ArtOfTest, Inc.

AUTOMATION DESIGN CANVAS 1.1 USER GUIDE

CONTENT

1 INTRODUCING AUTOMATION DESIGN CANVAS

Automation Design Canvas™ (Design Canvas) is the solution to many of today's tough test automation challenges faced by testers and developers who need to automate testing of Web 2.0 applications. Design Canvas is based on WebAii automation framework. For more information about WebAii automation framework for web testing go to http://artoftest.com/webaiifxproduct.aspx and http://www.artoftest.com/Resources/WebAii/Documentation/topicsindex.aspx

Design Canvas was developed after listening to and learning from the industry and WebAii customers; it solves tough web automation challenges. The ArtOfTest™ team analyzed numerous WebAii unit tests for repetitiveness and what causes them to be vulnerable and prone to failure. ArtOfTest determined what were the most common time consuming tasks when solving test failures and helps you solve these directly by using Design Canvas. Design Canvas was created to not only help you build automation *faster* but also help build *better* more *robust automation*. ArtOfTest does not claim to have solved every problem but believes that Design Canvas is a giant step forward in advancing test automation technology. Say farewell to the Dark Ages of web automation tools and say Hello to the future. ArtOfTest creators know that this tool will make your Web 2.0 testing tasks easier! Soon you will love Design Canvas as much as ArtOfTest customers do! For more information about Automation Design Canvas for web testing go to: http://artoftest.com/webaiidcproduct.aspx

1.1 About this Document

The Automation Design Canvas User Guide provides guidance on understanding and learning how to use Automation Design Canvas. It is intended to help both beginner and more advanced software testers automate the testing of their web sites and web applications. No prior test automation experience is needed in order to set-up, create, and run Design Canvas tests. Familiarity with Visual Studio® 2008 (VS) is helpful but not essential. If you need to customize automated tests then knowledge of C# or Visual Basic is helpful.

1.2 Why Use Automation Design Canvas

The following are just a few of the Design Canvas benefits you will notice immediately:

- **Build test automation in minutes vs. hours**. You will shave weeks off your release schedules without creating code or writing scripts.
- **Design Canvas supports rich Web 2.0 applications.** It uses an advanced designer surface that makes test automation as easy as point and click, allowing for seamless crafting of verifications and synchronizations for dynamic AJAX pages.
- **Simple but powerful**. You will not need to wade through large user manuals or take weeks of training to become an expert at Design Canvas. A quick start guide is all you need to start building test automation like a professional. See http://artoftest.com/webaiidcproduct.aspx

then select **Quick Start Guide** from the lower left hand navigation panel. Design Canvas is highly customizable allowing you to alter how recording is accomplished.

- **Built with maintenance in mind**. One of the most difficult aspects of test automation is that it must be constantly maintained. It must be updated as the product being tested or the test environment changes. Design Canvas has multiple built-in features that allow you to easily update tests, rapidly resolve test failures, and track how product changes affect your tests.
- **Visual Studio Team System integration**. Automation Design Canvas is integrated into Visual Studio (VS) Professional and Visual Studio Team System (VSTS) 2008. It adds another test type called "WebAii". It meshes with the existing test life cycle management that VS offers including test case management, test execution, source control, remote execution, and reporting using Team Foundation Server (TFS).
- **Scriptless/codeless recording**. To generate most of your automated tests all you need is to navigate using point and click. Run the scenario once in Design Canvas and an automated test generates. Then with Quick Execution, run the recorded test against Internet Explorer® (IE) or Mozilla Firefox® (Firefox) to catch product regressions and bugs. Each recorded step has properties that allow you to update and customize it without re-recording.
- **Optional Code-behind Model**. The optional code-behind model allows you to fully customize your tests as needed. You can write your own custom code methods in either C# or VB.NET, program it do whatever you need and have Design Canvas execute your custom method as a step in your test.

1.3 Automation Design Canvas Features

Here are the major Design Canvas features:

1.3.1 Design Canvas Recorder using Automation Overlay Surface™ (AOS)

Turn your webpage into a gallery of test automation tasks using Automation Overlay Surface™ (AOS). AOS allows you to highlight an element, locate it in the document object model (DOM) tree, and generate common verification and/or synchronization tasks. These tasks are automatically added to your test as a test step (for example, "Verify an item is selected in a dropdown" or "Wait for an element to become visible"). The task creators are initialized to the current state of the highlighted element. When you select an element an automation step is recorded against that element with the proper settings; you do not need to enter anything. Without Design Canvas testers must switch several times between the IE Developer Toolbar (or Firebug) and their recording tool. Section 5 describes how to use Design Canvas Recorder and AOS.

1.3.1.1 Visual Storyboard

As you record your test, a screenshot of your action on the target element is captured to the storyboard. This gives you a visual flow of your tests execution steps. This saves time as it helps

others understand the state of each test step at the time of recording along with what the test target elements are.

1.3.2 Sentence Based™ Verification User Interface

You can build your test rules as if you were writing a sentence. To make crafting verification and synchronization simple, the ArtOfTest team developed a Sentence Based™ user interface (UI) that guides you through creating verifications and test synchronization with elements. It is similar to an adaptive wizard that changes with the target element. Using the Sentence Based UI you can create a wide range of verification types such as attributes, styles, tables, select dropdowns, element visibility, and more. The Sentence Based UI guides you through crafting the verification criteria by first loading the state of the target element into the context of the rule being created. It then offers you verification criteria one step at a time until the verification step is complete. Section 6 provides more information about using the sentence based verification interface.

1.3.3 Web Element Abstraction

All webpage elements targeted for automation in your tests are abstracted out and filed in Elements Explorer. If there are multiple actions that use the same element, the element is referenced from Elements Explorer instead of being duplicated in the test. This abstraction saves significant test maintenance time and cost because now only need to maintain and update the one unique element in Element Explorer instead of multiple duplicate elements.

Another common significant web automation time-consuming task that testers complain about is determining how to uniquely locate a specific element on a page. In dynamic pages, this task is even more complex. Automation Design Canvas simplifies this task with an algorithm that automatically determines the best find parameters to use to locate a specific element on a specific page. The algorithm is configurable using Design Canvas settings. You can also create your own schemes and logic. Section 7 provides more information about using web element abstraction.

1.3.4 Data-Driven Testing

Design Canvas supports data-driven testing. All recorded steps (actions/verifications or synchronizations) come with data-driven properties that allow you to bind the steps to a data source. Design Canvas supports VSTS supported data sources (Database, CSV, and XML). It also has a built-in data grid that allows you to build your own data source in your test (without reverting to external sources) and manage external dependencies. Section 8 describes how to use data-driven testing.

1.3.5 Visual Studio Integration

Design Canvas allows WebAii tests to behave just like other built-in VSTS test types, such as Web Tests and Unit Tests. Also WebAii tests can be aggregated with other VSTS test types in the Run Manager, test lists, and test view. They act like other VS test types for participation in Source Control and Team Foundation Server (TFS) with Continuous Integration Server and remote execution and reporting. Section 9 describes how Design Canvas seamlessly integrates with Visual Studio.

1.3.5.1 Generating Unit Tests

Design Canvas enables you to take any of your recorded tests and generate coded unit tests. It supports generating NUnit as well as VS Unit tests. Unit tests can be generated in two modes:

- Leveraging Design Canvas to manage the elements. You can continue using Design Canvas to find elements and change/validate the 'find logic' of an element. Your unit test then performs the test logic as instructed by the recorded steps in your test.
- Enabling you to generate a fully self-contained unit test. These tests can be moved into other projects or run in isolation from your current test project.

1.3.6 Test Case Maintenance

Test case maintenance is time consuming. Design Canvas has several built-in features that target test failure-resolution of tests and test maintenance. For example, when a test fails because it cannot find an element Design Canvas helps to compare the page from the time of recording to the current failing page. This helps you quickly discover which change caused your failure. Also Design Canvas includes live validation for verification updates and 'element find parameter' updates hence you no longer need to repeatedly run tests over and over to validate your changes. Both these features save test maintenance time. Section 10 provides more details about test case maintenance.

1.3.7 Customizing and Extending Your Test Using Code

Automation Design Canvas supports a code-behind model for recorded tests. With this you can customize specific steps in a test using coded functions (in VB.NET or C#) which are automatically detected and added to Test Explorer. You can manage these steps in the same manner as your other recorded steps (that is, re-order them, enable them… and so on). Section 11 describes how you can exploit this Design Canvas test extension feature.

1.3.8 Generating Load Tests from WebAii Tests

Load tests can be auto-generated on demand. To generate a load test first select and convert a WebAii test into a VS Web Test using only one click - you can in turn use the generated VS Web Test to build your load test. This saves significant time as your WebAii tests can easily be used as the base for your load tests. By leveraging their ease of update and Elements Explorer

abstraction you will save a log of time compared to maintaining more cumbersome VS Web Tests. Section 12 describes how to generate load tests from WebAii tests.

1.4 Acronyms and Abbreviations

Acronym/Abbreviation	Description
AOS	Automation Overlay Surface™
DOM	Document Object Model
EULA	End user license agreement
HTML	HyperText Markup Language
IE	Internet Explorer
TFS	Team Foundation Server
UI	User Interface
URL	Universal Resource Locator
VS	Visual Studio
VSTS	Visual Studio Team System/Team Suite

1.5 References

The following items were either mentioned in this document or are listed here as additional resources:

Topic	URL
WebAii Quick Start Guide	http://artoftest.com/webaiidcproduct.aspx then select Quick Start Guide
WebAii How to videos	http://artoftest.com/webaiidcproduct.aspx
WebAii Reference	http://www.artoftest.com/Resources/WebAii/Documentation/topicsindex.aspx
JavaScript, DOM, XPATH, and CSS reference, examples, and tutorials	http://www.w3schools.com/jsref/jsref_events.asp
Net framework regular expressions	http://msdn.microsoft.com/en-us/library/hs600312.aspx

1.6 Need More Help?

For further WebAii or Automation Design Canvas assistance visit the Art of Test blogs and forums at: http://community.artoftest.com/blogs and http://community.artoftest.com/forums/ or ArtOfTest support at contact@artoftest.com.

2 GETTING STARTED

2.1 System Requirements

2.1.1 Hardware, Software, Operating System Requirements

The system requirements for Design Canvas are the same as those for Visual Studio 2008. These are listed at http://msdn.microsoft.com/en-us/vsts2008/products/bb933744.aspx

2.1.2 Supported Browsers

Design Canvas supports the following browsers:

- Internet Explorer® 7.0 and above (Internet Explorer® 6.0 is not supported)
- Firefox® 2.0 and above

2.2 Installing Automation Design Canvas

2.2.1 Prerequisites

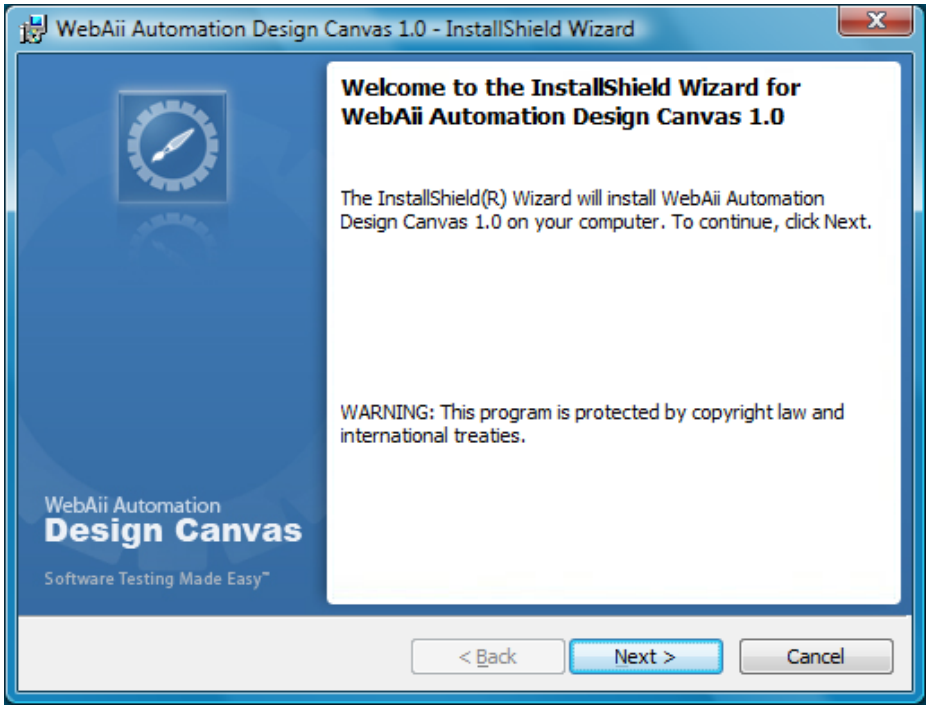
Automation Design Canvas requires you to first install Visual Studio 2008 Professional SP1 edition or Visual Studio 2008 Team System SP1 edition. Installing Design Canvas is simple and does not require preplanning.

Note: If you have the Beta or RC0 versions of Automation Design Canvas or WebAii 1.1 framework installed, please uninstall them before installing Automation Design Canvas. Simply run the Automation Design Canvas install file and follow the prompts.

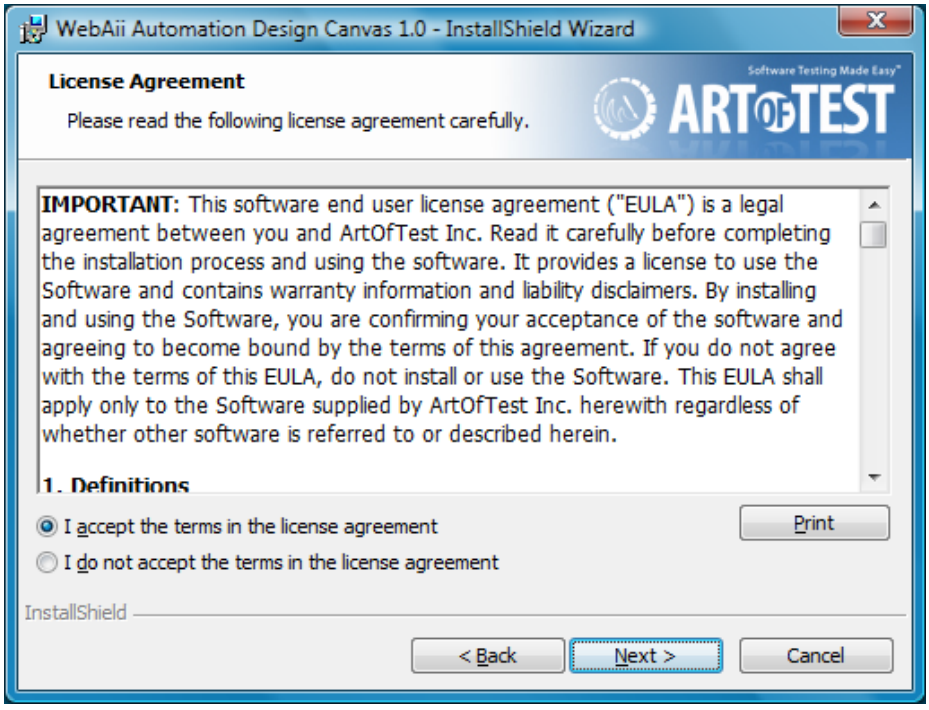
The installation program performs these steps:

- Installs Automation Design Canvas.
- Registers the Visual Studio Test templates used by Automation Design Canvas with Visual Studio.

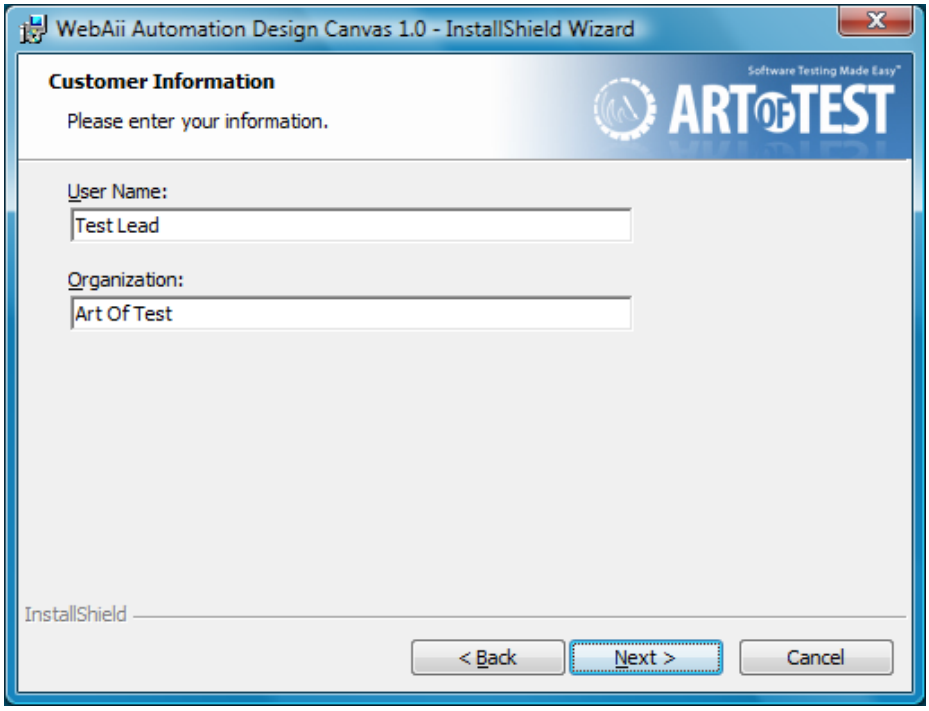
2.2.2 Installing Automation Design Canvas 1.1



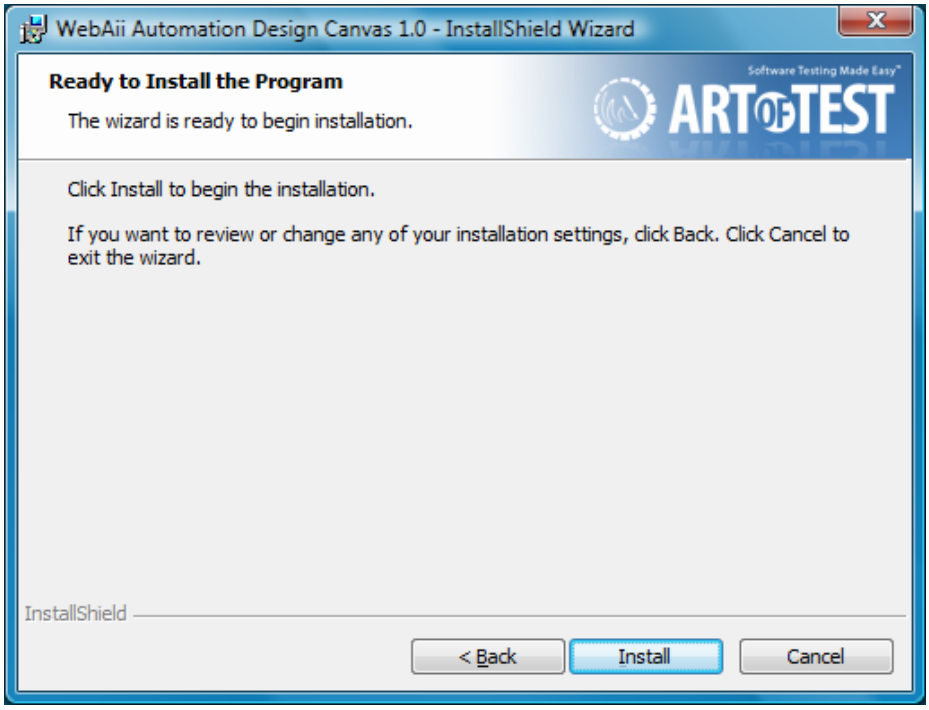
Click **Next** displays the License Agreement.



Read and accept the license terms and click **Next**. Then enter your name and the name of your group or company.

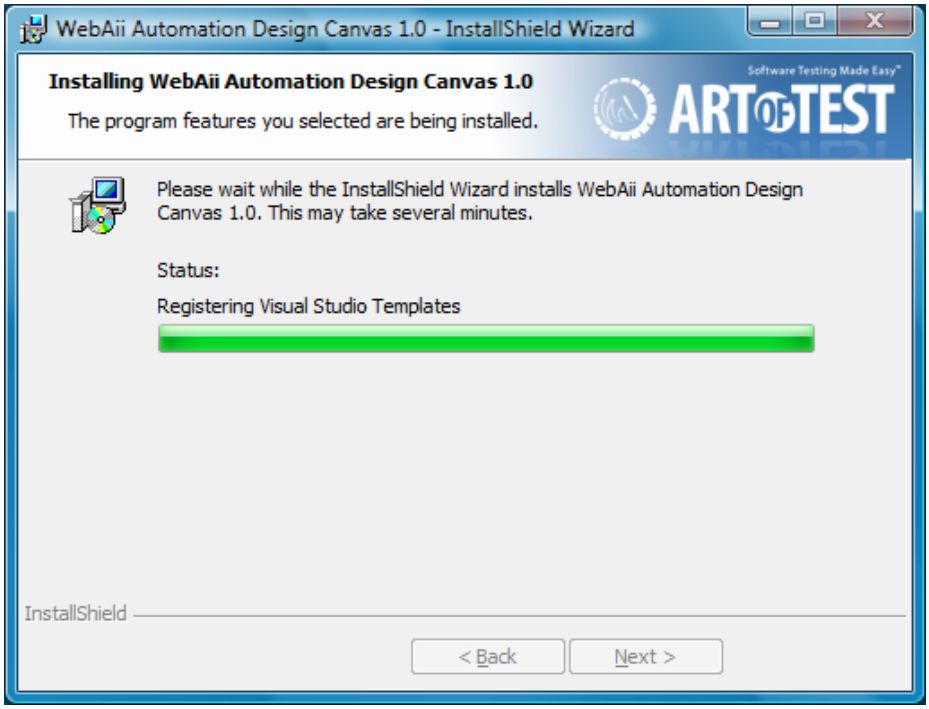


After entering this information click **Next**.

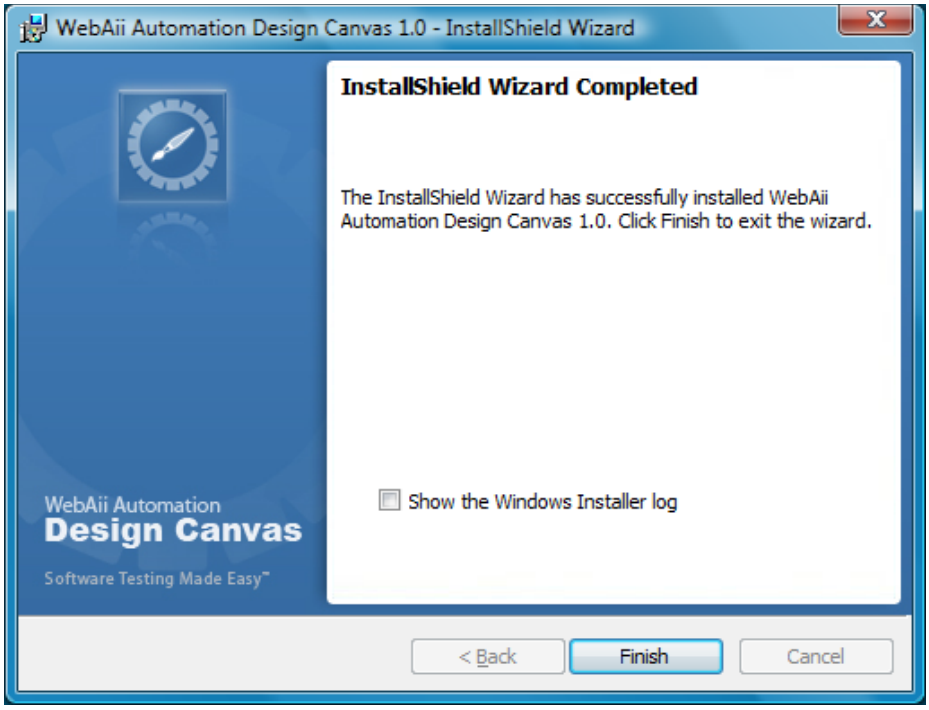


Click **Install**. The installer copies the required files to C:\Program Files\ArtOfTest Inc\WebAii Automation Design Canvas 1.0

NOTE: When the installer installs the necessary add-on to Visual Studio, it may appear to be hung and can take from about 30 seconds to 3 minutes depending on the speed of your computer.



After installation is complete you will see this confirmation:



Congratulations! Installation is complete and you are ready to begin using Automation Design Canvas!

2.3 Uninstalling Automation Design Canvas

Automation Design Canvas installs the WebAii framework with it. If you need to uninstall Automation Design Canvas it will not automatically uninstall WebAii. If you need WebAii uninstalled as well you must uninstall it separately from the Automation Design Canvas uninstallation.

2.4 Configuring Your Browser for Test Automation

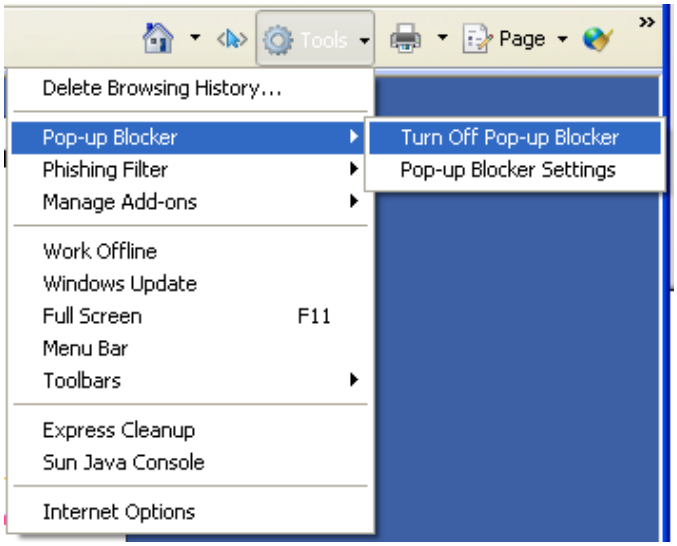
The default Internet Explorer and Firefox browser configurations are great for normal everyday standard use but present a few problems for smooth browser-driven test automation. Features such as popup blockers, IE's "gold bar", and security alert dialogs get in the way of fully unattended browser-driven test automation. This section describes how to change your browser configuration settings for a smoother test automation experience. We first describe how to configure IE 7 settings followed by Firefox 2.0 and 3.0 settings.

2.4.1 Internet Explorer

2.4.1.1 Disabling the Pop-Up Blocker

First disable the Internet Explorer (IE) pop-up blocker. This allows any HTML popup windows to open unhindered. If you have no pop-up windows to deal with, then you can skip this procedure. To disable the IE pop-up blocker, follow this procedure:

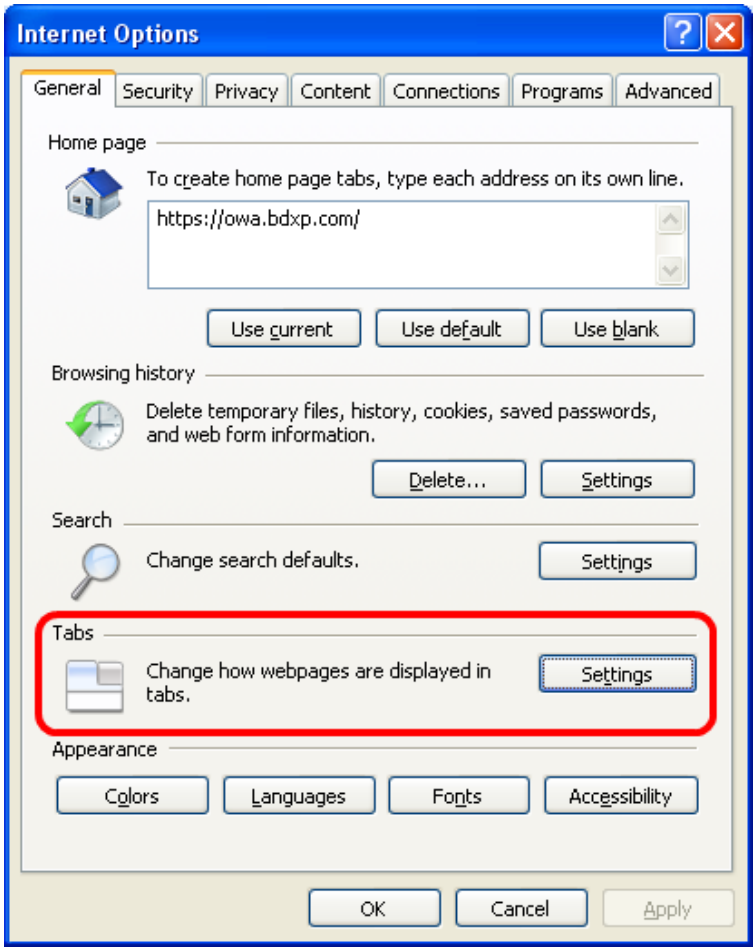
- In IE, click **Tools** then highlight Pop-up Blocker by moving the cursor over it.
- Click Turn Off Pop-up Blocker.



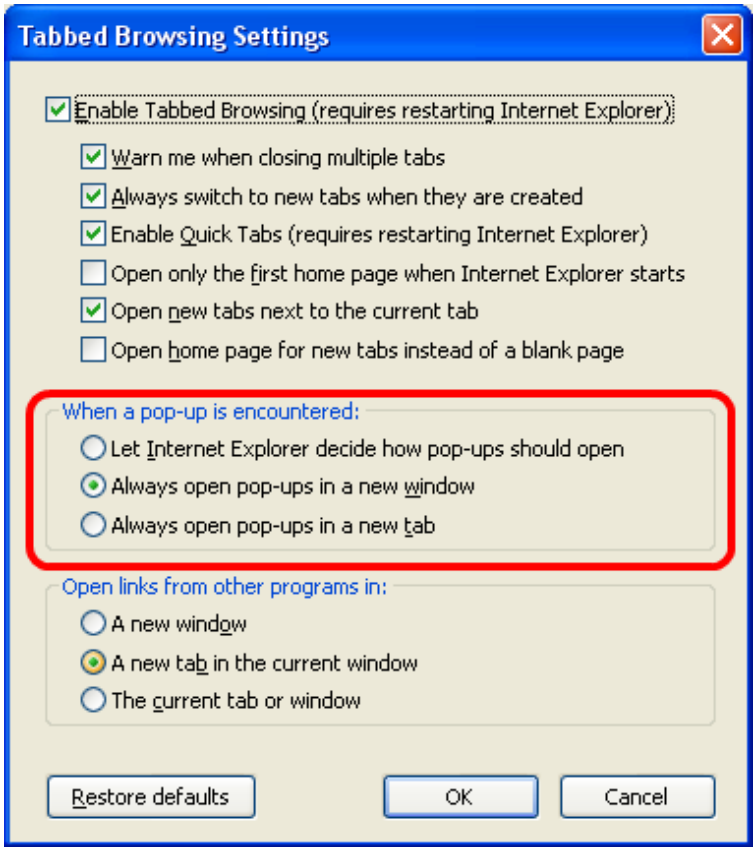
2.4.1.2 Opening Pop-ups in a New Window

WebAii requires pop-ups to open in a new window instead of a tab. Follow this procedure to change this setting in IE 7:

- In IE click Tools then Internet Options.
- The General tab should already be selected. If not click the **General** tab.
- Click **Settings** for "Change how WebPages are displayed in tabs".

- 

- Select "Always open pop-ups in a new window".

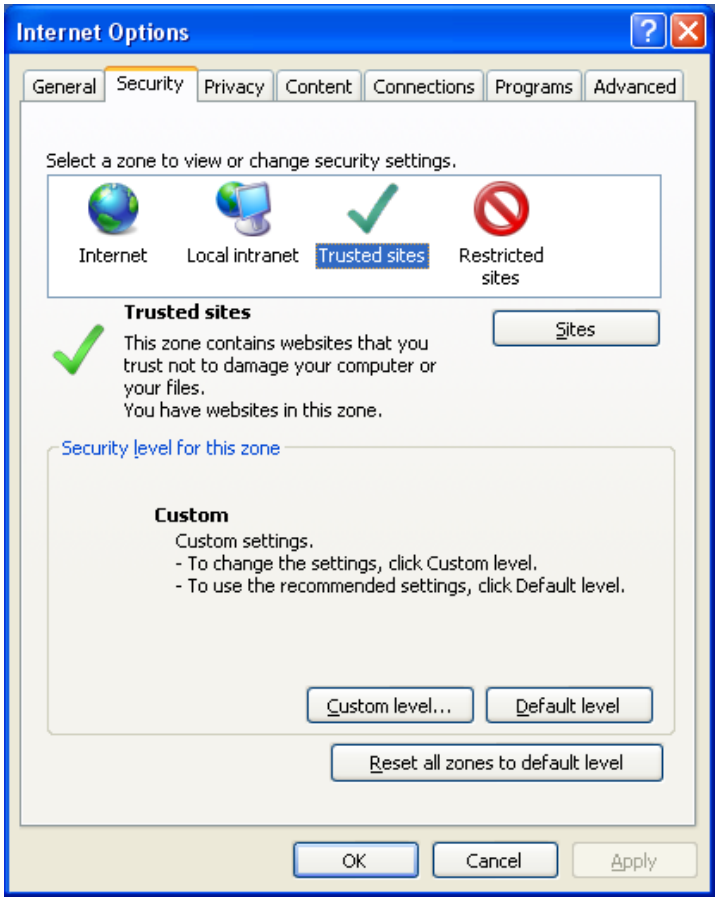


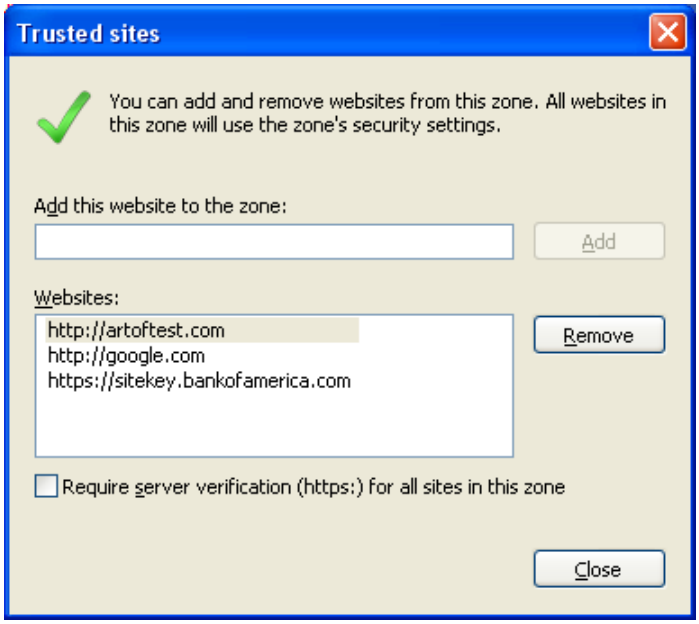
2.4.1.3 Adding Trusted Websites

The next thing you will need to do is to add the website(s) you will be testing to your list of trusted sites. This will eliminate the IE security warnings when switching between secure and non-secure pages or from one server to another in the middle of a test. To add a website to the trusted website list, follow this procedure:

- In Internet Explorer click **Tools** then **Internet Options**.
- Go to the Security tab of Internet Options.
- Click Trusted sites.
- Click **Sites**.
- Chances are your site is not a secure website (uses HTTPS protocol). If not then you will need to uncheck the "Require server verification..." checkbox.
- Enter the URL of your website in "Add this website..." text box. This text box will accept both DNS names and IP addresses.
- Click **Add**.

Repeat steps 4-5 for each website you will be testing.

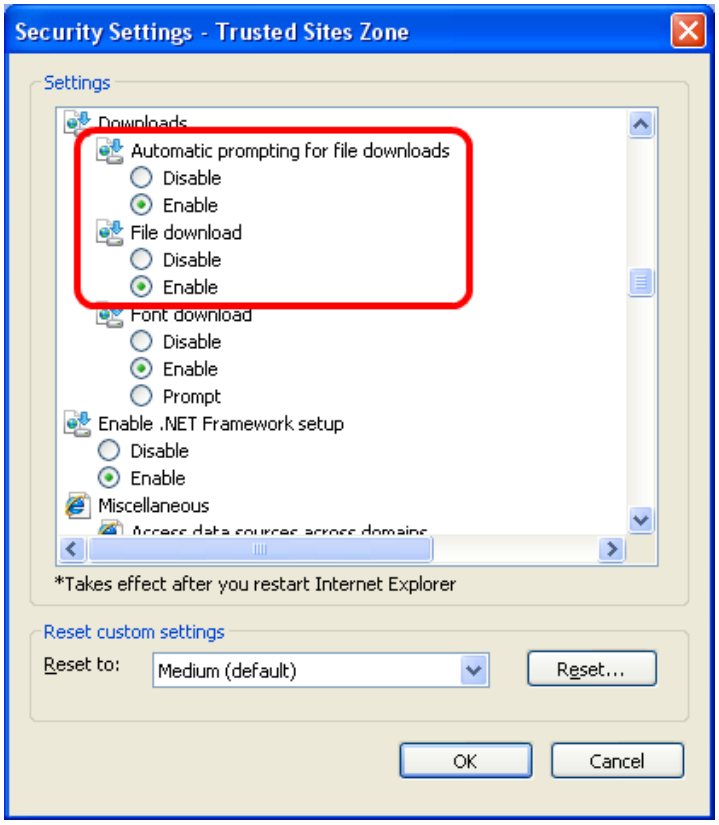

Internet Options
General
Security
Privacy
Content
Connections
Programs
Advanced
Select a zone to view or change security settings.
Internet
Local intranet
Trusted sites
Restricted sites
Trusted sites
This zone contains websites that you trust not to damage your computer or your files.
You have websites in this zone.
Sites
Security level for this zone
Custom
Custom settings.
- To change the settings, click Custom level.
- To use the recommended settings, click Default level.
Custom level...
Default level
Reset all zones to default level
OK
Cancel
Apply



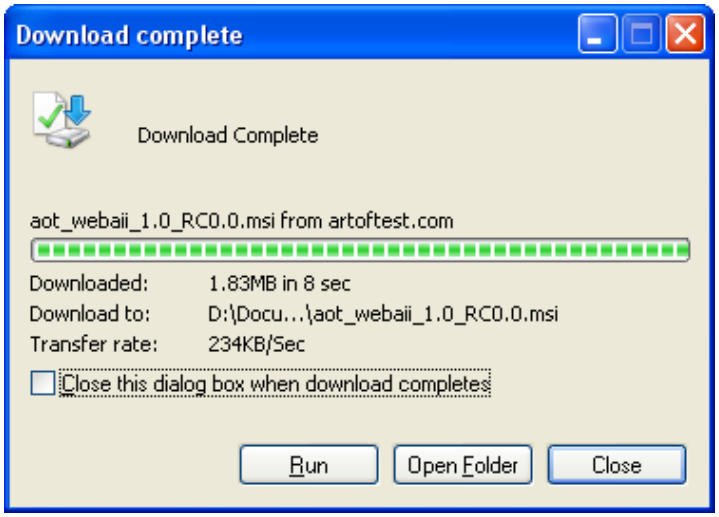
2.4.1.4 Enabling File Download and Automatic Prompting

If you will be performing downloads as part of your testing you must also make an adjustment to IE. Enable "File download" and "Automatic prompting for file downloads" in the security settings by following this procedure:

- In Internet Explorer click **Tools** and then **Internet Options**.
- Go to the Security tab of Internet Options.
- Click Trusted sites.
- Click Custom level....
- Select the two **Enable** settings shown in the following diagram.
- Click **OK**.
- Repeat steps 3 - 5 for the "Local intranet" and the "Internet" zones.



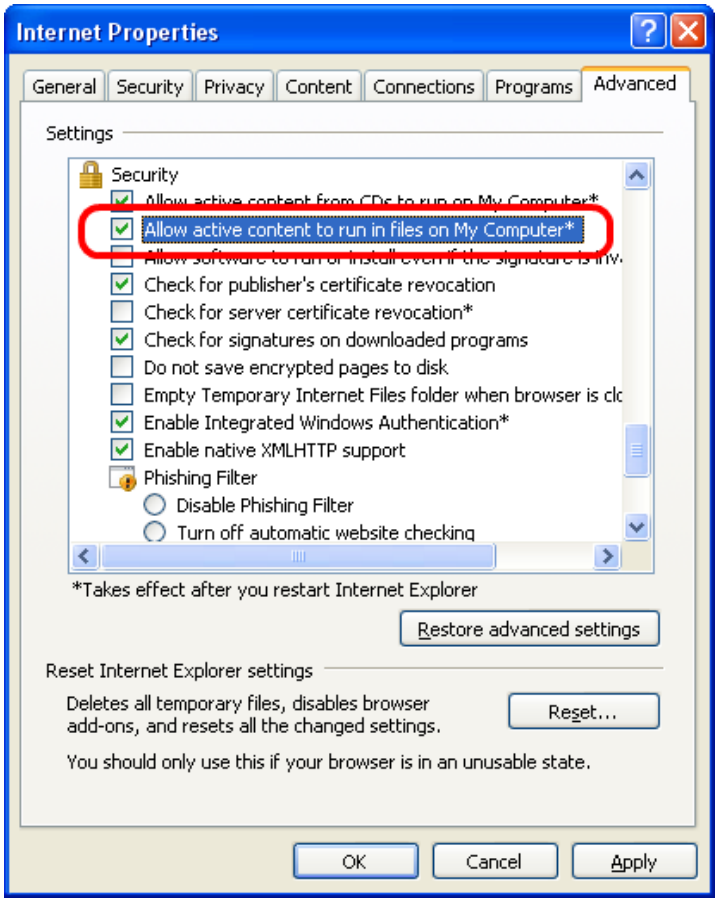
- The last setting to make for file downloads is to uncheck "Close this dialog box when download completes". The only way to do this is to actually start downloading something and uncheck this checkbox while the download is running.



2.4.1.5 Allowing Active Content to Run

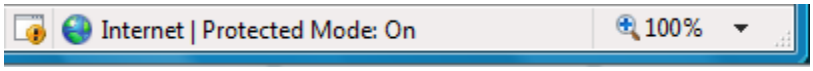
To allow local files (which are used extensively by the source code samples) to run unhindered in IE you must allow active content from local files as follows:

- In Internet Explorer click **Tools** and then **Internet Options**.
- Go to the Advanced tab under Internet Options.
- Check "Allow active content to run in files on My Computer"



2.4.1.6 Disable Protected Mode in Vista

If you are running Windows Vista you need to disable Protected Mode as this interferes with test automation. To determine whether or not Protected Mode is active look at Internet Explorer's status bar at the bottom of the window. If the status bar is not visible, turn it on by selecting View > Status Bar from the main menu. Here is how it looks with Protected Mode turned on:

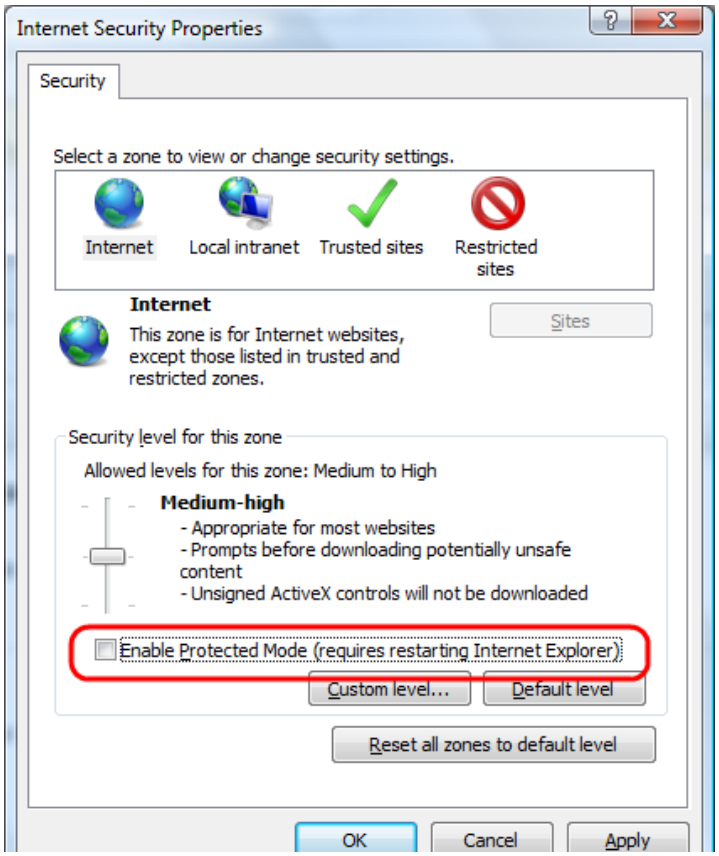


There are two ways to turn off protected mode:

2.4.1.7 How to turn off protected mode in Internet Explorer

Double click on the status bar where it says "Protected Mode" to open up the security settings for the current zone.

- Uncheck the "Enable Protected Mode" checkbox.

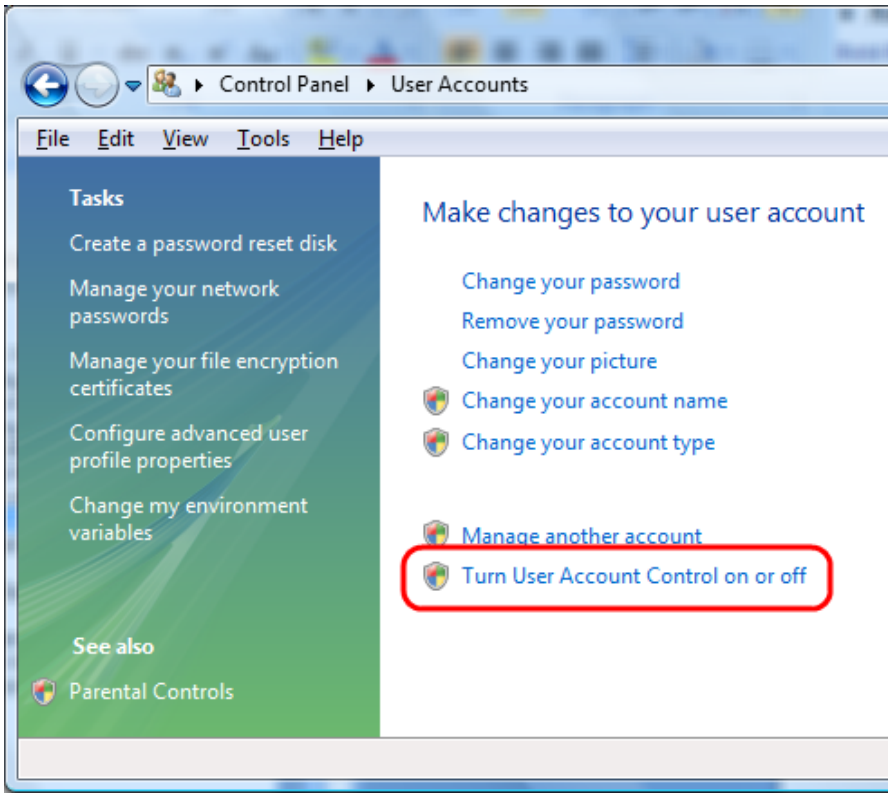


- Close and restart Internet Explorer

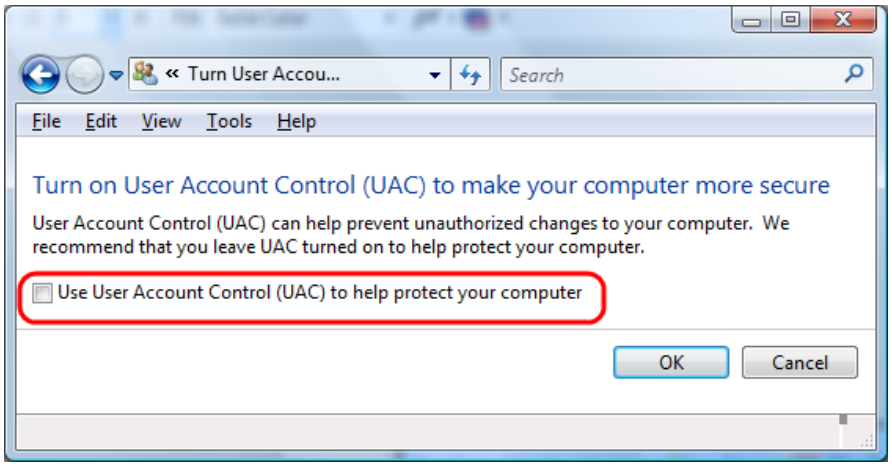
2.4.1.8 How to disable User Account Control

When User Account Control is turned off it automatically disables Protected Mode in Internet Explorer:

- Open the control panel by clicking **Start** > **Control panel**.
- Double click on **User Accounts**.
- Click Turn User Account Control on or off.



- Uncheck the checkbox “Use User Account Control (UAC)” and click **OK**.

