telerik' on
'IsALeadingDiv' - DataDriven: [$(CompareString)]
------------------------------------------------------
'6/18/2010 4:30:15 PM' - [Iteration #2: (CompareString=Silverlight)]
------------------------------------------------------
'6/18/2010 4:30:16 PM' - 'Pass' : 1. Navigate to : 'http://www.google.com/'
'6/18/2010 4:30:16 PM' - 'Pass' : 2. Set 'QText' text to 'Telerik'
'6/18/2010 4:30:18 PM' - 'Pass' : 3. Click 'BtnGSubmit'
'6/18/2010 4:30:18 PM' - 'Pass' : 4. Verify 'InnerText' 'Contains' 'Telerik' on
'IsALeadingDiv' - DataDriven: [$(CompareString)]
------------------------------------------------------
'6/18/2010 4:30:18 PM' - [Iteration #3: (CompareString=xyz)]
------------------------------------------------------
'6/18/2010 4:30:19 PM' - 'Pass' : 1. Navigate to : 'http://www.google.com/'
'6/18/2010 4:30:19 PM' - 'Pass' : 2. Set 'QText' text to 'Telerik'
'6/18/2010 4:30:20 PM' - 'Pass' : 3. Click 'BtnGSubmit'
'6/18/2010 4:30:20 PM' - 'Fail' : 4. Verify 'InnerText' 'Contains' 'Telerik' on
'IsALeadingDiv' - DataDriven: [$(CompareString)]
```

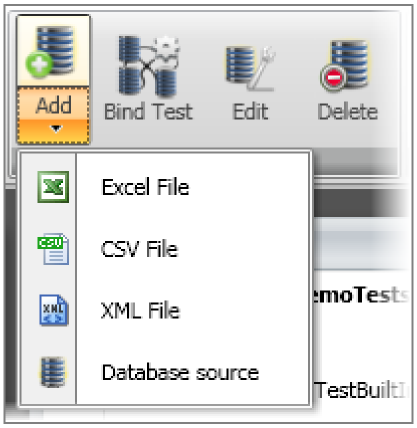
11.4 Connecting to External Data

You're not restricted to the built-in grid. Out of the box, you can connect to standard "*.csv" comma delimited 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.

The main steps to drive a test with data are:

- Add a data source.
- Bind the data source to a test as a whole
- Bind a specific piece or column of data to a property in the test.

At any time you can add a new data source from the **Project Tab**, **Add** button's drop down list. You can select Excel, comma delimited text (CSV), XML or a true database source. Selecting any of these will display the **Create New Data Source** dialog that will lead you through the process of defining where your data is coming from.



"Binding" associates a data source with a particular test. When you click the **Bind Test** button, the **Bind Test to Data Source** dialog walks you through choosing and configuring a data source. The **Edit** button lets you reconfigure an existing data source. The **Delete** button completely removes the data source permanently.

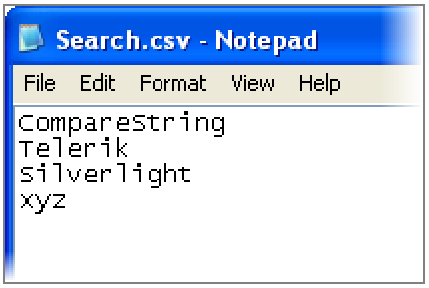
Binding a particular property is done with the **Bindings** property described in the "Built-In Grid" Topic.

Tip!

If you supply a data source with the same column names as the built-in grid example, you don't have to change the test steps.