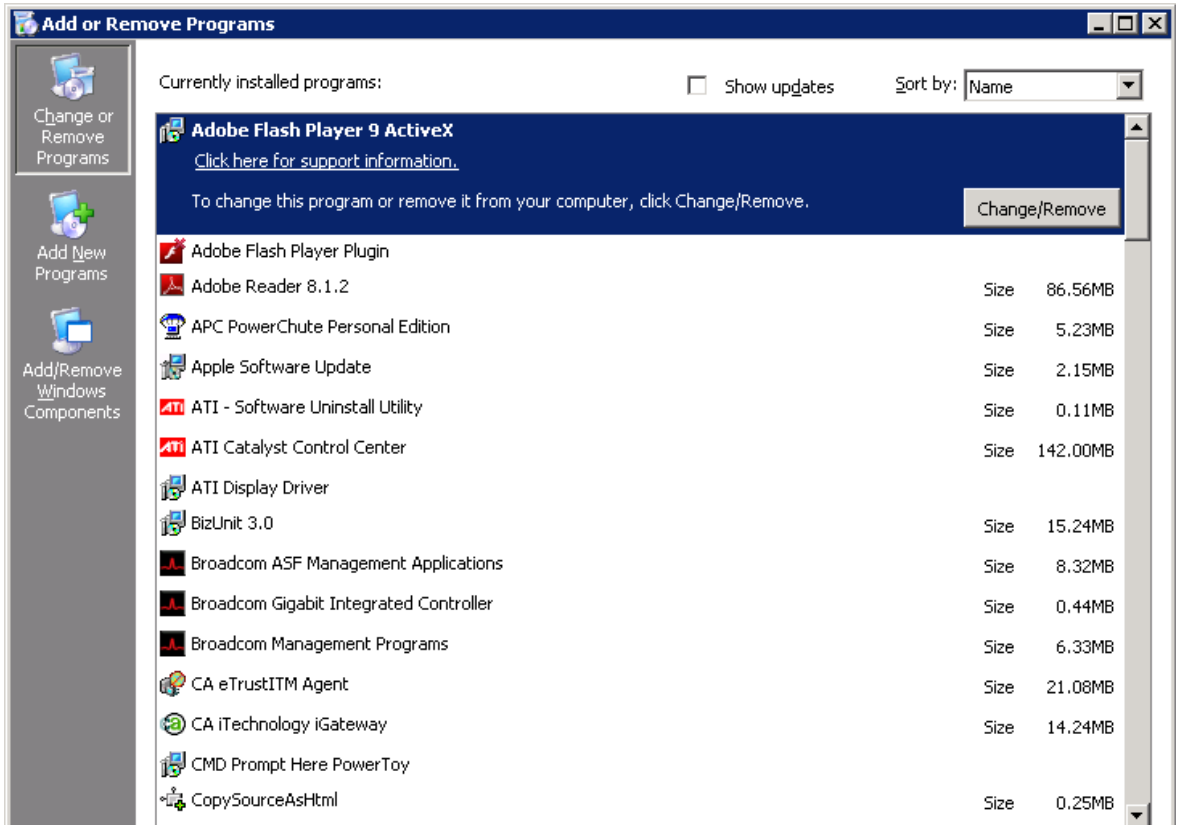


You will have to restart your computer before the change takes effect.

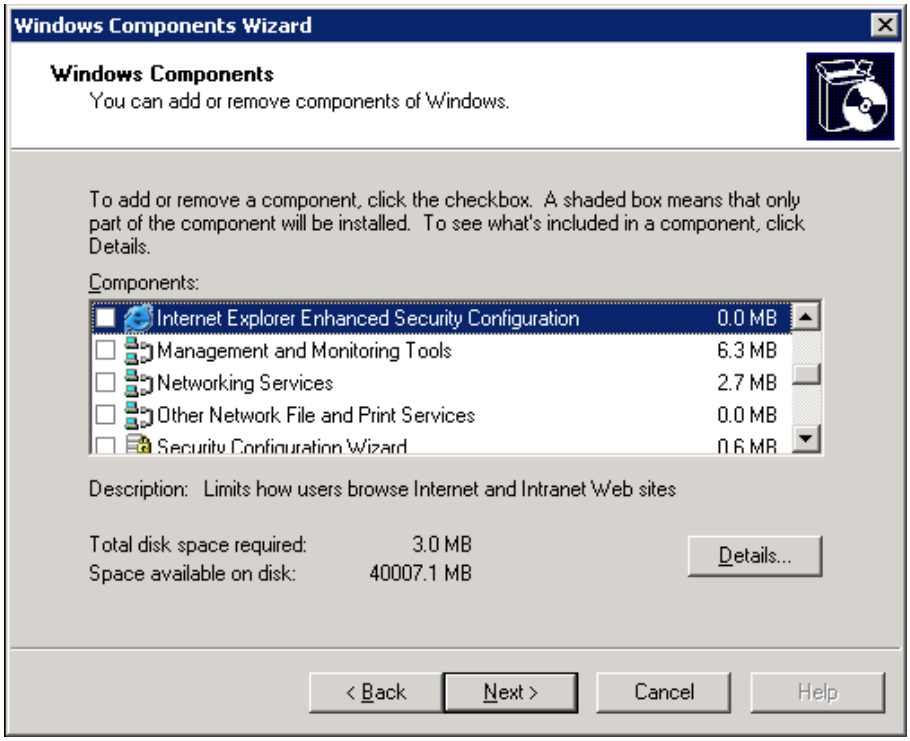
2.4.1.9 Uninstall Enhanced Security in Windows Server 2003

If you are running Windows Server 2003 you need to uninstall “Enhanced Security” as this interferes with test automation. To do this:

- Open the control panel by clicking **Start** then **Control Panel**.
- Double click Add or Remove Programs.
- Click Add/Remove Windows Components.



- Find and uncheck “Internet Explorer Enhanced Security Configuration”.



- Click **Next**.
- Let Windows perform the configuration change.

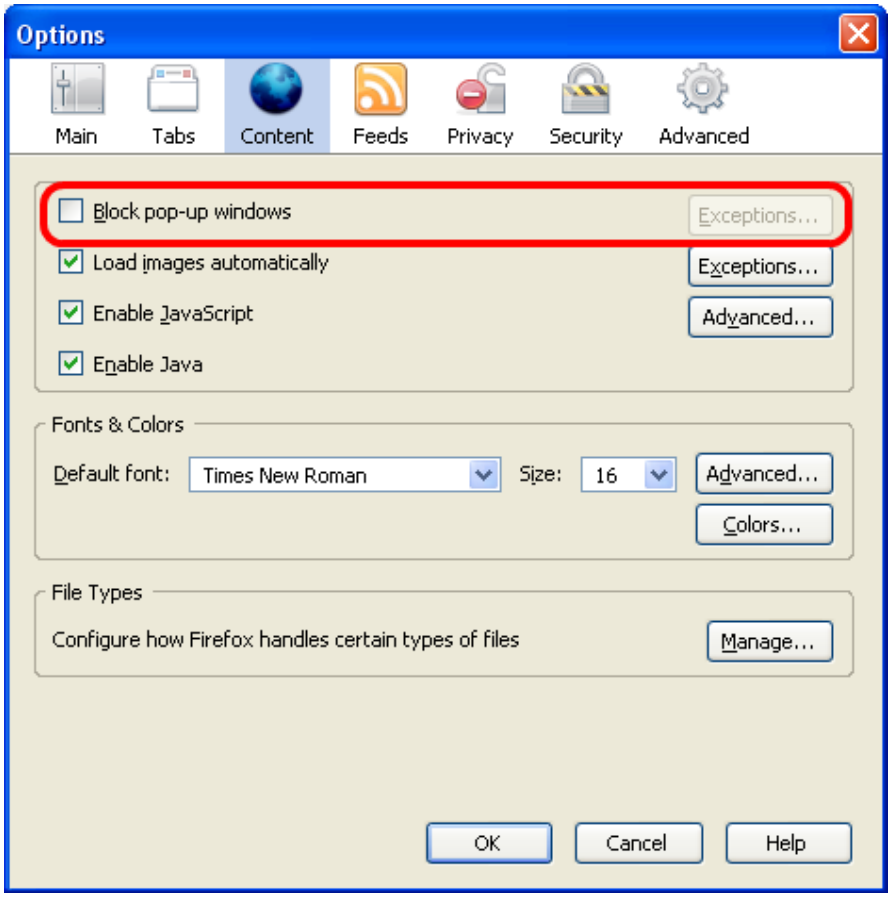
- Click **Finish**.

2.4.2 Firefox

2.4.2.1 Turning off Firefox Pop-up Blocker

Turn off the Firefox pop-up blocker by following this procedure:

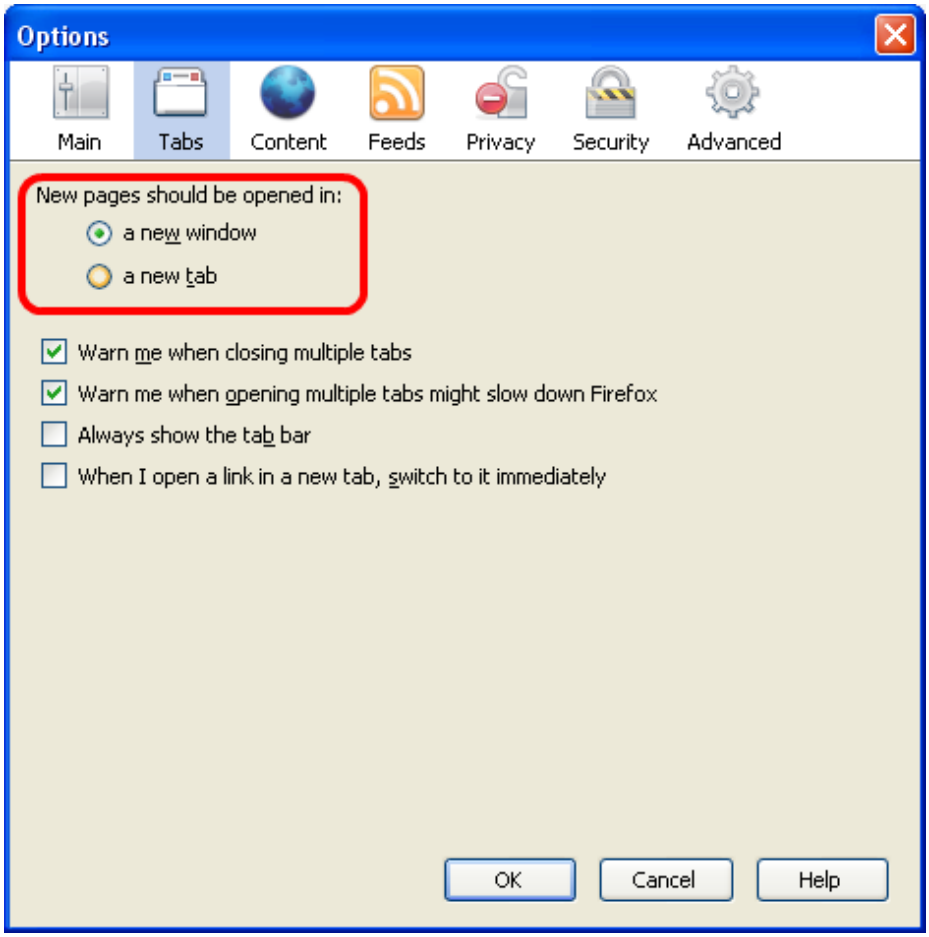
- On the main menu click **Tools** then **Options...**.
- Click the **Content** tab.
- Uncheck the Block pop-up windows checkbox.
- Alternatively if you prefer to keep the pop-up blocker enabled you can add all the addresses of your test websites to the Exceptions list. The problem with this approach is that any changes or any new test websites will need to be added to the list.



2.4.2.2 Opening pop-ups in a separate window

WebAii requires pop-ups to open in separate window instead of a tab. Follow this procedure to change this setting in Firefox:

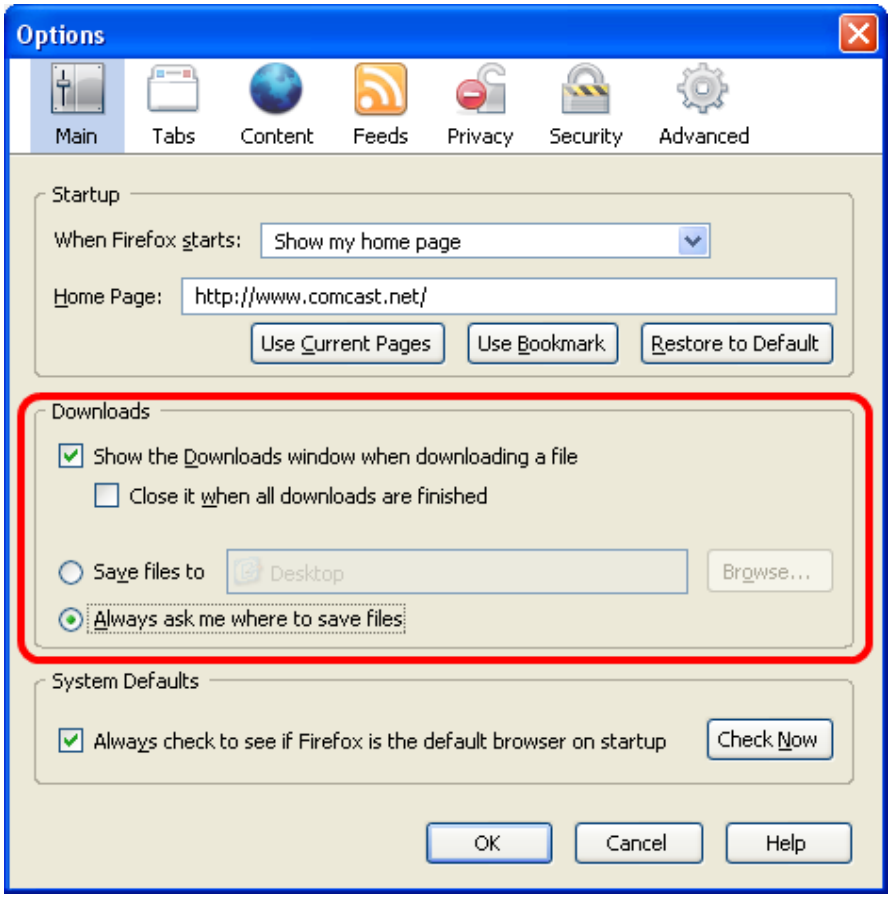
- On the main menu click on **Tools** and then **Options...**.
- Click the **Tabs** tab.
- Select “a new window” as shown in the following diagram.
- Click **OK**.



2.4.2.3 Setting the Show Downloads Window

Lastly if you will be performing downloads as part of your testing you need to make one more adjustment. Set Firefox to show the Downloads window, do not close it automatically, and ask where to save downloaded files by following this procedure:

- On the main menu click on **Tools** and then **Options...**.
- Click the **Main** tab.
- Check "Show the Downloads window...".
- Uncheck "Close it when all downloads are finished".
- Select "Always ask me where to save files".


Options
Main
Tabs
Content
Feeds
Privacy
Security
Advanced
Startup
When Firefox starts:
Show my home page
Home Page:
http://www.comcast.net/
Use Current Pages
Use Bookmark
Restore to Default
Downloads
Show the Downloads window when downloading a file
Close it when all downloads are finished
Save files to
Desktop
Browse...
Always ask me where to save files
System Defaults
Always check to see if Firefox is the default browser on startup
Check Now
OK
Cancel
Help

2.5 Setting up Your Automation Design Canvas Windows

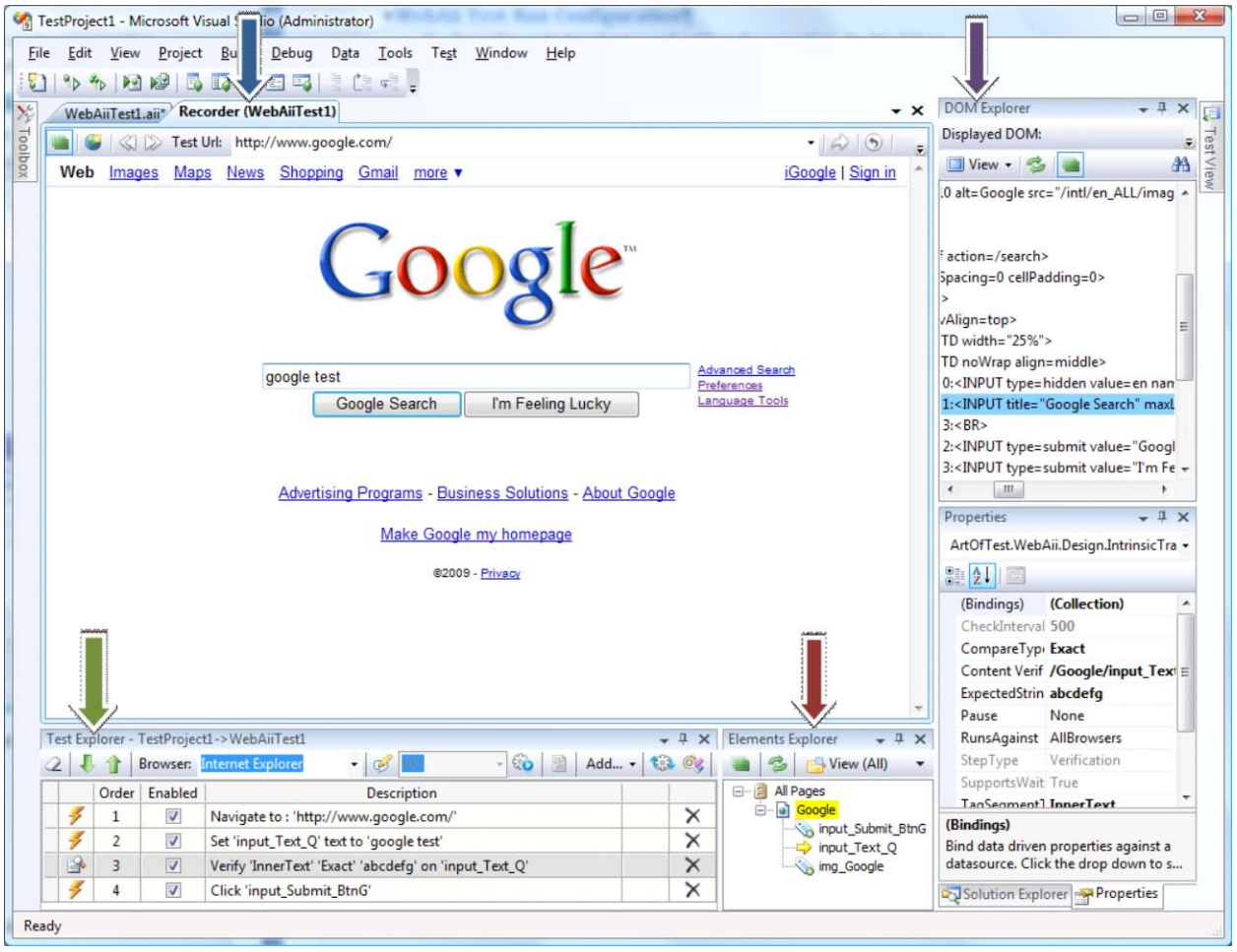
Automation Design Canvas provides the following main windows:

- Recording Surface (Document Window Docked)
- Elements Explorer (Dock-able floating tool window)
- Test Explorer (Dock-able floating tool window)
- DOM Explorer (Dock-able floating window)

Besides these it also provides a Storyboard window which you can use to get a visual perspective of your test's sequence of execution.

2.5.1 Suggested Design Canvas Window Layout

The following Design Canvas window layout is recommended to optimize space and recording efficiency.



Note: This is only a recommended layout. Customize your layout as desired.

2.5.2 Recording Surface

The **Recording Surface** is the Design Canvas window that you use for recording the steps of your test. You record both action steps (click, set text, select option, etc.) and verification steps using this window. It appears as its own document in the document pane in Visual Studio. There is only one **Recording Surface** and it is used by all WebAii tests in your project. Whichever WebAii test was the last test selected becomes the active test in the **Recording Surface**.

Section 5 illustrates a typical Recording Surface window.

2.5.3 Test Explorer

Test Explorer is a dock-able floating window that displays all test steps of the currently active WebAii test. Some tasks you can perform in this window are:

- Quick Execute the active test
- Change the order of the steps
- Change the name of the steps
- Delete a step
- Disable a step

Section 5.8 describes how to use **Test Explorer** window.

2.5.4 DOM Explorer

The **DOM Explorer** dock-able floating window displays the DOM (Document Object Model) of the webpage currently loaded in the Recording Surface. Use this window to study the elements and their hierarchy of the webpage. You can also select an element from this window that you cannot normally select in the Recording Surface to perform an action on. Section 5.10 explains how to use the **DOM Explorer** window.

2.5.5 Elements Explorer

Elements Explorer dock-able floating window displays the list of all elements used by all tests in your project. Some tasks you can carry out in this window are:

- Change the friendly name of the element
- Change the way an element is found
- Highlight the selected element in the **Recording Surface** to see where it is located on the webpage

Section 7.2 describes how to use **Elements Explorer** window.

2.5.6 Storyboard

The storyboard appears on the Steps tab of your test document window (in the document pane in Visual Studio). As you record your test, a screenshot of your action on the target element is captured to the storyboard. This gives you a visual flow of how your test has progressed. Section 5.6 describes the storyboard window in more detail.

3 PLANNING YOUR AUTOMATED TESTING

Before using Design Canvas to automate your tests it is often helpful if you prepare and plan for the automation by first obtaining the answers to these questions:

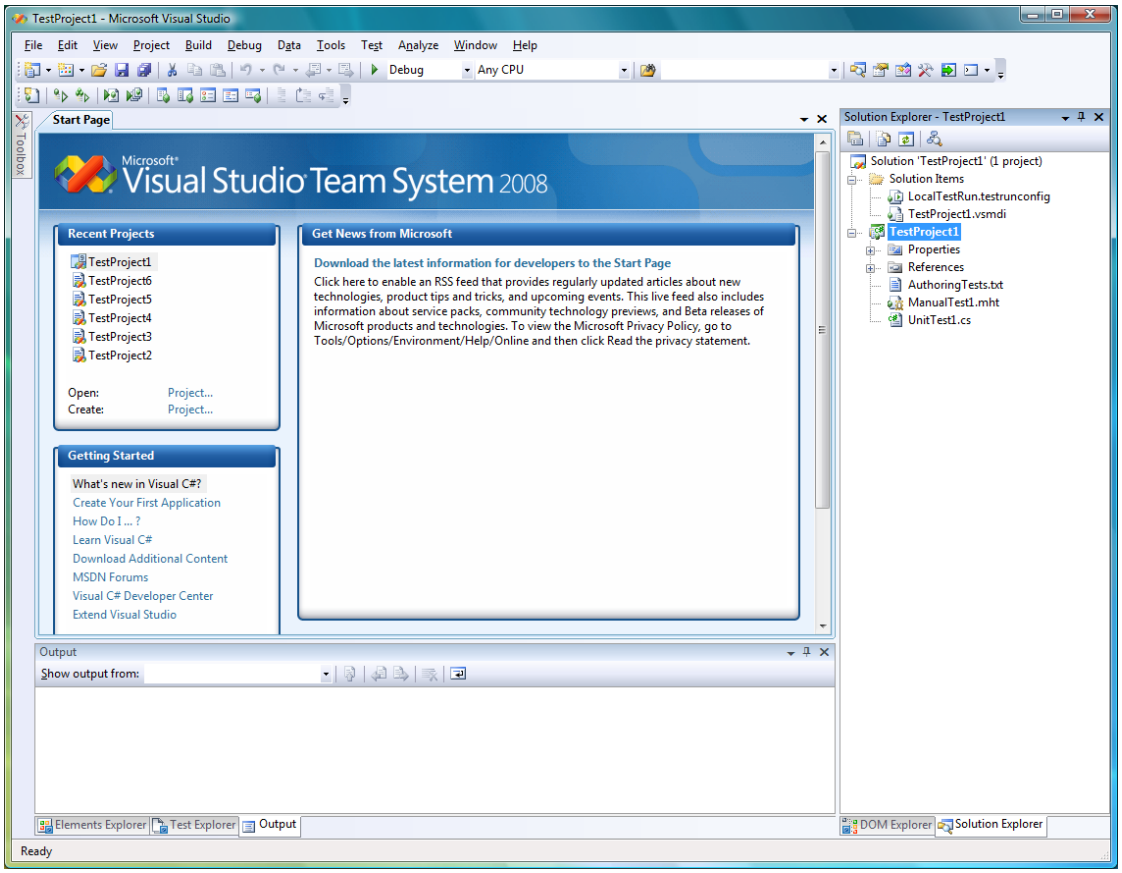
Question	Description
Web application name	
Site addresses of your web applications to be tested	
Determine which browsers you will be testing against	
Detailed listing of functions and features to be tested	
Documented test cases and scenarios if applicable	
Location of or documented data points?? To use in testing each field on each page	
Listing of expected results if not already included in the test cases	
Will your test involve downloading?	

4 RECORDING AND RUNNING YOUR FIRST TEST

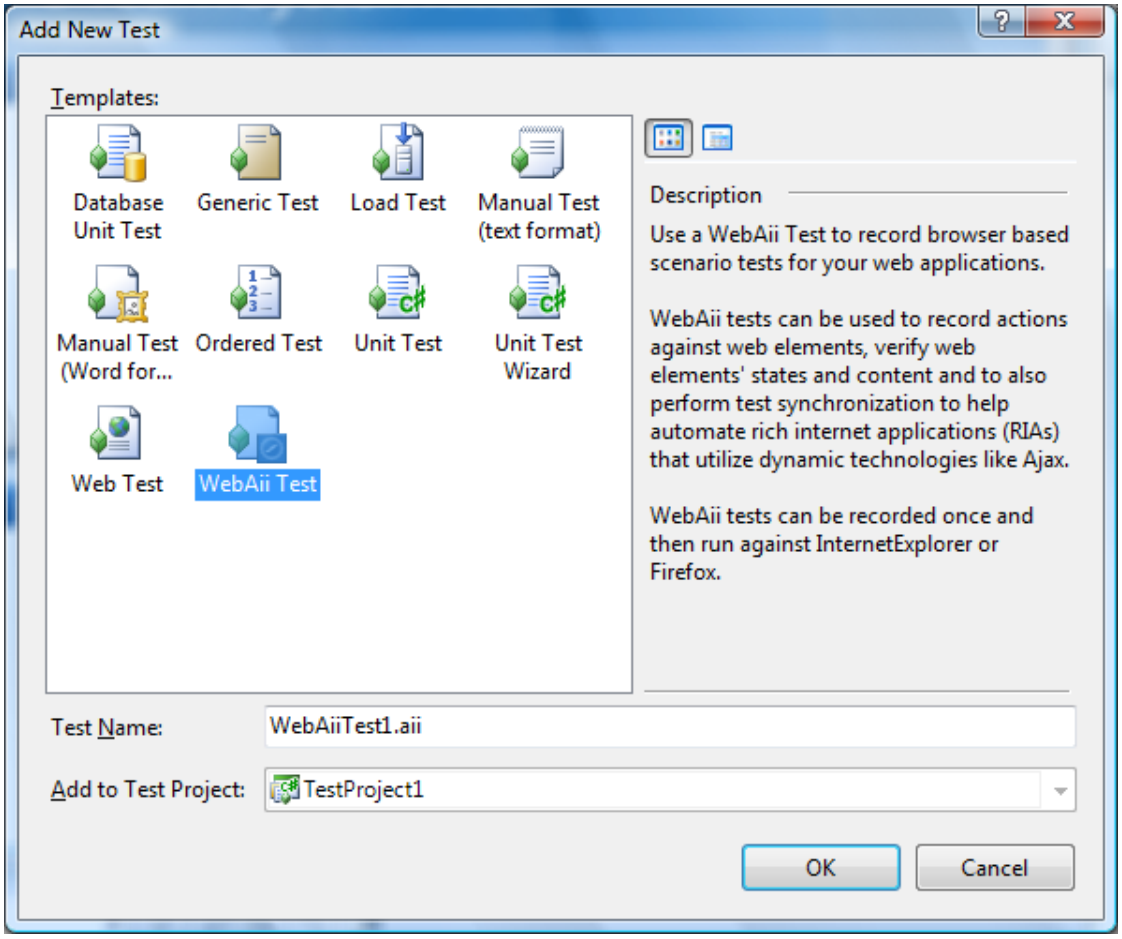
4.1 Creating a New WebAii Test

To create a new WebAii test using Automation Design Canvas follow these steps:

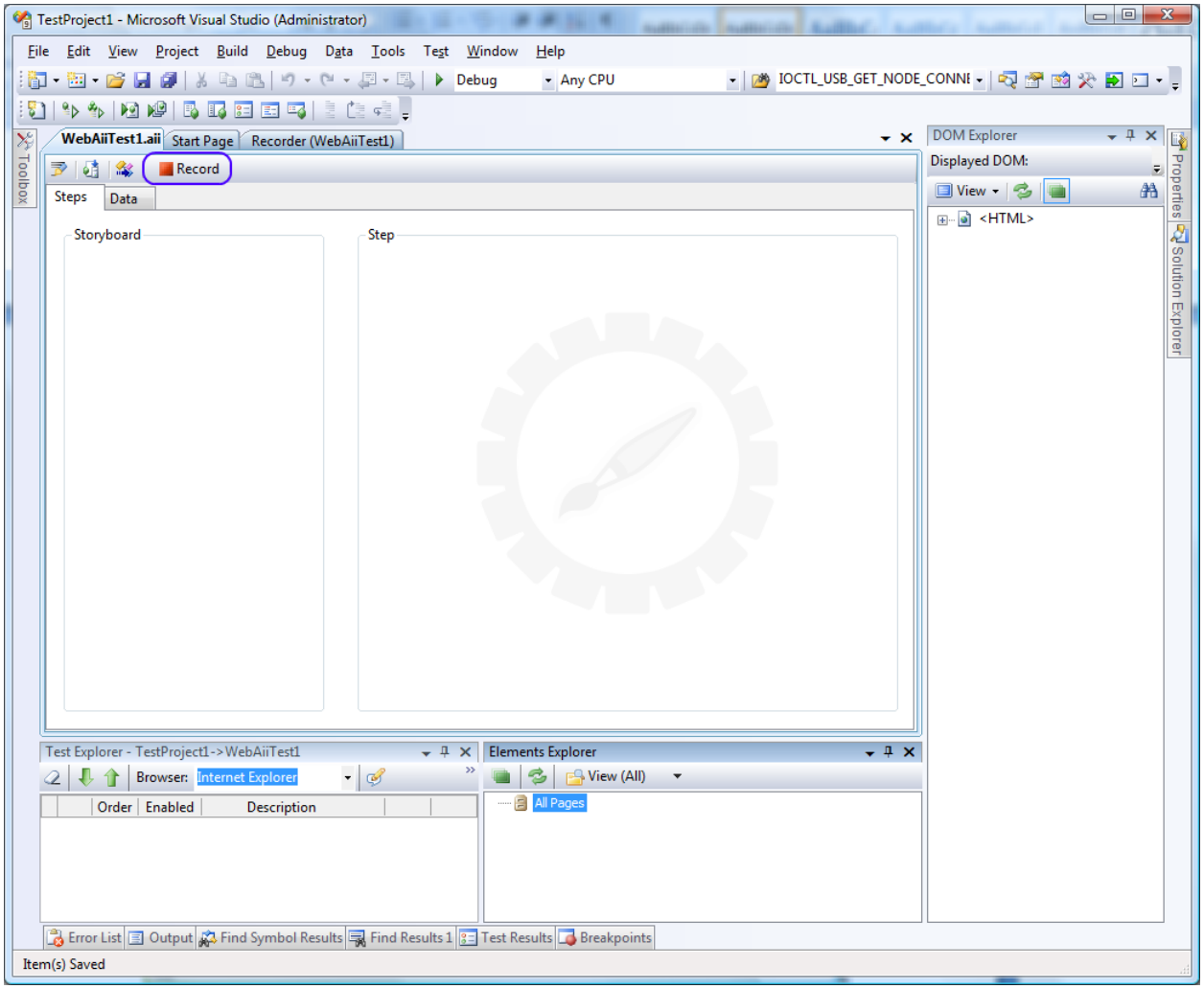
- Open Visual Studio and create a new Visual C# or Visual Basic test project.



- Right click on the project node Solution Explorer and select “Add > New Test…”. The Add New Test window displays.



- Click **WebAii Test** type, enter an appropriate test name that describes your test, and click **OK**.
- Your new empty WebAii test should automatically open. If not then double click the new WebAii Test in Solution Explorer or click the WebAii Test tab in the document windows.

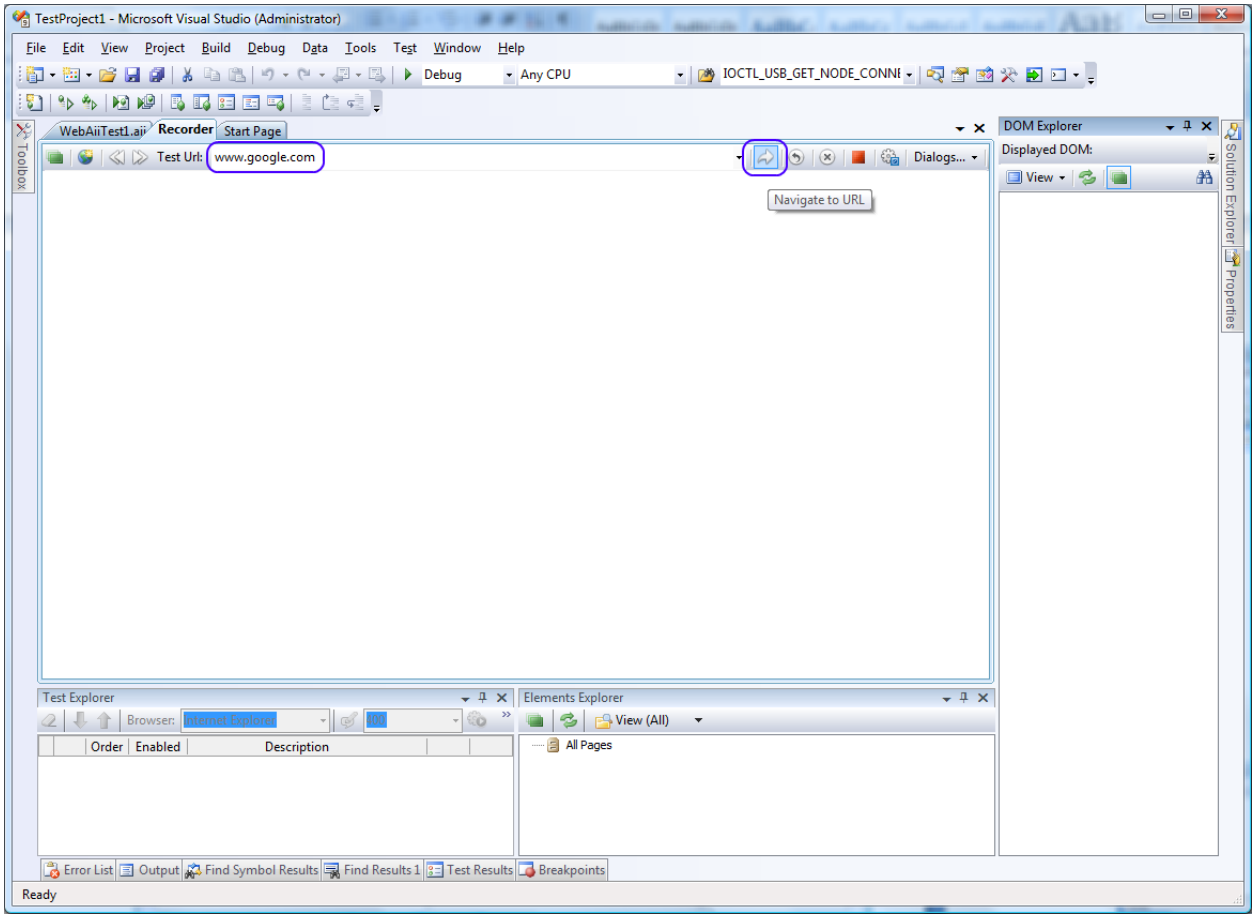


If this is your first time loading a WebAii Test see “Suggested Window Layout” in section 2.6.1.

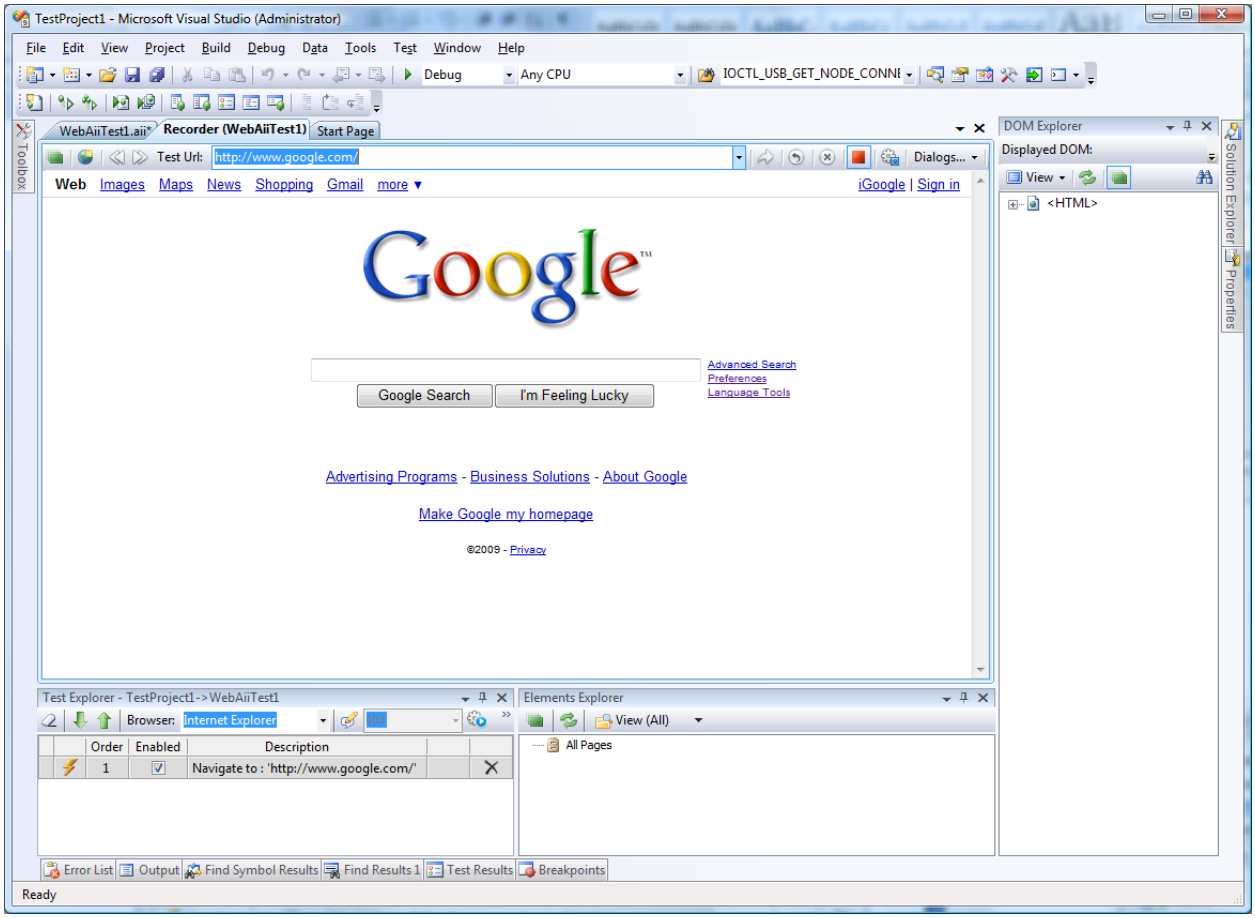
4.1.1 Recording a Test

To begin recording the steps of your test perform the following:

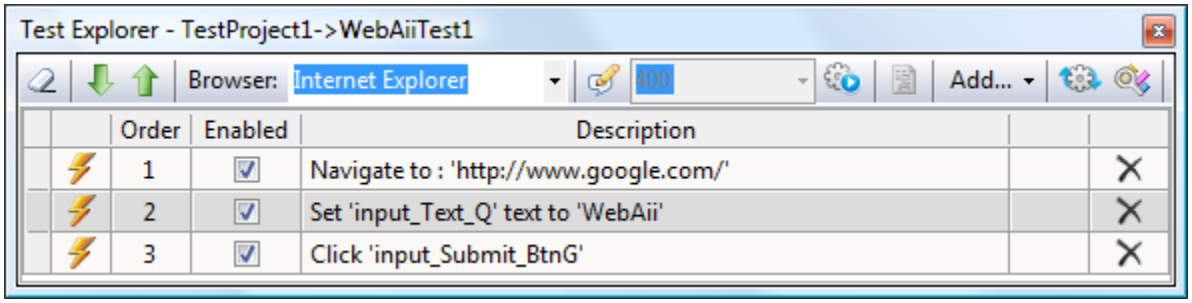
- Click **Record** (circled in blue above).
- The Recorder tab will automatically activate.



- In the Test Url box, enter for example, www.google.com.
- Click the Navigate to URL button.



- Notice that a recording step has been added to Test Explorer.
- Type “WebAii” in the Google search box and click the Search button.
- Notice that two more steps have been added to Test Explorer as shown:



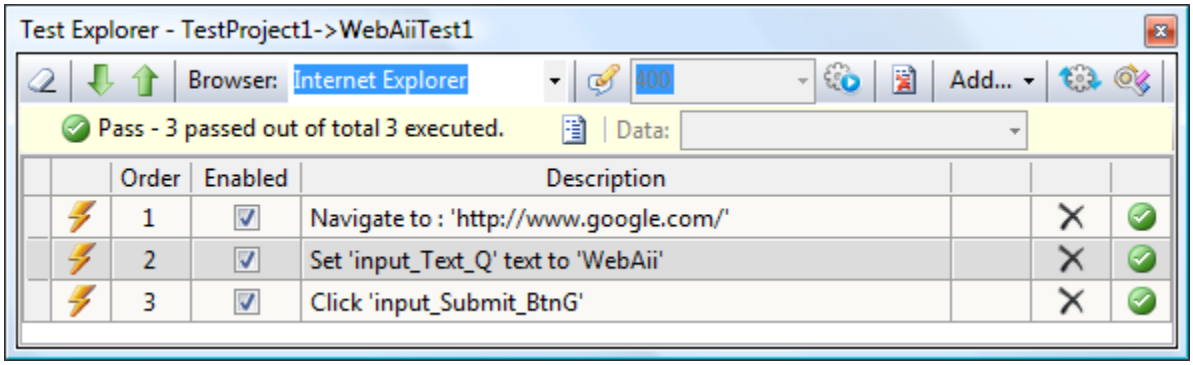
That’s how easy it is to create your first WebAii test! Save and build your project.

4.1.2 Running Your Test Using Quick Execute (Internet Explorer)

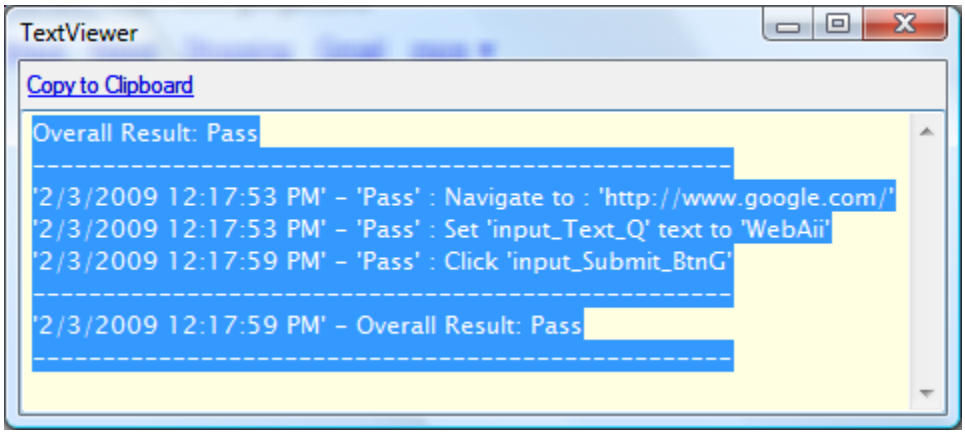
Performing a Quick Execute of your newly crafted test is a great way to verify that it performs as expected and does everything correctly. To ‘quick execute’ your test in IE follow these steps:

- Make sure Internet Explorer is selected as the browser in your Test Explorer window.
- Click the Quick Execute icon  in your Test Explorer window.

- An Internet Explorer window will open and the steps of your test automatically execute in the browser window. When the test completes the browser window automatically close and a summary of the test results is displayed in the Test Explorer window.



- To see detailed test results click the View Test Log button shown outlined in red in the Test Explorer window. A TextViewer window opens displaying the log of the test run.



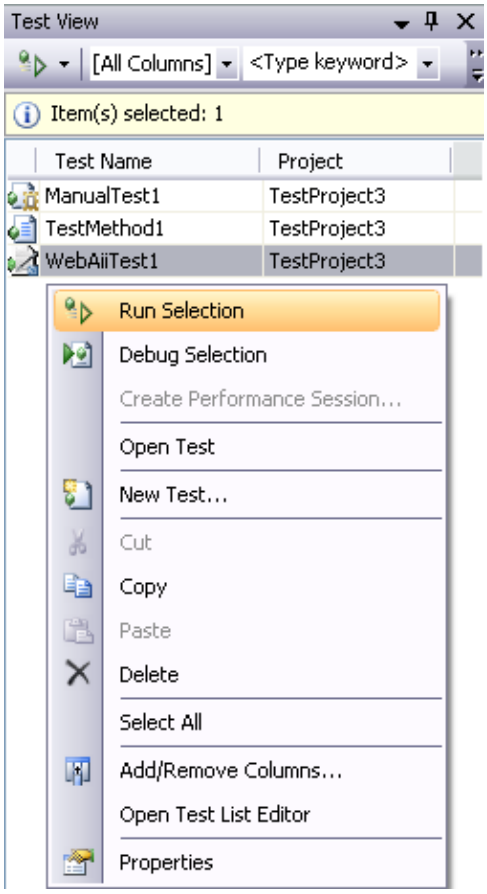
4.1.3 Running Your Test Using Quick Execute (Firefox)

Quick executing your test in Firefox is the same as running it in IE. Follow the procedure described in section 4.1.2 except select Firefox as the browser in Test Explorer instead of the default Internet Explorer.

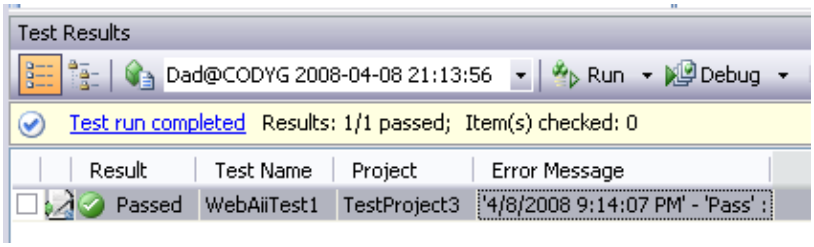
4.1.4 Running Your Test in Visual Studio Test Manager

To execute your test under Visual Studio Test Manager follow these steps:

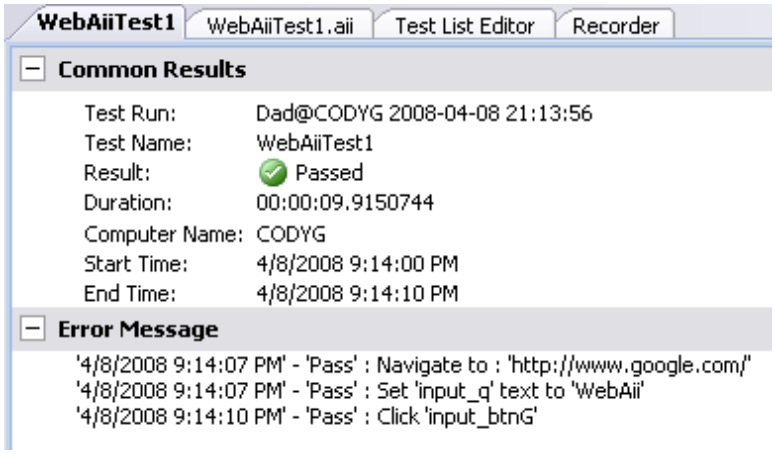
- Open the "Test View" tool window. This can be found by clicking **Test** in the main menu then **Windows** then **Test View**.
- Right click on your WebAii Test and click **Run Selection**.



- An Internet Explorer window opens and the steps of your test automatically execute in that browser window. When the test completes the browser window automatically closes.

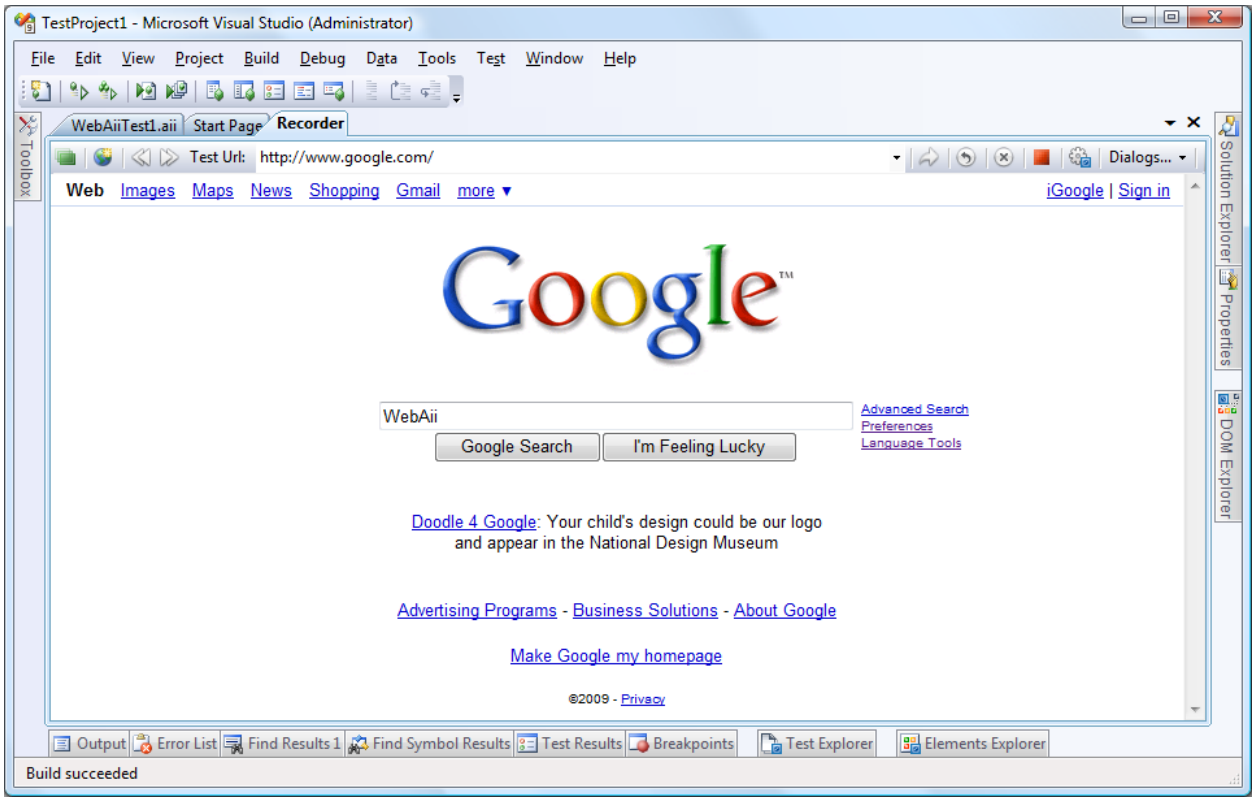


- After the test runs, double click on the results summary in the “Test Results” tool window to view the execution log.


WebAiiTest1
WebAiiTest1.aii
Test List Editor
Recorder
Common Results
Test Run: Dad@CODYG 2008-04-08 21:13:56
Test Name: WebAiiTest1
Result: Passed
Duration: 00:00:09.9150744
Computer Name: CODYG
Start Time: 4/8/2008 9:14:00 PM
End Time: 4/8/2008 9:14:10 PM
Error Message
'4/8/2008 9:14:07 PM' - 'Pass' : Navigate to : 'http://www.google.com/'
'4/8/2008 9:14:07 PM' - 'Pass' : Set 'input_q' text to 'WebAii'
'4/8/2008 9:14:10 PM' - 'Pass' : Click 'input_btnG'

5 DESIGN CANVAS TEST RECORDER

The Design Canvas Recording Surface window has an integrated control called the Automation Overlay SurfaceTM (AOS) that acts as a web browser with an activated component. It displays the current web page you are recording actions against which is then used to record actions on. Most actions performed against a web page can be recorded by just performing the action directly on the Recording Surface.



The following paragraphs explain each component of the Recording Surface window.

5.1 Test Url

This textbox is where you enter the starting URL for your test. Hitting ‘enter’ or clicking the “Navigate to URL” button after entering a URL while recording is enabled will record a “Navigate t to” step in Test Explorer.

You can navigate to other web pages by clicking on the elements (buttons, links, etc.) of the starting page and subsequent pages. However if you know the exact URL you need to navigate to in the middle of your test you can enter it in the Test Url field and click ‘Navigate to URL’ to go there directly.

5.2 Navigation buttons

5.2.1 Navigate to URL

Click this button to go to the URL you entered in the Test Url text field.

5.2.2 Refresh

Clicking this button reloads the current URL. This is useful when the webpage on the server has changed. The “Navigate to URL” button will reload the page from the local cache instead of refreshing it directly from the server. This button operates like the Refresh button in your browser.

5.2.3 Stop the current navigation

Clicking this button aborts loading the current URL to the Recording Surface browser. It is useful when the server seems to hang and not finish sending the contents of the current web page. This button works like the Stop button in your browser.

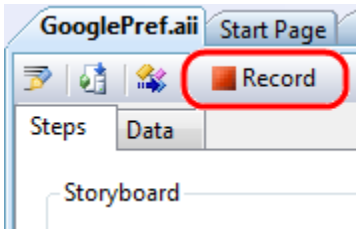
5.2.4 Back

Clicking this button navigates you back to the previous web page in the Recording Surface window. It works like the Back button in your browser.

5.2.5 Forward

Clicking this button navigates you forward to the next web page in the Recording Surface window. It works like the Forward button in your browser.

5.3 Record



Clicking this button activates the recording window and toggles recording on and off. When recording is enabled any actions performed on the Recording Surface are automatically recorded and displayed as steps (line items) in Test Explorer. In addition the elements that you perform an action on in the Recording Surface are added to the Elements Explorer window.

5.4 Launch External IE for Recording

Clicking this button starts a new instance of Internet Explorer outside of the Visual Studio IDE. Design Canvas then attaches to this new instance allowing you to record tests in the same manner as the IE instance hosted inside Visual Studio's IDE.

5.5 Dialogs

The Dialogs dropdown menu enables you to insert automatic dialog handler steps into your test as a test step in Test Explorer. Using this dropdown you can add automatic dialog handles for the most common Win32 dialogs displayed by the browser. Section 5.3.1 describes dialog handlers in more detail.

5.5.1 Handling Pop-Up Dialogs

Automation Design Canvas includes native support for some of the common browser dialogs including JavaScript alerts, file upload, file download, logon, and a special generic dialog handler. The process for adding any of the five dialog handlers is the same:

- Double click on the WebAii test in Solution Explorer that you want to add a dialog hander to. This opens the test document window.
- Click on the Record button located at the top of the .aii document window. This opens the Recording Surface with that test being the active test.
- Click on the Dialog… dropdown located in the toolbar of the Recording Surface.

- Click on the dialog hander you want added to your test. This adds a dialog handler for that type of dialog as the last step of your test. If the last step is not the correct location for the dialog handler step you can move it anywhere in the test sequence.

After adding the dialog hander step to your test you must open the properties and customize the step to match the requirements of your test.

5.5.1.1 JavaScript Alert dialogs

The JavaScript alert dialog handler is used as the standard JavaScript security alert dialog for both Internet Explorer and Firefox. It has two customizable properties:

- HandleButton – Specifies which button the dialog handler needs to click on to dispose of the dialog. Normally you would leave this at the default of OK to automatically handle JavaScript alert dialogs.
- HandleTimeout – Specifies the amount of time in milliseconds to wait for the dialog to close after dialog handler has tried to handle it. If the timeout expires the test step is set as having failed. This will abort the test, unless Continue on Failure is set on this step.

5.5.1.2 File upload dialogs

The file upload dialog handler handles the standard file upload dialogs for both Internet Explorer and Firefox. It has three customizable properties:

- File upload path – You must enter the path and filename for the handler to insert into the file upload's file text box. **Note**: Be sure to enter a path to a real file. If you do not the file upload dialog will reject the path and just pop right back up. This will cause the test to enter an infinite loop as it constantly tries to handle the dialog that will never go away permanently.
- HandleButton – Specifies which button the dialog handler needs to click on to dispose of the dialog. Normally you would leave this at the default of OPEN to automatically handle the file upload dialog.
- HandleTimeout – Specifies the amount of time in milliseconds to wait for the dialog to close after dialog handler has tried to handle it. If the timeout expires the test step is set as having failed. This will abort the test, unless Continue on Failure is set on this step.

5.5.1.3 File download dialogs

The file download dialog handler handles the standard file download dialogs for both Internet Explorer and Firefox. It has three customizable properties:

- Download path – You must enter the path and filename for the handler to insert into the file downloads file text box. **Note**: Be sure to enter a valid real path. If you don't the file download dialog will reject the path and just pop right back up. This will cause the test to enter an infinite loop as it constantly tries to handle the dialog that will never go away permanently.

- HandleButton – Specifies which button the dialog handler needs to click on to dispose of the dialog. Normally you would leave this at the default of SAVE to automatically handle the file upload dialog.
- HandleTimeout – Specifies the amount of time in milliseconds to wait for the dialog to close after dialog handler has tried to handle it. If the timeout expires the test step is set as having failed. This will abort the test, unless Continue on Failure is set on this step.

5.5.1.4 Logon dialogs

The logon dialog handler handles the standard file logon dialog for both Internet Explorer and Firefox. It has four customizable properties:

- Username – You must enter the username to be entered into the logon dialog by the dialog hander.
- Password – You must enter the password to be automatically entered into the logon dialog by the dialog hander. **Security Note**: Even though the password is not displayed in clear text here, it is stored in plain text in the project. It is **strongly** recommended to only use special test accounts for logging on since anyone with access to the project files will be able to discover the account logon credentials by picking apart the project files.
- HandleButton – Specifies which button the dialog handler needs to click on to dispose of the dialog. Normally you would leave this at the default of OK to automatically handle the logon dialog.
- HandleTimeout – Specifies the amount of time in milliseconds to wait for the dialog to close after dialog handler has tried to handle it. If the timeout expires the test step is set as having failed. This will abort the test, unless Continue on Failure is set on this step.

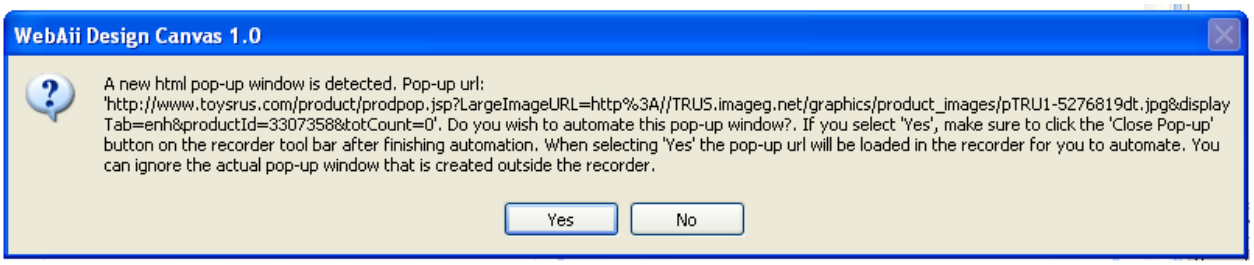
5.5.1.5 Generic dialog handler

The generic dialog handler is a special one that accepts additional properties allowing it to recognize and handle almost any simple Win32 dialog. It has seven customizable properties:

- Dialog title – This specifies the string that appears in the dialogs title bar. It can be the full exact string or a partial string depending on the “Match partial title” setting.
- Match partial title – When set to True, the dialog title must match exactly. When set to false the dialogs title must contain the string from the “Dialog title” property somewhere/anywhere in the title.
- Child window text content – This optional property may contain a text string that will appear in the dialog window. This can help isolate this dialog from other dialogs that might have the same or similar titles. If you leave this property blank it will be ignored.
- Handle button method – Specifies how to handle the dialog. It can be one of three settings:
- Button ID – The dialog handler will click on the button having the specified ID. You will need to use some sort of UI spy utility to determine the correct button ID.
- Button partial text – The dialog handler will click on the button that contains a text string.
- None close dialog – The dialog handler will not try to click on a dialog button. Instead it will close the dialog by clicking the close big red X in the upper right corner of the dialog.

- Button ID – Specifies the ID of the button to be clicked on by the dialog handler. This property is only used when “Handle button method” is set to “Button ID”.
- Button partial text – Specifies the text string the button will contain that the dialog handler is to click on. This property is only used when “Handle button method” is set to “Button partial text.
- HandleTimeout – Specifies the amount of time in milliseconds to wait for the dialog to close after dialog handler has tried to handle it. If the timeout expires the test step is set as having failed. This will abort the test, unless Continue on Failure is set on this step.

5.5.2 Handling HTML Pop-ups



Automation Design Canvas handles both HTML Pop-ups and Win32 Dialogs. HTML Pop-ups are automatically detected by the recording surface. Once detected, they are loaded in the surface if you with to automate them. Win32 Dialog Handlers can easily be added at any point in your test execution by adding one from the Dialogs drop down menu. Design Canvas also exposes a "Generic" dialog handler than can be customized to handle any Win32 dialog outside the out-of-box ones provided.

One of the biggest pain points in Design Canvas 1.0 was recording complex HTML pop-up sequences. To overcome this shortcoming in V1.0 we put together significant effort into Design Canvas 1.1 to support a flexible and rich environment for HTML pop-ups that can handle all of the possible HTML pop-up scenarios. We are pleased with the results and we believe you will be too.

5.5.2.1 Changes in 1.1

Prior to V1.1, the Design Canvas would intercept HTML pop-ups and attempt to load them inside the browser hosted inside of Visual Studio. That approach had several drawbacks especially in the following scenarios:

- Pop-ups that communicate between each other.
- Pop-ups that attempt to close the IE instance that opened it.

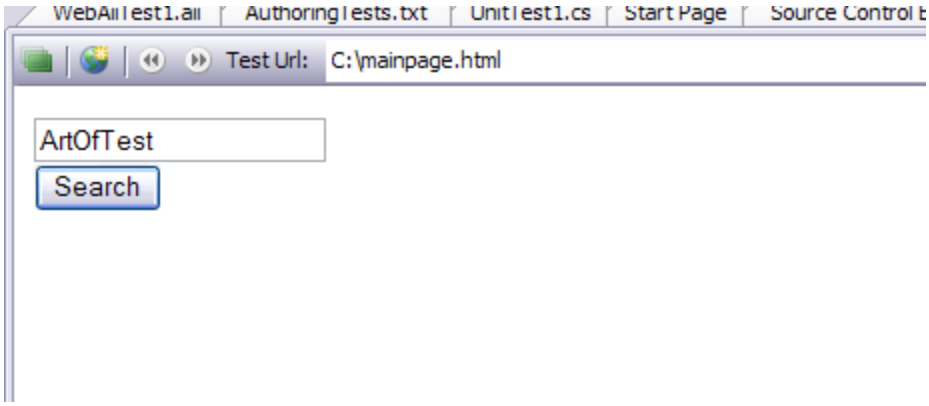
The biggest fundamental enhancement to Design Canvas V1.1 is the ability to record against an independent instance of Internet Explorer in place of an instance of IE that is hosted inside of Visual Studio. This enhancement enables recording from several instances of IE simultaneously which makes handling of pop-up support much easier.

We no longer attempt to intercept the HTML pop-up. Instead we simply allow the pop-up to pop naturally and attach a recorder to each instance as they open. This approach allows recording to continue as if they were all one instance. Furthermore, you can start recording from a standalone instance of IE instead of the one hosted inside of Visual Studio.

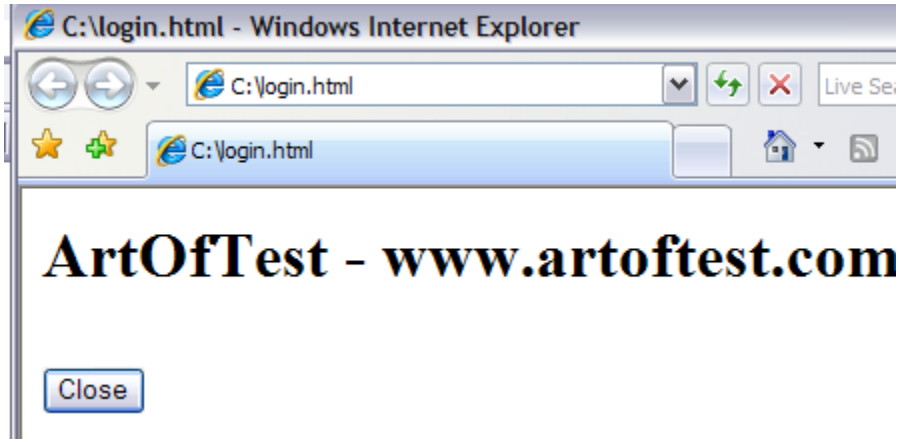
5.5.2.2 Handling HTML pop-up recording in V1.1

Let's look at an example. Suppose we have a page loaded in the hosted browser that pops-up an HTML page. The HTML page then closes itself by clicking the close button.

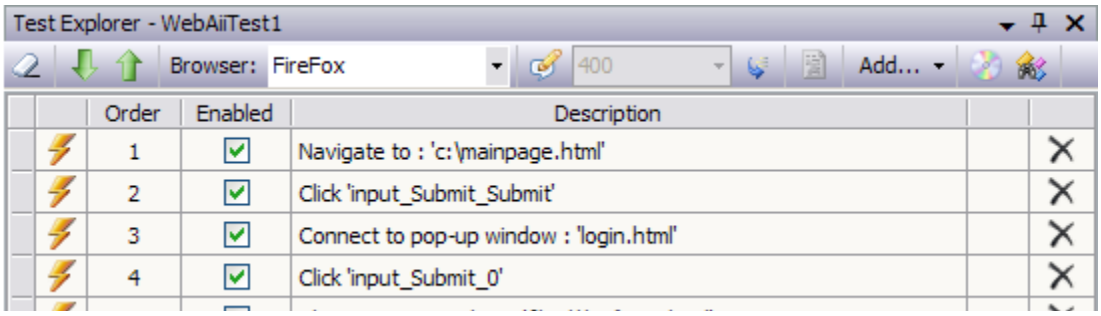
Mainpage.html



Clicking on the "Search" button launches the html pop-up (**login.html**)



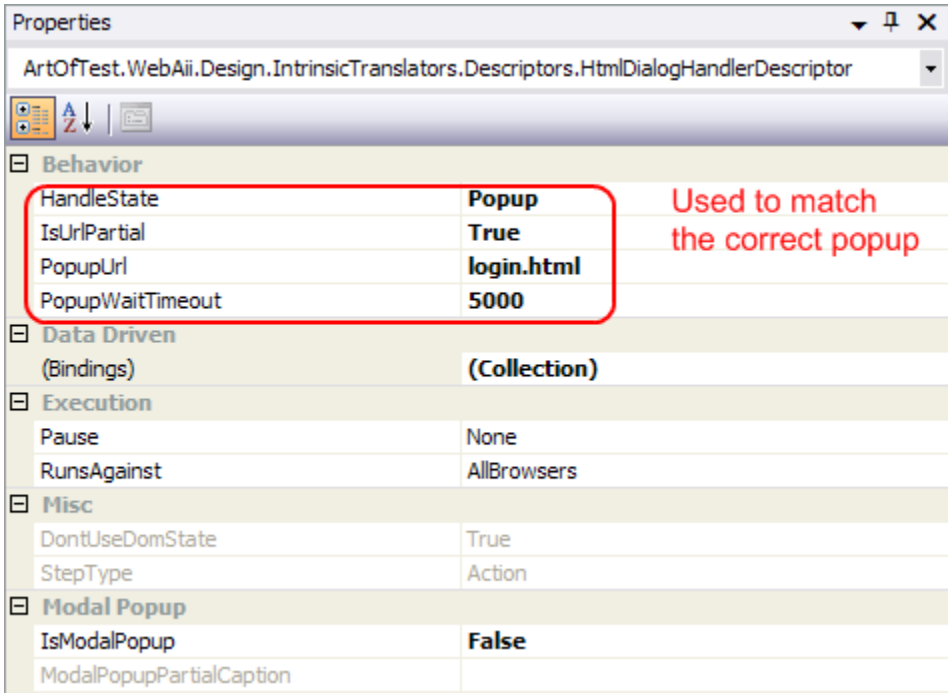
When recording this scenario inside the recorder we end up with the following steps:



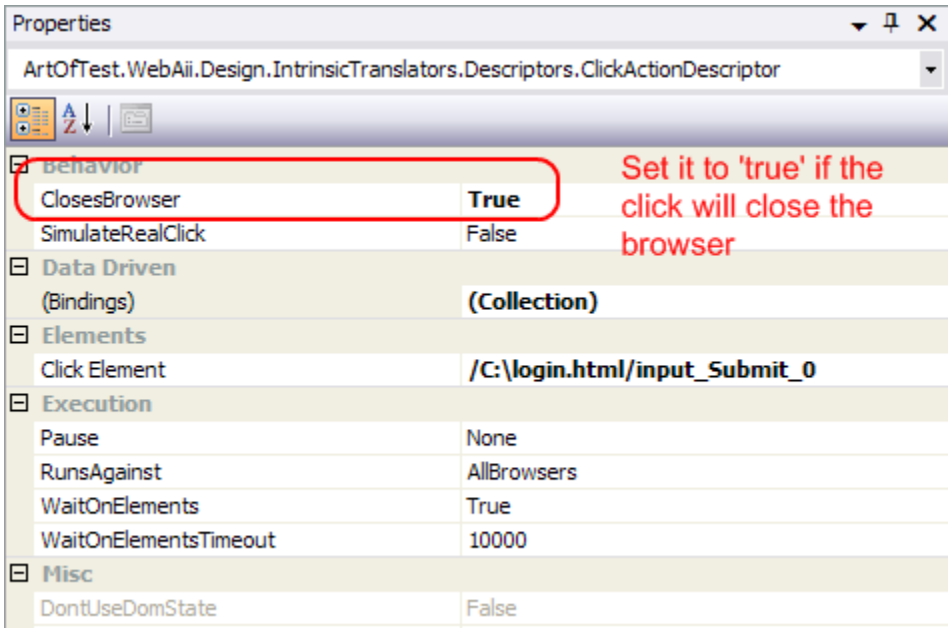
Once the recording is done, there might be few updates needed to each step to customize it to the current html pop-up behavior.

Steps:

- **Click 'input_Submit_Submit':** Clicks the button that launches the pop-up. In most cases there is no need to make any changes to the properties of this step.
- **Connect to the pop-up window: 'login.html'**. Make sure the Popup URL captured in the properties (as shown below) will match the popup



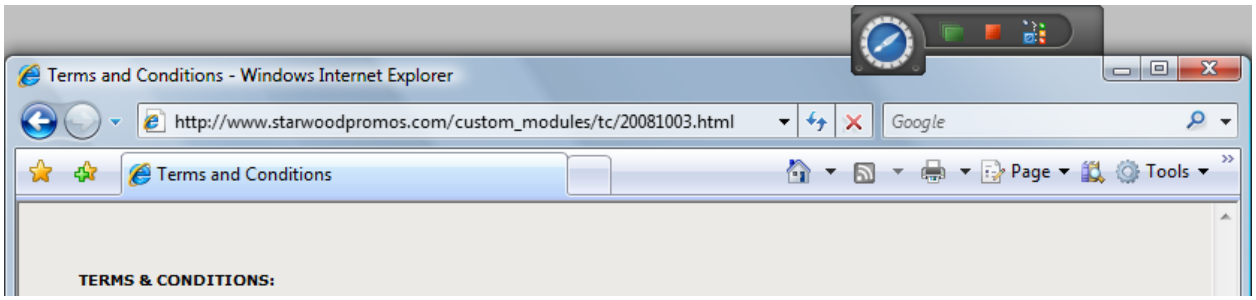
- **Click 'input_Submit_0':** This step will click the Close button on the popup page.



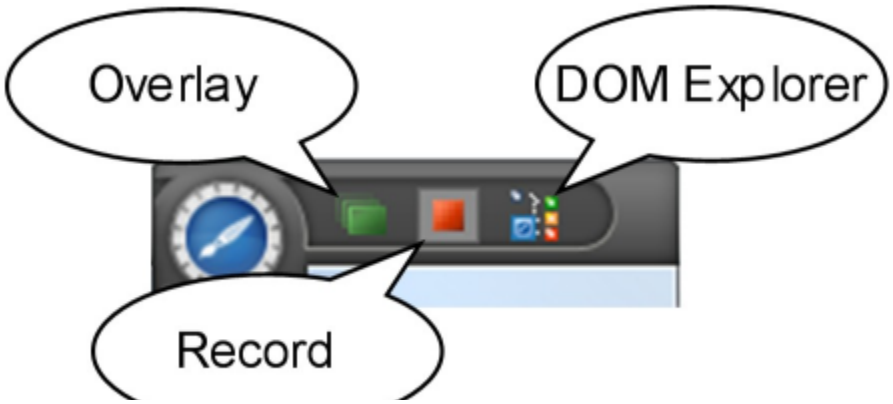
- In this scenario, this click will close the browser so we set 'ClosesBrowser' = true.
- **Close pop-up window.** This step will close the pop-up window.

5.5.2.3 Recording from disconnected browser instance

When an HTML pop-up is launched, you will notice an attached recorder tool bar with the Design Canvas logo attached to it. This toolbar allows you to enable/disable recording, enable/disable highlighting and show/hide the DOM Explorer. You have the same recording functionality for HTML pop-ups as the functionality available in the Visual Studio hosted browser including verification recording.

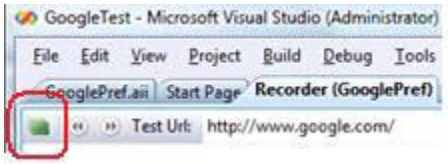


The buttons on the toolbar are defined in the diagram below:



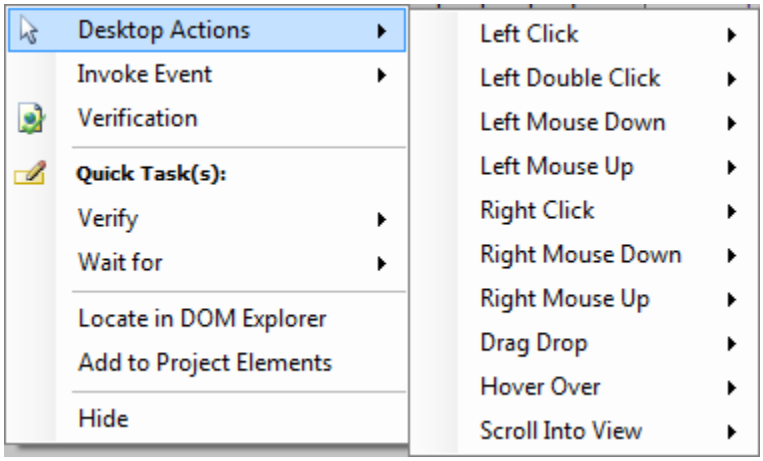
5.6 Automation Overlay Surface

Automation Overlay Surface™ (AOS) enables you to visually highlight an element, locate it in the DOM tree, and automatically generate common verification or synchronization tasks against it. These generated tasks are automatically added to your test as a step. For example, you can verify an item is selected in a dropdown or wait for an element to become visible. These tasks are automatically initialized to the current state of the highlighted element. As soon as you pick an action an automation step is instantly recorded against that element with the proper settings without the need to enter anything. You activate AOS by clicking the Enable Overlay button as shown in the following with a red outline:



Once enabled hovering the mouse over any element in the Recording Surface will highlight that element by drawing a rectangle around it. When an element is highlighted you can right click on it to open the context menu. The following sections describe the context menu.

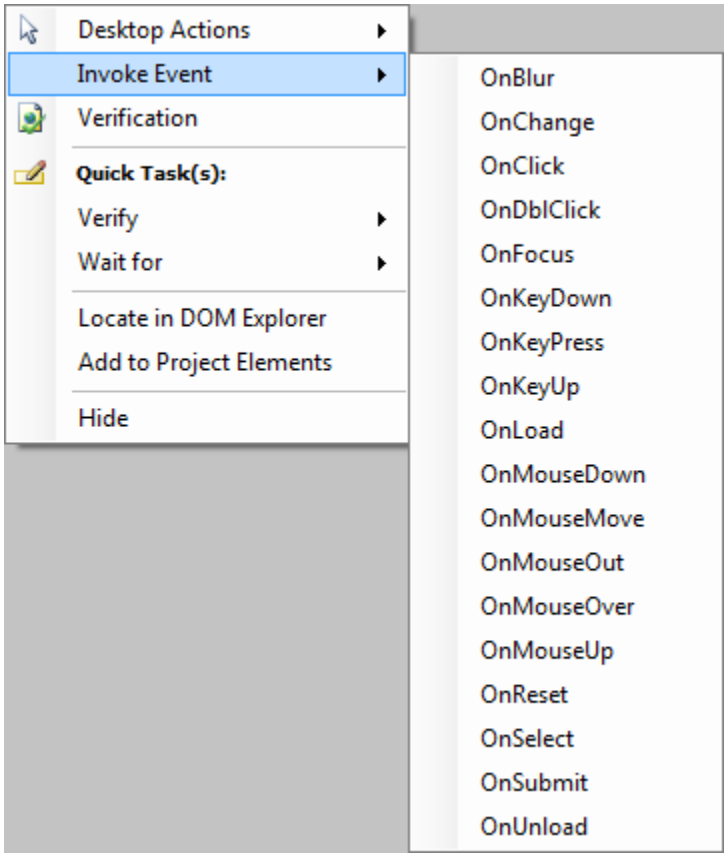
5.6.1 AOS context menu: Desktop Actions



Desktop actions are the following mouse and scroll window actions that can be performed on the Recording Surface highlighted element:

- Left click – Moves the mouse cursor to a point relative to the element and simulates a single left mouse button click. The point must be somewhere within the rectangular bounds of the element.
- Left double click – Moves the mouse cursor to a point relative to the element and simulates a double left mouse button click.
- Left mouse down – Moves the mouse cursor to a point relative to the element and simulates a left mouse button down event.
- Left mouse up – Moves the mouse cursor to a point relative to the element and simulates a left mouse button up event.
- Right click– Moves the mouse cursor to a point relative to the element and simulates a single right mouse button click.
- Right mouse down– Moves the mouse cursor to a point relative to the element and simulates a right mouse button down event.
- Right mouse up– Moves the mouse cursor to a point relative to the element and simulates a right mouse button up event.
- Drag drop – Performs a drag and drop operation. The starting point is always relative to the current element. The ending point can be either a point relative to the same element, a different destination element, or a browser window coordinate.
- Mouse Hover Over – Moves the mouse to a point relative to the element and leaves it there.
- Scroll Into View – Scrolls the browser window to make the element visible. This is a necessary or precautionary step to perform before acting on the element or perhaps before taking a screen shot.

5.6.2 AOS context menu: Invoke Event



Use Invoke Event to invoke any of the standard JavaScript events on the highlighted element in the AOS Recording Surface. Automation Design Canvas supports invoking the following JavaScript events:

- OnBlur
- OnChange
- OnClick
- OnDblClick
- OnFocus
- OnKeyDown
- OnKeyPress
- OnKeyUp
- OnLoad
- OnMouseDown
- OnMouseMove
- OnMouseOut
- OnMouseOver
- OnMouseUp
- OnReset
- OnSelect

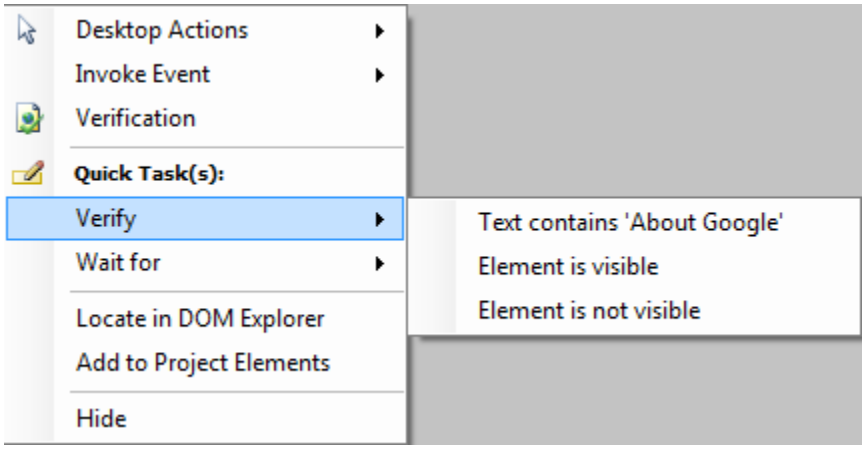
- OnSubmit
- OnUnload

See http://www.w3schools.com/jsref/jsref_events.asp for a technical description of each of these JavaScript events.

5.6.3 AOS context menu: Verification

Selecting Verification opens the Verification Builder dialog. Use Verification Builder to create almost any type of element verification step possible, such as attribute or style verification or attribute select drop-down. When defining your test completing Verification Builder adds verification step(s) against the highlighted element to Test Explorer. Section 6.1 describes how to use Verification Builder.

5.6.4 AOS context menu – Quick Tasks: Verify



Use "Verify" to quickly add common verification steps for the highlighted element to Test Explorer. The options available depend on the element selected. All elements have two options:

- Element is visible – This verification step assumes that the element is already present in the DOM. WebAii tests whether or not an element can be seen on the web page by analyzing its "visibility" (http://www.w3schools.com/CSS/pr_class_visibility.asp) and "display" (http://www.w3schools.com/CSS/pr_class_display.asp) attributes set for that element. WebAii follows the Cascading Style Sheets (CSS) chain as needed to determine the current visibility state of the element.
- Element is not visible – This verification step assumes that the element is already present in the DOM. It tests whether or not the element cannot be seen on the web page by analyzing the visibility attribute set for that element.

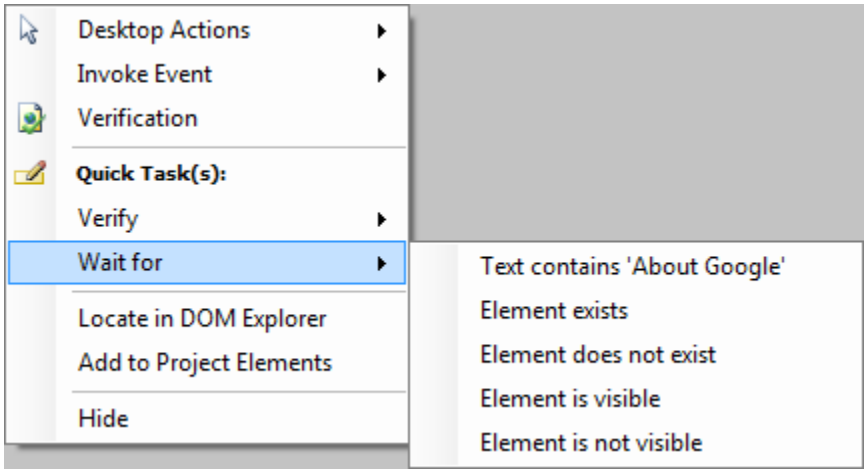
Besides these the following items will also display in the Verify menu:

- Elements that contain text will have a "Text contains <some text>" menu entry.
- Radio buttons and checkboxes will have a "Checked state is True" or "Checked state is False" menu entry depending on the current checked state of the element.

- Dropdown select elements will have a “Selection ‘Text’ is <some text>” menu entry and a “Selection ‘Value’ is <some value>” menu entry.
- Table elements will have a “Table has ‘n’ column(s).” menu entry and a “Table has ‘n’ row(s).” menu entry.

Parameters for the different types of “Verify” steps are detailed in Appendix A5.

5.6.5 AOS context menu – Quick Task(s): Wait for



AOS context menu Quick Task item “Wait for”, works like “Verify”. Use it to add a “wait for…” step to your test. All elements allow you to wait for one of the following four conditions to be true:

- Element exists – The test will wait until the element is present in the DOM or a timeout occurs. If a timeout occurs the test throws an exception causing the test to fail and abort unless “Continue on Failure” is selected. Section 5.8.2.5 describes “Continue on Failure”.
- Element does not exist – The test will wait until the element is no longer present in the DOM or a timeout occurs. If a timeout occurs the test will throw an exception which causes the test to fail and abort unless “Continue on Failure” is selected. Section 5.8.2.5 describes “Continue on Failure”.
- Element is visible – The test will wait until the element is visible in the browser window or a timeout occurs. WebAii tests whether or not an element can be seen on the web page by analyzing the “visibility” (http://www.w3schools.com/CSS/pr_class_visibility.asp) and “display” (http://www.w3schools.com/CSS/pr_class_display.asp) attributes set for that element. WebAii follows the CSS chain as needed to determine the current visibility state of the element. If a timeout occurs the test throws an exception which causes the test to fail and abort unless “Continue on Failure” is selected. Section 5.8.2.5 describes “Continue on Failure”.
- Element is not visible – The test will wait until the element is no longer visible in the browser window or a timeout occurs. If a timeout occurs the test throws an exception which causes the test to fail and abort unless “Continue on Failure” is selected. Section 5.7.2.5 describes “Continue on Failure”.

Besides these the following items will also display in the Wait for menu:

- Elements that contain text will also display, “Text contains <some text>”.
- Radio buttons and checkboxes will display, “Checked state is True” or “Checked state is False” depending on the current checked state of the element.
- Dropdown select elements will display, “Selection ‘Text’ is <some text>” and “Selection ‘Value’ is <some value>”.

Appendix A10 describes the parameters for the various types of “Wait for” steps.

5.6.6 AOS context menu: Locate in DOM Explorer

Clicking **Locate in DOM Explorer** in the AOS context menu sets the focus on that element in the DOM tree as displayed in the DOM Explorer window. This is useful when you want to see the element attributes or you want to navigate to the parent of the element. This is the easiest method to locate the <Table> element that contains the cell displayed in the browser window. Section 5.9 describes how to use the DOM Explorer window.

5.6.7 AOS context menu: Add to Project Elements

Clicking **Add to Project Elements** adds the highlighted element to the list of elements maintained in the Elements Explorer window. Normally you do not need to manually add elements to Elements Explorer. Elements are automatically added as you record your actions on the Recording Surface.

Suppose however you need to perform some task in a code-behind method. You probably need to refer to an element on the web page so you can perform an action or verification on it. Code-behind methods can reference elements in Elements Explorer more easily than if you hard code their FindParam or otherwise find the element programmatically in the browser’s DOM. Or you may need to modify the source and/or destination element of an already recorded Drag Drop step.

There are two advantages to keeping and using element references in Elements Explorer:

- You centralize maintenance of how to locate the element on the web page to one place. If all test steps, both recorded and code-behind methods, refer to the same element in Elements Explorer then you only need to update the element’s FindParam (how it is found) in one place (Elements Explorer) instead of having to update every place the element is used in your test.
- It is easier for code-behind methods to use element references from Elements Explorer than to use their own hard-coded methods.

5.6.8 Hide

Acts like “cancel”. Clicking on this menu item simply closes the context menu so you can continue your recording without having to select on something or click somewhere on the recording surface (which may have the side effect of recording an unwanted click event if recording is enabled at the time of the click).

5.7 Automation Design Canvas Settings

Clicking this button opens the Automation Design Canvas User Settings dialog. This is explained in detail in Appendix B.

5.8 Storyboard and the Steps Tab

Clicking the **Steps** tab brings up the storyboard.

5.8.1 Storyboard

The storyboard is a series of screenshot thumbnails showing your recorded test steps. As you record your test, a screenshot of your action on the target element is captured to the storyboard. This gives you a visual flow of how your test has progressed. The element being acted on is highlighted in the snapshot with a red rectangle:

The following is an example of what a WebAii document storyboard looks like:

5.8.2 Steps Tab

When you click on one of the thumbnails a large version of the snapshot is displayed in the Step panel:

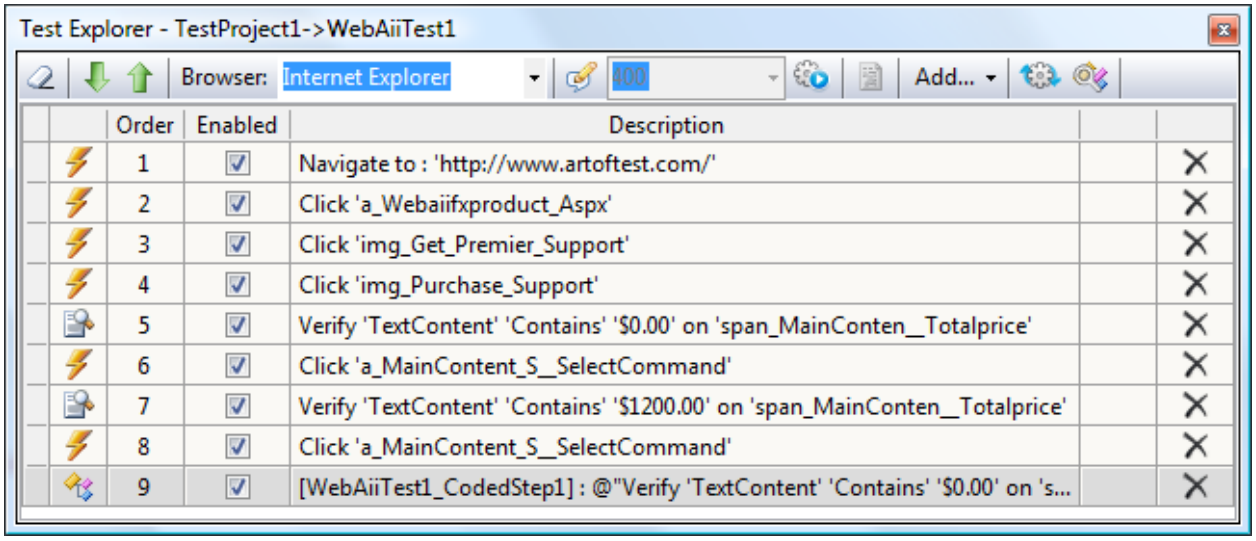
Step

11. [Enabled] - Check 'input_CheckBox_Lr_2' to be 'True'

5.9 Data tab

The data tab is used to create built-in data arrays for data-driven testing. Section 8.2 describes how to use the data tab to create and use your own data array.

5.10 Test Explorer Window

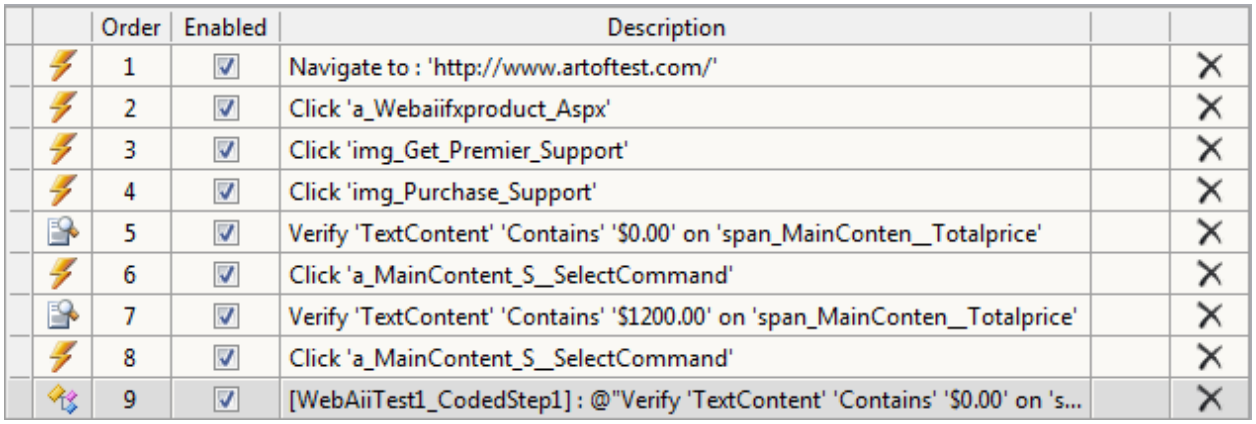


Test Explorer window is where you control all aspects relating to the execution of your test steps. It displays all the steps of the currently active test. Only one test is considered "active" at a time. Anytime you click on an .Aii Test Editor window or double click an .Aii document in the Solution Explorer window, that test becomes the "active" test.

Test Explorer allows you to modify your test steps (reorder, delete, change, etc.).

5.10.1 Steps

This part of the window displays all the steps that make up your test.



5.10.1.1 Step icons list

-  This icon represents an action step. Examples of actions include Navigate To, Click, Set Text, Set Check, Select, etc.
-  This icon represents a "Wait for" step. The step will wait until either the element is present and meets the condition or a timeout occurs.

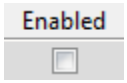
-  This icon represents a verify step. The step will verify that a setting of the specified element is true or that an element is present or absent. If the verification fails the test will abort at that step and not execute the rest of the steps (unless "Continue on failure" is turned on as described in section 5.7.1.5).
-  This icon represents a code behind step. Section 9.2.1 describes code behind steps.

5.10.1.2 Reordering test steps

There are two methods for reordering test steps:

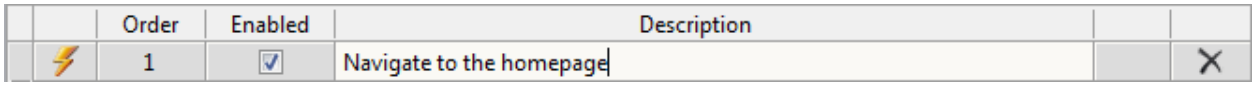
- You can use the "Move selected step up/down" arrows at the top of the window  . These move the currently selected test step either up or down in the sequence. They do not move multiple steps if you have more than one test step highlighted.
- You can drag and drop the test step that needs to be moved to its proper location. This method will only move one step at a time. If you have multiple test steps highlighted only the topmost step can be moved.
  If the location you want to move the step to is not in view drag the element over the scroll bar. The window will scroll up or down revealing the location you need to move the element to. Finally drop the element in the proper location.

5.10.1.3 Disabling



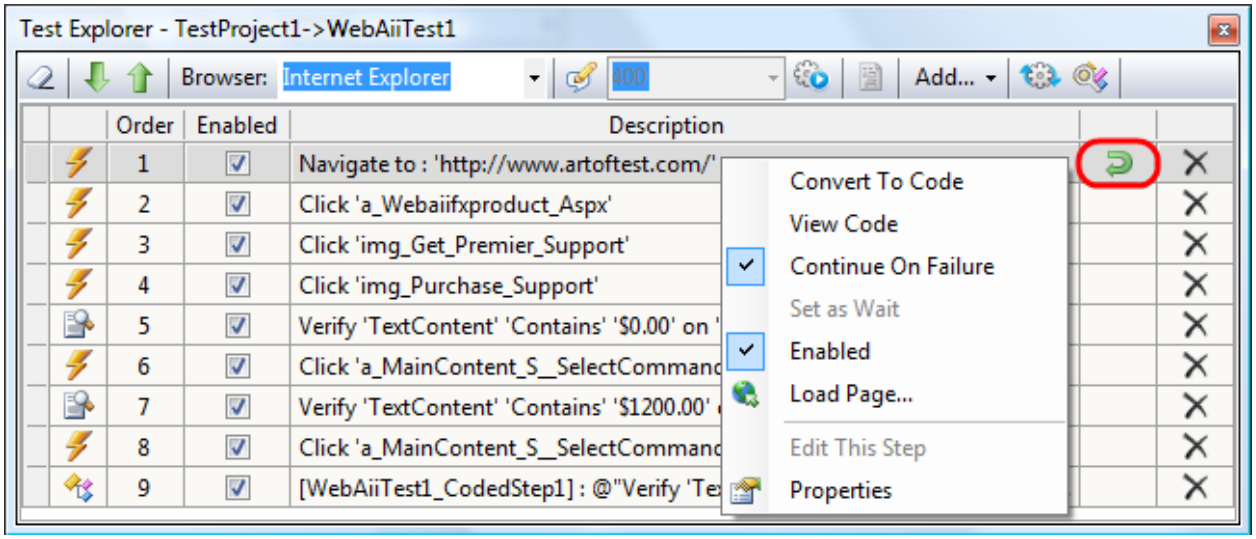
Every step can be enabled and disabled. Disabled steps are simply skipped when the test runs. This is useful during test development, troubleshooting or test case maintenance. To disable a test step uncheck the "Enabled" checkbox next to the step.

5.10.1.4 Changing test description



You can change the description of a test step by clicking on it then pressing F2. This puts the test step into edit mode allowing you to change the description to anything you like. You can also put the test step into edit mode by slowly double clicking on the step. If you click too fast Windows will interpret it as a double click (which is ignored by Test Explorer) instead of two single clicks.

5.10.1.5 Continue on failure



A test step that is marked "Continue on failure" will not cause the test to abort even if that step fails to execute normally. This is useful for rare or exceptional states. It does not mean that the test case has failed if that step alone fails. The test will still be marked as having failed when it runs however you will be able to see the complete results of the following test steps.

To mark a test step "continue on failure" right click on the step and select **Continue on failure**. Notice that the "Continue on failure" icon displays on the step when set. The icon is shown in the Test Explorer listing above as a green arrow, here outlined in red. To turn off "continue on failure" repeat the same steps.

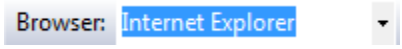
5.10.1.6 Deleting

Test steps can be deleted by clicking the 'delete' icon for the test step. **Note**: There is no undo for deletes. Once the step is deleted you must recreate it to get it back.

5.10.1.7 Clear all steps

Clicking the "Clear all steps" icon  deletes all of the steps in the test. **Note**: There is no undo. Once the steps are deleted you must recreate them to get them back.

5.10.2 Selecting browser for quick execute



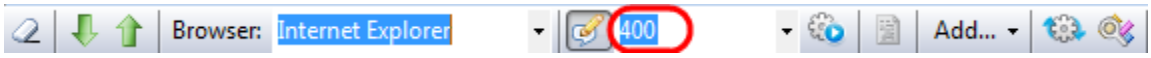
Automation Design Canvas supports running tests in Internet Explorer and Firefox. Which browser is used by Quick Execute is controlled by the selection in this dropdown.

5.10.3 Enable annotation

The annotator displays in the browser window which actions WebAii is performing. It also highlights the UI element that it is acting on. If the action does not involve a UI element (such as navigating to a URL or creating a cookie) the action is displayed at the top of the browser window. Here is what the annotator looks like in action:

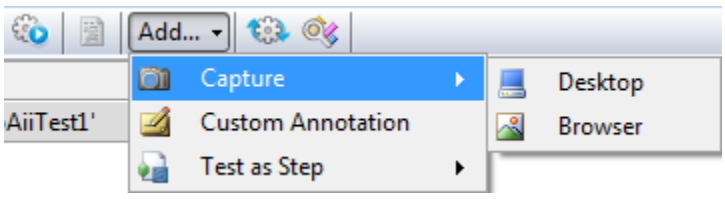
This icon  enables and disables annotation during quick execution. This can help you observe test execution. Section 9.6.2 describes how to enable annotation during test execution by Visual Studio.

5.10.4 Execution delay



This setting controls the execution delay (in milliseconds) between test steps during quick execution. This can help you observe test execution. Section 9.6.2 describes how to set execution delay during Visual Studio test execution.

5.10.5 Add additional steps dropdown



The “Add additional steps” dropdown adds a few special steps to Test Explorer. This is similar to how the Dialogs dropdown adds dialog handling steps to Test Explorer. The following sections describe the special steps you can add.

5.10.5.1 Capture Desktop

A Capture Desktop step takes and stores a screenshot of your desktop at that point in time. This can be useful for troubleshooting, verifying localization, and other tasks that require a snapshot for a visual inspection of the screen. After you add this custom step to Test Explorer you can go to the properties for this step to specify the path and filename to store the screenshot as. The default filename is “snapshot”. It is stored in the same folder where your VS test results are stored, typically:

```
“C:\Documents and Settings\testuser\My Documents\Visual Studio
2008\Projects\AoTWebsiteTests\TestResults\testuer_TESTPC 2008-06-13
14_34_38\Out”.
```

Note: Capture Desktop is only performed when the test is run from Test View or Test List Editor. It is not performed when it is run using Quick Execute.