Spreadsheet Files

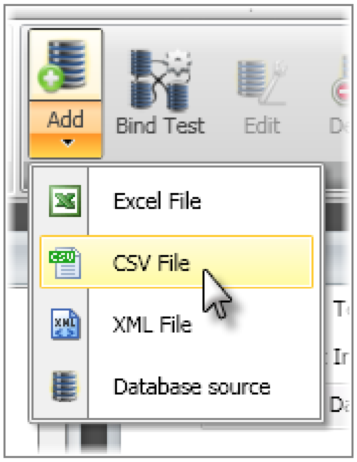
We can drive the entire test using an external spreadsheet with either a Comma Separated Value file (*.csv) or an Excel format file (*.xls, *.xlsx). Both types of files can be created in Excel. CSV files can actually be created in any spreadsheet application or directly in Notepad if you follow the *.csv formatting conventions (each line with comma separated values, the same number of commas in each line). In this example we will use a CSV file with the same "CompareString" rows as the "Built-in Grid" example. The data for a single column CSV file is shown in the screenshot below.



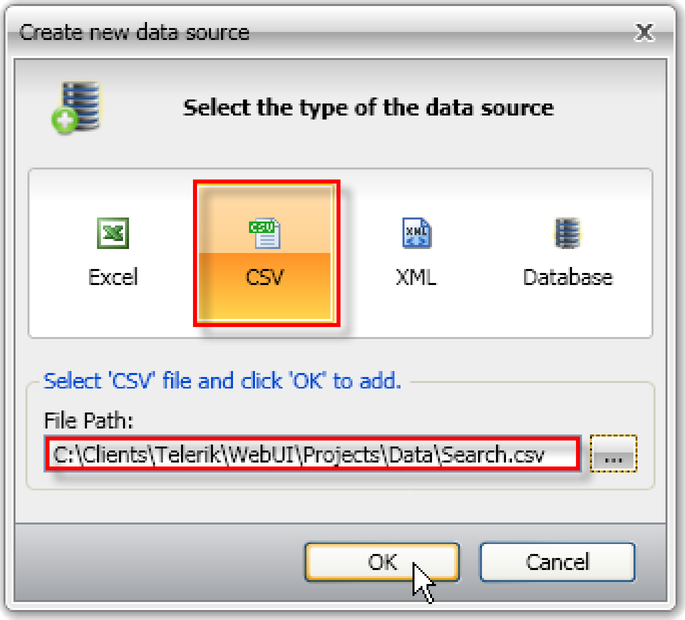
```
CompareString
Telerik
Silverlight
xyz
```

To bind a CSV file:

1. Click the **Add** button to drop down the list of data source choices. Click **CSV File** from the drop down menu. This action will display the "Create new data source" dialog.

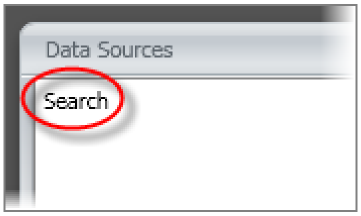


2. In the "Create new data source" dialog, select the "CSV" data source type icon. Use the browse button ... to locate a "*.csv" file and click the **OK** button. In this example, the "Search.csv" file has the same data as the built-in grid example.

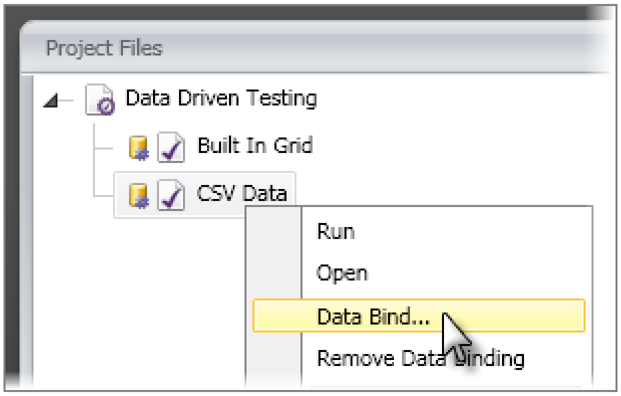

Create new data source
X
Select the type of the data source
Excel
CSV
XML
Database
Select 'CSV' file and click 'OK' to add.