5.10.5.2 Capture Browser

Just as with the Capture Desktop step, the Capture Browser step takes and stores a screenshot of the browser window at that point in time. This can be useful for troubleshooting, verifying localization, and other tasks requiring visual inspection of the screen at that point in time. After you add this custom step to Test Explorer you can go to the properties for this step to specify the path and filename for storing the screenshot. The default filename is “snapshot”. It is stored in the same folder where your VS test results are stored, typically:
“C:\Documents and Settings\testuser\My Documents\Visual Studio 2008\Projects\AoT_WebsiteTests\TestResults\testuer_TESTPC 2008-06-13 14_34_38\Out”.

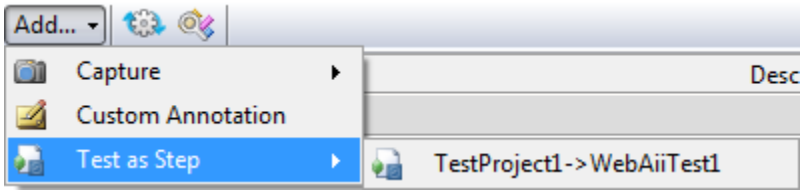
Note: Capture Browser is only performed when the test is run from Test View or Test List Editor. It is not performed when it is run using Quick Execute.

5.10.5.3 Custom annotation

In addition to the automation annotation described in section 5.7.3 you can have your own custom annotation displayed in the browser window. After you add this custom step to Test Explorer you can go to the properties for this step to specify:

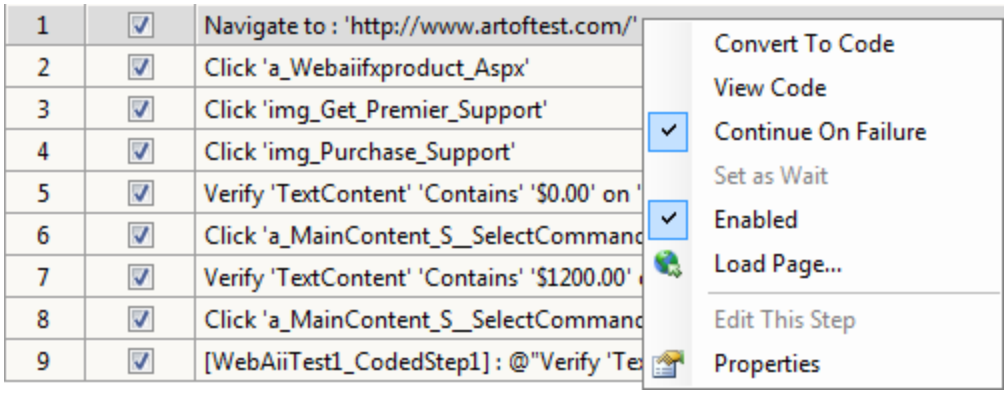
- The text to display.
- The location in the browser to display the text.
- The length of time (in milliseconds) to display the text.

5.10.5.4 Test as Step



"Test as Step" allows you to execute a previously crafted test as a step in the current test. This way you can modularize your test automation into smaller test blocks which are then incorporated into the main tests. For example, you can craft a smaller set of test steps that go through the logon sequence, or go through the purchase path of a website. Craft those test steps just once and incorporate them as a test step in the larger test case. In addition now you only need to update the smaller test block if that common sequence changes instead of having to replicate the change for all of your test cases.

5.10.6 Test Explorer Context menu



When you right click on a test step a context menu opens and presents you with additional settings and actions you can perform on that step. These are described in the following sections.

5.10.6.1 Test Explorer Context Menu: Convert to Code

The Convert to Code option converts that step into a code behind method. If a code behind file is not already attached to that WebAii test a new code behind file will be created, added to the project and attached to that test. Once a test step is converted into a code behind method it cannot be undone without re-recording the step.

When you convert a test step into a code behind method you can modify that step to do something you normally could not do with recorded steps alone. For example you may want a test step to conditionally perform an action depending on the presence or absence of an element on the web page. Since test steps do not support conditional execution (except for conditional browser execution) a code behind method is the only way to create the conditional execution.

5.10.6.2 Test Explorer Context Menu: View Code

When you right click on a step that refers to a code behind method and select the View Code option, the code behind file will automatically open and the cursor will be placed at the first line of that method. This is a shortcut to locating the referenced code behind method. It is very useful when you have numerous code behind files containing many code behind methods.

5.10.6.3 Test Explorer Context Menu: Continue On Failure

On occasion you may want the test to continue when a step fails. For example when you are verifying multiple items on a particular web page or a set of web pages and one failed verification item is not critical to the execution of the rest of the test. When you mark that step "continue on failure" the test will show as failed on completion but it will not abort the test if that step fails. In the test results you can see all failed steps allowing you to investigate the cause of each failure.

5.10.6.4 Test Explorer Context Menu: Set as Wait

The Set as Wait option is only available for WaitFor and Verification steps. Any verification step can be changed to a WaitFor step and vice versa. When you change a Verification step to a WaitFor step the test will not immediately fail if the condition being tested does not result in a True value. Instead WebAii will wait up to 10 seconds (the default setting) for the condition to be True. The default 10 second timeout can be change by opening the properties window for the step and changing the timeout value. Remember, the timeout value is in milliseconds, for example, 10 seconds = 10000 milliseconds.

5.10.6.5 Test Explorer Context Menu: Enabled

Every step can be enabled and disabled. Disabled steps are simply skipped when the test runs. This is useful during test development, troubleshooting, or test case maintenance. To disable a test step uncheck the "Enabled" in the context menu.

5.10.6.6 Test Explorer Context Menu: Load Page…

The menu item only appears for the "Navigate to" steps. Clicking on it will load that web page into the recorder window. Once loaded you can resume recording from that point.

5.10.6.7 Test Explorer Context Menu: Edit This Step

This menu item is enabled only for "Verify" steps. Clicking on it will open the "Sentence Verification Builder" wizard and initialize it with the parameters of that verification step. Now you can edit the parameters as needed and even test them (assuming the page containing that element is currently loaded in the recorder, otherwise the verification builder is set to read-only mode).

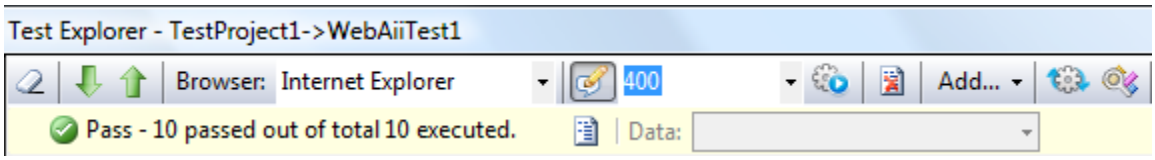
5.10.6.8 Test Explorer Context Menu: Properties

To view the properties for this test step select Properties. The properties window opens showing the properties and settings associated with this step. Appendix A describes all the possible test step properties.

5.11 Quick Execute

Clicking this icon  runs your test in the selected browser with the current annotation and execution delay settings, also called "quick executing" your test. It is important to note it is not being run with the normal VS testing framework. Test results will not be stored in the standard VS test results folder. This is intended for use only during test development. You can use it to quickly verify that your test performs properly before running it as part of a larger test case run.

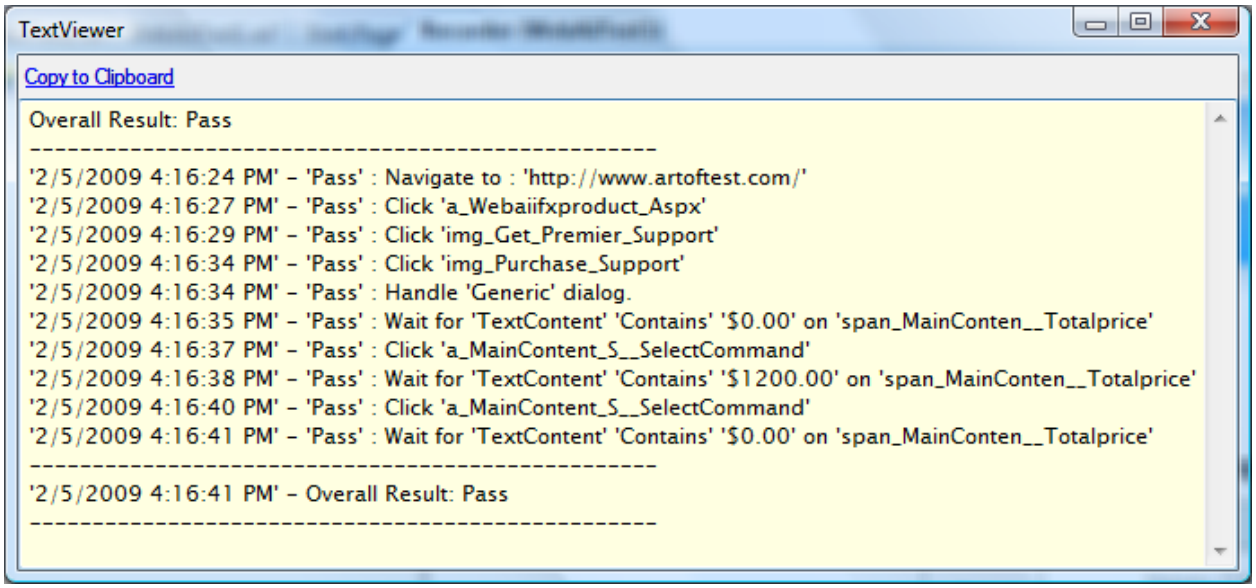
After your test runs you will see how many steps passed and how many failed, as shown in the following example:



The "Data" description shown in here is only displayed for data-driven tests. See also Section 8, Data Driven Testing.

5.11.1 Quick Execute: Results

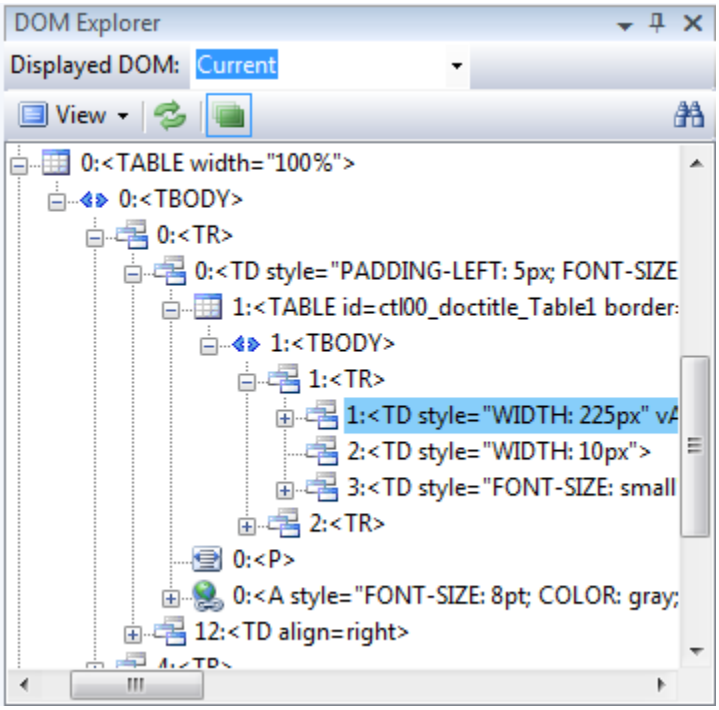
After you quick execute a test you get test results. To view the test results click this icon  . A text viewer opens displaying the detailed log of the quick execute test run. For example:



5.11.2 Quick Execute: Clear execution results

After a quick execute test run, clicking this icon  will clear the test results. The only reason for clearing the results is to make Test Explorer look less cluttered. Previous test results will automatically be cleared the next time you quick execute a test.

5.12 DOM Explorer



The DOM Explorer window displays the DOM (Document Object Model) of the web page currently loaded in the Recording Surface. The DOM is a platform and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure, and style of HTML documents. The document can be further processed and the results of that processing incorporated back into the presented page. Go to http://www.w3.org/DOM/ for a detailed description of the DOM and its technical specification.

5.12.1 DOM Explorer: Displayed DOM

The Displayed DOM: dropdown menu has four possible entries that appear under different conditions:

- Current – Displays the DOM for the web page currently loaded in the Recording Surface window. This is the only entry in the dropdown while recording a test.
- Execution – Displays the DOM captured during the execution of that step in the test. This option displays when WebAii fails to find the element on the web page which this test step references. Section 10.2.1 describes this in more detail.

- Recorded – Displays the original DOM captured during recording of this step during the test. This option displays when WebAii fails to find the element on the web page which this test step references. Section 10.2.1 describes this in more detail.
- (both) – Displays both the execution and the recorded DOM in DOM Explorer side-by-side. This allows you to view both and compare them. You are presented with this option when WebAii fails to find the element on the web page which this test step references. Section 10.2.1 describes this in more detail.

5.12.2 DOM Explorer: View

The View dropdown menu controls which view mode DOM Explorer displays. It can be one of two modes, 'Hierarchy' or 'TagName'.

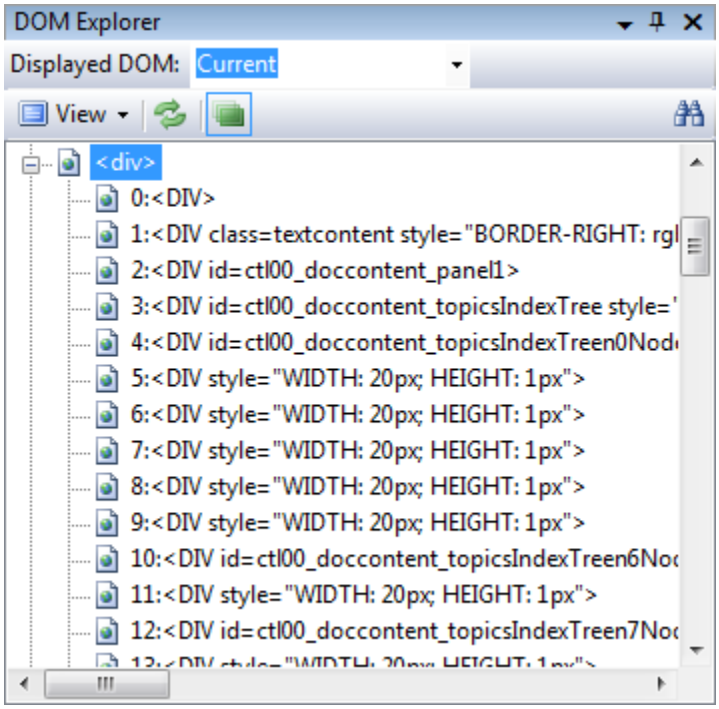
5.12.2.1 DOM Explorer –View Menu: Hierarchy

The Hierarchy mode setting in the View menu displays the DOM in a standard DOM hierarchical tree. At the top of the tree is an <html> node as the root node of the tree. Beneath this node a <head> and <body> node appears. What appears beneath these nodes depends on the content of the HTML document. You can drill further down the tree to find other elements contained in the DOM.

5.12.2.2 DOM Explorer –View Menu: TagName

The TagName mode setting in the View menu displays the DOM as a tree view showing all the different element tag types along with their tag index for the elements that appear on the web page currently loaded in the Recording Surface. Each different tag type is displayed as a root node in the tree. It only displays tags present in the HTML document and does not display any common HTML tags that are not actually present in the document.

Normally you will see <html>, <head>, and <body> tags in the tree even for an empty web page since this is the minimum required for a document to be a valid HTML document. What else appears in the tree depends on the actual content of your HTML document, for example:



5.12.3 DOM Explorer: Highlight selected element

This icon enables/disables highlighting of elements in the Recording Surface. When enabled and you click on an element in DOM Explorer a red rectangle will be drawn around it. The web page will automatically scroll up or down as needed in the Recording Surface to make the element visible if it happens to be outside the current viewport.

When the DOM Explorer view mode is set to TagName clicking on one of the tag nodes will highlight all elements in the Recording Surface that have that tag node name.

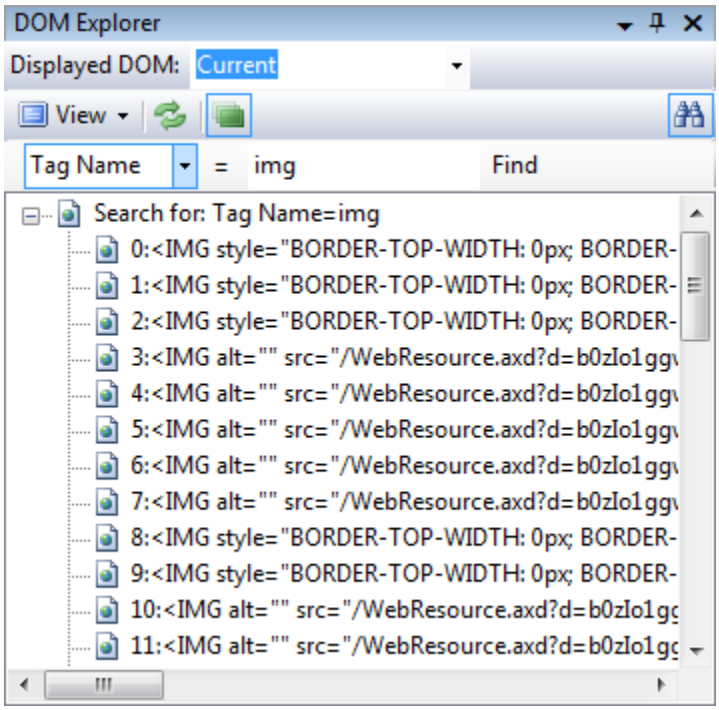
5.12.4 DOM Explorer: Refresh

Clicking on the Refresh icon causes Automation Design Canvas to reload the DOM of the web page displayed in the Recording Surface. You will rarely need to do this. Design Canvas usually

detects when the contents of the web page have changed and will automatically refresh the DOM tree. However you might be working with a dynamically updating web page and need to manually refresh the DOM tree by clicking on this icon.

5.12.5 DOM Explorer: Search

Clicking the 'Search DOM' icon opens the search toolbar where you can search for elements contained in the DOM. For example:



5.12.5.1 DOM Explorer – Search: Search type

The Search type dropdown menu specifies what type of content you want to find. It can be one of the following values:

- Tag Name – Select this when you want to search for elements having a particular tag, such as or <a> or <p>, etc.
- Id – Select this when you want to search for an element or elements have a particular ID.
- Class – Select this when you want to search for all elements that are associated with a particular class.
- Name – Select this when you want to search for an element or elements have a particular name.
- Content – Select this when you want to search for elements that have a specific text content.

5.12.5.2 DOM Explorer – Search: Exact or partial match

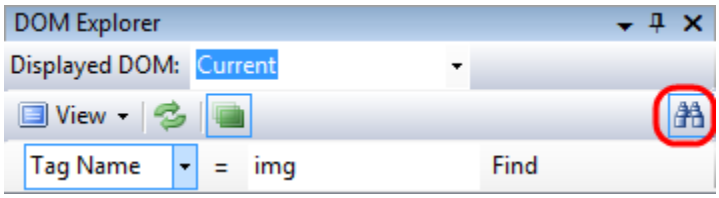
Clicking on this icon toggles between an exact match and a partial match search. When exact match search is active the attribute being searched must match the value being searched for

exactly. When partial match search is active, you will get all elements where the value being searched for is contained anywhere in the attribute of the element.

5.12.5.3 DOM Explorer – Search: Search string

Enter the value you want to look for in this text box.

5.12.5.4 DOM Explorer – Search: Find button



Clicking the Find button starts the search. The search results are displayed in DOM Explorer as a hierarchical tree. All elements that match the search criteria appear as nodes beneath the root of the tree.

5.12.6 DOM Explorer: Context menu

Whenever you right click on a node in DOM Explorer a context menu displays. The following sections describe the items contained in this context menu.

5.12.6.1 DOM Explorer – Context Menu: Record Options

Selecting the Record Options menu item opens the Automation Overlay Surface™ (AOS) menu for that element as described in section 5.3.2.

5.12.6.2 DOM Explorer – Context Menu: Add to project elements

Clicking on Add to Project Elements adds that element to the list of elements maintained in the Elements Explorer window. You do not typically need to manually add elements to Elements Explorer; elements are automatically added as you record your actions on the Recording Surface. Suppose however you must perform a task in a code behind method. You probably need to refer to an element on the web page so that you can perform an action or verification on it. Code behind methods can refer to elements in Elements Explorer much more easily than if you hard code their FindParam or otherwise find the element programmatically in the browser's DOM.

There are two advantages to keeping and using element reference from Elements Explorer:

- You can centralize the task of locating elements on the web page to one place. If all test steps, both recorded and code behind methods, all refer to the same element in Elements Explorer then you only need to update its FindParam in one place, Elements Explorer, instead of having to update every occurrence where the element is used in your test.
- It' i easier for code behind methods to use element references from Elements Explorer than to use their own hard-coded methods.

5.12.6.3 DOM Explorer – Context Menu: Properties

When you select Properties, the complete DOM properties for that element are displayed in the properties window. You cannot change any property values as they are in a read-only state but you can view them to get detailed DOM information about the selected element.

5.13 Taking Snapshots

Automation Design Canvas has the ability to take snapshots of the browser and the desktop at any point in time. It stores them as .png files on disk. The default behavior is to store the snapshot in the TestResults folder of your Visual Studio solution. On XP and Windows 2003 a typical folder is:

```
C:\Documents and Settings\<user>\My Documents\Visual Studio
2008\Projects\<project folder>\TestResults\<user>_<computer> 2008-06-17
15_51_22\Out\Snapshot0.png
```

On Windows Vista this folder is typically:

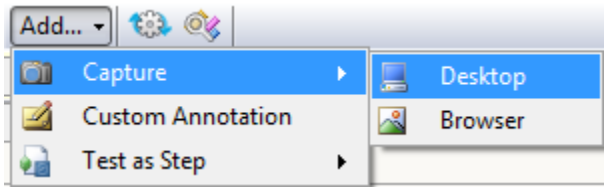
```
C:\Users\<user>\Documents\Visual Studio 2008\Projects\<project
folder>\TestResults\<user>_<computer> 2008-06-17 15_51_22\Out\Snapshot0.png
```

Note: When running the test using Quick Execute, Automation Design Canvas skips these steps. It will only take a snapshot when run through Test View or Test List Editor.

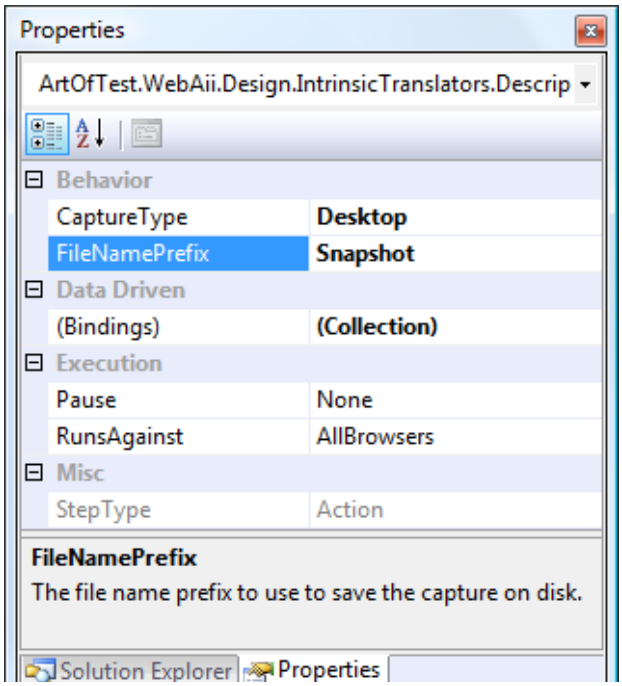
5.13.1 Snapshot of the desktop

To add a "Capture Desktop" step to your test:

- Click the **Add…** drop down found on the Test Explorer toolbar.
- Hover the mouse over Capture.
- Then click Desktop.



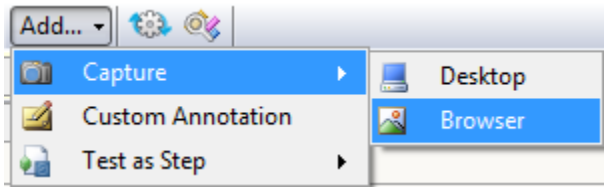
- A Capture Desktop step will be immediately added as the last step in your test. If the last step is not the correct location for the Capture Desktop step you can move it anywhere in the test sequence. Now you have the option to change the location and/or filename of the snapshot file.
- Open the properties for the new Capture Desktop step.
- In the FileNamePrefix property enter either a new filename or a path and filename.
  ***Note**: If you specify a hardcoded path, Automation Design Canvas will overwrite any existing files with the same name.*



5.13.2 Snapshot of the browser

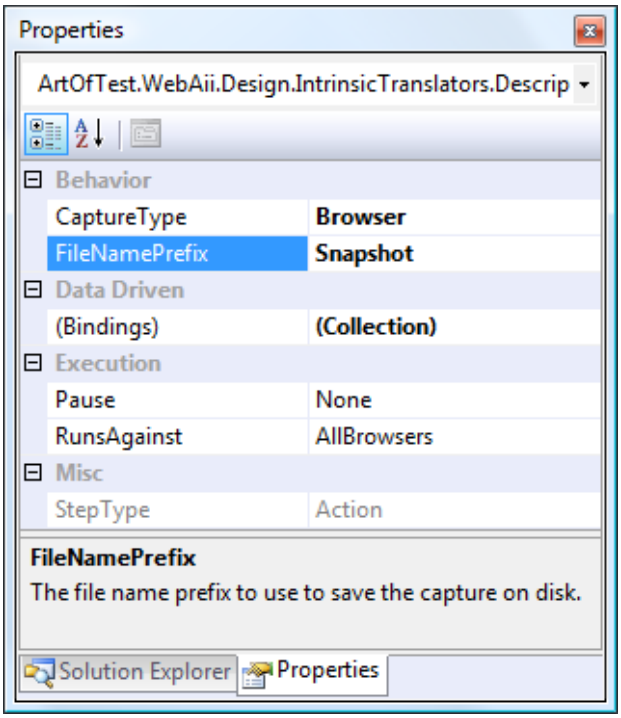
To add a "Capture Browser" step to your test:

- Click on the **Add…** drop down on the Test Explorer toolbar.
- Hover the mouse over Capture.
- Click on Browser.



- A Capture Browser step will be immediately added as the last step in your test. If the last step is not the correct location for the Capture Desktop step you can move it anywhere in the test sequence. You now have the option to change the location and/or filename of the snapshot file.
- Open the properties for the new Capture Browser step.
- In the FileNamePrefix property enter either a new filename or a path and filename.
  ***Note**: If you specify a hardcoded path, Automation Design Canvas will overwrite any*

existing files with the same name.



5.14 Re-Using Test Cases

Another time-saving Design Canvas feature is the ability to reuse test cases. For example your test may require you to login first before you can access other web pages but you do not want each test to include this log in step each time. Most test tools require you to navigate to the URL, enter the userid and password, and login for every test. But in Design Canvas you only need to define this test with the login steps once. All other tests that require logging in can simply refer to this 'login test'.

To re-use the test, once you have defined your login test for example, and you are defining subsequent tests, in Test Explorer simply:

- Select the test step that needs to include the login step.
- Click **Add** then click **Test as Step**, illustrated below. This displays a list of all test steps in your project.
- Select the test step that has the login steps.

In this way you have allowed one test to be referenced (re-used) by other tests.

5.14.1 Setting IsTestFragment to True or False

IsTestFragment can be set to True or False in the test view properties window. The default is False. If IsTestFragment is set to True then:

- It will be acknowledged if run as a Visual Studio test.
- It will be ignored if run from Test Explorer.

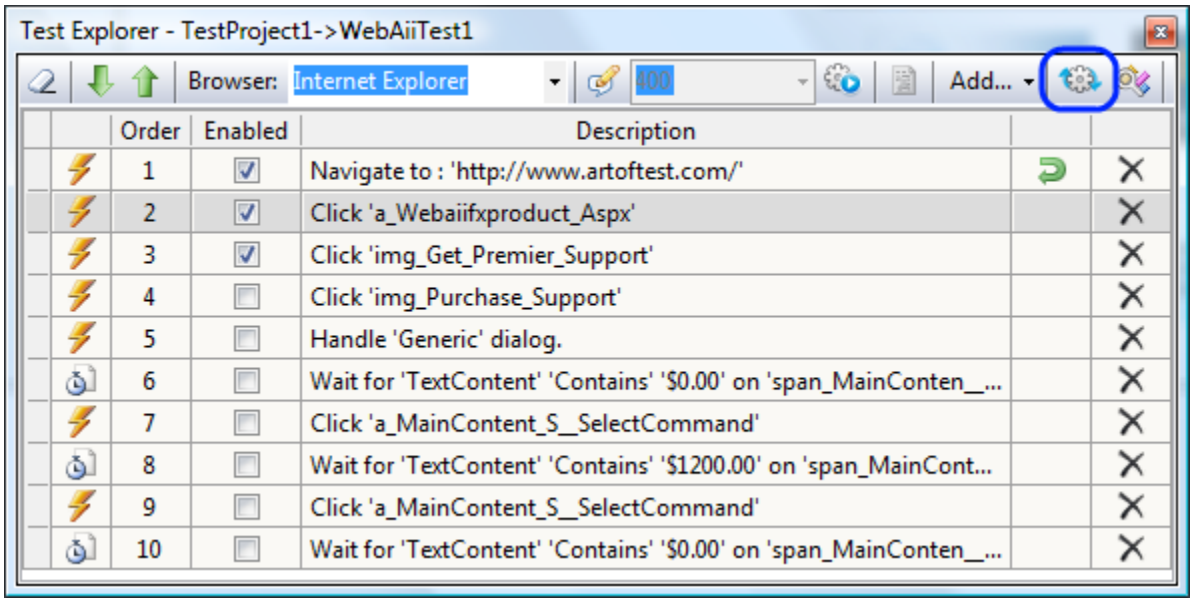
This flag can be used to inform the execution not to run this test as a stand-alone test and to only run it if it is part of another test. (i.e. used as a step in another test)

5.15 Re-capture test browser states

When you initially create your test, Design Canvas does not capture the browser state (i.e. the DOM tree) as you are recording your test. After you have crafted your test and verified it runs reliably and predictably, it's a good idea to capture (or re-capture) the browser state at each test step.

Capturing the browser state isn't required to run your tests. In fact Design Canvas doesn't even use the captured states for test execution. However you can use them to resolve test failures if/when they occur. By capturing the browser states you will then have the ability to compare what was the state when the test was created to what the state of the browser at the point of a test failure. This will assist you to determine what the root cause of a failure is.

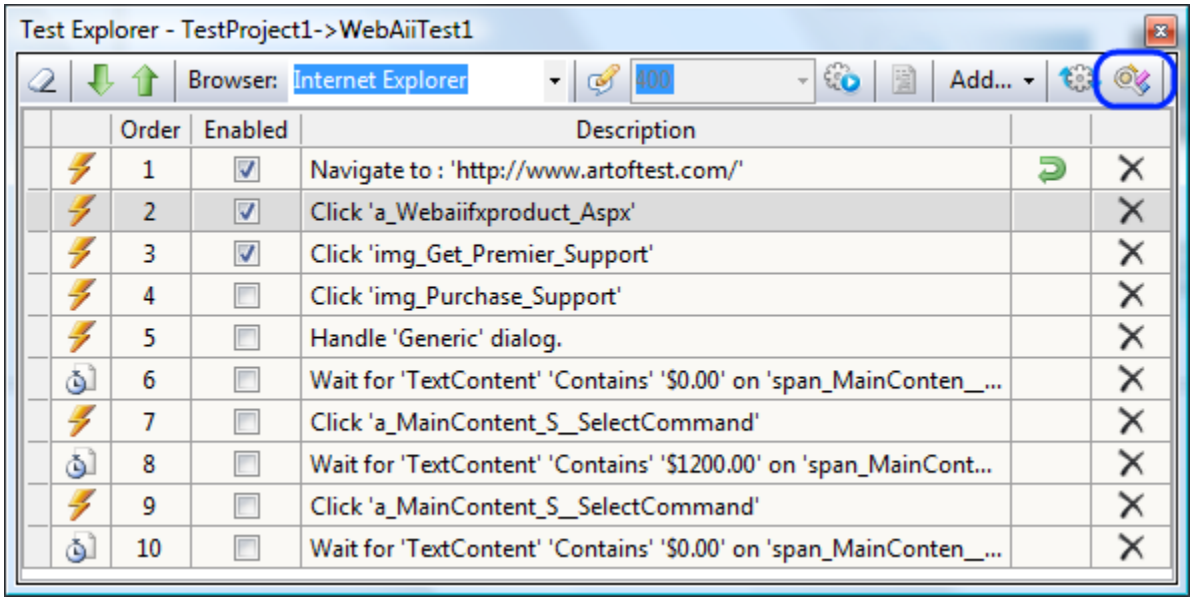
To re-capture the browser states, click the recapture button:



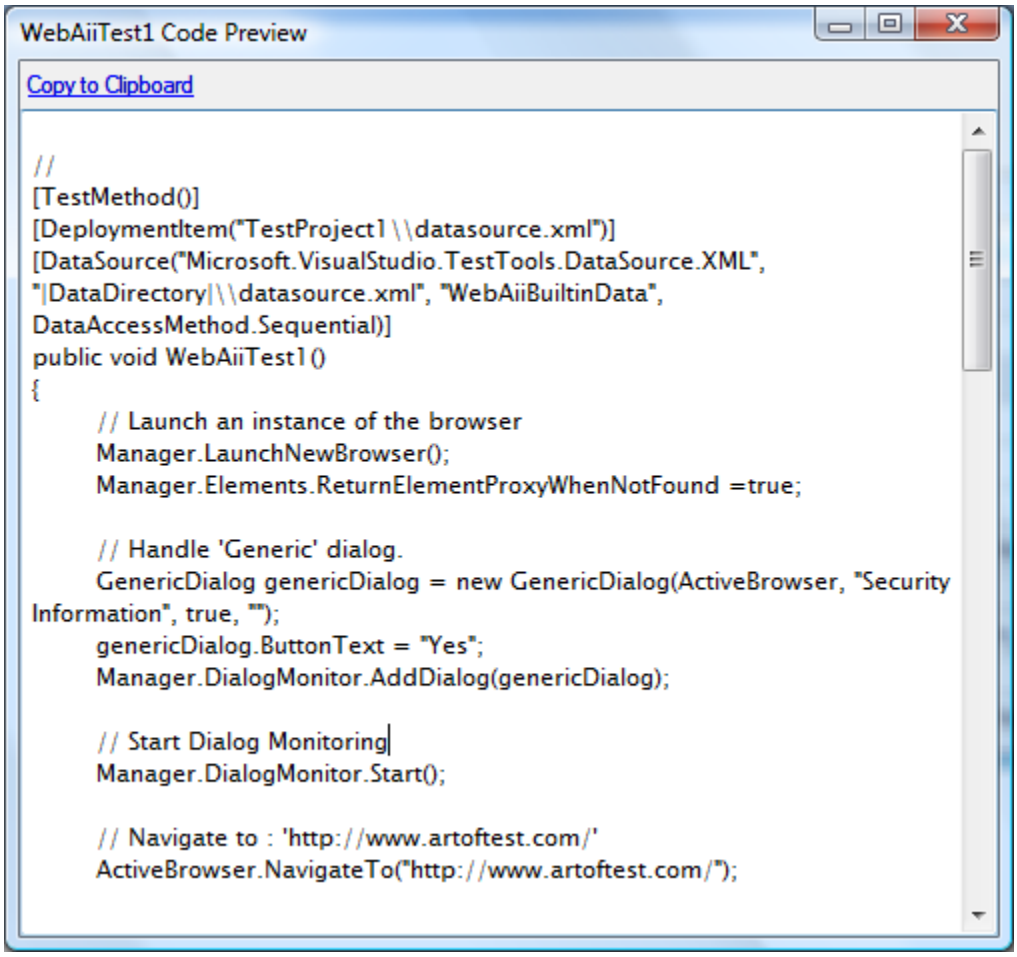
5.16 Preview Code for Test

Optionally you can preview the code that Design Canvas will generate in order to run your test. This is useful if you want to see how the test makes use of the WebAii library. This step is not required for your tests to run. Design Canvas will generate and execute the proper code automatically as needed.

To preview the code, click the Preview Code button:



Clicking this button will open a new window showing the code something like this:



6 SENTENCE BASED VERIFICATION

6.1 Verification Builder Overview

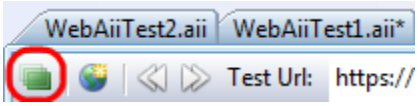
With Verification Builder you can create your own verification rules as if you were writing a sentence. To make crafting verification and synchronization as simple as possible, ArtOfTest developed an innovative Sentence Based™ user interface (UI) that guides you through crafting verifications and test synchronization with elements. It is similar to an adaptive wizard that changes depending on the target element. Using the Sentence Based UI you can create a wide range of verification types such as attributes, styles, tables, select dropdowns, element visibility, and more. The Sentence Based UI guides you through crafting the verification criteria by first loading the state of the target element into the context of the rule being crafted. It then offers you verification criteria one step at a time until the verification rule is complete.

Using Verification Builder you can create as many different element verification rules as needed for your test. Every element verification rule that you create in Verification Builder is added as a separate verification step in Test Explorer once complete.

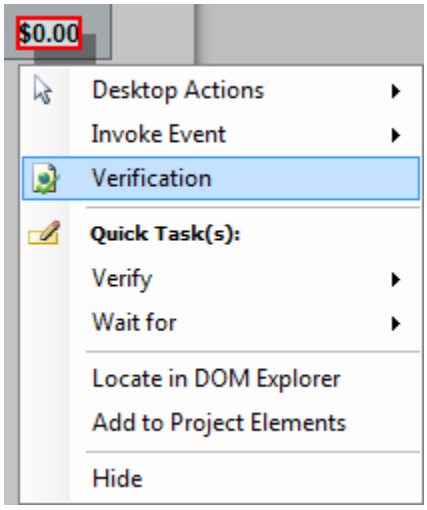
6.2 Opening Verification Builder

Follow these easy steps to open Verification Builder:

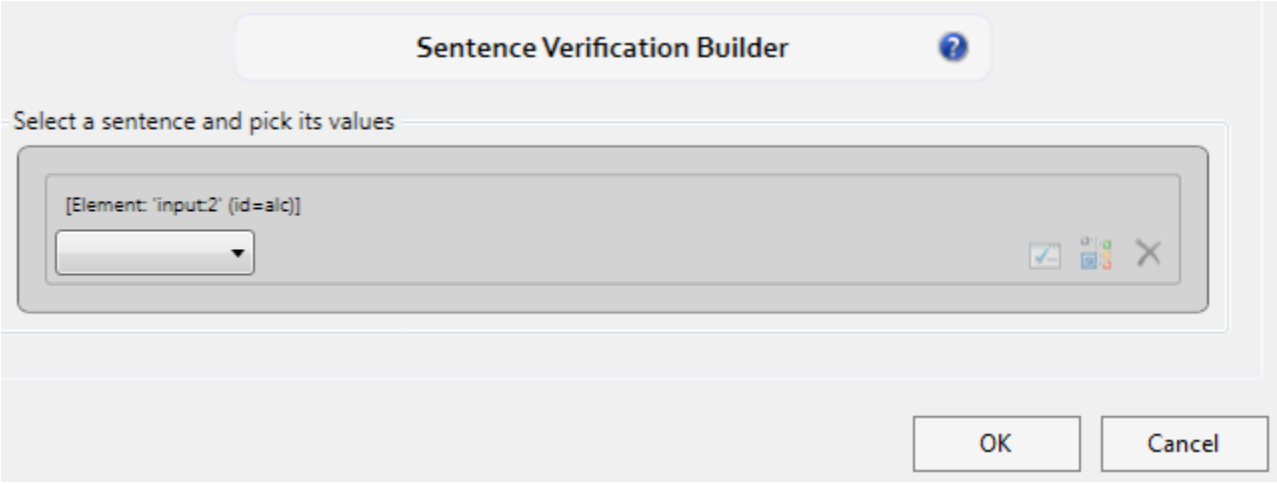
- Navigate to the web page containing the element that you want to create a verification rule for.
- In Design Canvas Recorder, enable the overlay highlighting by clicking the green overlay button on the top left-hand corner of the Recorder.



- Right click on the element and select Verification from the context menu.

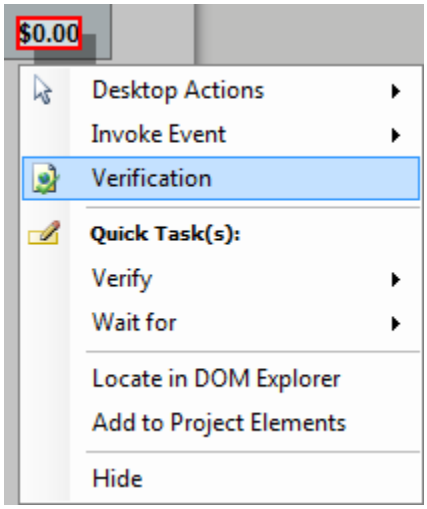


- Verification Builder opens with that element loaded into it.

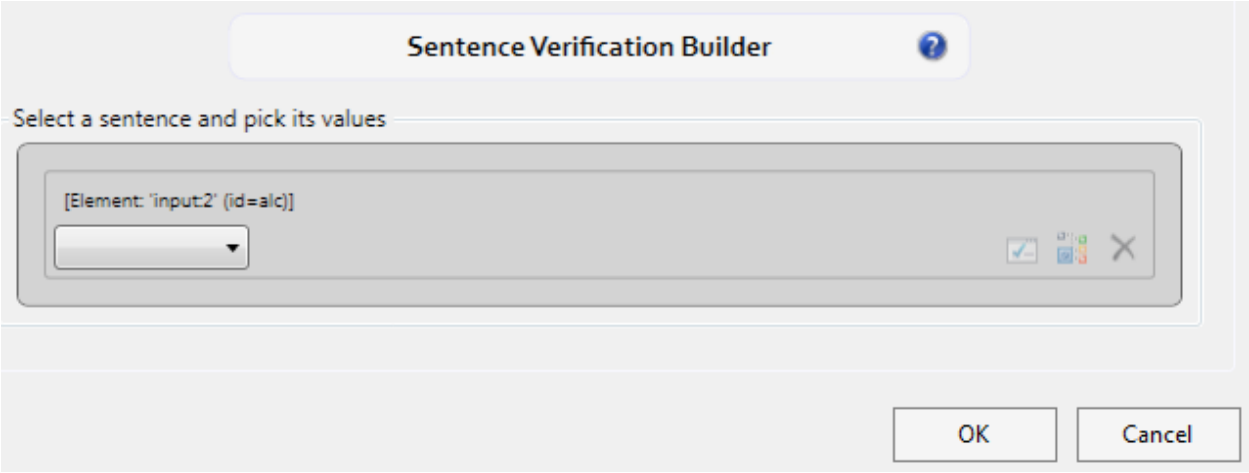


Another way to open Verification Builder is to use DOM Explorer. This is the method you must use when you are not able to click on the element in the browser window for example when it is hidden behind another element (such as a <table> element) or when the element is simply not clickable in the browser window.

- In the DOM Explorer window drill down to the element that you want to create a verification rule for. Sometimes a convenient shortcut is to right click on an element in the browser window that is close to the element you want (for example, a table cell element when you want to get to the owning table element) and select "Locate in DOM Explorer". From here you can more easily locate the element you want instead of having to drill down from the very top.
- Right click on the element you want and select Record Options. This will open the same context menu as opened using overlay highlighting.
- Select Verification from the context menu.



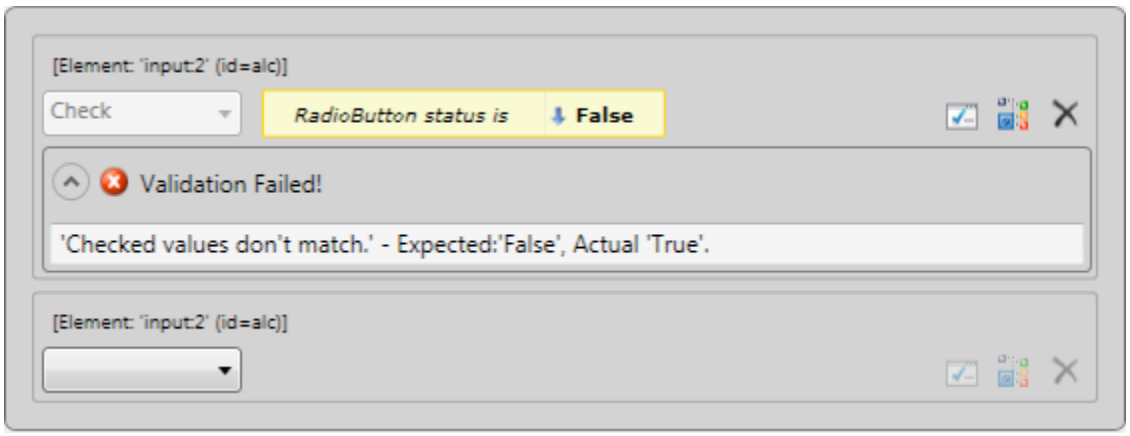
- Verification Builder opens with that element loaded in it.



Each verification rule that you create in Verification Builder has three icons you can use:

-  – Verifies that the rule is valid against the current page. If the rule passes then Design Canvas displays:

If the rule fails then Design Canvas displays:



-  – Highlights the element in the DOM tree as displayed in the DOM Explorer window. Useful if you need to refer to the full HTML of the element as you are creating your element verification rule.

-  – Deletes this element verification rule. ***Note****: There is no warning or undo so be certain this is what you want.* Of course you can always recreate the rule since the HTML and DOM are not changed by the delete.

Clicking **OK** closes Verification Builder and adds each element verification rule as a verification step in the Test Explorer window. You can optionally change any verification step in Test Explorer into a "Wait For" step as described in section 5.7.6.4.

Clicking **Cancel** closes Verification Builder and immediately discards any element verification rules you were working on.

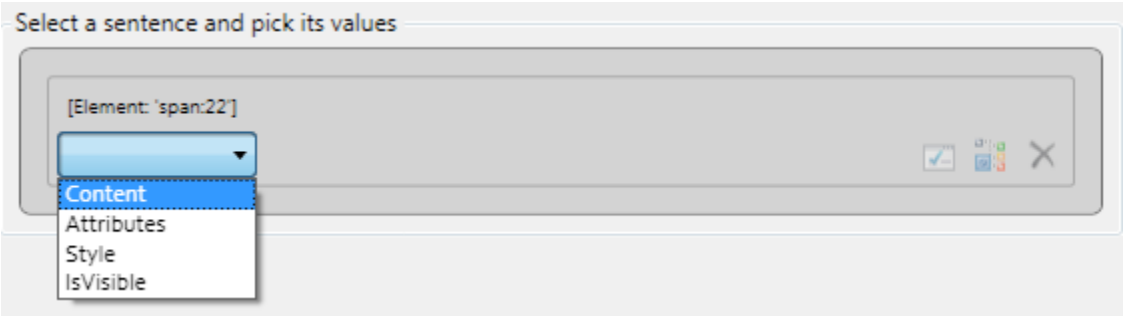
6.3 Verification Builder: Verifying Different Types of Elements

With sentence based verification you can verify the following element features:

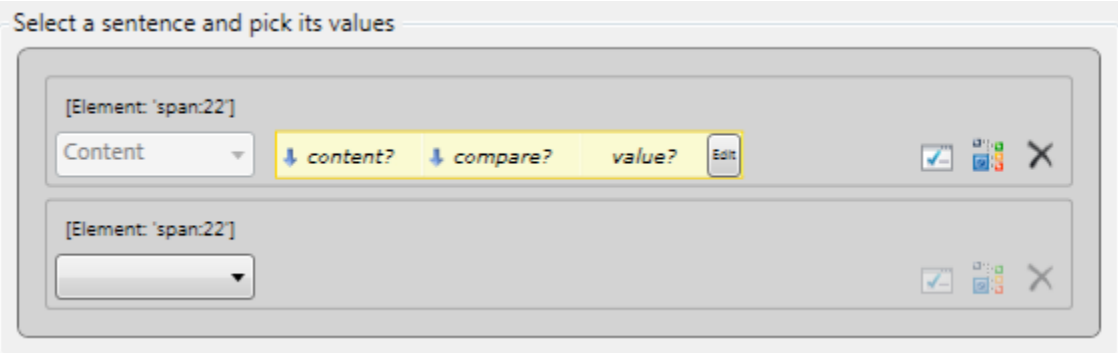
- Verifying element content
- Verifying element attributes
- Verifying element style
- Verifying element visibility
- Verifying select dropdown
- Verifying checkboxes and radio buttons
- Verifying tables

6.3.1 Verifying element content

All elements have the ability to verify element content, both HTML and text content. Start by selecting "Content" from the first dropdown:



You will be presented with three options that must be specified:



6.3.1.1 Content

Select what type of content to verify. This can be one of the following:

- InnerText – Compares the inner text of the tag. For example, “Data Display”. Be careful because InnerText is recursive. It includes not only the text of the current element but all the text for all children elements.
- InnerMarkup – The inner markup of a tag. For example, “ ”. The same care must be taken for this type of compare as for InnerText.
- OuterMarkup – The outer markup of a tag. For example, “ Some Data <input attr1="Button1" type="button" value="Click Me" onclick="clicked () ;"/>”.The same care must be taken for this type of compare as for InnerText.
- TextContent – Text content of the tag only without all its children. For example, “Some Data”. This type of compare is less risky than the above methods because it is non-recursive. The above methods are all recursive, that is, include their child elements.
- StartTagContent – The raw start tag content.

6.3.1.2 Compare

Select which compare type to perform. You can select from:

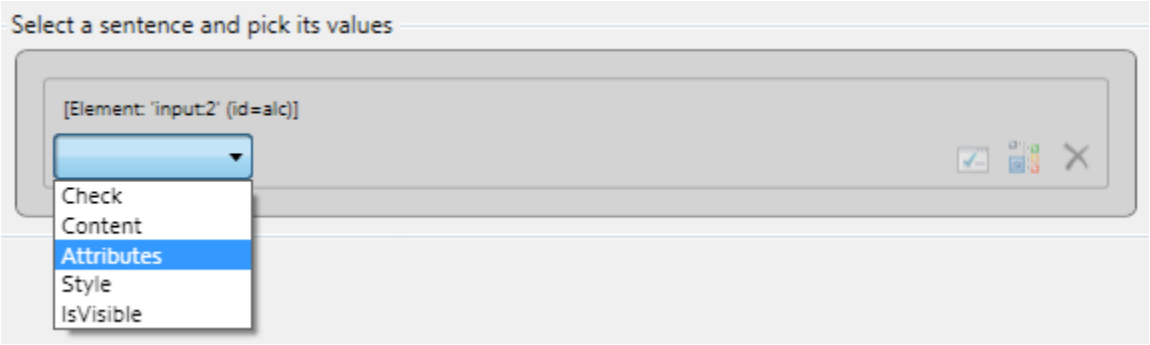
- Exact – Verify the attribute value matches exactly with the expected value.
- Contains – Verify the attribute value contains the expected value. Used mostly for those attributes having string values rather than numeric values.
- NotContain – Verify the attribute value does *not* contain the expected value. Used mostly for those attributes having string values rather than numeric values.
- StartsWith – Verify the attribute value starts with the expected value. Used mostly for those attributes having string values rather than numeric values.
- EndsWith – Verify the attribute value starts with the expected value. Used mostly for those attributes having string values rather than numeric values.
- RegEx – Verify the attribute value matches a regular expression. WebAii takes advantage of the .NET Framework Regular Expressions as detailed here: http://msdn.microsoft.com/en-us/library/hs600312.aspx. The string you enter as the expected value is used as the expression in the match comparison.

6.3.1.3 Value

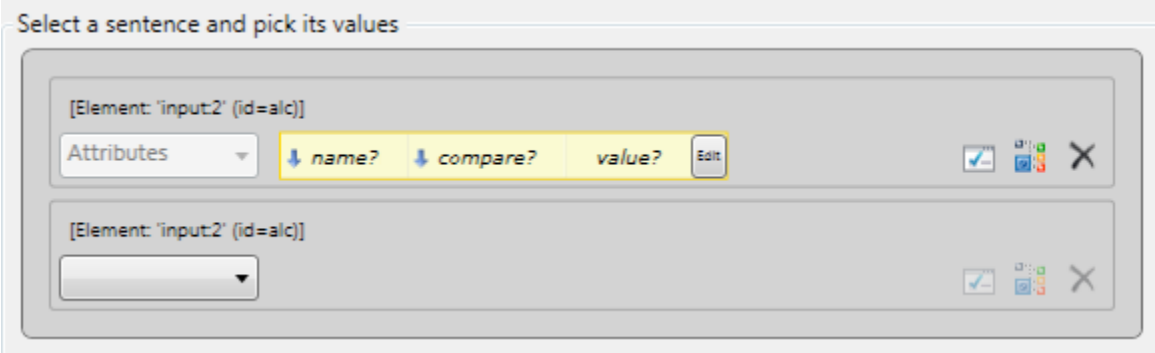
This shows the expected value to use in the comparison. The value string is prefilled with the current setting of the attribute selected in the Name dropdown. How this value is used by the verification depends on the comparison type selected as described in the prior paragraphs.

6.3.2 Verifying element attributes

Using Verifications Builder you can create a verification rule that evaluates any attribute set on the element. Start by selecting 'Attributes' from the first dropdown:



You will be presented with three options that you must specify:



6.3.2.1 Name

Name –Select which attribute you want to create a verification rule for. This dropdown will be filled only with the attributes currently set on the element. If necessary the attribute name can be changed later via the properties window after you have saved the verification rule.

6.3.2.2 Compare

Compare – Select which compare type to be performed. You can select from:

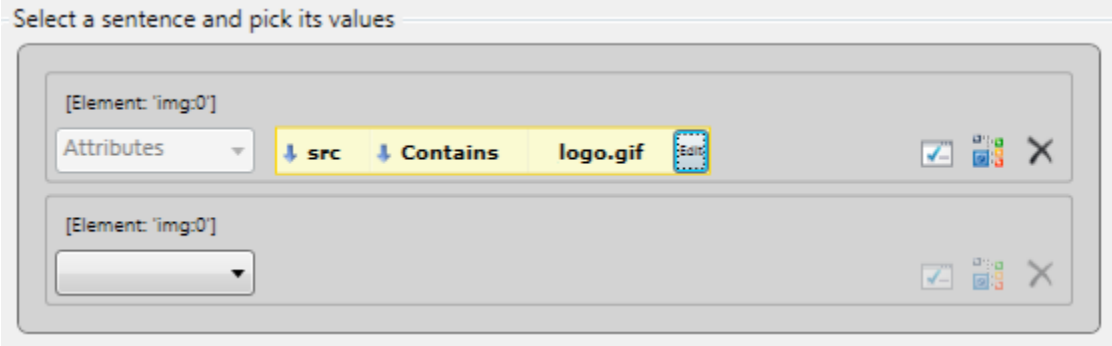
- Exact – Verify the attribute value matches exactly with the expected value.
- Contains – Verify the attribute value contains the expected value. Used mostly for those attributes having string values rather than numeric values.
- NotContain – Verify the attribute value does *not* contain the expected value. Used mostly for those attributes having string values rather than numeric values.

- StartsWith – Verify the attribute value starts with the expected value. Used mostly for those attributes having string values rather than numeric values.
- EndsWith – Verify the attribute value starts with the expected value. Used mostly for those attributes having string values rather than numeric values.
- RegEx – Verify the attribute value matches a regular expression. WebAii takes advantage of the .NET Framework Regular Expressions as detailed in: http://msdn.microsoft.com/en-us/library/hs600312.aspx. The string you enter as the expected value is used as the expression to use in the match comparison.

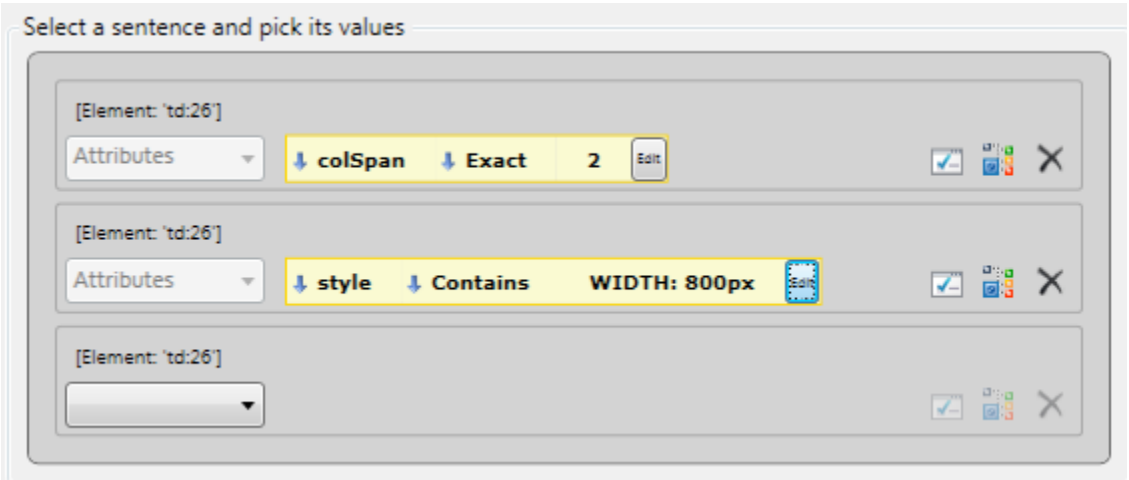
6.3.2.3 Value

Value – This is the expected value to use in the comparison. The value string is prefilled with the current setting of the attribute selected in the Name dropdown. How this value is used by the verification depends on the comparison type selected as described above. Here are a couple of examples of completed element verification rules:

- This verifies that the src attribute for the <img> element contains the string "logo.gif":



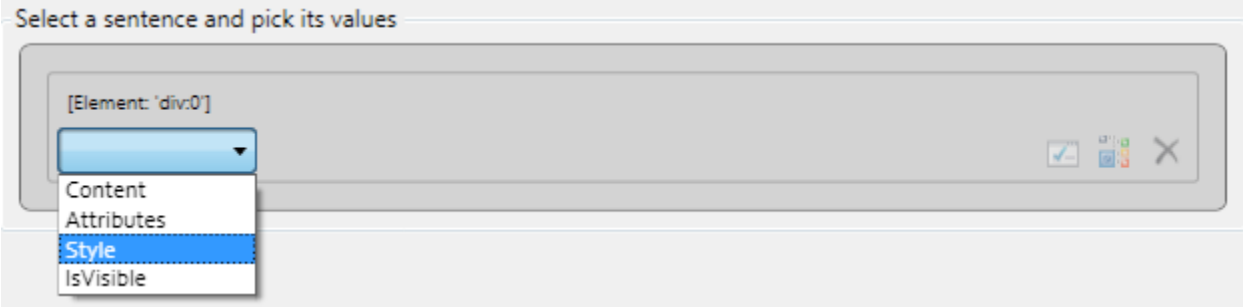
- This verifies that the colSpan attribute equals 2 and that the style attribute contains the string "WIDTH: 800px". Both element verification rules will examine the same <td> element. When saved the two verification steps are added to Test Explorer, one for each element verification rule:



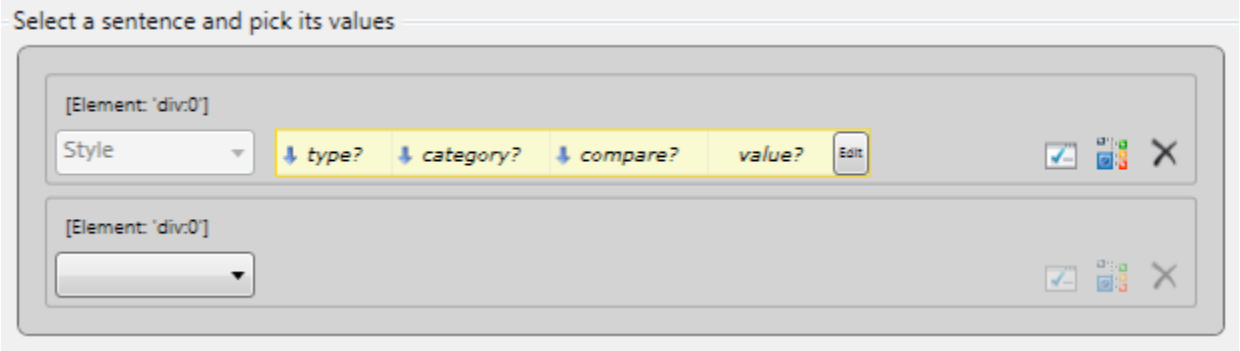
6.3.3 Verifying element style

Using Verifications Builder you can create a verification rule that evaluates almost any style applied to the element as defined by the CSS2 standard: http://www.w3schools.com/css/css_reference.asp. Any element may have a style applied to it hence this element rule verification applies to all elements.

Start by selecting Style from the first dropdown:



You will be presented with four options that you must specify:



6.3.3.1 Type

Type – This can be either:

- Computed – WebAii will follow the CSS chain to get the currently active style setting.
- Inline – WebAii will only look at the style applied directly to the element, if present.

6.3.3.2 Category

Category – Defines which style category you want to select from. It can be one of:

- Font – When you select Font as the category an additional "Font" dropdown appears. In this additional dropdown you can select from:
- Family
- Style
- Variant
- Weight
- Size
- Stretch

- ColorAndBackground – When you select ColorAndBackground as the category an additional "color/background" dropdown appears. In this additional dropdown you can select from:
- Color – Note: The color styles are abstracted by translating them to the #rrggbb format. This relieves the tester of having to worry about the different formats that can be used to specify color.
- BackgroundColor – **Note**: The color styles are abstracted by translating them to the #rrggbb format. This relieves you of having to worry about the different formats that can be used to specify color.
- BackgroundImage
- BackgroundRepeat
- BackgroundAttachment
- BackgroundPosition
- Background
- Text – When you select Text as the category an additional "Text" dropdown appears. In this additional dropdown you can select from:
- WordSpacing
- LetterSpacing
- WhiteSpace
- TextTransform
- TextAlign
- TextIndent
- TextDecoration
- TextShadow
- Display – When you select Display as the category an additional "Display" dropdown appears. In this additional dropdown you can select from:
- Width
- Height
- LineHeight
- VerticalAlign
- MinWidth
- MaxWidth
- MinHeight
- MaxHeight
- Display
- Position
- Top
- Left
- Bottom
- Right
- Float

- zIndex
- Box – When you select Box as the category an additional “Box” dropdown appears. In this additional dropdown you can select from:
- MarginTop
- MarginRight
- MarginBottom
- MarginLeft
- Margin
- PaddingTop
- PaddingRight
- PaddingBottom
- PaddingLeft
- Padding
- BorderTopWidth
- BorderRightWidth
- BorderBottomWidth
- BorderLeftWidth
- BorderWidth
- BorderTopColor
- BorderRightColor
- BorderBottomColor
- BorderLeftColor
- BorderColor
- BorderTopStyle
- BorderRightStyle
- BorderBottomStyle
- BorderLeftStyle
- BorderStyle
- BorderTop
- BorderRight
- BorderBottom
- BorderLeft
- Border
- List – When you select List as the category an additional “List” dropdown appears. In this additional dropdown you can select from:
- ListStyleType
- ListStyleImage
- ListStylePosition
- ListStyle

6.3.3.3 Compare

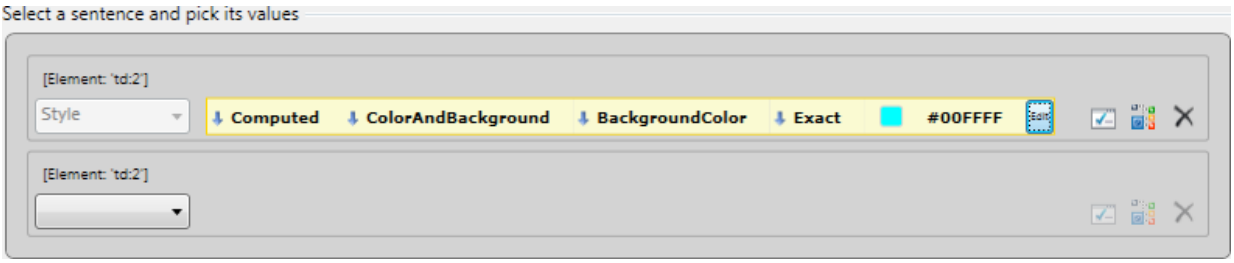
Compare – Select which compare type to be performed. You can select from:

- Exact – Verify the style value matches exactly with the expected value.
- Contains – Verify the style value contains the expected value. Used mostly for those styles having string values rather than numeric values.
- NotContain – Verify the style value does *not* contain the expected value. Used mostly for those styles having string values rather than numeric values.
- StartsWith – Verify the style value starts with the expected value. Used mostly for those styles having string values rather than numeric values.
- EndsWith – Verify the style value starts with the expected value. Used mostly for those styles having string values rather than numeric values.
- RegEx – Verify the style value matches a regular expression. WebAii takes advantage of the .NET Framework Regular Expressions as detailed here: http://msdn.microsoft.com/en-us/library/hs600312.aspx. The string you enter as the expected value is used as the expression to use in the match comparison.

6.3.3.4 Value

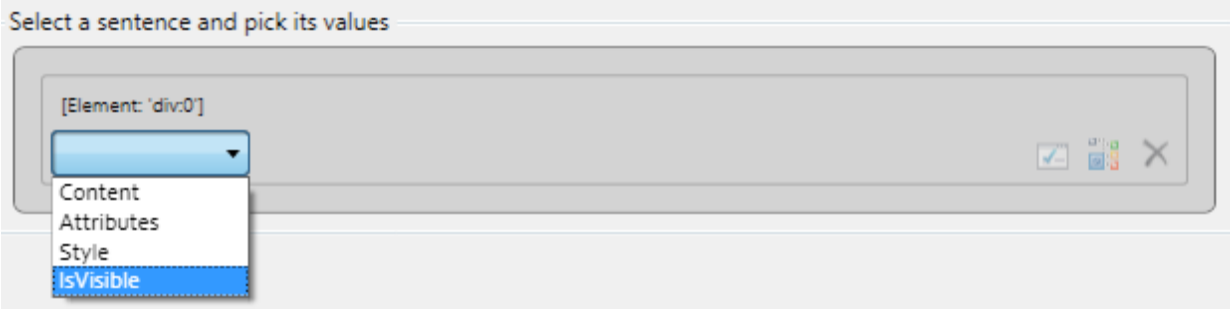
Value – This is the expected value to use in the comparison. The value string is prefilled with the current setting of the style selected by the category dropdown and that categories style dropdown. How this value is used by the verification depends on the comparison type selected as described above.

In the following example the computed style of the background color is verified to equal the hex value #00FFFF. Notice how the color was translated from the HTML
<td class="s" width="25%" bgcolor="aqua"> to its hex equivalent:



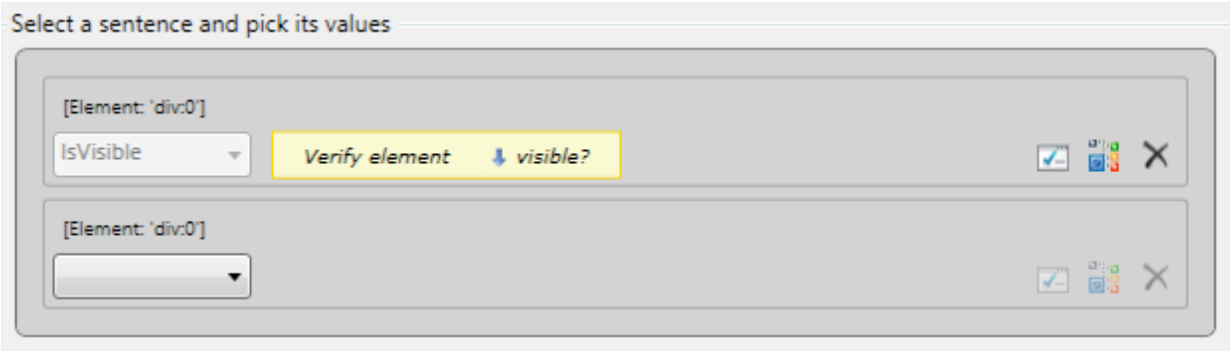
6.3.4 Verifying element visibility

Using Verifications Builder you can create a verification rule that evaluates the visibility state of the element. WebAii tests whether or not an element can be seen on the web page by analyzing the "visibility" (http://www.w3schools.com/CSS/pr_class_visibility.asp) and the "display" (http://www.w3schools.com/CSS/pr_class_display.asp) attributes set for that element. WebAii follows the CSS chain as needed to determine the current visibility state of the element.
Start by selecting Attributes from the first dropdown:



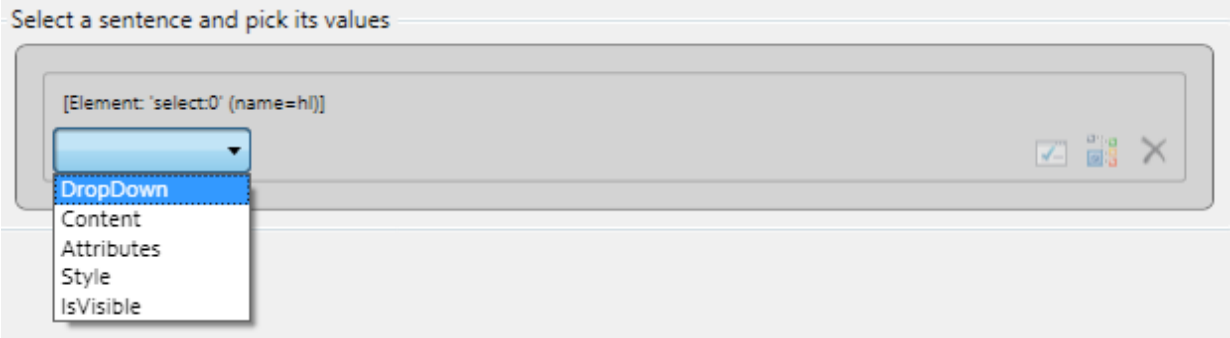
6.3.4.1 Visible

Now all you need to specify is "Visible" or "NotVisible" for the "visible" option displayed:

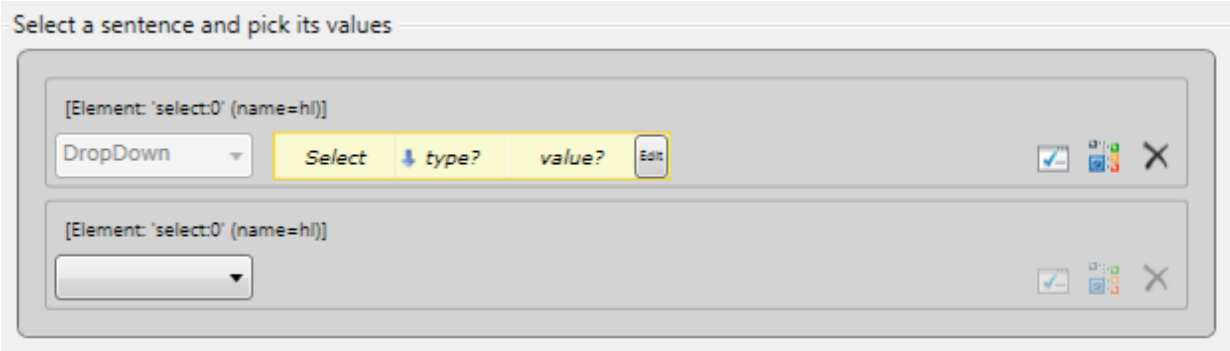


6.3.5 Verifying a select dropdown element

Using Verifications Builder you can create an element verification rule to validate what is currently selected for a <select> element. Start by selecting 'DropDown' from the first dropdown.



At first you will be presented with two options that you must specify:

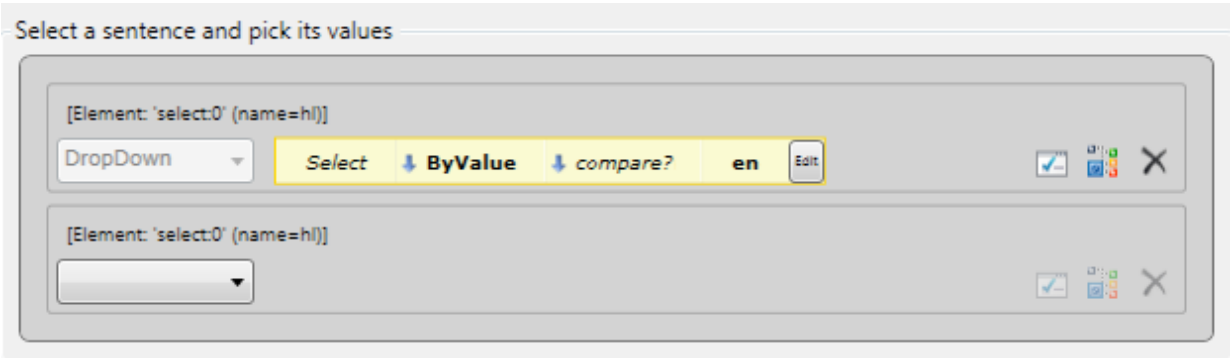


6.3.5.1 Type

Type – This is the type of comparison to perform. You can select from:

- ByIndex – Verifies which dropdown index is currently selected. Note: the index value is zero based. Thus if the first item in the dropdown list is selected the value is 0.
- ByValue – Verifies which selection value is currently selected. Remember the list of possible values come from the <option> elements that are children of the <select> element, not what's displayed in the browser window. For example <option value=4>.
- ByText – Verifies the text that is displayed for the current selection.

After selecting Type, one of two different Compare dropdowns will be displayed for you to select from:



6.3.5.2 Compare

When you select ByIndex then you can select from the following for the compare type to be performed:

- Equals – Verify the count matches exactly with the expected value.
- LessThan – Verify the count is less than the expected value.
- GreaterThan – Verify the count is greater than the expected value.
- LessThanOrEqual – Verify the count is less than or equal to the expected value.
- GreaterThanOrEqual – Verify the count is greater than or equal to the expected value.
- NotEqual – Verify the count is not equal to the expected value.

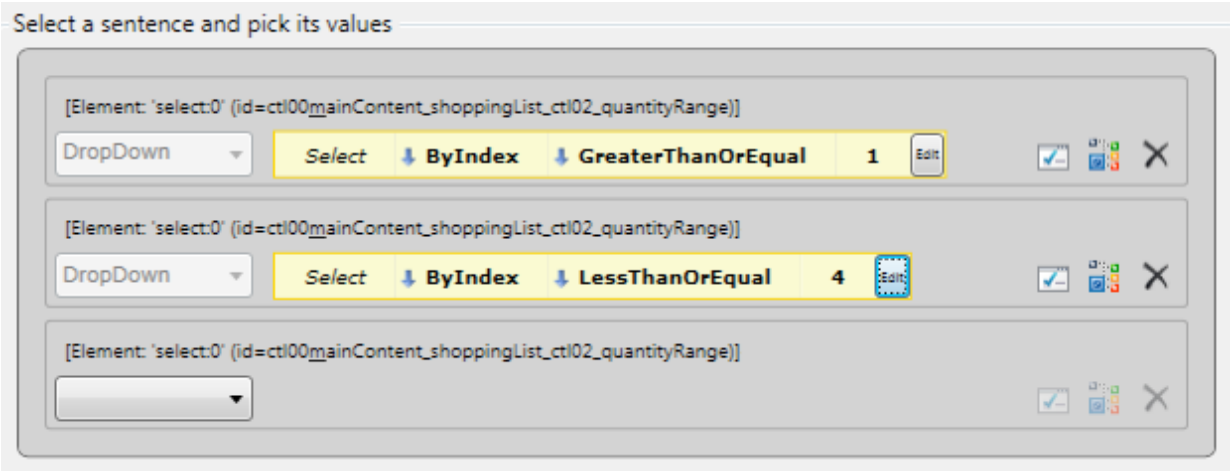
When you select ByValue or ByText you then can select from the following for the compare type to be performed:

- Exact – Verify the selection matches exactly with the expected value.
- Contains – Verify the selection contains the expected value. Used mostly for ByText verifications.
- NotContain – Verify the selection does *not* contain the expected value. Used mostly for ByText verifications.
- StartsWith – Verify the selection starts with the expected value. Used mostly for ByText verifications.
- EndsWith – Verify the selection starts with the expected value. Used mostly for ByText verifications.
- RegEx – Verify the selection matches a regular expression. WebAii takes advantage of the .NET Framework Regular Expressions as detailed at: http://msdn.microsoft.com/en-us/library/hs600312.aspx. The string you enter as the expected value is used as the expression to use in the match comparison.

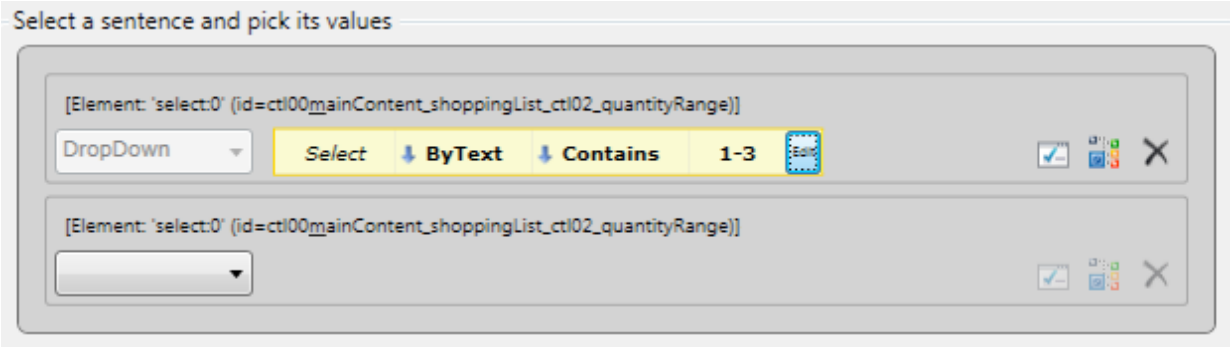
6.3.5.3 Value

Value – This is the expected value to use in the comparison. The value string is filled with the current selection setting as chosen by the Type dropdown, that is, ByIndex will fill the value with the current selected index value, ByValue will fill the value with the current value selected, ByText will fill the value with the string currently displayed for the selection. How this value is used by the verification depends on the comparison type selected as described above. Here are two examples of completed element verification rules:

This example verifies that the selected index is between 1 and 4:

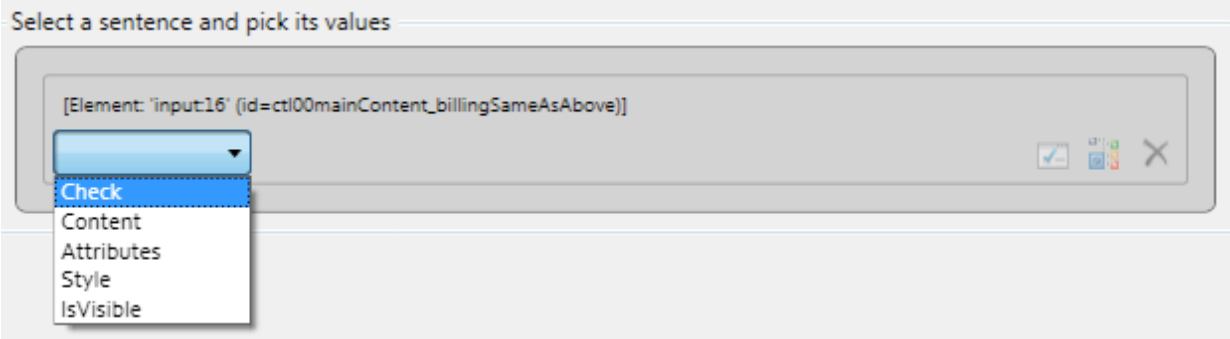


This example verifies that the current selection displays the text “1-3”:

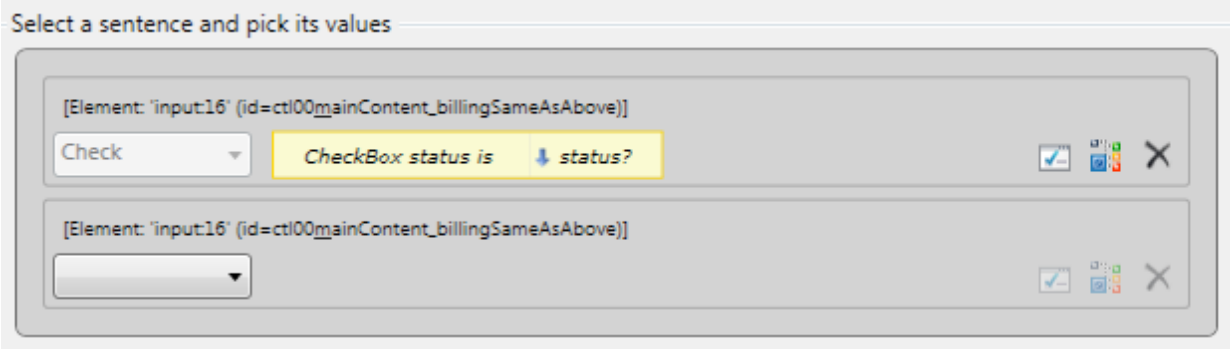


6.3.6 Verifying checkboxes and radio buttons

Using Verification Builder you can create an element verification rule to test whether or not a checkbox or radio button is checked or selected. Start by selecting Check from the first dropdown.



You will be presented with the following selections:



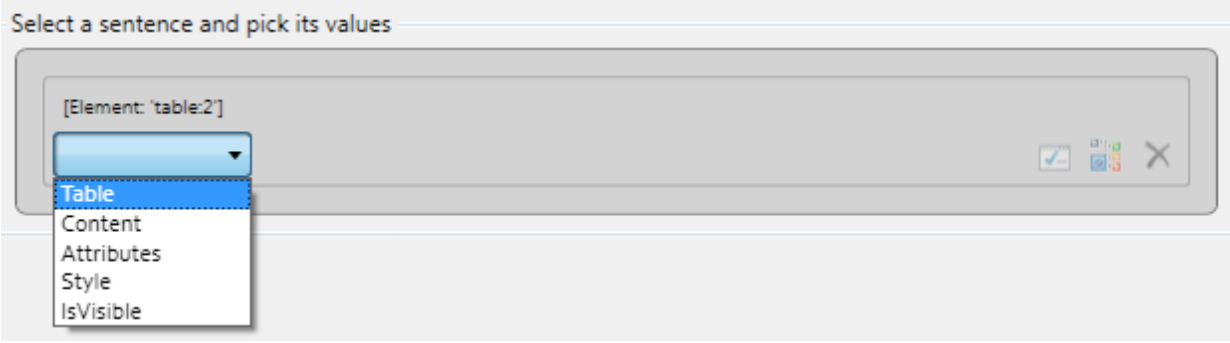
6.3.6.1 Status

Status – Choose either:

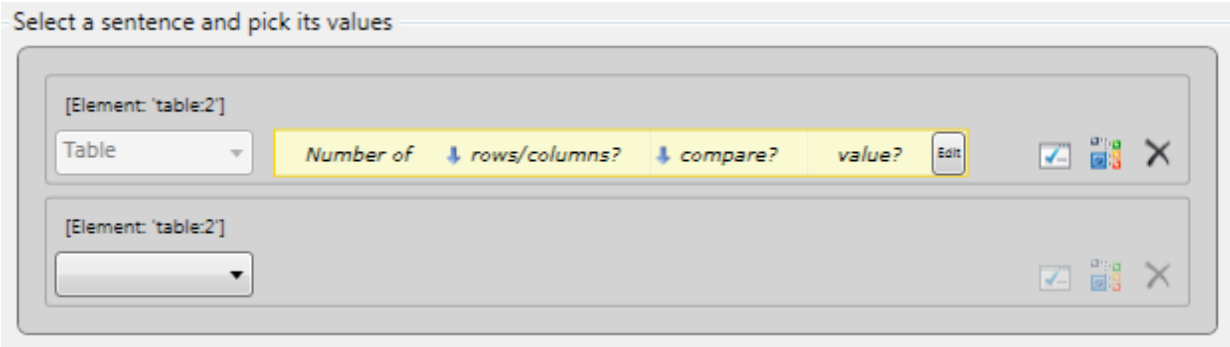
- True – The checkbox is checked. The radio button is selected.
- False – The checkbox is unchecked. The radio button is not selected.

6.3.7 Verifying tables

Using Verification Builder you can create an element verification rule to test the number of rows or the number of columns in a table. Start by selecting Table from the first dropdown.



You will be presented with three options that you must specify:



6.3.7.1 Rows/columns

Rows/Columns – Select either:

- RowsCount – Verify the number of rows in the table.
- ColumnsCount – Verify the number of columns in the table.

6.3.7.2 Compare

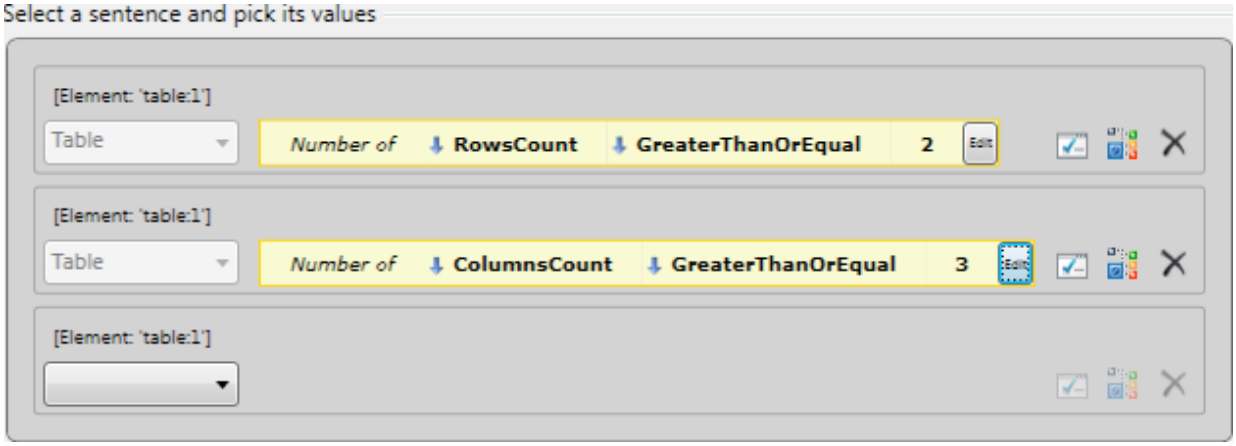
Compare – Select which compare type to be performed. You can select from:

- Equals – Verify that the count matches exactly with the expected value.
- LessThan – Verify that the count is less than the expected value.
- GreaterThan – Verify that the count is greater than the expected value.
- LessThanOrEqual – Verify that the count is less than or equal to the expected value.
- GreaterThanOrEqual – Verify that the count is greater than or equal to the expected value.
- NotEqual – Verify that the count is not equal to the expected value.

6.3.7.3 Value

Value – This is the expected value to use in the comparison. The value is filled with the current count of either number of rows or the number of columns depending on the “rows/columns” selection. How this value is used by the verification depends on the comparison type selected as

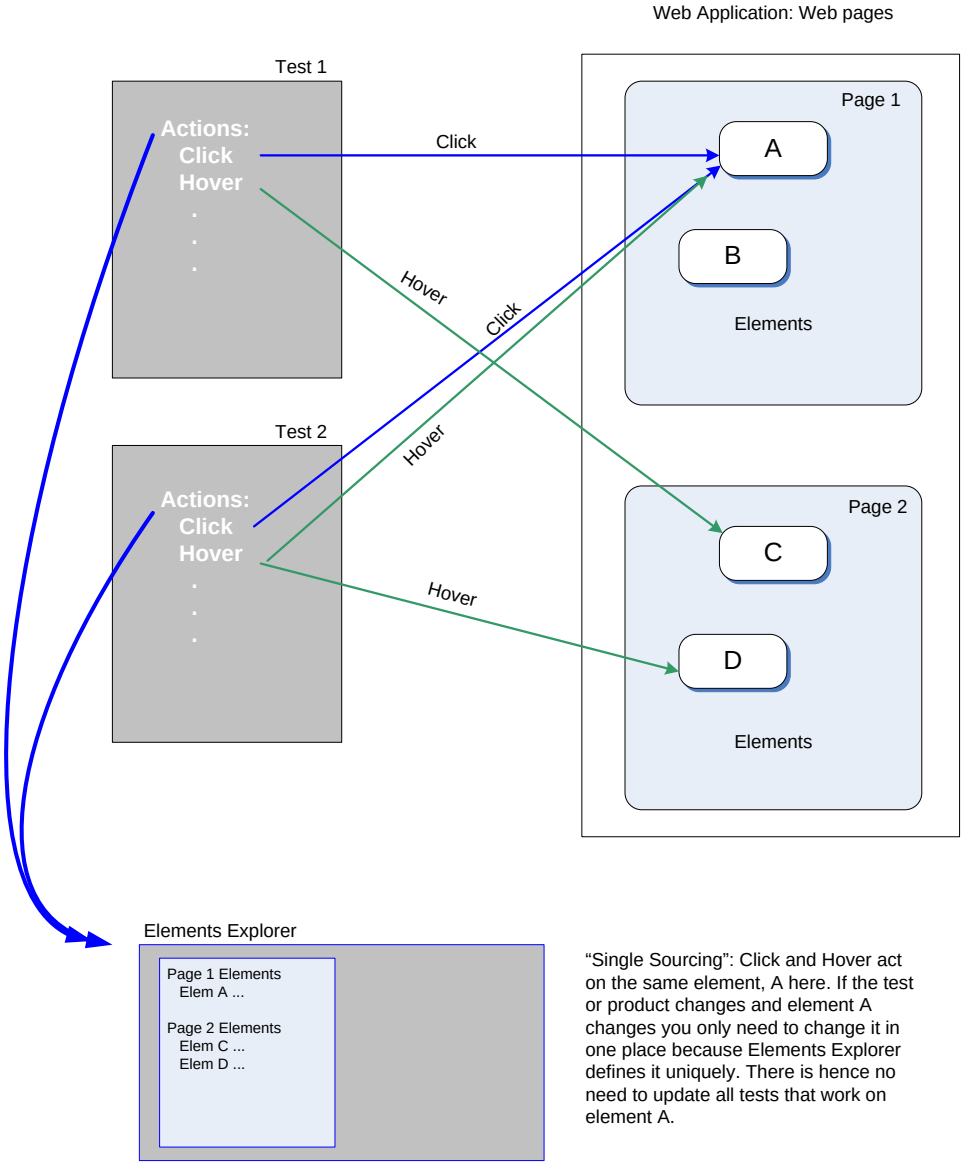
described above. The following example shows that it verifies that the number of rows is at least 2 and the number of columns is at least 3.



7 WEB ELEMENT ABSTRACTION

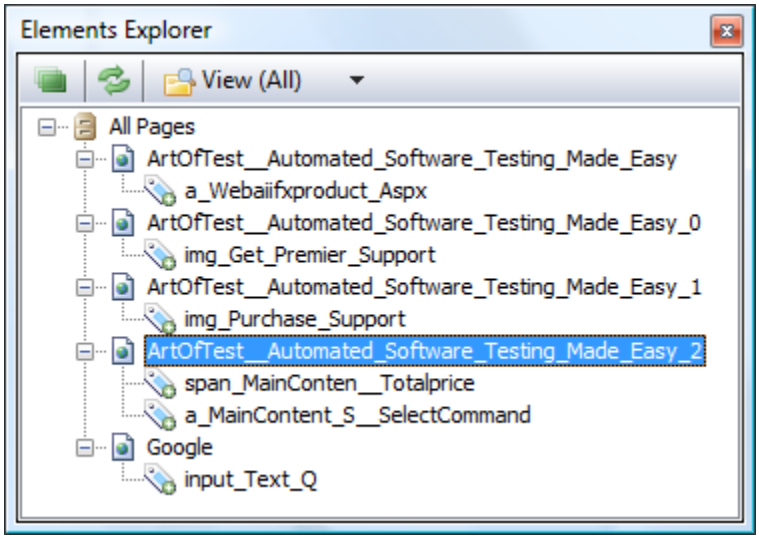
7.1 Introducing Web Element Abstraction

The following diagram illustrates part of the web element abstraction concept. Because elements are uniquely defined in Elements Explorer regardless of how the web page or web page tests change or how often and from where each web page element is acted on in the test, you only need to modify that element in one place, Elements Explorer.



Automation Design Canvas: Web Element Abstraction

7.2 Elements Explorer Window



All web page elements targeted for automation in your tests are abstracted out and filed under a specific page/frame in the Elements Explorer window. If there are multiple actions that use the same element, the element can be referenced from Elements Explorer instead of being duplicated in the test. This allows you to maintain just one unique element and update it once here whenever there is a required change instead of having to update multiple duplicate elements. This is huge time saver and can greatly reduce the cost of test maintenance.

Another common significant time consuming task that testers complain about with web automation is determining how to uniquely locate a specific element on a page. For dynamic pages, this task becomes even more complex and time consuming. Automation Design Canvas makes this task almost trivial. It is empowered with an algorithm to automatically determine the best find parameters to use to uniquely locate a specific element on a page. This algorithm is configurable using Design Canvas settings as described in Appendix B. In addition you can create your own schemes and find logic.

7.2.1 Elements Explorer: Highlighting elements on the active page

Clicking on this icon allows you to visually highlight elements in the Recording Surface. Likewise, when this is enabled, clicking on the various respective elements displayed in Elements Explorer also highlights them in the Recording Surface. Of course it will only highlight elements for the currently loaded web page. Clicking elements in Elements Explorer for other web pages/frames will not load those pages or highlight anything in the Recording Surface.

7.2.2 Elements Explorer: Refresh

Clicking on this icon refreshes the display of the elements in Elements Explorer. You seldom should have to do this because Design Canvas normally automatically refreshes the window properly.

7.2.3 Elements Explorer: Views

Elements Explorer has two viewing modes, View All and View Current Page Only. Choose the view mode by selecting it from the View dropdown menu in Elements Explorer:

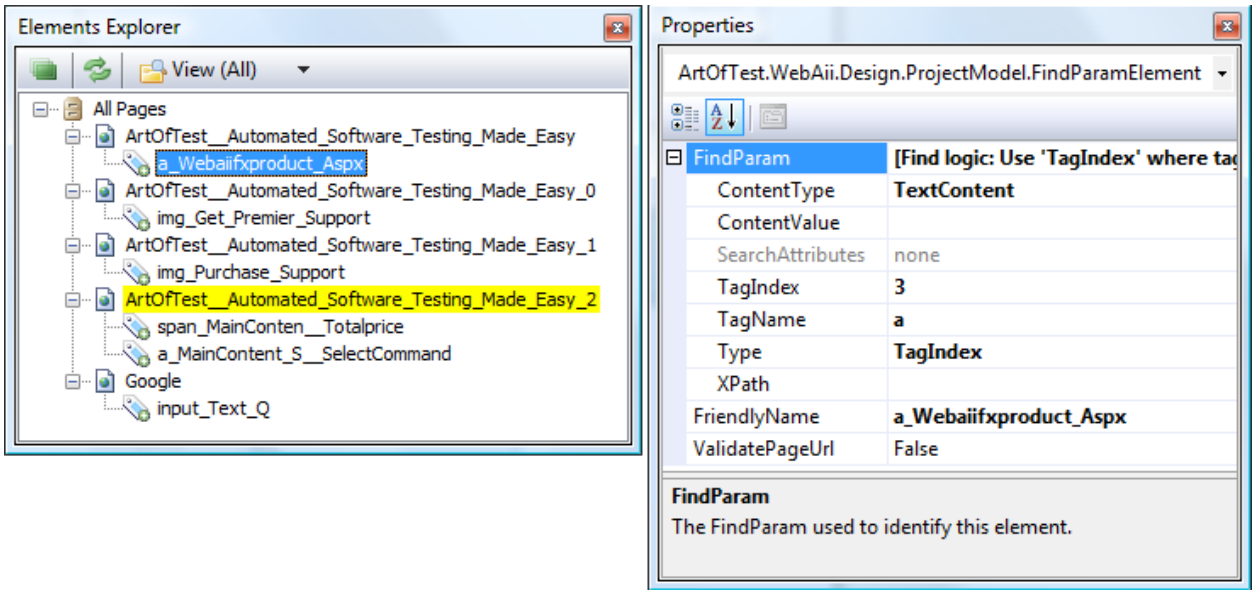
7.2.3.1 View (All)

The 'View (All)' mode displays all elements that have been added to Elements Explorer from all web pages and frames in a hierarchical format.

7.2.3.2 View current page only

'View (Current)' page only mode displays in Elements Explorer only the elements contained in the currently loaded web page. This is useful when you have added many elements from many web pages to Elements Explorer. By limiting the display to just the currently loaded web page it becomes much easier to find a particular element in the hierarchy.

7.2.4 Elements Explorer: Element properties



All elements in Elements Explorer have a common set of properties that can be viewed and set in the properties window. To see the properties of an element first open the properties window, then click an element in Elements Explorer. The properties for that element will be displayed

allowing you to modify them as needed. Each property is described in the following sections.
***Note**: If the properties window remains blank when you click an element, click on a test step then click on the element again. Occasionally VS does not recognize the focus change and hence may not properly display the properties for the element having the current focus.*

7.2.4.1 Element properties: FriendlyName

FriendlyName is the name displayed in Elements Explorer for that element. When the element is first added to Elements Explorer it is given a default unique name based on the properties of the FindParam used to locate the element. After the element is added you can optionally change the name to something more meaningful, shorter, or easier to remember. This is not required however.

7.2.4.2 Element properties: ValidatePageURL

When this property is set to 'true' Design Canvas will first validate this elements parent page URL before trying to find it. This applies only when the steps 'WaitOnElements' is set or an 'Exists' step is used.

7.2.4.3 Element properties: FindParam properties

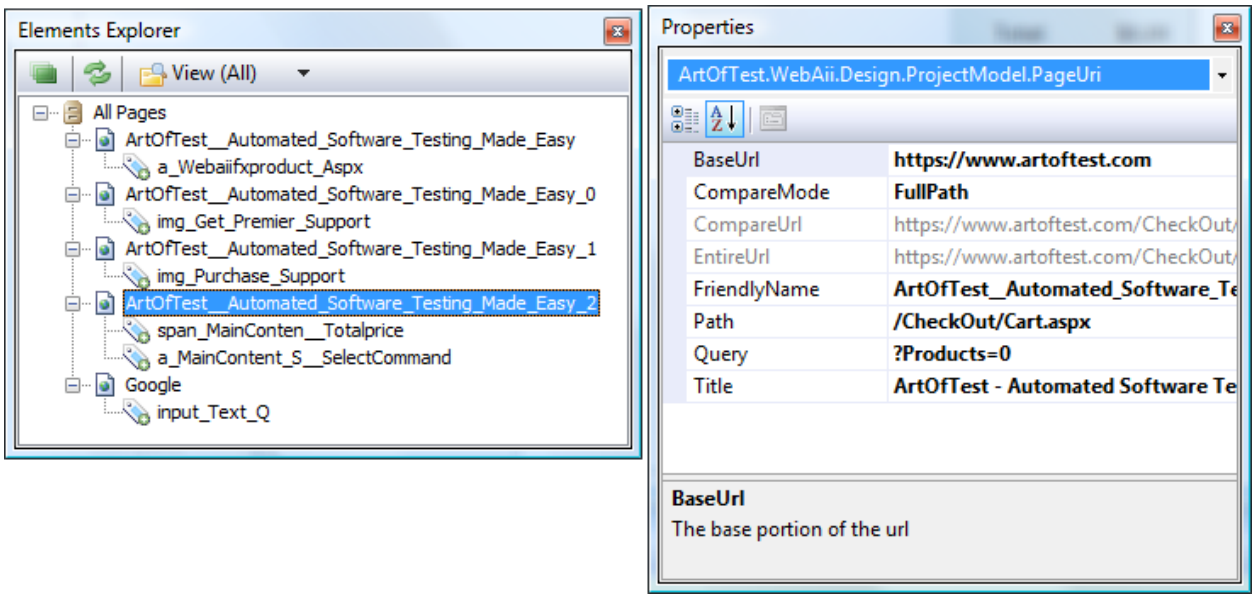
The FindParam set of properties controls how WebAii finds the element on the web page. They work together to define characteristics of the element so WebAii can correctly locate it from all other elements on the web page.