OK
Cancel

File Path: C:\Clients\Telerik\WebUI\Projects\Data\Search.csv

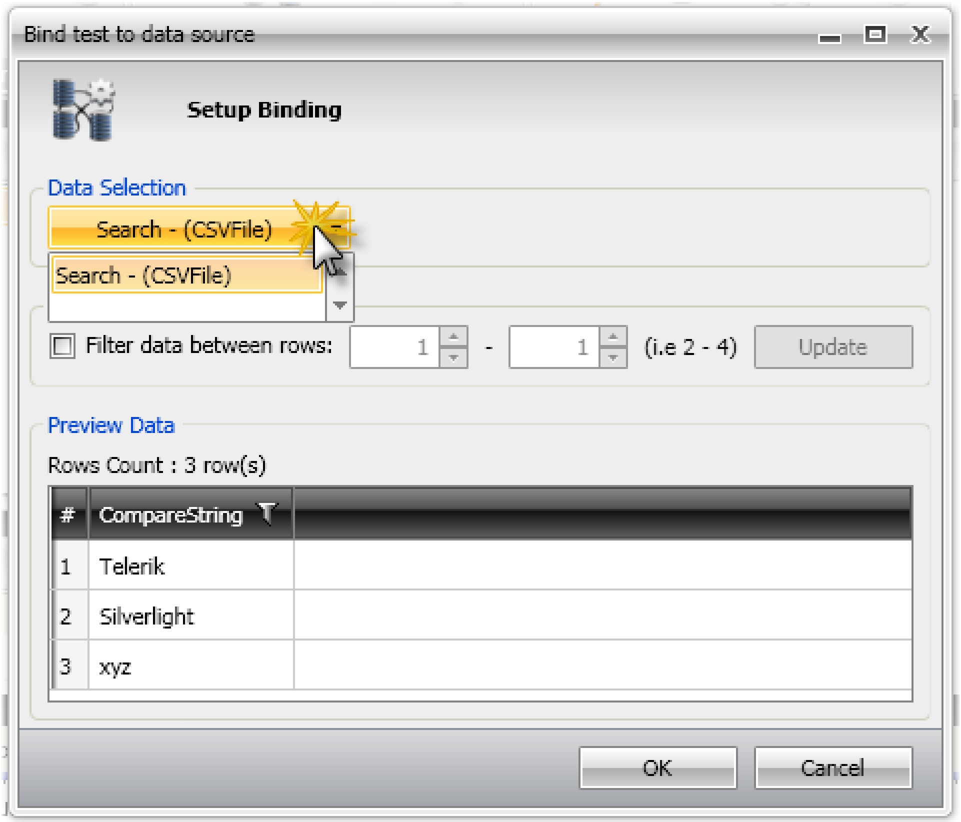
3. Notice that the data source has been added to the list:



4. Right click the test in the Project Files Pane and select **Data Bind...** from the context menu.. This will display the **Bind Data To Data Source** dialog.



5. In the **Bind Data To Data Source** dialog, drop down the **Data Selection** list and pick the CSV data source. Your data will display in the Preview Data section of the dialog. Click the **OK** button to bind the data source to the test and close the dialog.



Search - (CSVFile)

□ Filter data between rows: 1 - 1 (i.e 2 - 4)

Rows Count : 3 row(s)

#	CompareString
1	Telerik
2	Silverlight
3	xyz

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.

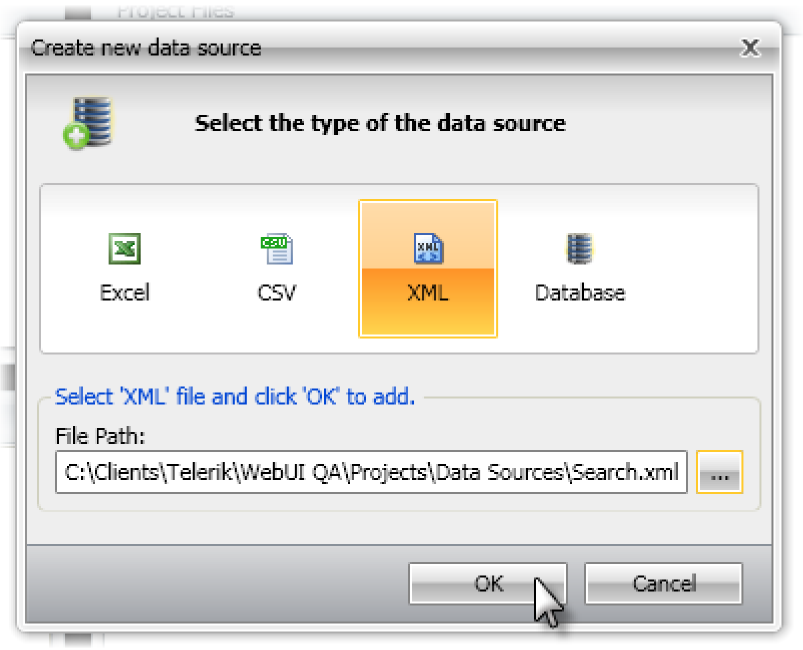
The test is now bound to the "Search.csv" file. You can rerun the test and get the same results as the "Built-in Grid" example, but in this case the data will be coming from the csv file.

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.

```xml
<Searches>
  <Search CompareString="Telerik" />
  <Search CompareString="Silverlight" />
  <Search CompareString="xyz" />
</Searches>
```

When you click the **OK** button on the Create New Data Source dialog the data source is added directly without preview.



Create new data source

Select the type of the data source

Select 'XML' file and click 'OK' to add.

File Path: C:\Clients\Telerik\WebUI QA\Projects\Data Sources\Search.xml

Once again, if the column names match the data bindings from the built-in test, then the test steps and property bindings don't need to be changed. Running this test should return the same results as the built-in grid example.

Database Tables

You can modify the data connection of the test to use the "Database" option and 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.

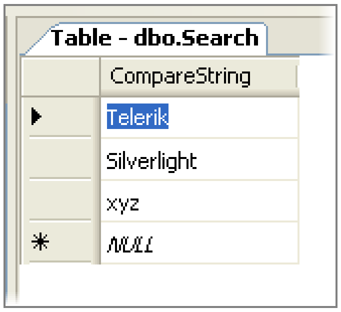


Table - dbo.Search

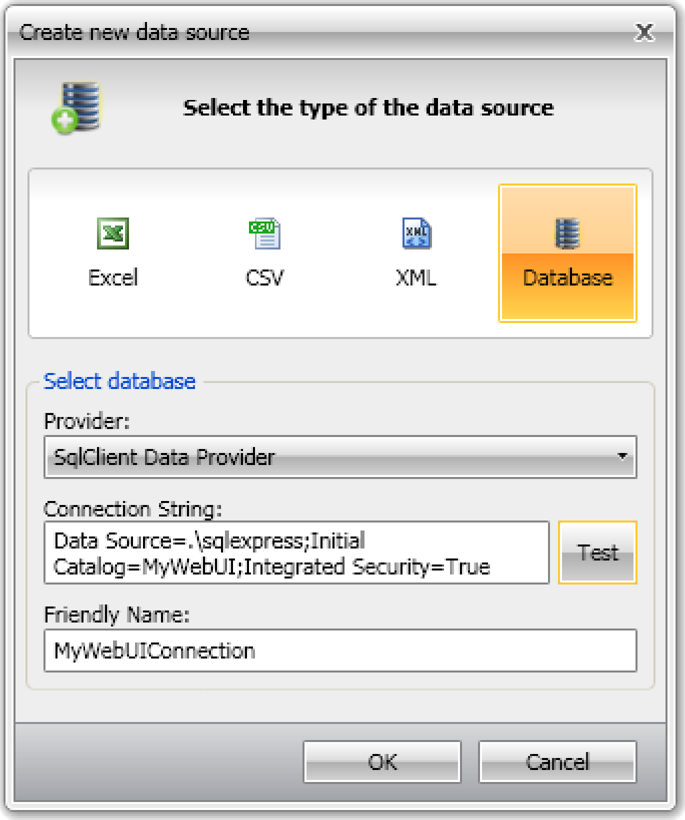
	CompareString
▶	Telerik
	Silverlight
	xyz
*	NULL

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('Telerik')
INSERT INTO Search VALUES('Silverlight')
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".