FindParam Property	Description
Type	The PageParam property Type specifies the type of search to perform when searching for the element on the web page. It can be one of the following: • TagIndex – Locates an element according to its occurrence (index) within the DOM. For example the 14th <a> element, or the first element. It is important to note that WebAii uses zero-based indexing. Thus the first element has an index 0. • AttributesOnly – Locates elements by matching its tag along with one or more of the elements attributes. For example an <a> element having a particular href attribute, or an <input> with a specific id attribute. • XPath – XPath will find the element using the XML XPath specified. See http://www.w3schools.com/XPath/default.asp for a description of XPath and how to use it. • Content – Locates elements by searching for a matching string.
SearchAttributes	The SearchAttributes property specifies the attributes and values to search for when WebAii is performing searching for elements by their attributes. It can only be set via the Element editor as described in section 7.3. It stores a list of attributes and values that WebAii will look for when looking for that element on the web page. It displays the attributes and values in the form

FindParam Property	Description
	"attribute1=value1; attribute2=value2…"
ContentType	When WebAii is doing a search for a specific content string, this parameter specifies which type of content it is looking for. It can have one of the following values: • InnerText – Compares the inner text of the tag. For example, "Data Display". Be careful because InnerText is recursive. It includes not only the text of the current element but all the text for all child elements. • InnerMarkup – The inner markup of a tag. For example," ". The same care must be taken for this type of compare as described for InnerText. • OuterMarkup – The outer markup of a tag. For example, " Some Data <input attr1="Button1" type="button" value="Click Me" onclick="clicked () ;"/>".The same care must be taken for this type of compare as described for InnerText. • TextContent – Text content of the tag only without all its children. For example, "Some Data". This type of compare is less risky than the above methods because it is non-recursive. The above methods are all recursive, that is, they include their child elements. • StartTagContent – The raw start tag content. **Note**: There is a difference between InnerText and TextContent that is worth noting: Example: Text1 Text2 The InnerText for div1 is "Text1Text2" that is it is recursive. The TextContent of div1 is Text1, that is, it is non-recursive. For any other search type this property is ignored.
ContentValue	The ContentValue property specifies the content string to search for when WebAii is performing a content search. For any other type of search this property is ignored.
TagName	This property specifies the tag to search for when WebAii is searching by TagIndex or tag occurrence within the web page. For any other type of search this property is ignored. For example, the 14th <a> element or the first element.
TagIndex	This property specifies the index value when WebAii is searching by TagIndex or tag occurrence within the web page. It's important to note that WebAii uses zero-based indexing, thus the first element has an index of 0. For any other type of search this property is ignored.
XPath	This property specifies the XPath string to use when WebAii is performing an XPath search. See http://www.w3schools.com/XPath/default.asp for a description of XPath and how to use it. For any other type of search this

FindParam Property	Description
	property is ignored.

7.2.5 Elements Explorer: Page properties



When you click on one of the pages shown in Elements Explorer the properties for that web page are displayed in the properties window.

7.2.5.1 Page Properties: FriendlyName

FriendlyName is the name displayed in Elements Explorer for that web page. When the web page is first added to Elements Explorer it is given a default unique name based on the title and URL of the web page. After the web page is added you can optionally change the name to something more meaningful, shorter, and easier to remember. This is not required however.

7.2.5.2 Page Properties: Page properties

Page properties define the characteristics of the web page so that WebAii can tell when that web page is active versus any other web page that your test uses.

Page Property	Description
Title	This property specifies the title of the web page to look for when WebAii is performing a title comparison. The title to compare against is extracted from the <title> element in the <head> section of the web page and compared to this property when WebAii is trying to determine which web page is the active web page.
BaseUrl	The BaseUrl property is the base part of the URL. It contains the protocol and host name part of the URL.

Page Property	Description
Path	The Path property is the relative path part of the URL. It contains the part of the URL after the host name but before the query part of the URL.
Query	The Query property is the query part of the URL, that is, the part that comes after the '?' in the URL.
EntireUrl	You cannot directly change the EntireUrl property. Automation Design Canvas displays the full URL which is simply the three properties BaseUrl, Path, and Query concatenated together to form one long string. This property is not actually used anywhere. It is displayed only for reference and user validation when entering the first three properties.

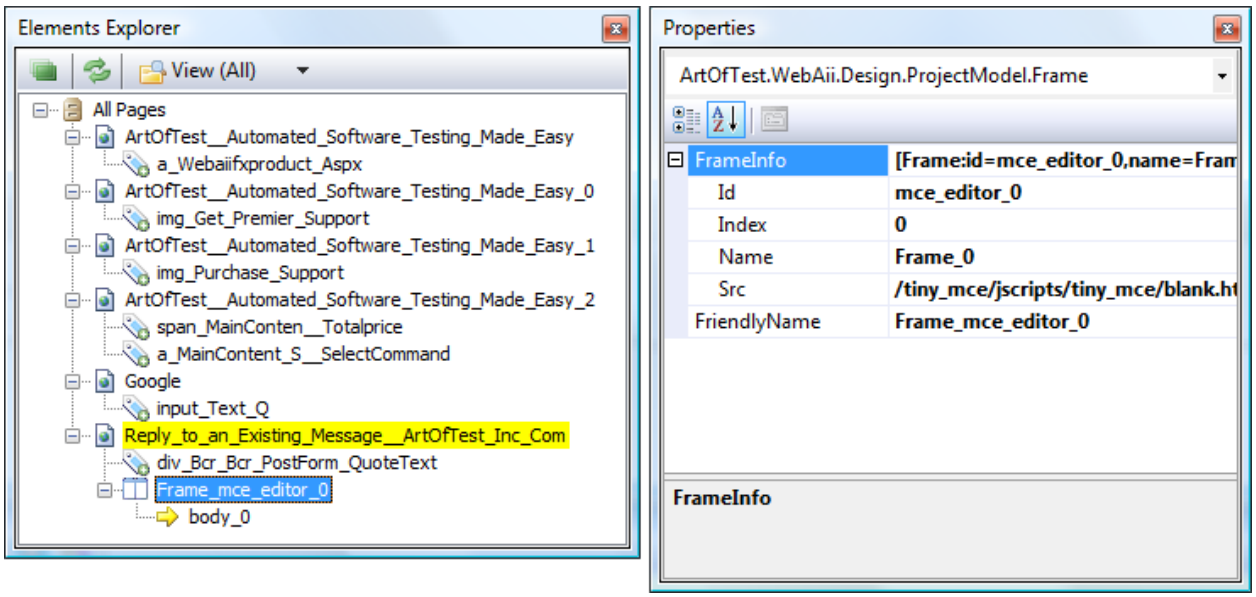
7.2.5.3 Page Properties: CompareUrl

You cannot directly change the CompareUrl property. Design Canvas displays the part of the full URL that it will use when trying to determine which web page is active. The part used depends on the CompareMode as described in the next section.

7.2.5.3.1 Page Properties – CompareUrl: CompareMode

- Title. WebAii determines if this page is active by looking at the web page title as specified in the header of the HTML.
- FullPath. WebAii determines if this page is active by comparing the current URL in the web browser to the BaseUrl and Path properties of this web page. The Query portion of the URL in the web browser will be ignored.
- FullPathAndQuery. WebAii determines if this page is active by comparing the current URL in the web browser to the entire URL of this web page.
- RelativePathOnly. WebAii determines if this page is active by comparing the current URL in the web browser to the Path property of this web page. The BaseUrl and Query portion of the URL in the web browser will be ignored.
- RelativePathAndQuery. WebAii determines if this page is active by comparing the current URL in the web browser to the Path and Query properties of this web page. The BaseUrl portion of the URL in the web browser will be ignored.

7.2.6 Elements Explorer: Frame properties



When a web page contains frames an additional 'Frames' node is added to Elements Explorer as elements are added. A Frames node has the following properties associated with it:

7.2.6.1 Frame Properties: Frame info

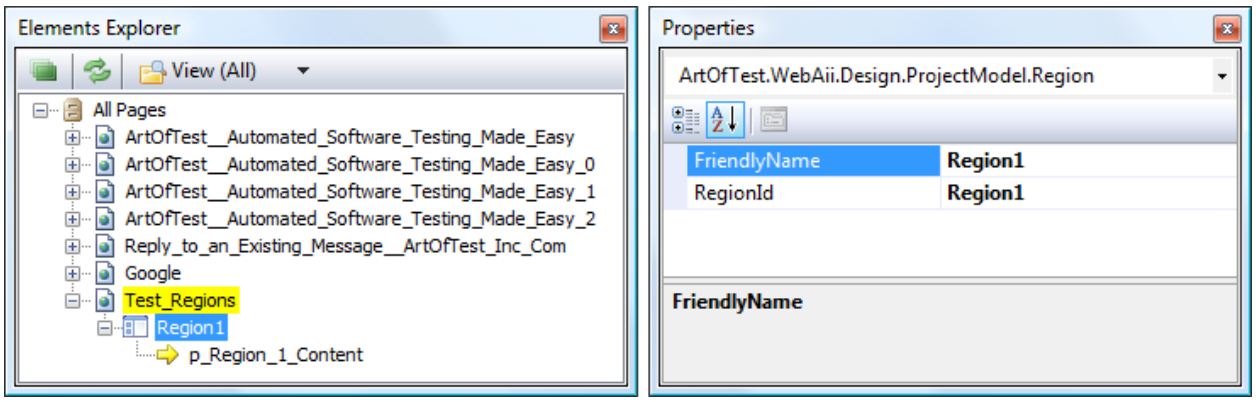
The Frame Info set of properties is used to uniquely identify that frame on the web page.

- ID – This parameter specifies the ID attribute attached to the frame. This ID attribute appears in the parent frame HTML document. If the ID attribute is not present, WebAii checks the name and then the index trying to identify which frame it is.
- Name – This parameter specifies the Name attribute attached to the frame. This Name attribute appears in the parent frame HTML document. If the Name attribute is not present, WebAii resorts to checking the index trying to identify which frame it is.
- Index – This parameter specifies the index of this frame as WebAii sees it in the flat list of frames on the web page. This is used as the last resort to locate the frame if ID or Name has not been set.
- Src – The path to the frame as contained on that page.

7.2.6.2 Frame Properties: FriendlyName

This is the name displayed in Elements Explorer for that frame. When the frame is first added to Elements Explorer it is given a default unique name based on the name and frame index of the frame. After the frame is added you can optionally change the name to something more meaningful, shorter and easier to remember. This is not required however.

7.2.7 Elements Explorer: Test Regions properties



When a web page contains one or more test regions than an additional 'Regions' node is added to Elements Explorer as elements are added. The Regions node has he following properties associated with it:

7.2.7.1 Regions Properties: FriendlyName

FriendlyName is the name displayed in Elements Explorer for that test region. When the region is first added to Elements Explorer it is given a default unique name based on the ID assigned to the test region. After the region is added you can optionally change the name to something more meaningful, shorter and easier to remember. This is not required however.

7.2.7.2 Regions Properties: RegionID

This property specifies the actual ID assigned to the region in HTML. This is what WebAii uses to locate elements in that region.

7.2.8 Elements Explorer: Highlight selected element

This icon  enables/disables highlighting of elements in the Recording Surface. When enabled and you click on an element in Elements Explorer that is contained on that web page a red rectangle will be drawn around it. The web page will automatically scroll as needed in the Recording Surface to make the element visible if it happens to be outside the current viewport.

7.2.9 Elements Explorer: Context menu

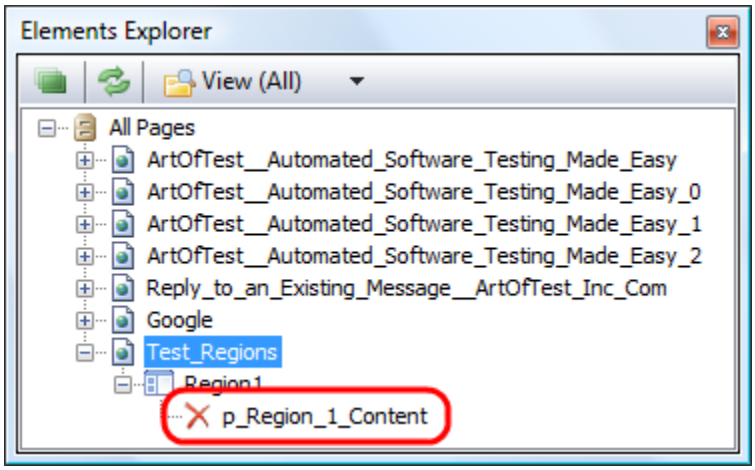
When you right click on any node in Elements Explorer a context menu is displayed. The following sections describe each menu item contained in this context menu.

7.2.9.1 Context Menu: Edit

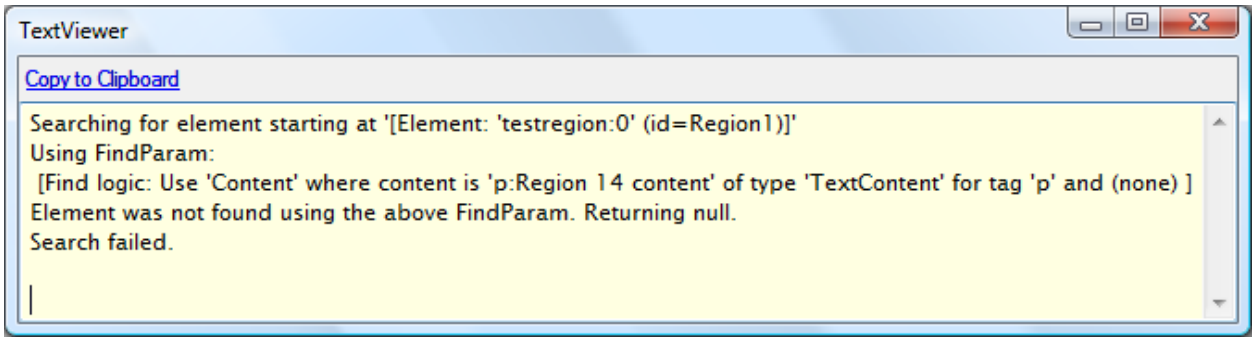
When you right click on an element the Edit menu option will be enabled. By selecting this option the Element editor UI will open. This UI is described in detail in section 7.3.

7.2.9.2 Context Menu: Validate all elements

This menu item is only available for the web page currently loaded in the Recording Surface because it needs access to the live HTML to operate. What it does is verify that all the elements recorded from that page can be found using their current FindParam settings. Any elements that cannot be found on the web page will have an error icon displayed next to them as shown here:



You can right click on these elements and select View Error to see a detailed error message that could prove helpful in explaining why that element cannot be found on the web page. For example:



7.2.9.3 Context Menu: Delete

When you right click on an element which has a reference count of 0 and the ReferencedByCode property is false, then you can select Delete. By clicking on this option that element will be permanently removed from Elements Explorer. Most of the time you do not need to deal with this. Elements are automatically removed when the last test step that refers to them is deleted.

However if you were referencing this element in a code behind method and have removed that method, the element will not be automatically deleted. In addition elements that were manually added to Elements Explorer and still have a reference count of 0 will not be automatically deleted.

It is not absolutely necessary to remove unused elements from Elements Explorer. It is helpful to be able to remove clutter however so that you can minimize the list of elements listed when you need to search for a particular one from the list.

7.2.9.4 Context Menu: Locate in DomTree

When you right click on an element contained on the currently loaded web page the Locate in DomTree menu item is enabled. By selecting this menu item that element will be given focus in DOM Explorer. This allows you to view the full HTML code for that element. DOM Explorer is detailed in section 5.9.

7.2.9.5 Context Menu: Load page

When you right click on an element or a web page in Elements Explorer the Load page menu item is enabled. By selecting this menu item that web page will be loaded into the Recording Surface. This is very useful when you want to continue a test you have been working on, need to perform maintenance on a test, or need to start a new test in the same project.

7.2.9.6 Context Menu: Properties

The Properties menu item is always enabled; however it only does something when you click on a web page node, a region node, a frame node, or an element node. When you select Properties, the properties for that node are displayed in the properties window allowing you to view and edit them as needed.

7.2.10 Elements Explorer: How elements are found

7.2.10.1 FindParam description

Whenever a web page element has an action recorded against it or you explicitly add an element to Elements Explorer, a “FindParam” function is generated that tells the framework how to find that element. The FindParam function uses one of four approaches to finding elements on a web page. Design Canvas tries to intelligently determine which approach works best on that particular web page for finding the element. This intelligence is configurable as described in Appendix B2.

The four approaches that can be used are:

- TagIndex – Locates an element according to its occurrence (that is, index) within the DOM. For example the 14th <a> element or the first element. It is important to note that WebAii uses zero-based indexing. Thus the first element has an index 0.
- AttributesOnly – Locates elements by matching its tag along with one or more of the elements’ attributes. For example an <a> element having a particular href attribute or an <input> having a certain id attribute.
- XPath – XPath will find the element using the XML XPath specified. See http://www.w3schools.com/XPath/default.asp for a description of XPath and how to use it.

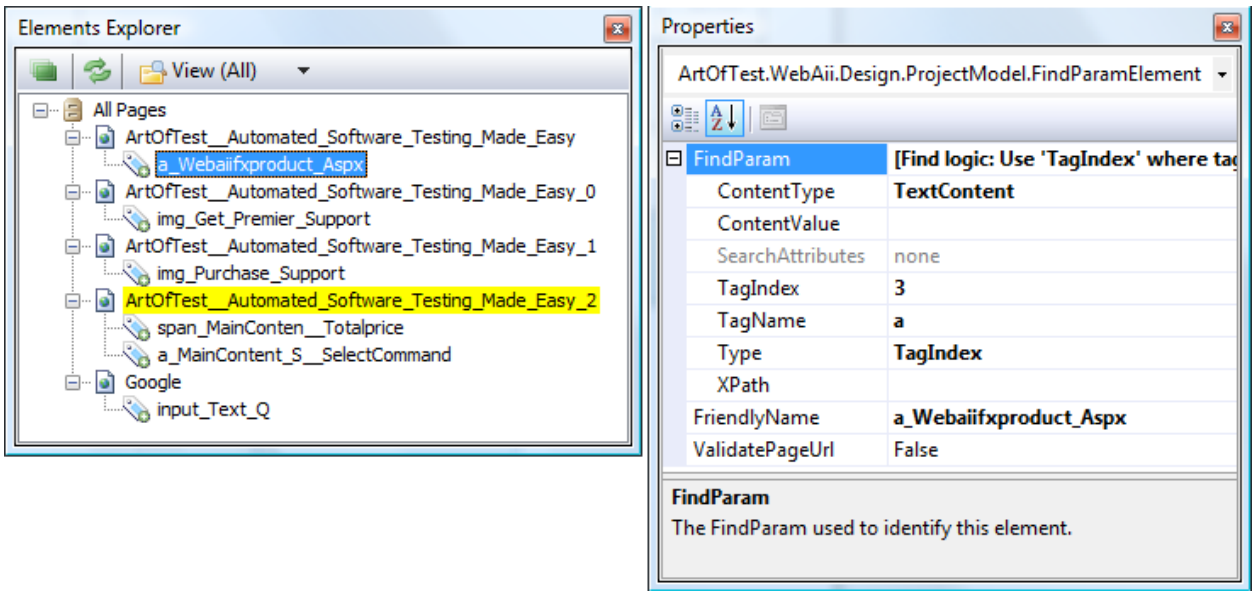
- Content – Locates elements by searching for a matching string. WebAii can perform five types of content searches:
- InnerText – Finds the first element in the DOM tree that contains the matching inner text of a tag, for example, “Data Display”. Be careful as this performs a recursive type of search. It will find the first element that contains the text, even if the text is buried in the middle of the text string. If you have multiple elements with the same tag on your web page you may wind up getting the wrong element returned.
- InnerMarkup – The inner markup of a tag, for example, “ ”. The same care must be taken for this find method as for InnerText.
- OuterMarkup – The outer markup of a tag, for example, “ Some Data <input attr1="Button1" type="button" value="Click Me" onclick="clicked();"/>”. The same care must be taken for this find method as for InnerText.
- TextContent – Text content of the tag only without all its children, for example, “Some Data”. This find method is less risky for finding the wrong element than the above methods as it is non-recursive. The above methods all perform recursive searching.
- StartTagContent – The raw start tag content.

Note: There is a difference between InnerText and TextContent that is worth noting: Example: Text1 Text2
The InnerText for div1 is “Text1Text2” i.e. it is recursive. The TextContent of div1 is: Text1 i.e., non-recursive.

7.2.10.2 Changing how elements are found

There are two different methods you can use to change FindParam’s. Both work equally well. You can either click on the element in Elements Explorer then go to the property window to edit the options directly or you can right click on the element in Elements Explorer and select Edit… to open the Element Editor.

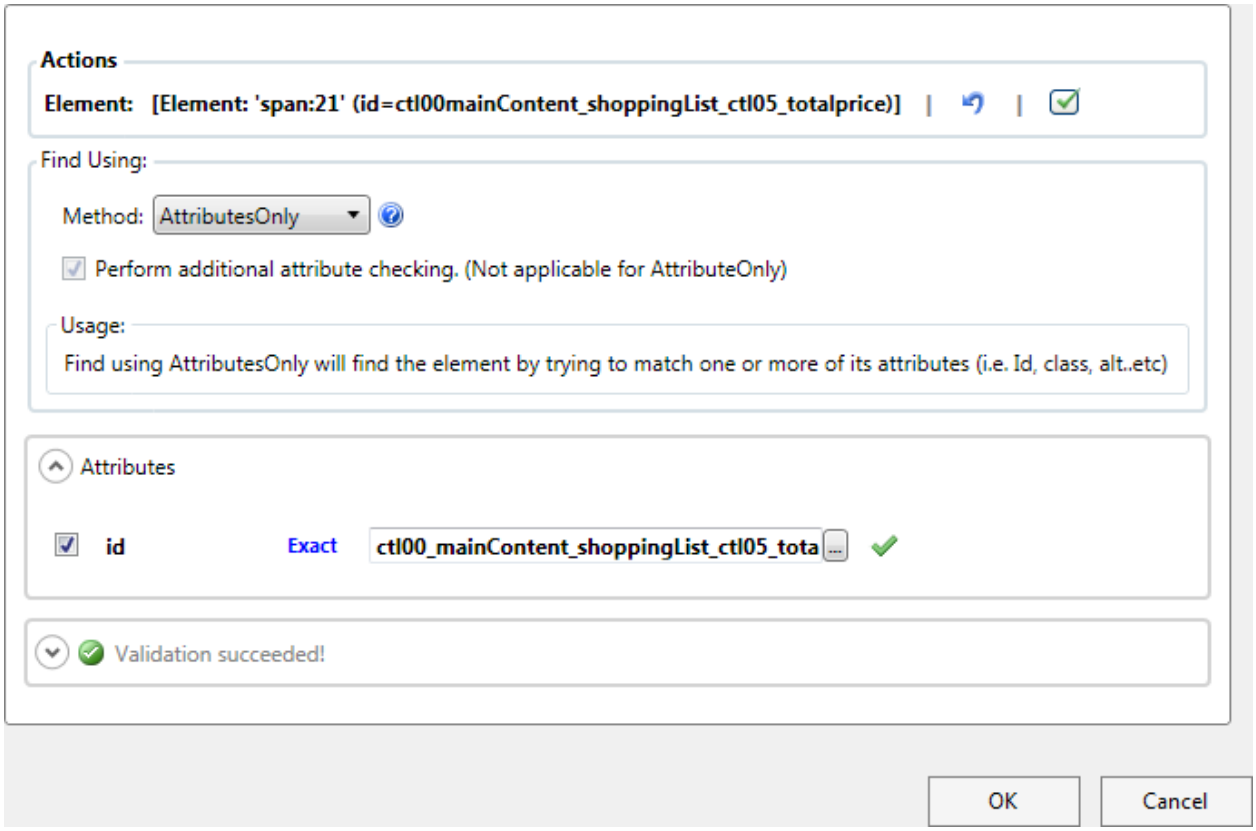
7.2.10.3 Using the Element's Property window



Section 7.2.4.explains how to use the property window.

7.3 Elements Explorer: Element Editor

Use the Element Editor to edit the FindParam properties. You can also test the FindParam properties against the web page loaded in the Recording Surface to verify that it works. Activate Element Editor by right clicking on one of the elements in Elements Explorer and selecting Edit… from the context menu that appears.



The Element Editor window is grouped into four sections:

- Find Using – In this section you specify what type of search to perform to find that element.
- Attributes – In this section you specify the attributes that must match when looking for that element.
- Actions – In this section you can reset the FindParam properties back to their original value and test if the FindParam properties will find the correct element.
- Validation – This section displays the results of testing the FindParam properties.

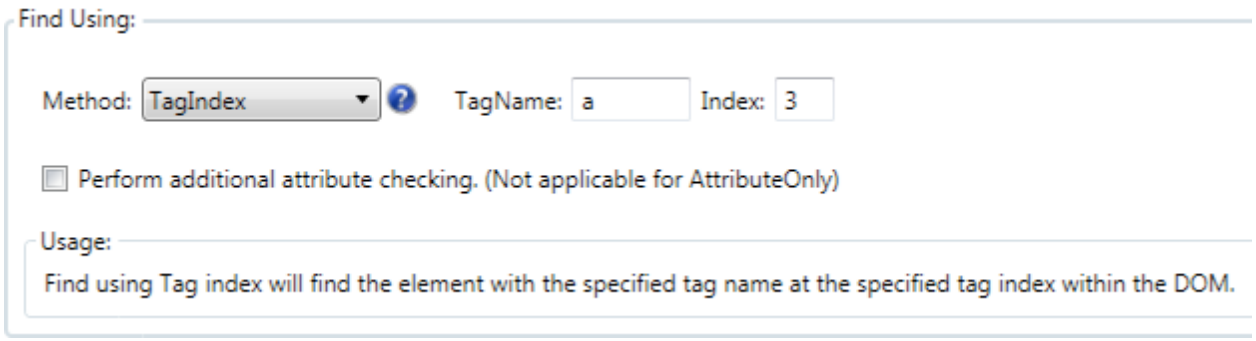
7.3.1 Element Editor: Find Using Section

The Method dropdown selects which type of search to perform when searching for the element on the web page. It can be one of the following:

- TagIndex – Locates an element according to its occurrence (i.e., index) within the DOM. For example the 14th <a> element, or the first element. It's important to note that WebAii uses zero based indexing. Thus the first element has an index 0.

- AttributesOnly – Locates elements by matching its tag along with one or more of the elements attributes. For example an <a> element having a particular href attribute, or an <input> having a certain id attribute.
- XPath – XPath will find the element using the XML XPath specified. See http://www.w3schools.com/XPath/default.asp for a description of XPath and how to use it.
- Content – Locates elements by searching for a matching string.

7.3.1.1 Element Editor: Find by TagIndex



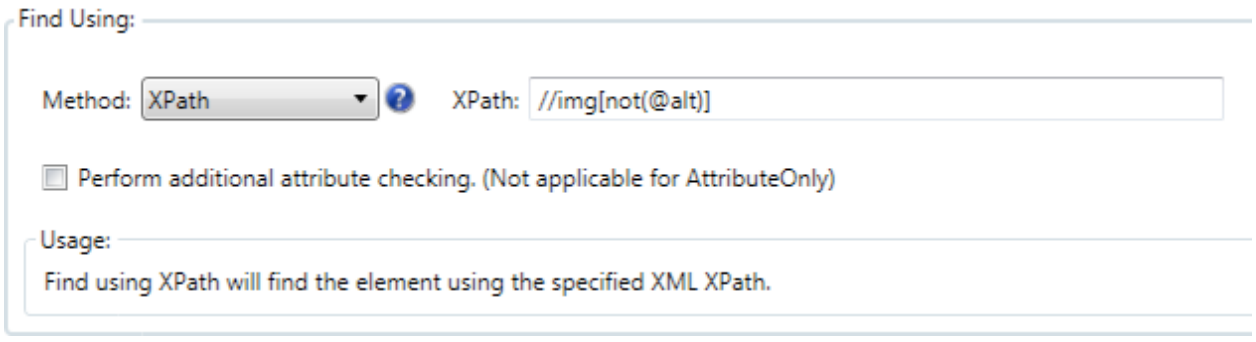
7.3.1.2 Element Editor: TagName

The TagName textbox is shown when TagIndex is selected as the method. Enter the tag to look for when searching for this element, for example, a or div or img or p, etc…. Only one tag can be entered. WebAii cannot look for multiple tags at the same time for the same element.

7.3.1.3 Element Editor: Index

The Index textbox is shown when TagIndex is selected as the method. This property specifies the index value when WebAii is searching by TagIndex or tag occurrence within the web page. It's important to note that WebAii uses zero-based indexing, thus the first element has an index of 0.

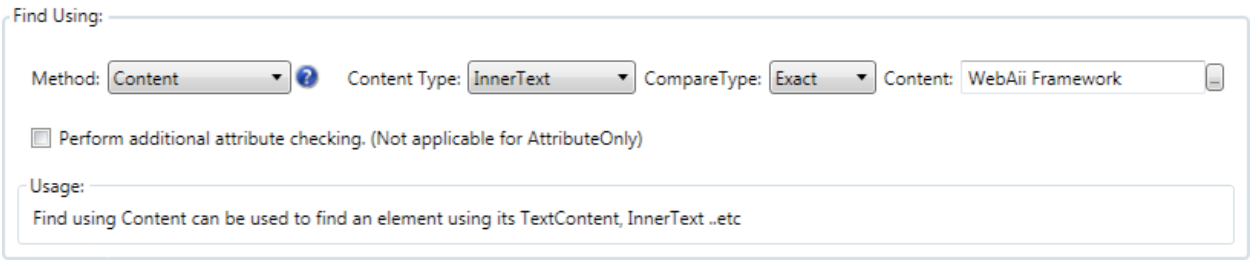
7.3.1.4 Element Editor: Find by XPath



7.3.1.5 Element Editor: XPath

The XPath textbox is shown when XPath is selected as the method. This property specifies the XPath string to use when WebAii is performing an XPath search. See http://www.w3schools.com/XPath/default.asp for a description of XPath and how to use it.

7.3.1.6 Element Editor: Find by Content



7.3.1.7 Element Editor: ContentType

The ContentType dropdown is shown when Content is selected as the method. When WebAii is doing a search for a specific content string, this parameter specifies which type of content it is looking for. It can be one of these values:

- InnerText – Compares the inner text of the tag. For example, “Data Display”. Be careful because InnerText is recursive. It includes not only the text of the current element but all the text for all children elements.
- InnerMarkup – The inner markup of a tag. For example,“ ”. The same care must be taken for this type of compare as described for InnerText.
- OuterMarkup – The outer markup of a tag. For example,“ Some Data <input attr1="Button1" type="button" value="Click Me" onclick="clicked () ;"/>”.The same care must be taken for this type of compare as described for InnerText.
- TextContent – Text content of the tag only without all its children. For example,“Some Data”. This type of compare is less risky than the above methods as it is non-recursive. The above methods are all recursive, that is, they include their child elements.
- StartTagContent – The raw start tag content.

Note: There is a difference between InnerText and TextContent:
For example: Text1 Text2
The InnerText for div1 is “Text1Text2” that is, it is recursive.
The TextContent of div1 is Text1 which is non-recursive.

7.3.1.8 Element Editor: CompareType

The CompareType dropdown is shown when Content is selected as the method. Select which of the following compare types is to be performed:

- Exact – The content must match exactly with the expected value.
- Partial – The content must contain the expected value somewhere/anywhere within the content.
- RegEx – The content must match the regular expression. WebAii takes advantage of the .NET Framework Regular Expressions as described at: http://msdn.microsoft.com/en-us/library/hs600312.aspx. The string you enter as the expected value is used as the expression to use in the match comparison.

7.3.1.9 Element Editor: Content

The Content textbox is shown when Content is selected as the method. Enter the content that must match when looking for this element. The content string is prefilled with the current value of the element when you select the content type from the ContentType dropdown. How this value is used during the search depends on the CompareType selected as described in the previous paragraph.

7.3.1.10 Element Editor: Perform additional attribute checking

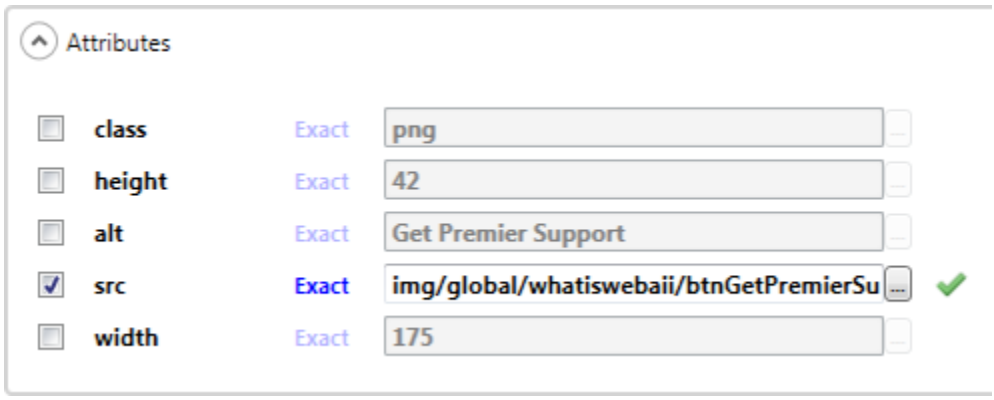
This checkbox is enabled when the find method selected is anything but AttributesOnly. When this checkbox is checked, WebAii will perform additional attribute checking as specified in the next section during its search for that element. Otherwise the WebAii will decide it has found the element based on the other parameters only.

7.3.1.11 Element Editor: Validation icon

When you test finding the element an icon is displayed in the fifth column:

- When the attribute and its value was found.
- When the attribute or value could not be found.

7.3.2 Element Editor: Attributes Section



The Attributes section is enabled when you select Find by AttributesOnly or the “perform additional attribute checking” checkbox is checked. It will be filled with an array of attributes currently found on the element from the Recording Surface.

7.3.2.1 Enable checkbox

In the first column of the array is a checkbox. When this checkbox is checked WebAii will look for that attribute when searching for the element on the web page. When unchecked that attribute will be ignored.

7.3.2.2 Attribute name

In the second column is the name of the attribute. You cannot edit the attribute name.

7.3.2.3 Exact or Partial match

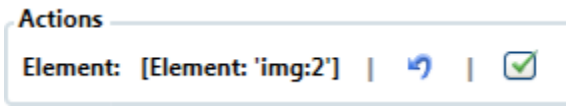
In the third column is the word Exact or Partial. It is only enabled when the checkbox for that attribute is checked. By clicking on it the name will toggle between the two values.

- Exact – When Exact is displayed the attributes value must match exactly with the expected value.
- Partial – When Partial is displayed the attributes value must contain the expected value somewhere/anywhere within the content.

7.3.2.4 Attribute value

In the fourth column is the string value of the attribute to match. It is only enabled when the checkbox for that attribute is checked. You can enter any string into this checkbox. It is prefilled with the current value of that attribute from the element in the Recording Surface. Normally you can leave the value alone but sometimes (such as for ID and Name attributes on compiled ASPX pages) you may need to change from Exact to Partial and trim the prefilled attribute value to the last part of the compound/compiled name.

7.3.3 Element Editor: Actions Section



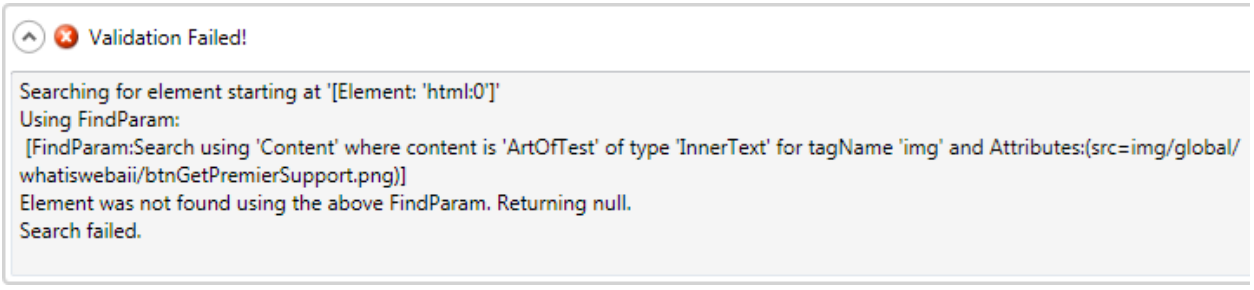
The Actions section displays the tag name and tag index of the element along with two icons that you can click on:

- Clicking the discard changes icon resets all of the FindParam properties back to their original saved settings.
- Clicking the Validate icon tests finding the element using the FindParam properties you have set in this Element Editor. If successful “Validation succeeded!” is displayed in the Validate section. If the search failed “Validation Failed!” is displayed in the Validate section.

7.3.4 Element Editor: Validate Section

The Validate section displays the results of an element search test. If the search is successful "Validation succeeded!" is displayed in this section. 

If the search failed "Validation Failed!" is displayed in this section.



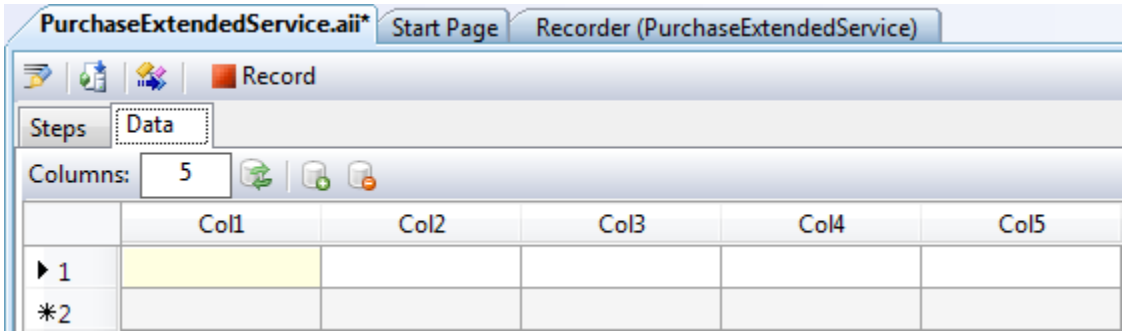
8 DATA-DRIVEN TESTING

Automation Design Canvas supports data-driven testing. All recorded steps (Actions/Verifications or synchronizations) come with data-driven properties that allow you to bind them to a data source. Design Canvas supports all data sources that come with Visual Studio Team System (Database, CSV, XML). In addition it has a built-in data grid that allows you to easily build your own data source right inside your test without the need to use or manage external sources or dependencies.

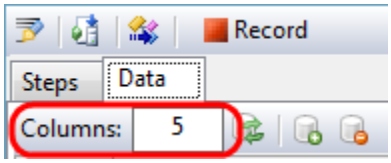
Each test can either be bound to a different data source or have its own data grid. Different tests can share the same external data source however they cannot share the same internal data grid. Each data grid is specific to the test it is contained in only.

8.1 Built-In Data Grid

To add and manipulate the internal data grid (data array) for data-driven testing you must first open the Data tab for the .aii document as explained in section 5.6.

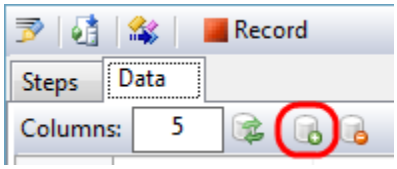


8.1.1 Number of columns



Enter the number of columns you want for your data grid in the Columns textbox. This number is used when creating or updating the size of your data grid.

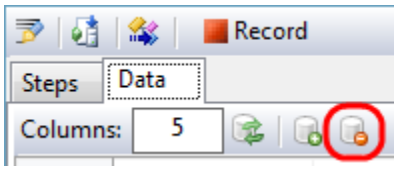
8.1.2 Create new data table



Click the **Create Data Table** icon. This adds a new data grid to that test only. It will be created with the number of columns specified by the Columns textbox and one empty row. The columns

names will be given default column names. You can change them later.
***Note**: This data grid cannot be shared with other tests.*

8.1.3 Remove data table



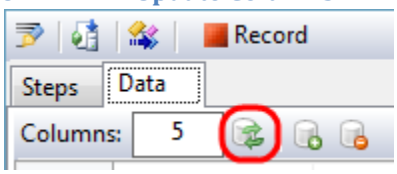
To remove the data grid from the test simply click the **Remove Data Table** icon. You will be prompted to confirm the delete.
Note: Once the data grid is deleted all the data it contained is immediately discarded. There is no way to undo the delete so be certain you want to discard the data grid!

There is no harm in keeping a data table in your test that is not used. It simply adds takes a few more KB of disk space and memory to store the data. It will not affect the performance of your test in any way.

8.1.4 Deleting columns

There are two methods for deleting columns you no longer need which are explained in the following two sections.

8.1.4.1.1 Update Columns

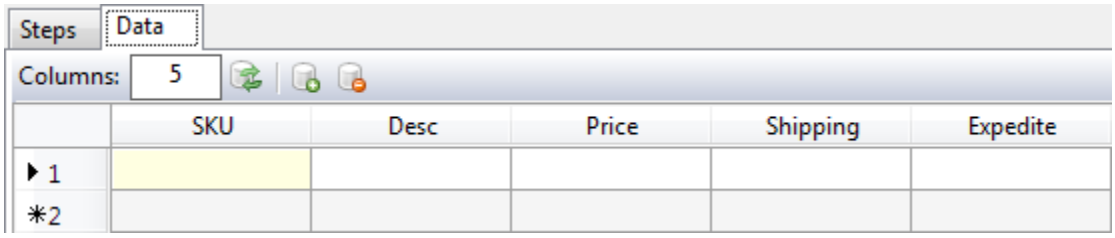


You can change the number of columns kept by your data grid. You can add or remove columns. You do not have a choice of which columns will be removed however. Enter the number of columns in the Columns textbox then click the **Update Columns** icon. If you are adding columns they will be immediately appended to the columns already there and given default column names. If you are deleting columns you will first be prompted to confirm the delete. When you click **Yes** to confirm the delete the columns on the right will be deleted until the new size is reached, for example, if the data grid was 5 columns wide and you specify 3 columns to be the new size, the 2 columns on the right will be discarded. ***Note**: Once the columns are deleted all the data they contained is immediately discarded. There is no way to undo the delete so be certain you want to discard the columns!* There is no harm in keeping data in your test that is not used. It simply adds takes a few more KB of disk space and memory to store the data. It will not affect the performance of your test in any way.

8.1.4.1.2 Delete specific column

To delete a specific column right click on the column name and select Delete.
Note: Once the column is deleted all the data it contained is immediately discarded. There is no way to undo the delete so be certain you want to discard that column! There is no harm in keeping data in your test that isn't used. It simply adds takes a few more KB of disk space and memory to store the data. It will not affect the performance of your test in any way.

8.1.5 Changing column names



You have the ability to change the name of each column. To change the name right click on the column name and select Rename. A dialog box will popup allowing you to enter the new name. ***Note**: All column names must be unique because the column name is what you use to bind to test properties and code behind methods.* Automation Design Canvas will prevent you from accidentally giving two columns the same name.

8.1.6 Adding data and rows

The data grid acts like Microsoft Access. To add or modify data on an existing row simply click on one of the cells for that row and type the data you want entered. To add a new row simply click on one of the cells in the last empty row at the bottom of the data grid and enter the data you want. A new row will automatically be added to the data grid.

Automation Design Canvas also supports copy & paste into the data grid. Copy the data from some external data source (for example, an Excel spreadsheet, text file, etc.) and paste it into the data grid at the proper location.

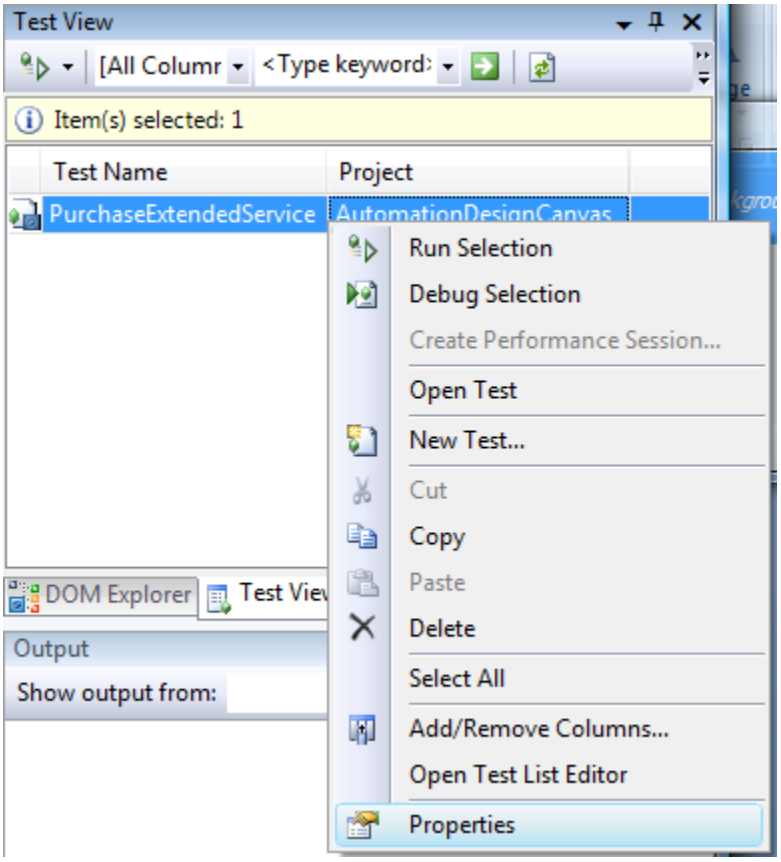
8.1.7 Deleting rows

You can delete one row at a time from the data grid. Simply right click on the row and select Delete. **Note**: Once the row is deleted all the data it contained is immediately discarded. There is no way to undo the delete so be certain you want to discard that row!

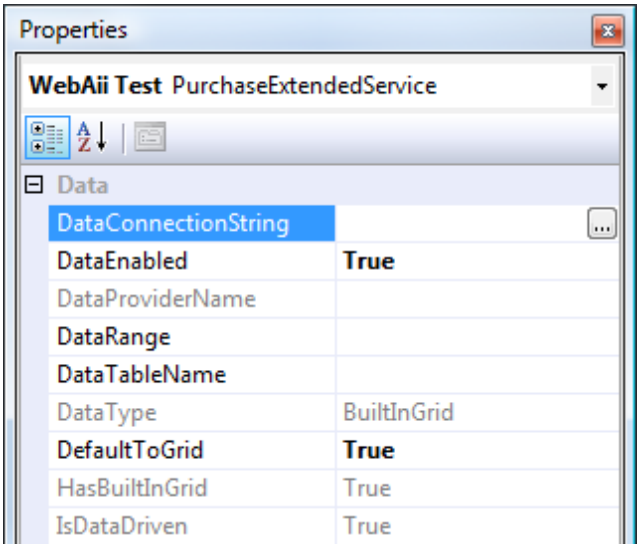
8.2 Binding test step properties to an external data source

How to bind the properties of a test step to an external data source in test project is not obvious. Following these steps will make it easier:

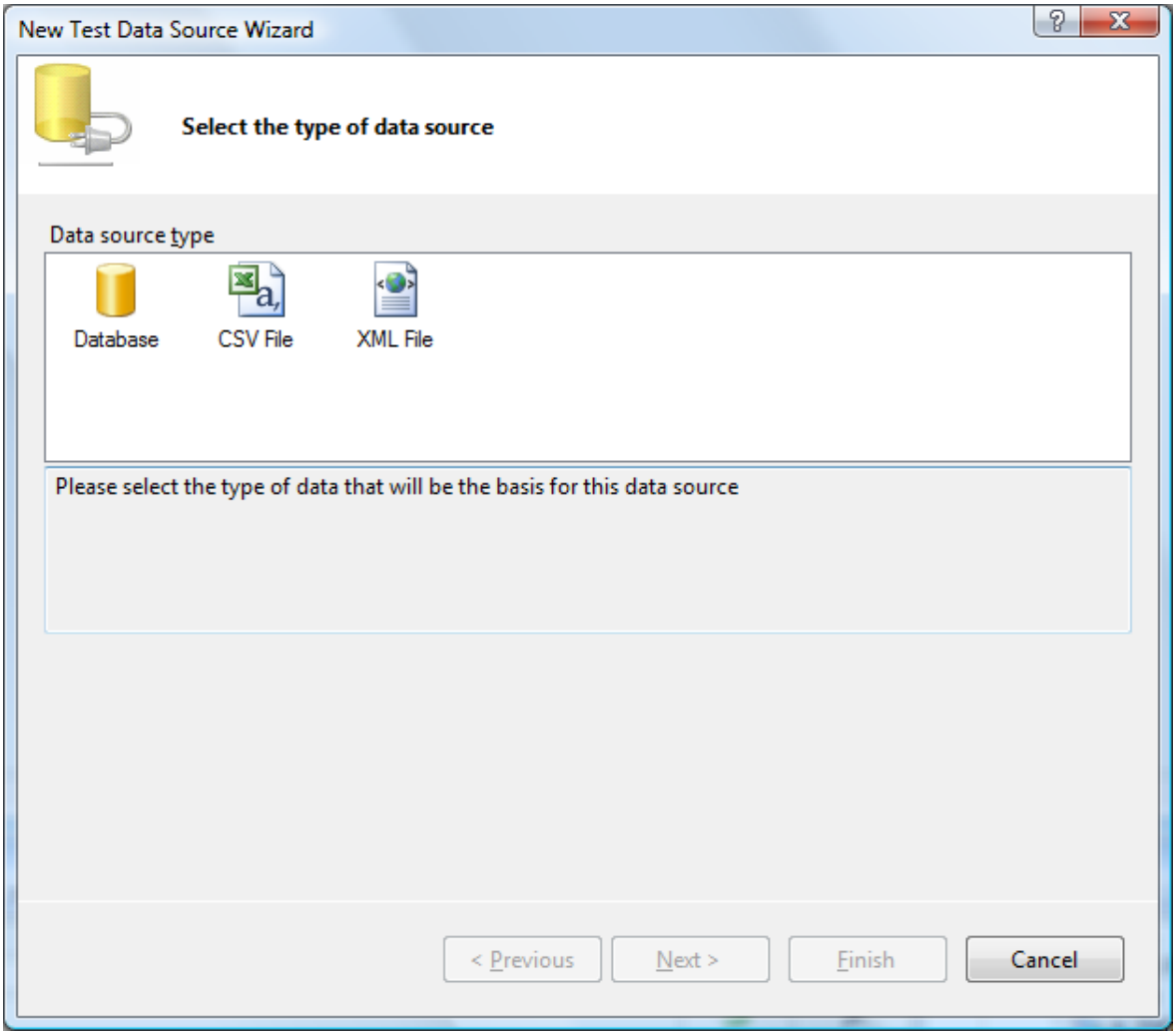
- Open the Test View window or the Test List Editor window in Visual Studio. A list of tests present in your test project will be displayed.