Provider: SqlClient Data Provider

Connection String: Data Source=.\sqlexpress;Initial Catalog=MyWebUI;Integrated Security=True

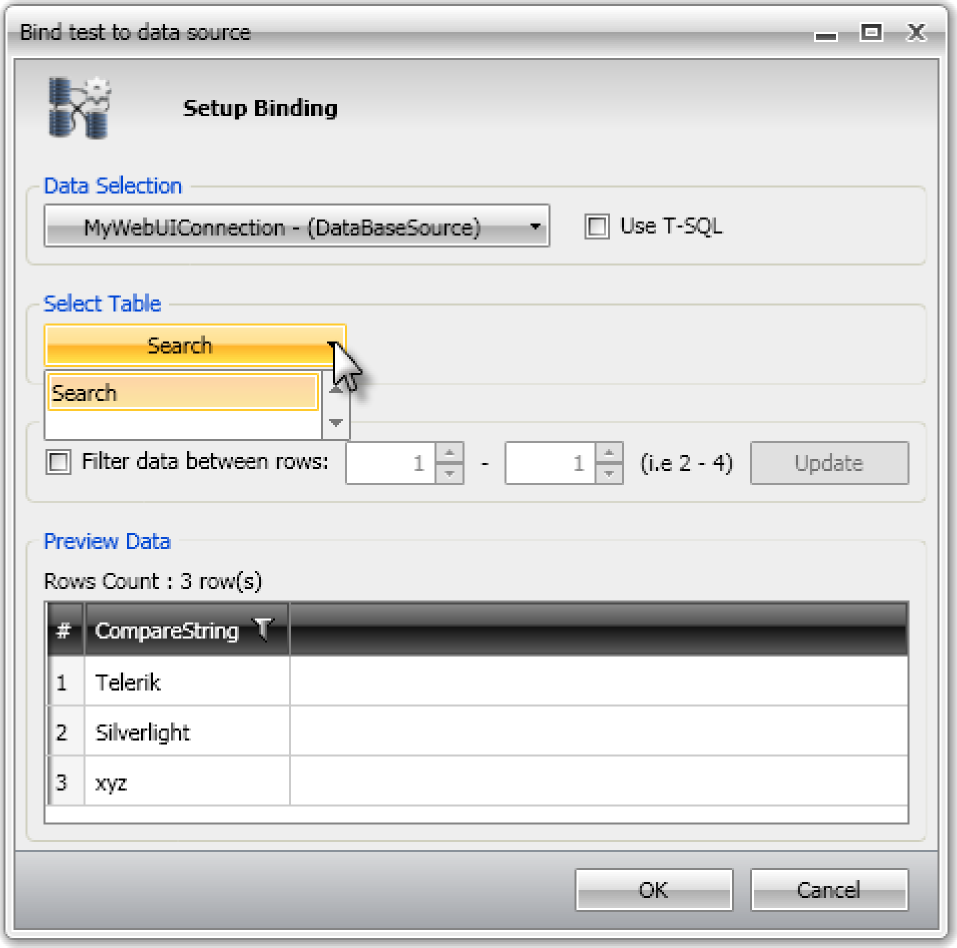
Friendly Name: MyWebUIConnection

The **Connection String** must be entered manually. Finally, enter a **Friendly Name** for the connection

Gotcha!