- Right click the test that you would like to start using an external data source for and select Properties. This opens the Properties window showing all the properties for that test.
- Look for the Data section in the properties window. If this section is not already expanded, then expand it by clicking the + icon.
- At the top of this section is the DataConnectionString property. Click this row to reveal the … icon.



- Click the … icon to open the “New Test Data Source Wizard”.



- You can select to use either a Database source (Access database file, ODBC Data Source, SQL Server, etc.), a CSV file, or an XML file as your external data source. Click on one of the three then click **Next** to proceed to the next step of the wizard.
- Complete all the steps in the wizard.
- You now have a connection pipeline between that test and the external data source.

Another excellent resource for understanding data-driven testing in Visual Studio is at: http://msdn.microsoft.com/en-us/library/ms182519.aspx and http://msdn.microsoft.com/en-us/library/ms182528.aspx

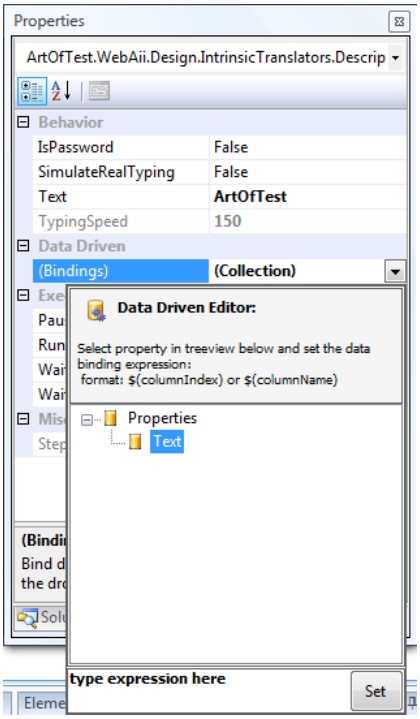
8.3 Reference the data array in a recorded step

Section 9.2.4 explains how to reference the data from your data grid.

8.4 Referencing the data array in a code behind method

Whether you are binding to an internal data grid or an external data source the process is the same. Suppose you have a “Set Text” step in your test that you need to be filled with data from your internal data grid or an external data source. To bind the “Text” property of this step to your data follow these steps:

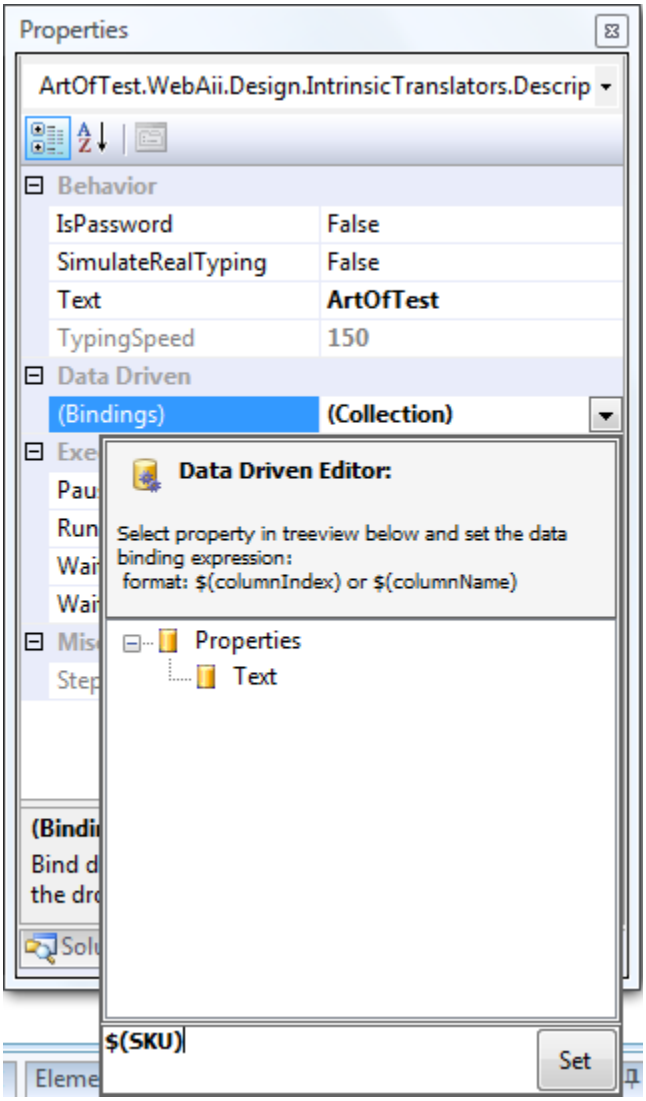
- Open the properties window for that test step as described in section 5.7.6.6.
- Look for the Data-driven section in the properties window. If this section is not already expanded, then expand it by clicking on the ⊟ icon.
- Click on the (Bindings) row to reveal a dropdown arrow.
- Click on the dropdown arrow to open up the Data Driven Editor.



- The Data Driven Editor presents a list of test step properties that can be bound to data. Some steps have multiple properties while others have only one. In this example, a Set Text step, there is only one property that can be bound… the Text property.
- Select the property to be bound by clicking on it… the Text property in this example.
- Now click where it says “type expression here” and enter the binding string. The binding string takes the format: ($columnIndex) or ($columnName). So if the first column in your data grid was a column named SKU you could enter either:

($0)
($SKU)

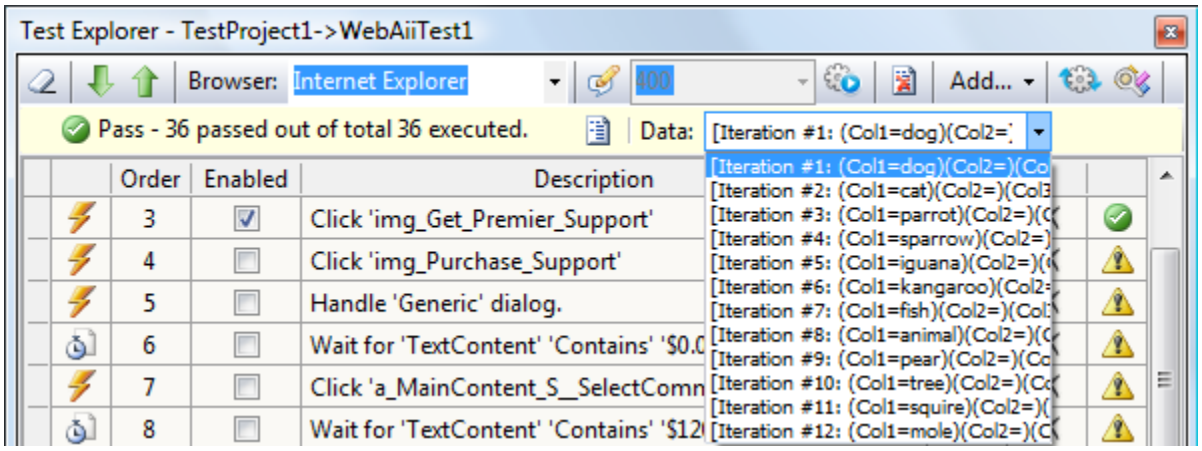
- Use whichever form you prefer.



- Click Set. This binds the property you selected in step 6 (the Text property in this example) to the first column or the column named ‘SKU’.
- The Data Driven Editor window stays open after you click Set. This allows you to bind data grid columns to other properties that may be present on that test step. Clicking anywhere else will automatically close the Data Driven Editor window.

8.5 Quick Executing a Data-Driven Test

When a data-driven test is run using Quick Execute the results of each iteration are displayed in a drop down box as shown here:



The totals shown as Passed and Executed reflect the total across all iterations. Thus for a three step test with 12 iterations (that is, 12 rows of data) you will get 36 steps executed and (hopefully) up to 36 passed.

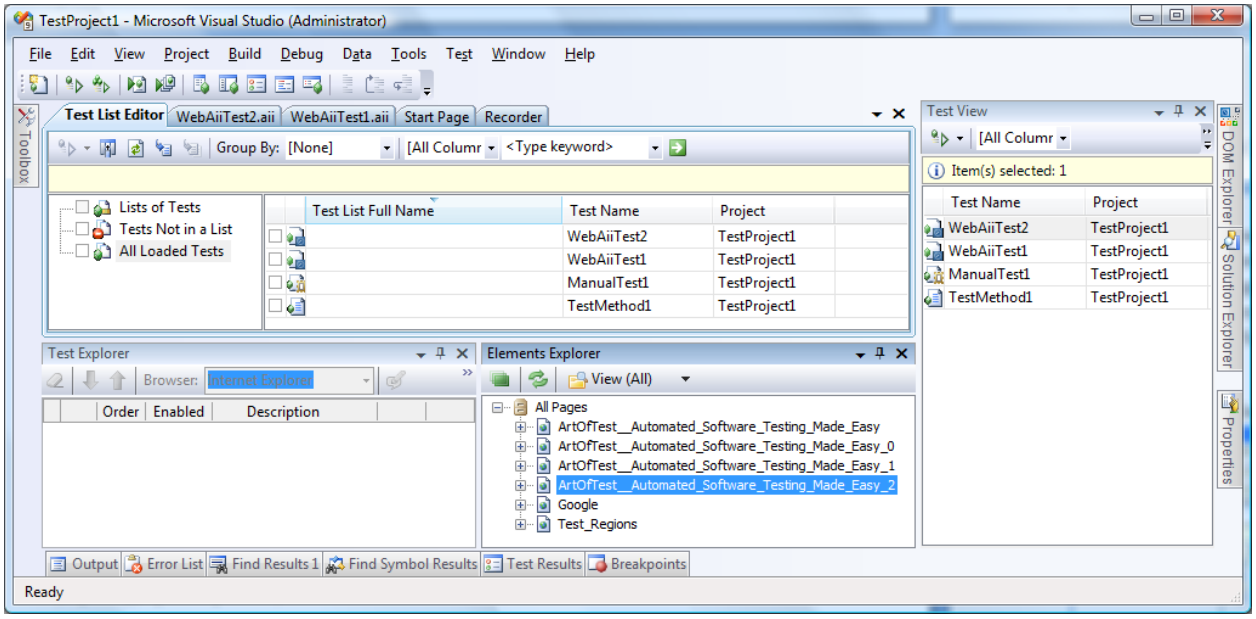
When you select one of the iterations from the iteration drop down list the pass/fail results for each step in that iteration are displayed in Test Explorer. If one of them failed you can click on it to see the details of the failure in the debug UI. Hopefully you will have enough information to determine the cause of the failure and fix it right there. Or perhaps you have discovered a bug in the website that needs to be filed in your bug tracking system.

9 VISUAL STUDIO INTEGRATION

9.1 Managing WebAii Tests Using the Visual Studio Test Window

WebAii tests behave exactly like other built-in Visual Studio (VS) Team Suite test types such as Web Tests and Unit Tests and you can manage your WebAii test using the VS test windows. WebAii tests can be aggregated with other VS Team Suite test types in the run manager, test lists, and test view. They also act like other VS test types and can participate in source control and Team Foundation Server's with Continuous Integration Server and remote execution and reporting.

The following screen depicts the ability to use VS test windows for WebAii tests:



10 TEST CASE MAINTENANCE – RESOLVING TEST FAILURES

Automated testing is not only about how fast you can record tests but also about how fast you can analyze and resolve test failures. Automation Design Canvas helps testers maintain, analyze, and fix their automated tests by providing the following features:

- **Visual State Capturing:** As you record each step, Design Canvas captures the browser state at that moment with the target element highlighted. Each visual step is then added to the storyboard so that you can return to it at will and view a graphical depiction of the test steps. Each pictorial step is also tied to the actual step so as you re-order your steps or delete them the visual storyboard of your test is kept up to date. The goal of this feature is to help you understand where each element is at the time of recording and what the entire page looks like at that moment. In some cases, the graphical depiction of the test steps can help you pin-point a failure by simply looking at the captured images.
- **DOM State Capturing:** In many cases, failures are due to issues such as ID or attribute value changes on HTML elements and such failures rarely exhibit visual changes. For this Design Canvas also captures the DOM state at that moment to help you understand changes in the DOM.
- **DOM Tree Diff View:** When a failure is loaded into Design Canvas, the DOM Tree tool window enables a "DOM Diff" view to help you understand differences between the recorded and executed DOMs. This side-by-side view allows you to easily inspect property changes or DOM structure changes between the recorded and executed test steps.
- **DOM Element Index View:** Each element in the DOM Tree is prefixed with its TagIndex. Together with the DOM Tree Diff View, this allows you to quickly detect and match two DOM nodes for the recorded and executed DOM trees.
- **Live Validation:** Both Element Identification and Verification Rules can be validated 'live'. As you investigate your test failures, you can readily edit the failed identification or verification rule and validate it against the executed DOM.

10.1 Resolving Test Step Failures

Automation Design Canvas offers built-in support through its execution engine to help solve some of the most common web automation failures and provides contextual information to help you readily resolve any issues. Typical web automation test step failures occur when:

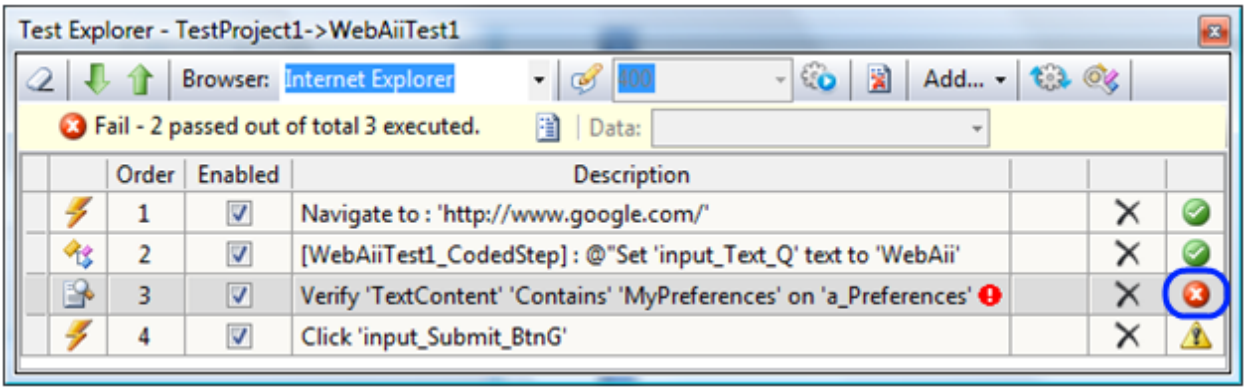
- The test could not find an element on a web page it was looking for, or
- An element was not in the correct state causing a validation failure.

You, the test engineer, must deduce which is the root cause and then determine the correct fix for it. Design Canvas helps you a) deduce which is the root cause, and b) modify the test to apply the correct fix.

Hopefully, before you run into your first test failure, you already captured the browser states during the creation/recording of your test as described in section 5.15. If you didn't then the only

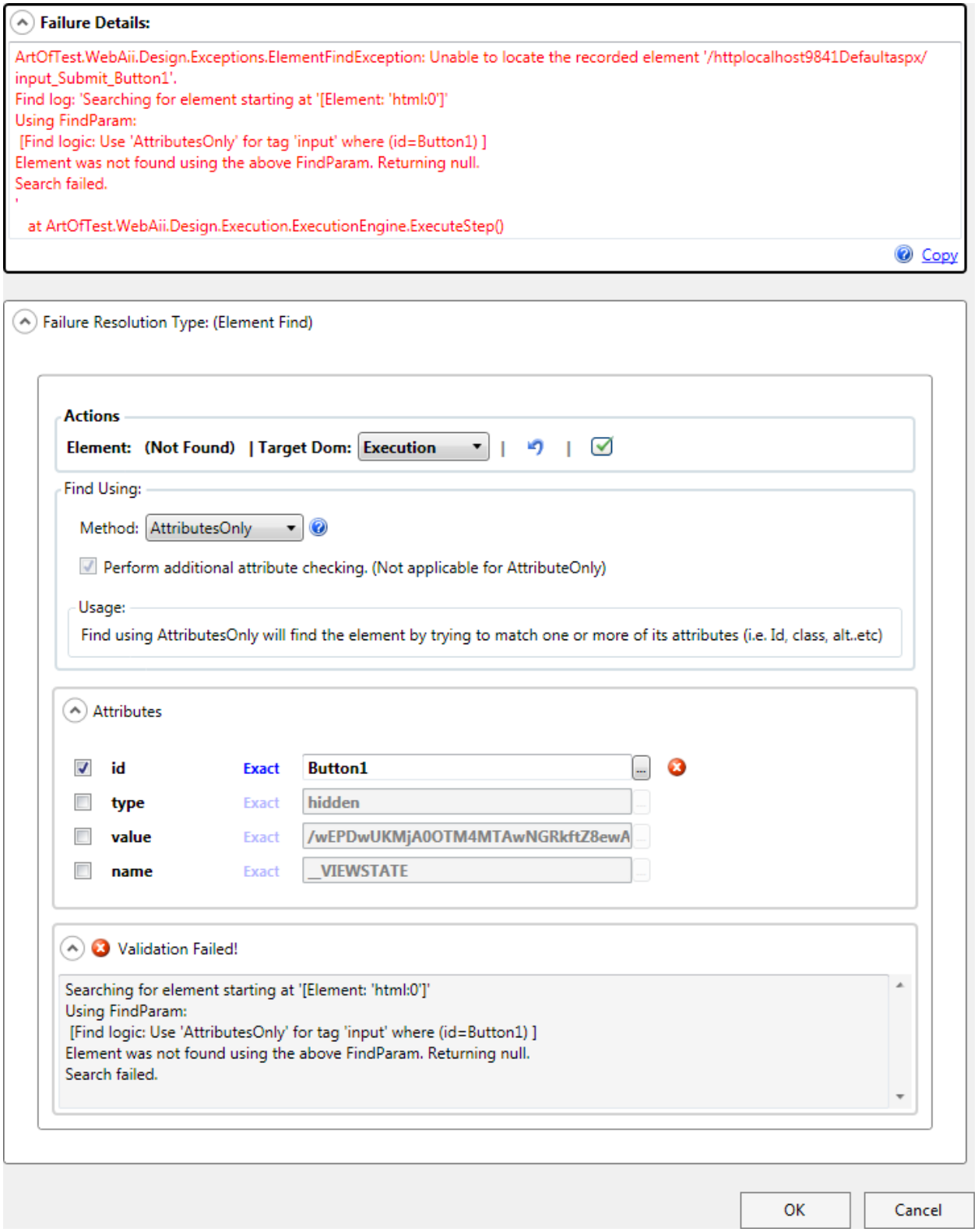
the execution version of the DOM will be available for viewing while investigating your test failure.

If you have not already done so, try running the failing test using Quick Execute. When the test fails the step that failed will have a red X against it. Double clicking on the red X opens up either the Verification Builder or the Element Editor depending on which of the two failure types this one is.



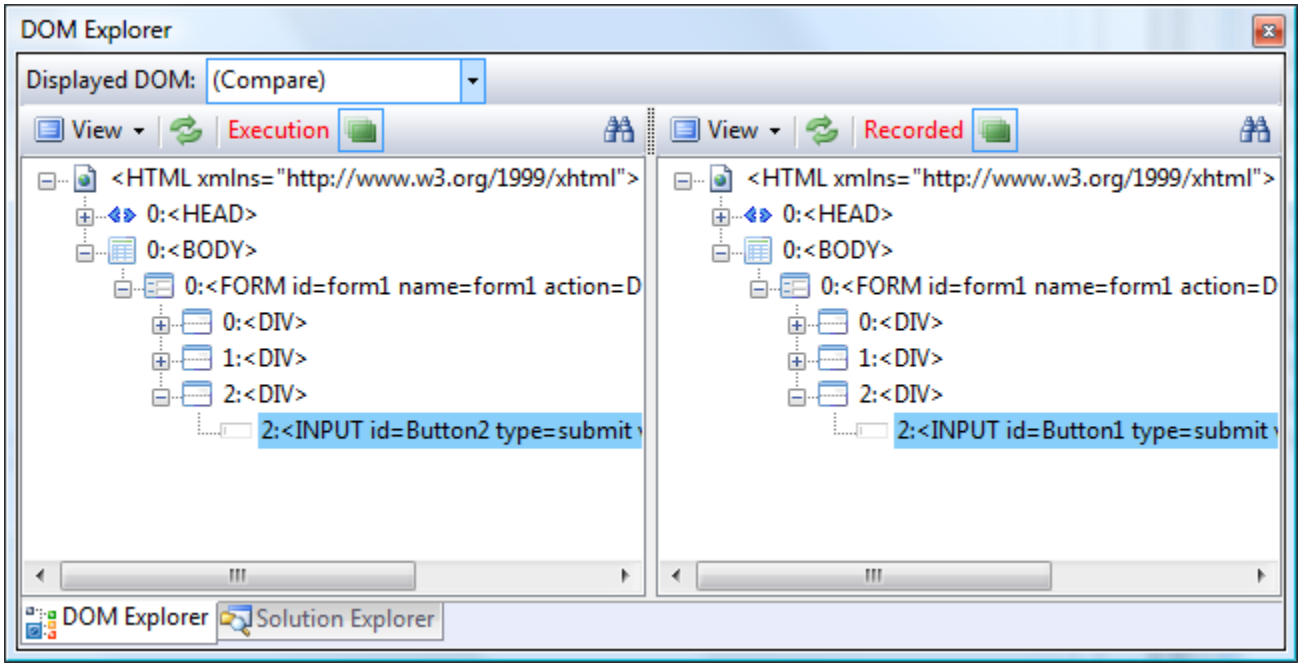
10.2 Resolving element find failures

If Automation Design Canvas was unable to find the element the Element Editor will open but its display is different from what you see when recording a test. The top of the Element Editor shows details of the failure that Design Canvas found. For example:



Here you see that Design Canvas was unable to find an element with an ID of “Button1”. Another significant difference is that in the Actions section you can choose whether to validate against the DOM that was captured at the time of test execution or at the time the test was recorded. This helps you to see how the DOM may have changed between when the test was recorded and when it was executed. ***Note**: this dual DOM display is only available if you took the extra step of capturing the browser state during the recording of your test as described in section 5.15.*

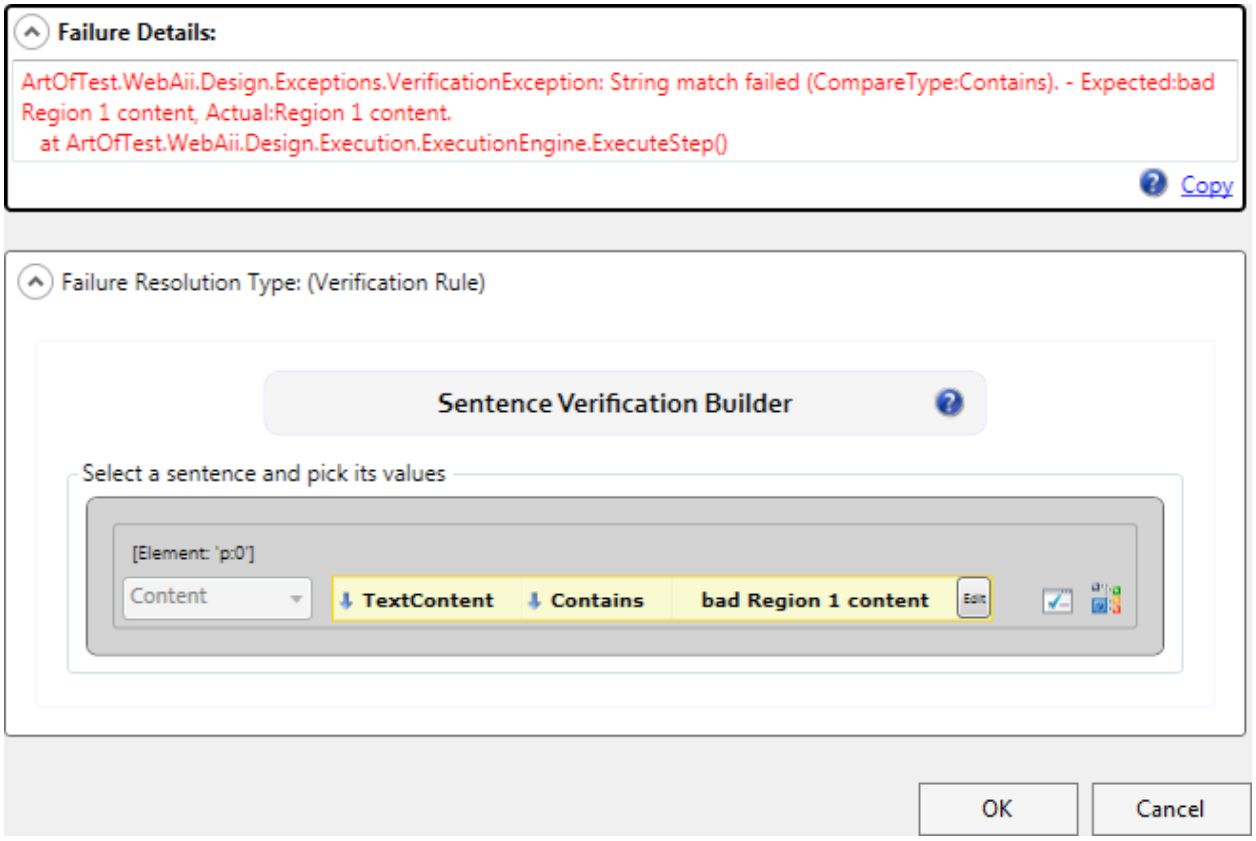
Also note that DOM Explorer has switched modes allowing you to display either the DOM at the time the test was recorded or at the time of test execution or both as a side by side comparison. Studying what has changed in the DOM can help you determine why Design Canvas was unable to find the element when the test ran. This feature is enabled only if you have captured DOM states for the test before execution (See Section 5.15). Here is an example of the executed and recorded versions in DOM Explorer side by side:



Careful study of the DOM trees reveals that the ID of our button has changed from “Button1” to “Button2”. Since the Element Editor is already open you can proceed to change the rule on how to find this element. If you also loaded the web page into the Recording Surface you can now test the change to the verification rule to verify that it fixed the problem.

10.3 Resolving verification failures

If Automation Design Canvas found the element but could not validate some property of the element, Verification Builder will open but is different from what you see when recording a test. The top of Verification Builder now shows details of the failure that Design Canvas encountered as illustrated by the following example:



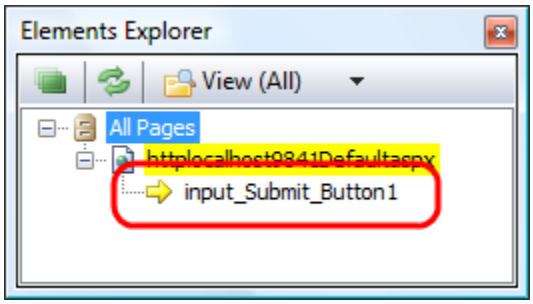
Here you see that Design Canvas was expecting the string “bad Region 1 content” but instead found the string “Region 1 content”. Since we are already in Verification Builder we can proceed to change the verification rule as needed. If you also loaded the web page into the Recording Surface you can test the change to the verification rule to verify that it fixes the problem.

Clicking OK saves your changes to the verification test step in the project.

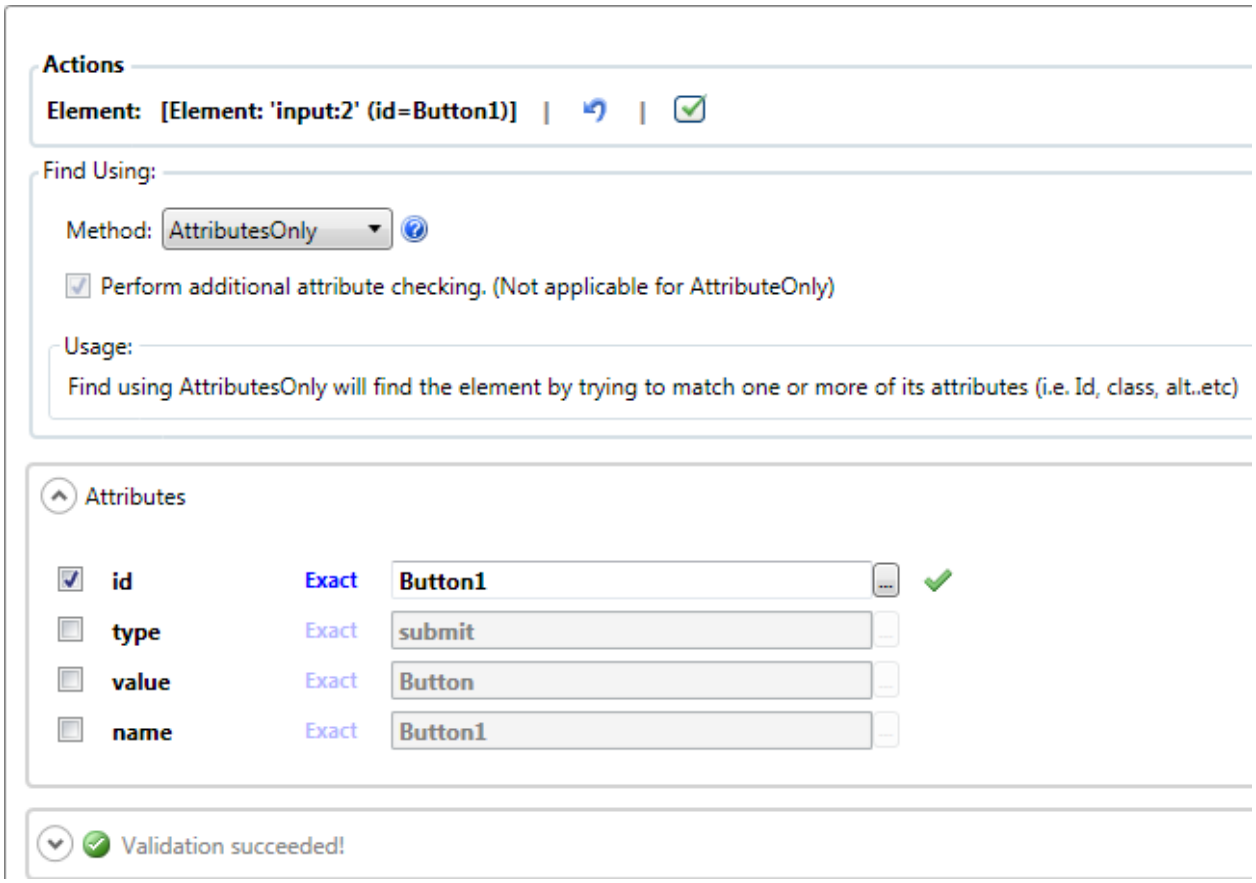
10.4 Case Study: Resolving Failures

To help demonstrate these features, consider a simple common failure example in web automation: element ID changes. For example in the following HTML Page:

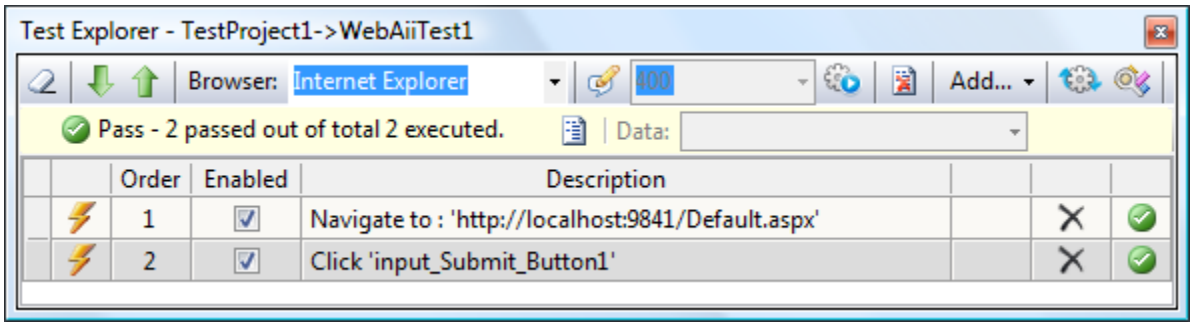
The input button has a ‘Button1’ ID associated with it. When you load this page in Automation Design Canvas 1.1, record the clicking of the button. Design Canvas determines that the best method to identify this element on the page is using its ID (Button1) since the ID is currently defined on the page and is unique. The following Elements Explorer view depicts how this element appears. For more information about how Design Canvas builds identifications for elements including how to customize them, see Section 7.



Here is what the FindParam editor looks like:

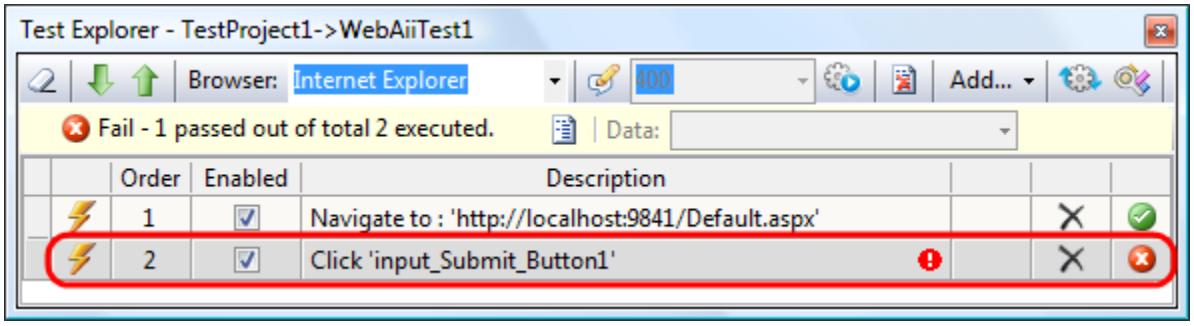


If you run the test, obviously everything should pass…

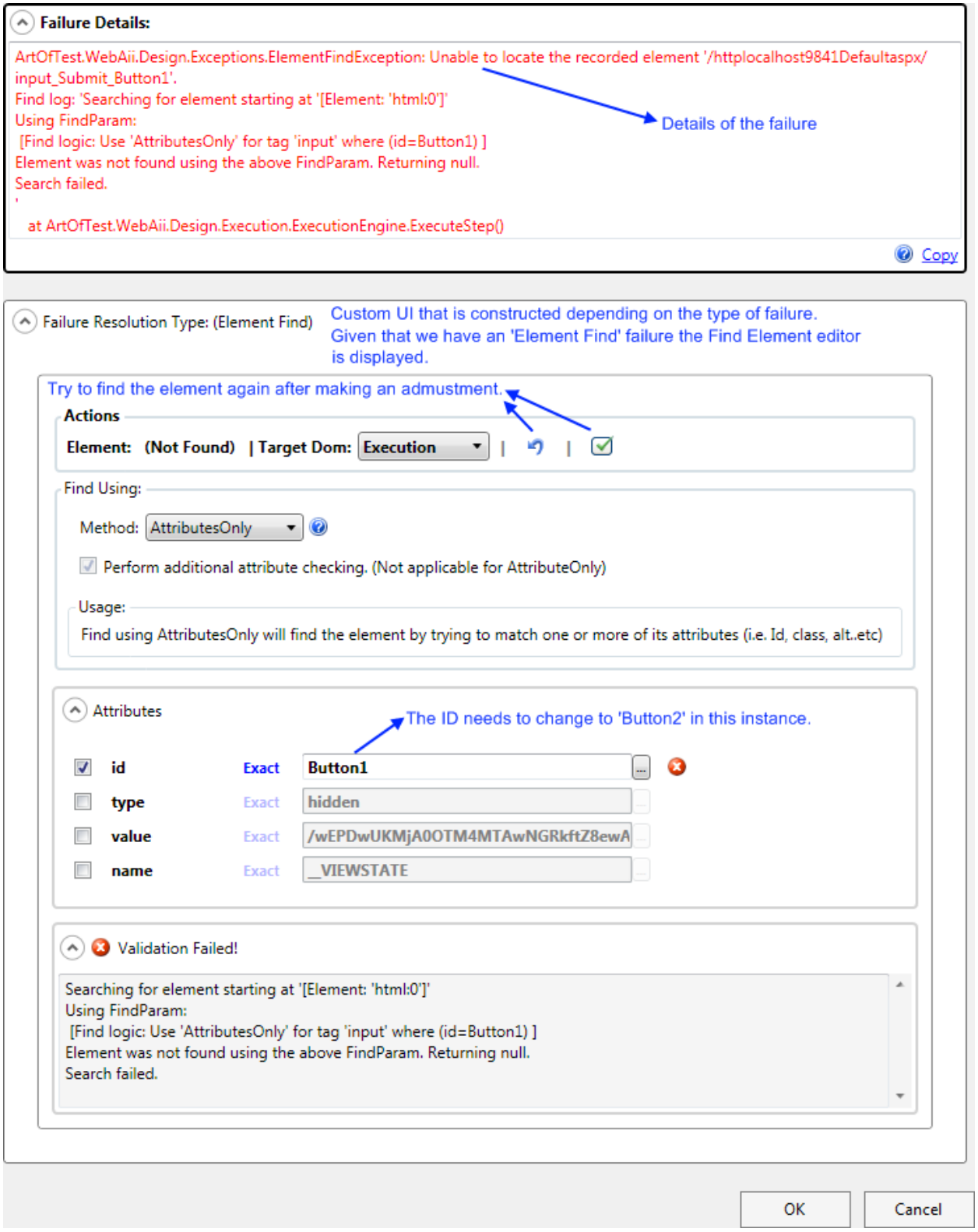


Now before moving on. Click on the Capture button to capture the states of this testcase (Section 5.15)

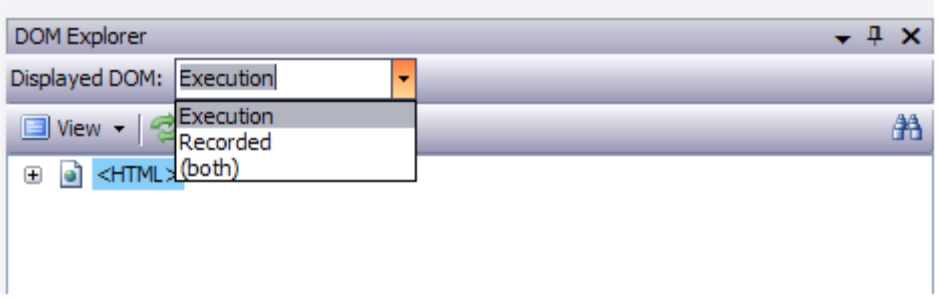
Now let’s assume the developer decides to change the button's ID from "Btn1" to "Btn2". When you run the test again an error occurs:



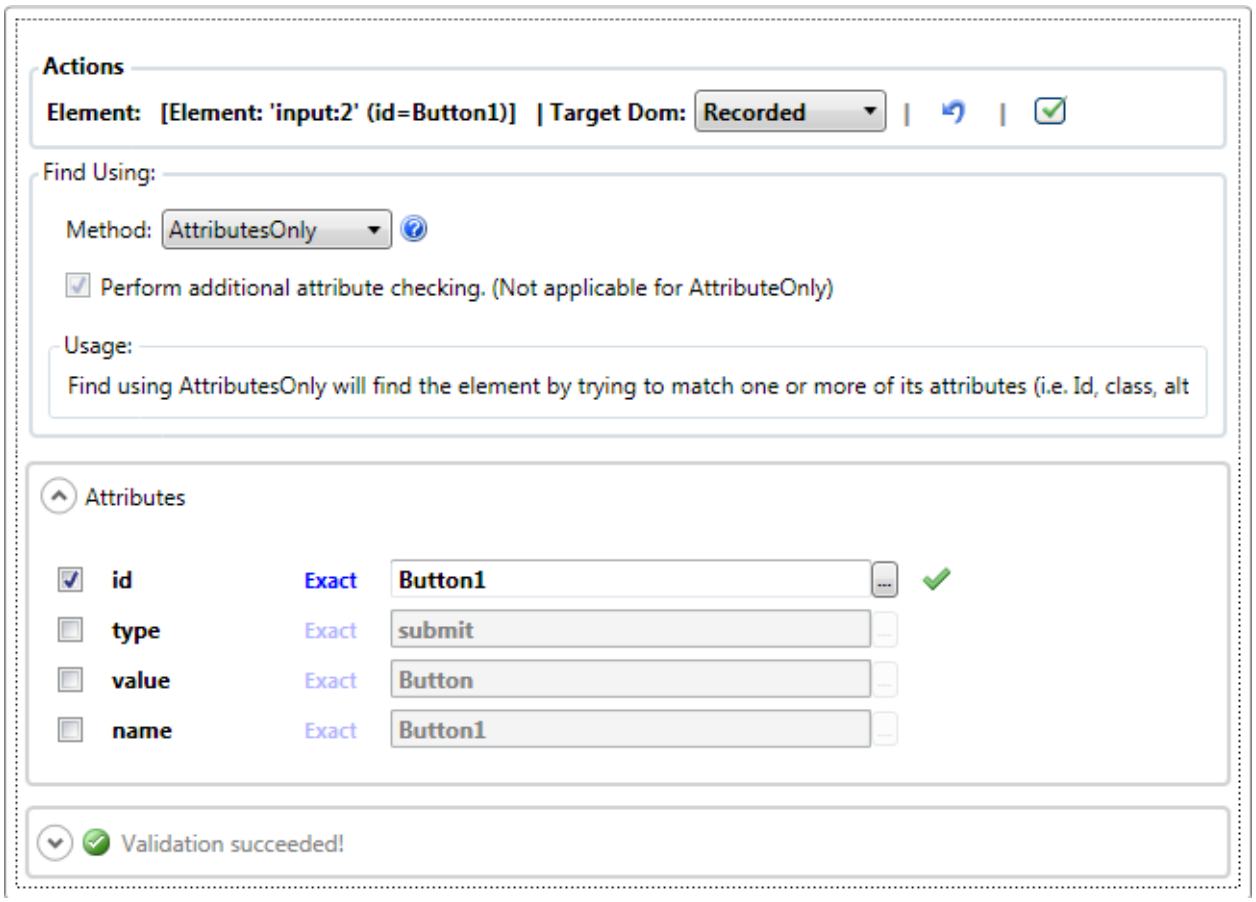
How does Design Canvas help you resolve this issue? To debug a failure in Design Canvas you simply double click on the red x right next to the failing step since Design Canvas supports a step-level debug capability. When you click on the red x, Design Canvas opens the following Debug screens:



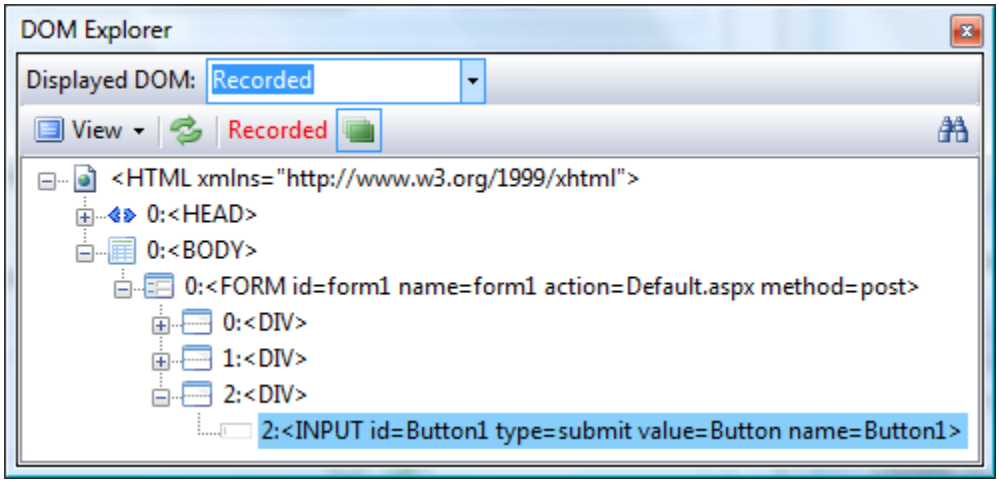
When you open DOM Explorer you will see that it has a drop-down similar to the one listed above in the ‘Actions’ section:



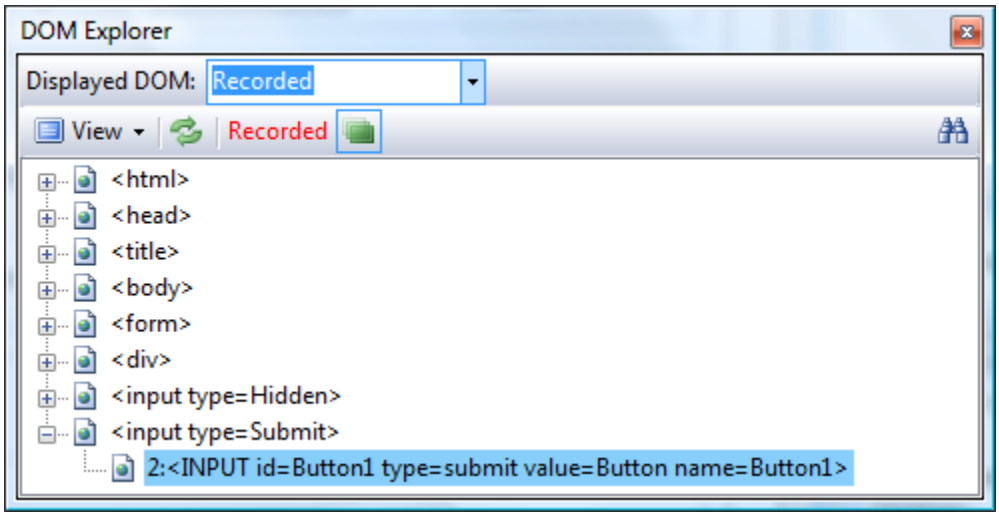
The first thing that comes to mind when debugging such a failure is, ‘What was the state of the DOM when I recorded my test?’ and then, ‘Why can't my test find the "Button1" ID?’ [As a tester, you might not be aware that this ID has changed]. If you switch the Displayed DOM to ‘Recorded’ and then click on ‘Validate’, see if you can still find the element in the DOM when you recorded it:



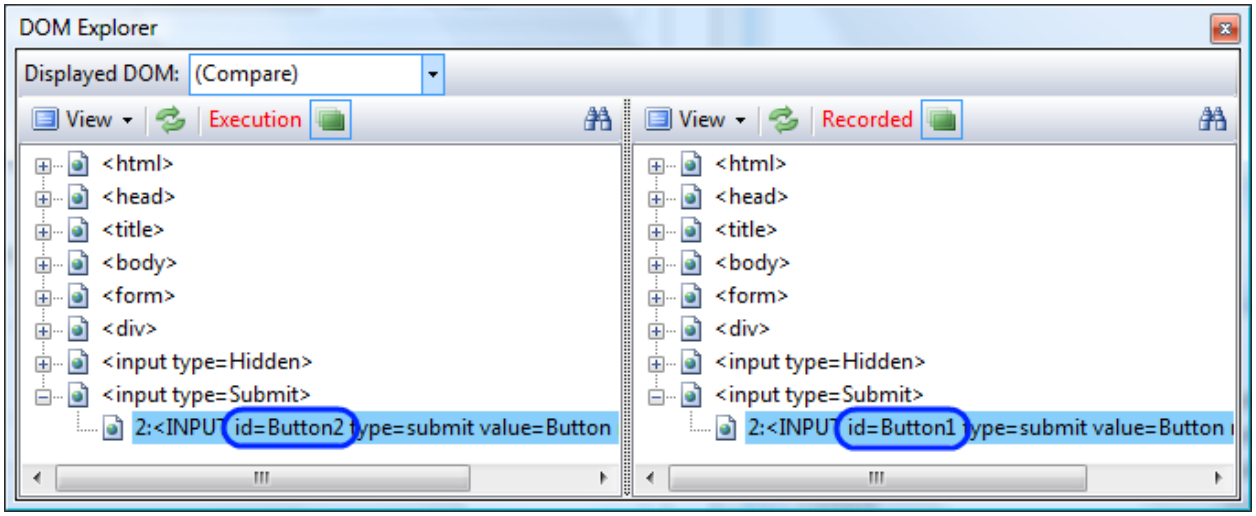
As you can see, the Design Canvas was able to find this element against the DOM that was captured at the time of recording. If you now open DOM Explorer and switch the Displayed DOM to 'Recorded', you will see that element highlighted in the tree:



We now at least know that the FindParameter used to identify this element is correct and that the failure must be due to a change in the page, not a problem in how we identify it. The next question is "What changed and what the new value for the change is?" To resolve this switch DOM Explorer view from 'Recorded' to 'TagName' so that you get a flat list of each tag with its index that looks like this:



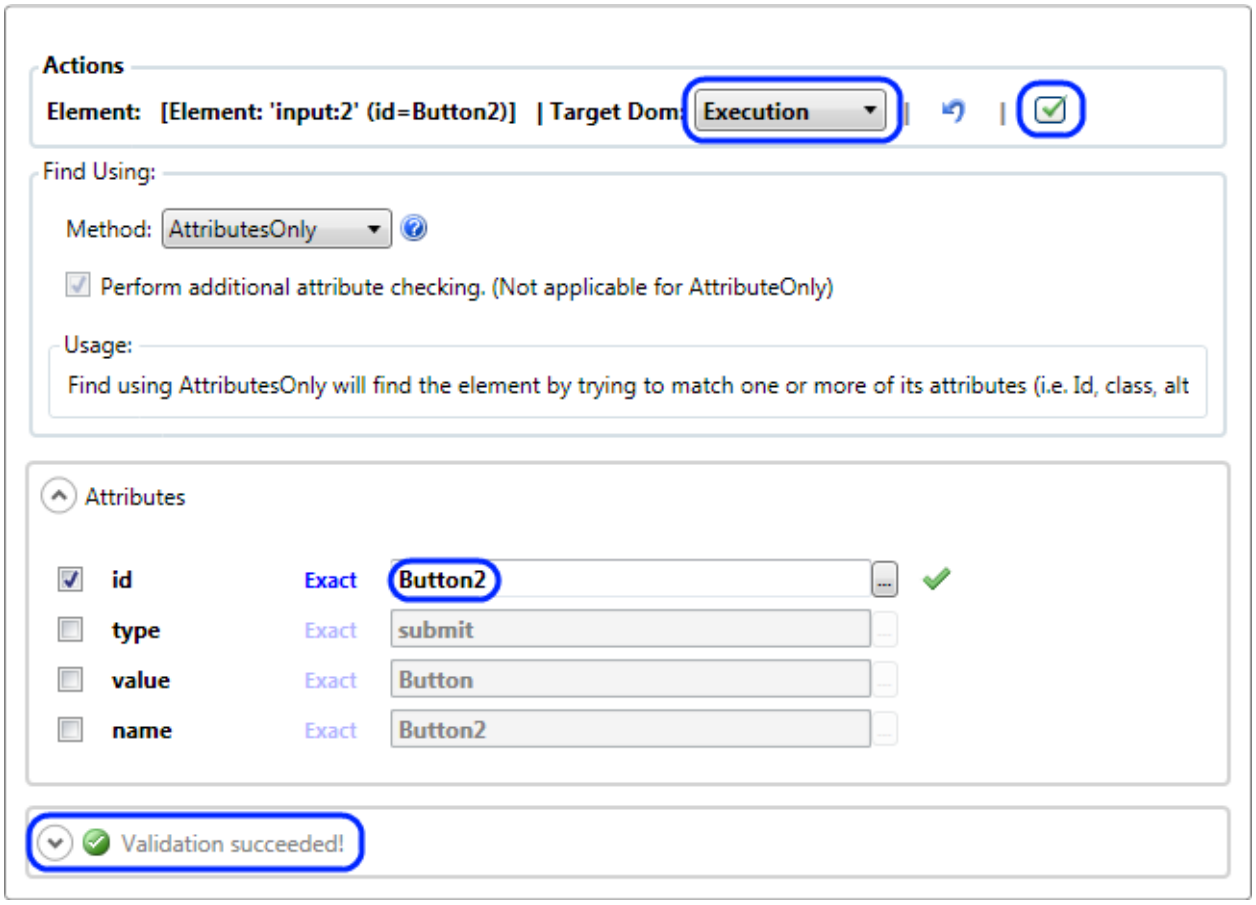
Next we compare this element with the actual ‘Execution’ DOM that the test failed against. To do so, simply switch the "Displayed DOM" to (both). Also switch the "Execution" view to ‘TagName’. DOM Explorer then displays both the recorded and executed DOM side by side as shown here:



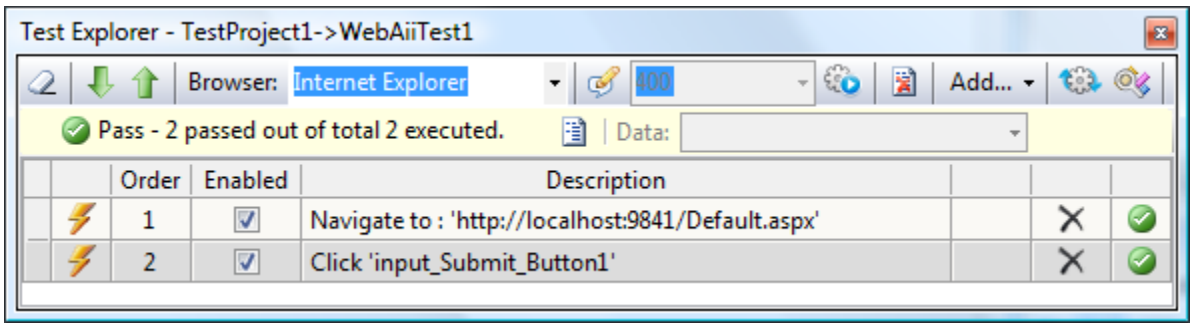
You can now easily navigate in the Execution DOM to the INPUT tag with index "2".
Now you can see that this element's id is ‘Button2’ not ‘Button1’. For the next step you have one of two choices:

- Report this as a bug and wait to see how the developers want to proceed.
- Acknowledge this as an acceptable change and simply update your automated test.