As of this writing, WebUI 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



Bind test to data source

Setup Binding

Data Selection

MyWebUIConnection - (DataBaseSource) □ Use T-SQL

Select Table

Search

Search

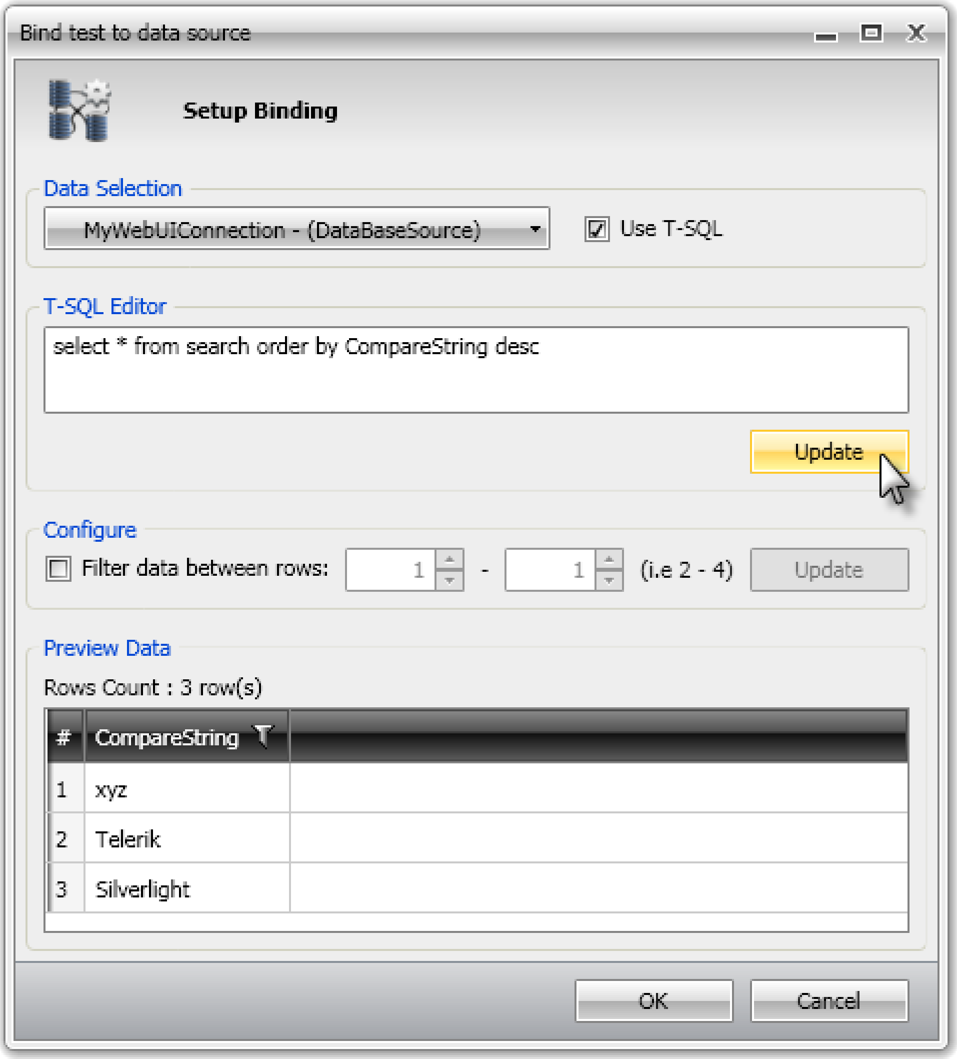
□ Filter data between rows: 1 - 1 (i.e 2 - 4) Update

Preview Data

Rows Count : 3 row(s)

#	CompareString
1	Telerik
2	Silverlight
3	xyz

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 reverse alphabetical order.



MyWebUIConnection - (DataBaseSource) ☑ Use T-SQL

```
select * from search order by CompareString desc
```

□ Filter data between rows: 1 - 1 (i.e 2 - 4)

Rows Count : 3 row(s)

#	CompareString
1	xyz
2	Telerik
3	Silverlight

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.

11.5 Summary

In this chapter you learned how to drive your tests using data sources. You learned how to add a data source, connect the data source to a test and how to bind data to properties in the test. First, you used the built-in grid to build simple data-driven tests without needing an external data source. You also learned how to drive tests with external data from spreadsheet files, comma delimited files, XML and database tables.

Part XII

About Telerik

12 About Telerik

12.1 History

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.

12.2 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.

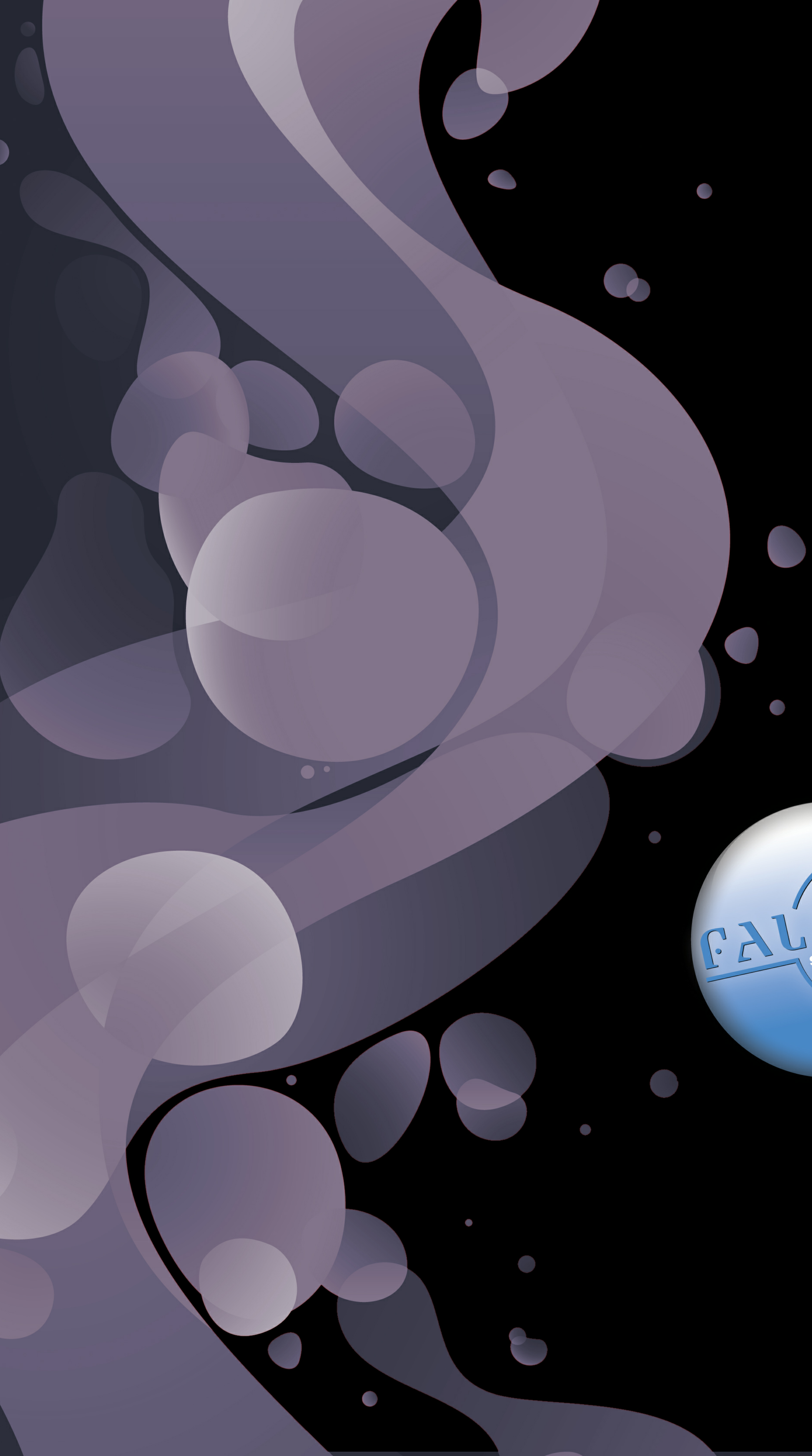
WebUI Consulting Express

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



FALAFEL software

www.falafel.com