To update your automated test, simply go to the Element Editor UI and change the ID attribute from ‘Button1’ to ‘Button2’. Once you make that change don’t forget to validate it against the execution DOM. Simply select ‘Execution’ for the Target DOM in the editor and click on the ‘Validate’ button. The element should validate correctly. Now you are good to go.



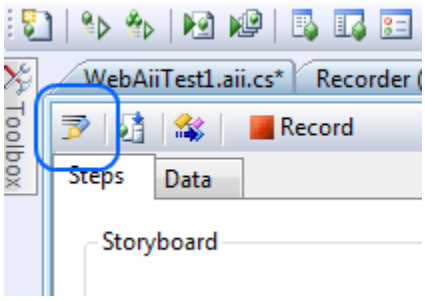
Click OK on the debug UI and you should be able to run the test again and it should pass.



Note: As soon as you click OK, the old recorded DOM is discarded and replaced by the Execution DOM since this now is the DOM that the element was last validated against successfully.

11 Test Customization: Adding a Code Behind File

Automation Design Canvas supports a code-behind model for recorded tests. In this model you can programmatically customize specific steps in a test using coded functions (VB.NET or C#). These coded functions are automatically detected and added to Test Explorer as test steps. You can manage these steps just like other recorded steps (for example, re-order them, enable them… etc.). Each WebAii test can have its own code-behind file. Click this button to add a code-behind file to this test:



Detailed information on how to use code-behind files and methods is contained in section 9.2.1.

11.1 Code Behind Methods

Automation Design Canvas supports a code behind model for recorded tests. In this model users can customize specific steps within a test using coded functions (in either VB.NET or C#). These coded functions are automatically detected and added as test steps in Test Explorer. You manage these steps just like the rest of your recorded steps (you can re-order them, enable them… etc.).

There are two approaches to creating code behind methods:

- You can convert an existing recorded step into a code behind method. Section 5.8.7.1 describes how to do this.
- You can manually add a code behind file and then manually insert your own code behind methods to this file. Section 9.2 describes how to manually add a code behind file.

11.2 CodedStep Attribute

Automation Design Canvas recognizes code behind methods by looking for the CodedStep attribute. Here's an example of a complete code behind method with its CodedStep attribute underlined and highlighted in yellow:

```
public class WebAiiTest1 : BaseWebAiiTest
{

    [CodedStep(@"Wait for 'TextContent' 'Contains' '$0.00' on
'span_MainConten__Totalprice'")]
    public void WebAiiTest1_CodedStep2()
    {
```

```
        // Wait for 'TextContent' 'Contains' '$0.00' on
'span_MainConten__Totalprice'
        HtmlSpan span_MainConten__Totalprice =
Pages.ArtOfTest__Automated_Software_Testing_Made_Easy_2.span_MainConten__Tota
lprice;
        span_MainConten__Totalprice.Wait.ForExists(10000);
        Wait.For<HtmlSpan>(c =>
c.AssertContent().TextContent(ArtOfTest.WebAii.Controls.HtmlControls.HtmlAsse
rts.StringCompareType.Contains, "$0.00"), span_MainConten__Totalprice, 10000,
500, this);

    }
}
```

The text passed in as a parameter to the CodedStep attribute represents the description for that code behind method. It can be any string you like.

With the above code added to a code behind file a test step will appear in Test Explorer like this:

Notice how the name of the code behind method appears in brackets and the description from the CodedStep attribute also appears in the description for the test step.

11.3 How to Reference Elements Contained in Elements Explorer

Automation Design Canvas makes referencing elements kept in Elements Explorer by code behind methods easy. Simply use the syntax:

```
Pages.<web page name string>.<element name string>
```

Use the friendly name of the web page as displayed in Elements Explorer as the element within the Pages object. Use the friendly name of the element as displayed in Elements Explorer as element to the web page object. This returns a strongly typed WebAii HTML object. Once you have you this object you can do all the normal things possible using the WebAii framework as described in: http://www.artoftest.com/Resources/WebAii/Documentation/topicsindex.aspx.

11.4 How to Reference Data from the Data Table

Whether you're referencing data from the internal data grid or an external data source the process is exactly the same. To reference data from either your internal data grid or an external data source use the "Data" property followed by the index of the data column. For example:

```
[CodedStep(@"Set 'input_Text_Q' text to 'webaii' - DataDriven: [$(col1)]")]
public void WebAiiTest2_CodedStep()
{
    // Set 'input_Text_Q' text to 'webaii'
    HtmlInputText input_Text_Q = Pages.Google.input_Text_Q;
    input_Text_Q.Wait.ForExists(10000);
```

```
    // You can reference the column by index
    input_Text_Q.Text = ((string)(Data[0]));
    // or by name
    input_Text_Q.Text = ((string)(Data["col1"]));
}
```

Where the data property code is shown underlined and highlighted in yellow. Section 8 describes how to create a data array or connect to an external data source.

12 Using VS Web Tests and Generating VS Load Tests

12.1 Converting WebAii Tests to VS Web Tests

Load tests can be auto-generated on demand. With one click of a button a VS WebTest is generated from your recorded WebAii test which in turn can be used for load testing. All your WebAii tests can easily be used as the base for your load tests. Section 12.1.2 explains how to do this.

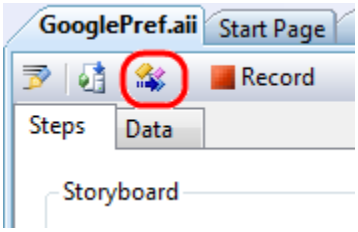
It is important to remember that once generated the new VS Web Test becomes its own independent test in your test project. Any changes to the Web Test or WebAii test are not automatically carried over from one to the other or back. The best practice is to use your WebAii test as the master source, make any required changes to it, then regenerate a VS Web Test when finished. By leveraging their ease of update and the element explorer abstraction you save significant time as compared to having to maintain more cumbersome VS Web Tests.

12.2 Generating Unit Tests

Automation Design Canvas enables you to take any of your recorded tests and generate coded unit tests from them. It supports generating fully self contained NUnit tests as well as fully self contained VS Unit tests. These tests can be moved into other projects or run in isolation of your current test project.

To generate a self contained Unit Test follow these steps:

- Double click on the WebAii test in Solution Explorer that you want to generate a unit test from. This opens the test document window.
- Click the Generate a Unit Test icon.



- If the WebAii test contains a data array you will be prompted for the name of an XML file to save the data into. The generated unit test will read from this file during execution.

This takes the existing WebAii test and generates a new unit test code file and adds it to the project. The original WebAii test is left intact.

It is important to remember that once generated the new unit test becomes its own independent test in your test project. Any changes to the unit test or the WebAii test are not automatically carried over from one to the other or back. The best practice is to use your WebAii test as the master source, make any required changes to it, then regenerate a unit test when finished. By

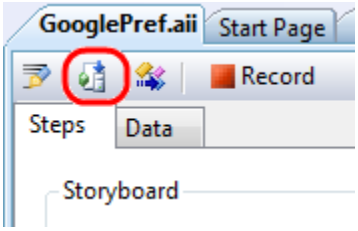
leveraging their ease of update and the element explorer abstraction you save significant time as compared to having to maintain more cumbersome unit tests.

12.3 Generating Visual Studio Web Tests

With one click of a button a VS WebTest can be generated out of your recorded WebAii test which can then be used in load testing. All your WebAii tests can easily be used as the base for your load tests. By leveraging their ease of update and the element explorer abstraction you save significant time compared to having to maintain more cumbersome VS Web Tests.

To generate self-contained VS WebTest follow these steps:

- Double click on the WebAii test in Solution Explorer that you want to generate a unit test from. This opens the test document window.
- Click the **Convert test to VS WebTest** icon. An Internet Explorer window will open and start playing back the WebAii test while recording – it as a VS WebTest at the same time.



When the recording is complete the Internet Explorer window will automatically close. Note that a new VS WebTest has been added to your project. The original WebAii test is left intact.

It is important to remember that once generated the new VS Web Test becomes its own independent test in your test project. Any changes to the Web Test or the WebAii test are not automatically carried over from one to the other or back. The best practice is to use your WebAii test as the master source, make any required changes to it, then regenerate a VS Web Test when finished. By leveraging their ease of update and the element explorer abstraction you save significant time as compared to having to maintain more cumbersome VS Web Tests.

12.4 Debugging Your Test Using VS Debugger

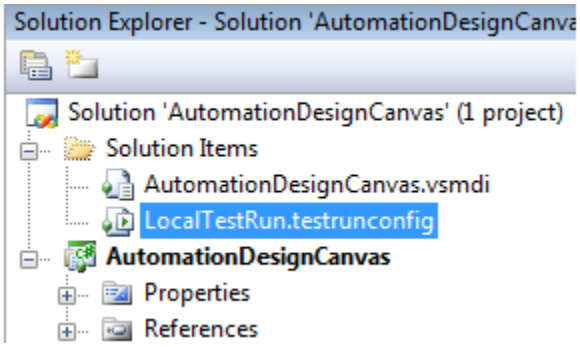
You can debug your code behind methods the same way you debug any other coded unit test with Visual Studio. You can set breakpoints in your code behind methods, and then execute them in debug mode. When execution reaches the breakpoint, the WebAii test will stop on that line.

From there you can inspect the values contained in variables, single step, etc. The full power and flexibility of the Visual Studio debugger is at your disposal.

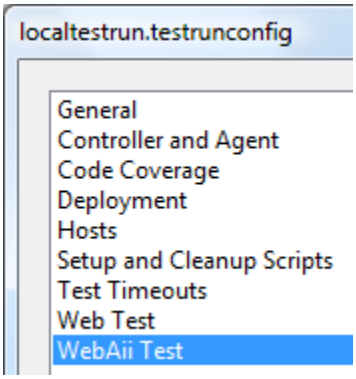
12.5 Configuring and Executing Your Tests with Visual Studio Test Run (Configuration Settings)

When WebAii tests run as unit tests within Visual Studio (instead of running via Quick Execute) they use the WebAii configuration that is kept in the test run configuration file for the project. To get to these configuration settings follow these steps:

- Open the Solution Explorer window.
- Locate the test run configuration file for your project. The default is named LocalTestRun.testrunconfig.



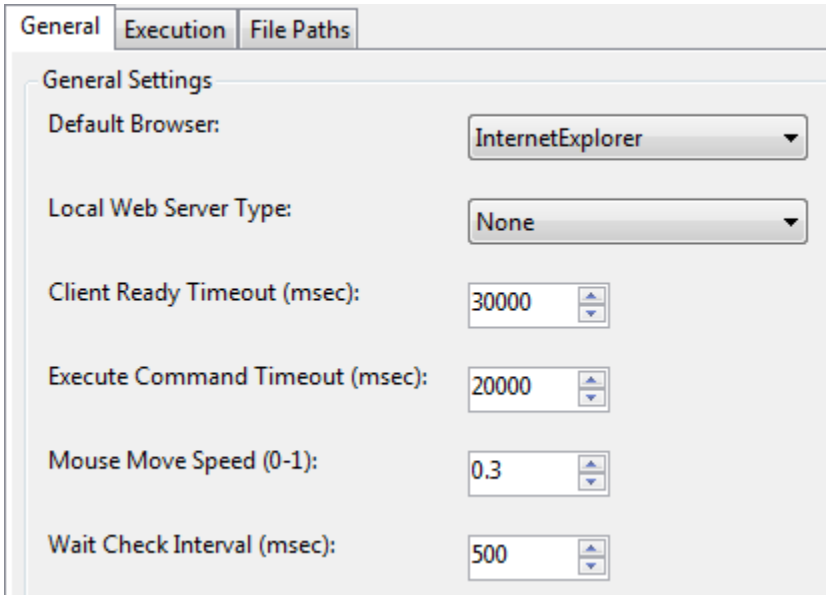
- Double click your test run configuration file. This will open the test run configuration dialog.
- Click **WebAii Test** which appears in the configuration sections list on the left.



There are three tabs containing WebAii configuration settings. Each tab is explained in the following sections. Additional information about configuring test execution in Visual Studio can be found at:
http://msdn.microsoft.com/en-us/library/ms182479.aspx

12.6 General settings tab

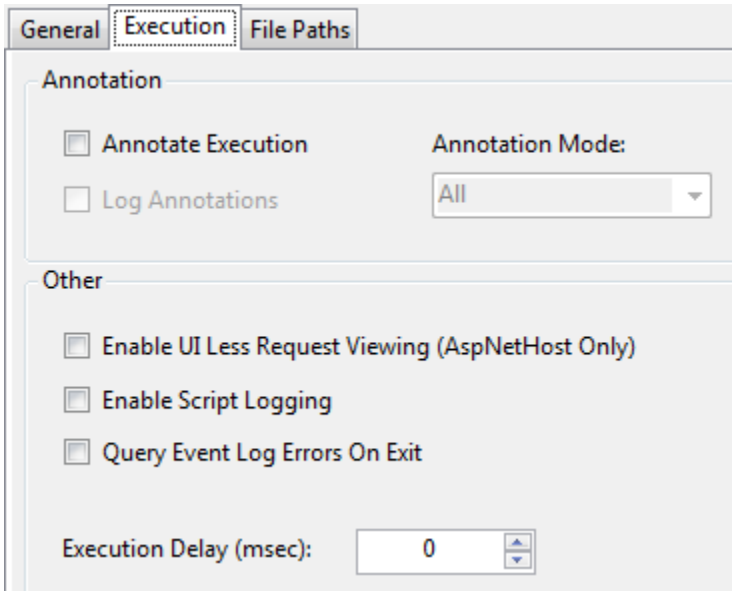
The General tab holds configuration settings that are used in most WebAii testing environments.



- Default Browser – Controls which web browser will be used by the recorded steps. You can override this setting in code behind methods.
- Local Web Server Type – Controls whether or not to use the local ASP.NET development server. When set to AspNetDevelopmentServer WebAii tests will connect to the local ASP.NET development server. Be sure to set Web App Physical Path on the File Paths tab to the full physical path of the root of your ASP.NET development server. When set to None WebAii tests will expect the browser to connect to external web servers.
- Client Ready Timeout – Timeout (in milliseconds) used to wait for a client (a browser) to be ready after it is first launched or after executing a command. For example after launching the browser if the browser is not ready in this timeout the framework will throw a **TimeoutException**.
- Execute Command Timeout – The amount of time to wait for a response to be received from the browser (in milliseconds) after sending it a command request. For example, if there is no response from the browser within this amount of time after sending it a click request, the framework will throw a **TimeoutException**.
- Mouse move Speed – Sets the simulated mouse move speed for mouse and DragDrop operations in pixels/millisecond. Typical values are between 0.1f - 0.5f.
- Wait Check Interval – Controls the rate that the condition is tested by “Wait For” steps. This value specifies the number of milliseconds to wait between tests of the condition.

12.6.1 Execution tab

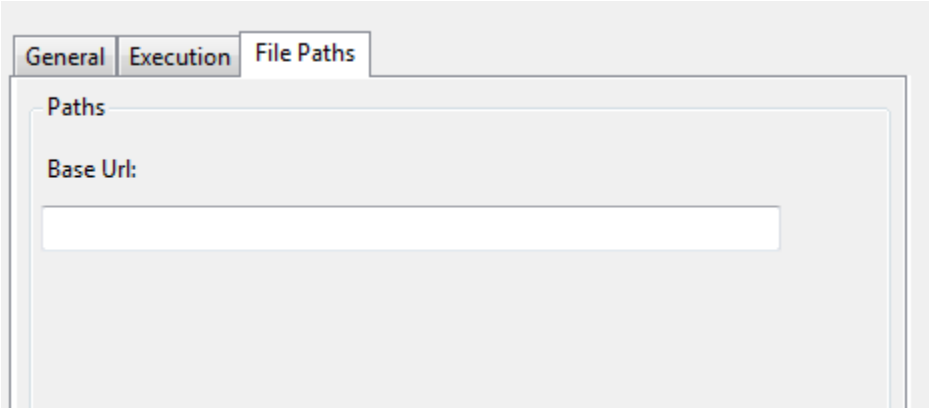
The Execution tab holds configuration settings that control how WebAii tests are executed.



- Annotate Execution – Controls whether or not annotation is turned on during execution. The annotator is described in sections 5.8.4 and 0201250847 \r \h * MERGEFORMAT 7.5 and 0.
- Log Annotations – Controls whether or not to write annotations to the WebAii test log file.
- Annotation Mode – Controls what types of annotation will be displayed in the browser. You can select from All, Native Only – only WebAii automatic annotations will be displayed, Custom Only – only Custom Annotation steps you added to your test will be displayed.
- Enable UI Less Request viewing – Controls whether or not to allow debugging of UI-Less page requests using a UI browser like Internet Explorer. This setting only applies when Default Browser is set to AspNetHost, which is a headless browser (that is, a browser type that has no actual visible UI).
- Enable Script Logging – Globally enables or disables logging of JavaScript execution.
- Query Event Log Errors – Controls whether or not Automation Design Canvas will query the Windows Application Event log on exit for any error logged from automation clients. Any errors will be appended to the test log.
- Execution Delay – Sets the amount of execution delay (in milliseconds) between commands. This can help in observing test execution and sometimes helps troubleshoot automation timing issues.

12.6.2 File Paths tab

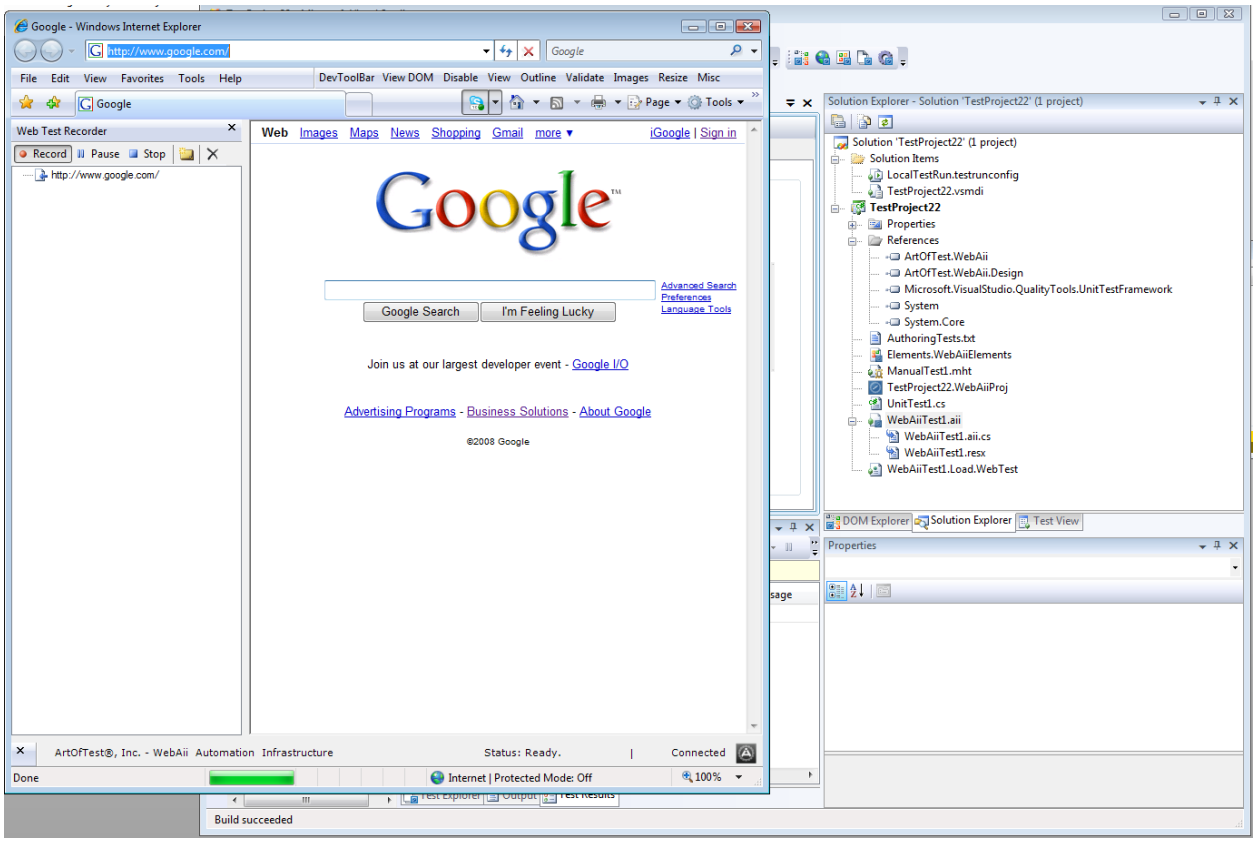
The File Paths tab holds WebAii test URL and physical file location configuration settings.



- Base Url – Sets the base URL to use for all Navigate To steps. When set your Navigate To steps should use a relative URL (for example,"~/default.aspx").

12.7 Generating Load Tests from WebAii Tests

Load tests can be auto-generated on demand! With one click of a button a VS WebTest is generated from your recorded WebAii test which can then be used in load testing. This is yet another time saver! All your WebAii tests can easily be used as the base for your load tests. By leveraging their ease of update and the Elements Explorer abstraction you save significant time compared to having to maintain more cumbersome VS WebTests. The following window shows how WebAii tests can be incorporated into VS load tests as with other VS test types.



A Test Step Properties Reference

A.1 Properties common to all steps

A.1.1 ClosesBrowser

When set to True it tells Design Canvas that this click step will force the browser to close (that is, it calls window.close). This is typical in HTML pop-ups. Design Canvas will watch for the browser window to close and handle it appropriately.

A.1.2 sPause

Controls whether or not the test will pause at this step. The available options are:

- None – Does not pause the test at this step.
- Before – Pauses the test just before this step executes.
- After – Pauses the test just after this step executes.

Note: This property is only used by Quick Execute. Running the test via Test View or Test List Editor does not use this property.

A.1.3 RunsAgainst

Defines which browser(s) this step will run in when the test runs. The available options are:

- AllBrowsers – The step will run regardless of which browser the test is executing in.
- InternetExplorerOnly – The step will only run when the test is running in an Internet Explorer browser window.
- FirefoxOnly – The step will only run when the test is running in a Firefox browser window.

A.1.4 WaitOnElements

When set to True, Automation Design Canvas will wait on the steps elements to exist on the web page before executing the step. If ValidatePageUrl is also set to True for the steps elements Design Canvas will wait for the pages CompareUrl to match before executing the step.

When set to False, Design Canvas assumes the steps elements already exist on the web page and will immediately execute the test step.

A.1.5 WaitOnElementsTimeout

When WaitOnElements is set to true this setting controls the amount of time, in milliseconds, that Design Canvas will wait for the steps elements to exist before executing that step.

A.2 NavigateTo step

A.2.1 NavigateUrl

Specifies the URL to navigate to. If BaseURL is set in the testrunconfig file this value needs to be a relative URL.

A.3 Click step

A.3.1 SimulateRealClick

When set to True the click is performed using the mouse rather than directly invoking a click event on the DOM element.

A.4 Set text step

A.4.1 IsPassword

When set to True it informs A Design Canvas that this text field is a password field. The string entered into the text field will be masked and Design Canvas will mask the data as it enters into the input field during execution.

A.4.2 SimulateRealTyping

When set to True Design Canvas will simulate real text typing one character at a time with a speed set by TypingSpeed.

A.4.3 Text

The text to be entered in the input control.

A.4.4 TypingSpeed

The typing speed, in milliseconds, to use when SimulateRealTyping is set to true. It is only enabled when SimulateRealTyping is set to true, otherwise the value is not used.

A.5 Verify Content step

A.5.1 CompareType

Selects which compare type is to be performed from the following options:

- Exact – Verify the attribute value matches exactly with the expected value.
- Contains – Verify the attribute value contains the expected value. Used mostly for those attributes having string values rather than numeric values.
- NotContain – Verify the attribute value does not contain the expected value. Used mostly for those attributes having string values rather than numeric values.
- StartsWith – Verify the attribute value starts with the expected value. Used mostly for those attributes having string values rather than numeric values.
- EndsWith – Verify the attribute value starts with the expected value. Used mostly for those attributes having string values rather than numeric values.
- RegEx – Verify the attribute value matches a regular expression. WebAii takes advantage of the .NET Framework Regular Expressions as detailed here: http://msdn.microsoft.com/en-us/library/hs600312.aspx. The string you enter as the expected value is used as the expression to use in the match comparison.

A.5.2 ExpectedString

The expected string to compare against.

A.5.3 TagSegmentType

Select what type of content to verify. This can be one of the following options:

- InnerText – Compares the inner text of the tag. e.g., “Data Display”. Be careful because InnerText is recursive. It includes not only the text of the current element but all the text for all children elements.
- InnerMarkup – The inner markup of a tag. For example,“ ”. The same care must be taken for this type of compare as what was just described for InnerText.
- OuterMarkup – The outer markup of a tag. For example,“ Some Data <input attr1="Button1" type="button" value="Click Me" onclick="clicked () ;"/>”.The same care must be taken for this type of compare as what was just described for InnerText.
- TextContent – Text content of the tag only without all its children. For example, “Some Data”. This type of compare is less risky than the above methods because it is non-recursive. The above methods are all recursive i.e., include their child elements.
- StartTagContent – The raw start tag content.

A.6 Verify Attributes

A.6.1 AttributeName

The name of the attribute associated with the element to be verified.

A.6.2 AttributeValue

The value of the attribute to compare against.

A.6.3 CompareType

Select which compare type to be performed. You can select from:

- Exact – Verify the attribute value matches exactly with the expected value.
- Contains – Verify the attribute value contains the expected value. Used mostly for those attributes having string values rather than numeric values.
- NotContain – Verify the attribute value does *not* contain the expected value. Used mostly for those attributes having string values rather than numeric values.
- StartsWith – Verify the attribute value starts with the expected value. Used mostly for those attributes having string values rather than numeric values.
- EndsWith – Verify the attribute value starts with the expected value. Used mostly for those attributes having string values rather than numeric values.
- RegEx – Verify the attribute value matches a regular expression. WebAii takes advantage of the .NET Framework Regular Expressions as detailed here: http://msdn.microsoft.com/en-us/library/hs600312.aspx. The string you enter as the expected value is used as the expression to use in the match comparison.

A.7 Verify Style

A.7.1 CompareType

Select which compare type to be performed. You can select from:

- Exact – Verify the attribute value matches exactly with the expected value.
- Contains – Verify the attribute value contains the expected value. Used mostly for those attributes having string values rather than numeric values.
- NotContain – Verify the attribute value does *not* contain the expected value. Used mostly for those attributes having string values rather than numeric values.
- StartsWith – Verify the attribute value starts with the expected value. Used mostly for those attributes having string values rather than numeric values.
- EndsWith – Verify the attribute value starts with the expected value. Used mostly for those attributes having string values rather than numeric values.
- RegEx – Verify the attribute value matches a regular expression. WebAii takes advantage of the .NET Framework Regular Expressions as detailed here: http://msdn.microsoft.com/en-us/library/hs600312.aspx. The string you enter as the expected value is used as the expression to use in the match comparison.

A.8 StyleDescriptor

The style descriptor on the element to be verified.

A.8.1 Value

The value of the style to compare against.

A.9 Verify IsVisible

A.9.1 IsVisible

The visibility value to compare against (True or False).

A.10 Wait for step

All four verify step types can be changed into a Wait For step. When set as a Wait For they have these additional properties:

A.10.1 CheckInterval

How frequently the element will be checked to see if it is in the indicated state. The value is in milliseconds.

A.10.2 SupportsWait

Indicates whether or not this verification type supports the Wait For mode. This property is read-only, it cannot be changed.

A.10.3 Timeout

Specifies the amount of time, in milliseconds, that Automation Design Canvas will wait before timing out and setting that step as failed.

A.11 Set checkbox/radio button step

A.11.1 CheckedState

The checked state to set the element to (True or False).

A.11.2 InvokeOnClick

When set to True Automation Design Canvas will invoke the OnClick JavaScript event on the element.

A.12 Select dropdown step

A.12.1 SelectDropDownType

The type of comparison to perform. You can select from:

- ByIndex – Verifies which dropdown index is currently selected. Note: the index value is zero based. Thus if the first item in the dropdown list is selected the value is 0.
- ByValue – Verifies which selection value is currently selected. Remember the list of possible values come from the <option> elements that are children of the <select> element, not what is displayed in the browser window. For example, <option value=4>.
- ByText – Verifies the text that is displayed for the current selection.

A.12.2 SelectionIndex

The selection index to select. Only used when SelectDropDownType is set to ByIndex.

A.12.3 SelectionText

The selection text to select. Only used when SelectDropDownType is set to ByValue or ByText.

A.12.4 InvokeSelectionChanged

When set to True Automation Design Canvas will invoke the OnChange JavaScript event on the element.

A.13 Invoke Event step

A.13.1 EventType

Specifies which JavaScript event to invoke. Automation Design Canvas supports invoking the following JavaScript events:

- OnBlur
- OnChange
- OnClick
- OnDblClick
- OnFocus
- OnKeyDown
- OnKeyPress
- OnKeyUp
- OnLoad

- OnMouseDown
- OnMouseMove
- OnMouseOut
- OnMouseOver
- OnMouseUp
- OnReset
- OnResize
- OnSelect
- OnSubmit
- OnUnload

A.14 Desktop action step

A.14.1 DesktopCommandType

Specifies the desktop actions to perform on the element. These actions include:

- Left click – Moves the mouse cursor to a point relative to the element and simulates a single left mouse button click. The point must be somewhere within the rectangular bounds of the element.
- Left double click – Moves the mouse cursor to a point relative to the element and simulates a double left mouse button click.
- Left mouse down – Moves the mouse cursor to a point relative to the element and simulates a left mouse button down event.
- Left mouse up – Moves the mouse cursor to a point relative to the element and simulates a left mouse button up event.
- Right click– Moves the mouse cursor to a point relative to the element and simulates a single right mouse button click.
- Right mouse down– Moves the mouse cursor to a point relative to the element and simulates a right mouse button down event.
- Right mouse up– Moves the mouse cursor to a point relative to the element and simulates a right mouse button up event.
- Drag drop – Performs a drag & drop operation. The starting point is always relative to the current element. The ending point can be either a point relative to the same element or a different destination element or a browser window coordinate.
- Mouse Hover Over – Moves the mouse to a point relative to the element and leaves it there.
- Scroll Into View – Scrolls the browser window to make the element visible. This is a necessary or precautionary step to perform before acting on the element or perhaps before taking a screen shot.

A.14.2 Focus

When set to True Automation Design Canvas will scroll the element into view before performing the specified action on it.

A.14.3 Offset Group

A.14.4 ClickUnitType

Specify either:

- Pixel – The offset values are considered to be pixels.
- Percentage – The offset values are considered to be a percentage of the total size of the element.

A.14.4.1 OffsetReference

Specifies the starting point to calculate the offset from. Specify one of the following:

- TopLeftCorner
- BottomLeftCorner
- TopRightCorner
- BottomRightCorner
- TopCenter
- LeftCenter
- RightCenter
- BottomCenter
- AbsoluteCenter

A.14.4.2 X

Specifies the X offset value.

A.14.4.3 Y

Specifies the Y offset value.

A.15 Scroll into view step

A.15.1 ScrollToVisibleType

Specifies which type of scroll to perform. Select one of:

- ElementTopAtWindowTop
- ElementBottomAtWindowBottom

A.16 Capture snapshot step

A.16.1 CaptureType

Specifies the type of window to take a snapshot of. Specify either:

- Browser
- Desktop

A.16.2 FileNamePrefix

The filename prefix to use to when saving the snapshot to disk.

A.17 Annotation step

A.17.1 AnnotationText

The text to be displayed in the annotation.

A.17.2 DisplayLocation

The location to display the annotation at. You can choose from:

- TopLeftCorner
- BottomLeftCorner
- TopRightCorner
- BottomRightCorner
- TopCenter
- LeftCenter
- RightCenter
- BottomCenter
- AbsoluteCenter

A.17.3 DisplayTime

Specifies the length of time, in milliseconds, to display the annotation.

A.18 Alert handler dialog step

This step handles the standard Java alert popup dialog.

A.18.1 HandleButton

Specifies which button to click to close the dialog. Specify one of:

- OK
- Cancel
- Open
- Yes
- No
- Close
- Run
- Save
- NotSet

A.18.2 HandleTimeout

Specifies the amount of time, in milliseconds, that Automation Design Canvas will wait for the dialog to close after being handled.

A.19 File upload dialog step

This step handles the browsers standard file upload popup dialog box. After adding the dialog handler step to Test Explorer you need to modify the step properties to insert the full path to the file to be entered in the file upload dialog.

A.19.1 HandleButton

Specifies which button to click to close the dialog. Specify one of:

- OK
- Cancel
- Open
- Yes
- No
- Close
- Run
- Save
- NotSet

A.19.2 HandleTimeout

Specifies the amount of time, in milliseconds, that Automation Design Canvas will wait for the dialog to close after being handled.

A.19.3 FileUploadPath

The path and filename to enter into the upload dialog.

A.20 Logon dialog step

This step handles the browsers standard logon popup dialog. After adding the dialog handler step to Test Explorer you need to modify the step properties to provide the username and password to be entered into the dialog.

A.20.1 HandleButton

Specifies which button to click to close the dialog. Specify one of:

- OK
- Cancel
- Open
- Yes
- No
- Close
- Run
- Save
- NotSet

A.20.2 HandleTimeout

Specifies the amount of time, in milliseconds, that Automation Design Canvas will wait for the dialog to close after being handled.

A.20.3 UserName

The user name to enter into the logon dialog.

A.20.4 Password

The password to enter into the logon dialog.

A.21 Download dialog step

This step handles the browser's standard file download popup dialog box. After adding the dialog handler step to Test Explorer you must modify the step properties to insert the full path of the location to store the downloaded file. This path will be entered into the file download dialog.

A.21.1 HandleButton

Specifies which button to click to close the dialog. Specify one of:

- OK
- Cancel
- Open
- Yes
- No
- Close
- Run
- Save
- NotSet

A.21.2 HandleTimeout

Specifies the amount of time, in milliseconds, that Automation Design Canvas will wait for the dialog to close after being handled.

A.21.3 DownloadPath

The download location to enter into the down dialog.

A.22 Generic dialog step

This is a special dialog handler with multiple properties that will enable you to create your own automatic popup dialog handler to handle most simple Win32 popup dialogs. After adding the dialog handler step to Test Explorer you must modify the following step properties to suit your needs:

A.22.1 HandleTimeout

Specifies the amount of time, in milliseconds, that Automation Design Canvas will wait for the dialog to close after being handled.

A.22.2 Dialog Title

The dialog title to look for to identify this dialog from all other windows currently open.

A.22.3 MatchPartialTitle

When set to True Automation Design Canvas will use partial text matching when looking at dialog titles.

A.22.4 ChidWindowTextContent

A partial text to look for within the dialog window. This can help isolate the correct dialog from other dialogs that might have similar dialog titles.

A.22.5 HandleButtonMethod

Specifies how to handle the dialog. Specify one of:

- NoneCloseDialog – will send the dialog a close command to close it.
- ButtonId – Will click on the button having the correct button Id to close it.
- ButtonPartialText – Will click on the button containing the correct text to close it.

A.22.6 ButtonId

The ID of the button to click on. Only used when HandleButtonMethod is set to ButtonId.

A.22.7 ButtonPartialText

The button text contained on the button to click on. Only used when HandleButtonMethod is set to ButtonPartialText.

A.23 Drag Drop step

A.23.1 Focus

When set to True True Automation Design Canvas will scroll the drag element into view before performing the drag drop.

A.23.2 DragElement

Specifies which element to drag. Clicking on the … icon will open the Projects Elements Selector dialog showing all the elements contained in Elements Explorer. Pick the desired element and click OK.

A.23.3 Offset group

A.23.3.1 ClickUnitType

Specify either:

- Pixel – The offset values are considered to be pixels.

- Percentage – The offset values are considered to be a percentage of the total size of the element.

A.23.3.2 OffsetReference

Specifies the starting point to calculate the offset from. Specify one of the following:

- TopLeftCorner
- BottomLeftCorner
- TopRightCorner
- BottomRightCorner
- TopCenter
- LeftCenter
- RightCenter
- BottomCenter
- AbsoluteCenter

A.23.3.3 X

Specifies the X offset value.

A.23.3.4 Y

Specifies the Y offset value.

A.23.3.5 DropElement

Specifies which element to drop onto. Clicking on the … icon will open the Projects Elements Selector dialog showing all the elements contained in Elements Explorer. Pick the desired element and click OK.

A.23.3.6 DropOffset group

TBD

A.23.3.7 ClickUnitType

Specify either:

- Pixel – The offset values are considered to be pixels.
- Percentage – The offset values are considered to be a percentage of the total size of the element.

A.23.4 OffsetReference

Specifies the starting point to calculate the offset from. Specify one of the following:

- TopLeftCorner
- BottomLeftCorner
- TopRightCorner
- BottomRightCorner
- TopCenter

- LeftCenter
- RightCenter
- BottomCenter
- AbsoluteCenter

A.23.4.1 X

Specifies the X offset value.

A.23.4.2 Y

Specifies the Y offset value.

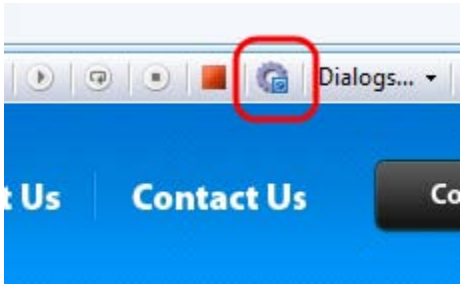
A.23.4.3 DroptargetType

Specifies the type of target to drop onto. Select either:

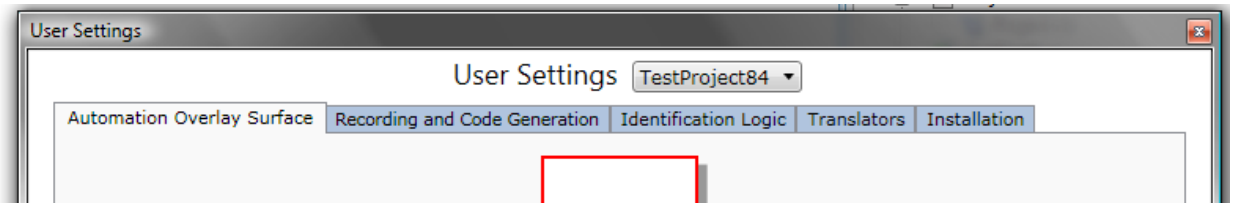
- Element – Drops onto the specified element using the specified drop offset values.
- Window – Ignores the specified drop element and uses the drop offset values to be values within the browser window.

B Automation Design Canvas Settings

You get to the Automation Design Canvas by clicking the Design Canvas Setting icon located in the toolbar of the Recording Surface window:

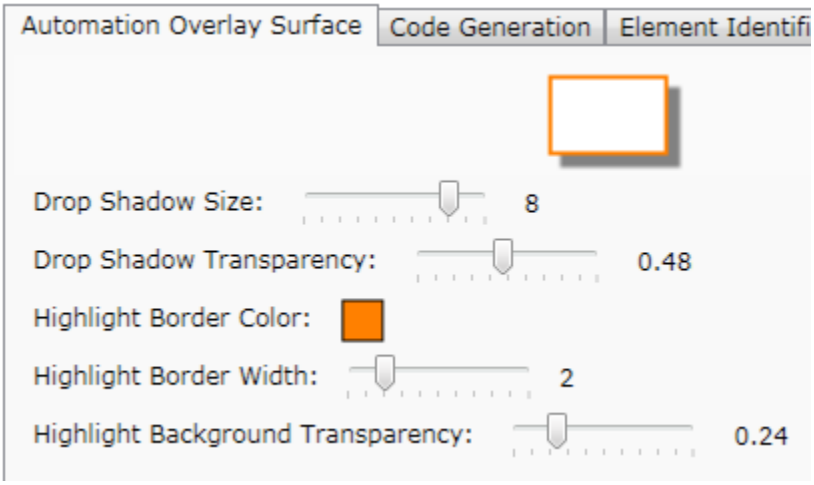


In this dialog you can change the WebAii project settings for all of the WebAii projects that are in your solution. Once the dialog opens you need to select which project you want to change the settings of. Select the project you are going to work with using the dropdown at the top of the dialog:

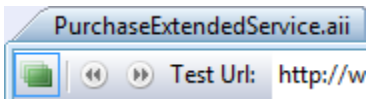


The rest of the sections in this appendix describe how to use each tab in the dialog.

B.1 Automation Overlay Surface tab



This tab controls all the visual display aspects of the overlay surface, that is, when this icon is clicked:



B.2 Drop Shadow Size

This controls the size, in pixels, of the drop shadow displayed around highlighted elements on the Recording Surface.

B.2.1 Drop Shadow Transparency

This controls the transparency level of the drop shadow displayed around highlighted elements on the Recording Surface. 0 means fully transparent. You won't even see the drop shadow. 1 means fully opaque. The drop shadow will be completely solid and you won't be able to see what the drop shadow covers up in the Recording Surface. Any value between 0 and 1 will show allow some amount of the UI underneath the drop shadow to show through.

B.2.2 Highlight Border Color

This color selection control allows you to change the color of the rectangle displayed around highlighted elements on the Recording Surface.

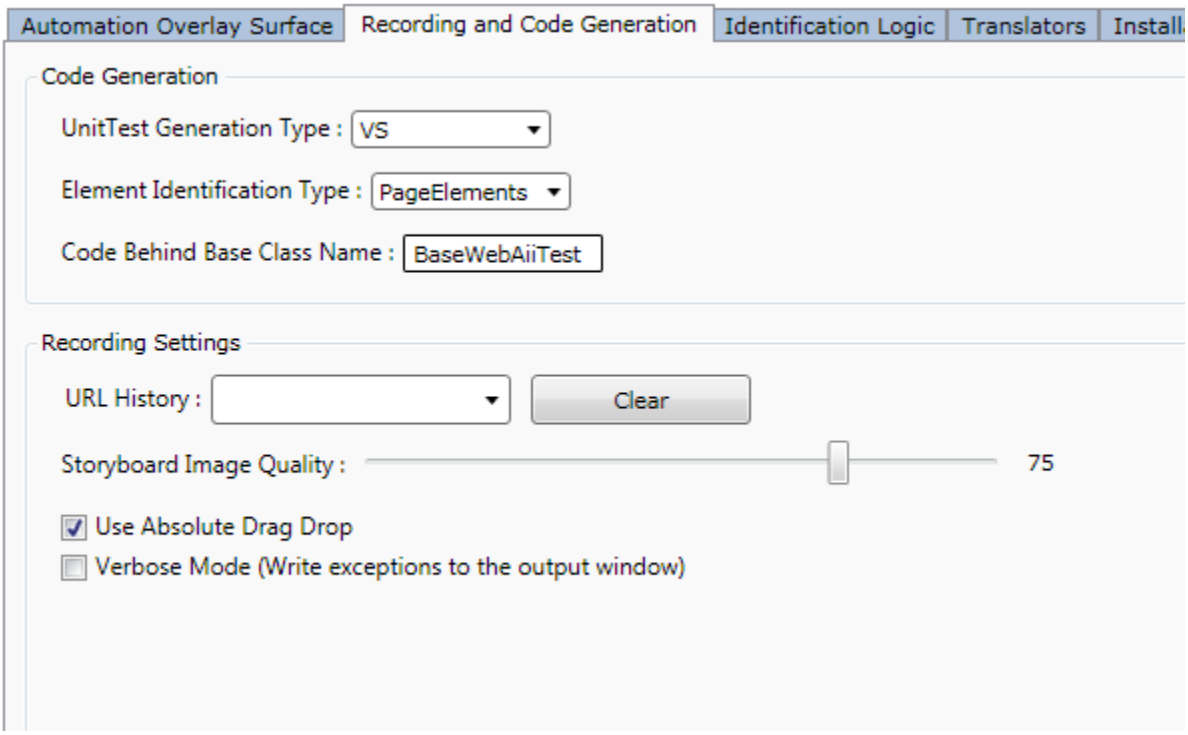
B.2.3 Highlight Border Width

This controls the width, in pixels, of the border of the rectangle displayed around highlighted elements on the Recording Surface.

B.2.4 Highlight Background Transparency

This controls the transparency level of the background drawn on the Recording Surface when the element context menu is displayed, that is, when you right click on an element in the recording while overlay highlighting is enabled. 0 means fully transparent. You won't even see the background. 1 means fully opaque. The background will be completely solid and you won't be able to see what the background covers up in the Recording Surface. Any value between 0 and 1 will show allow some amount of the UI underneath the background to show through.

B.3 Recording and Code Generation tab



B.3.1 Unit Test Generation Type

This controls the target testing engine the code needs to be generated for. You can select either VS for Visual Studio or NUnit. This controls the text fixtures that get generated for your unit tests.

B.3.2 Element Identification Type

This controls what type of HTML element identification code will be generated.

B.3.2.1 PageElements

When set to "PageElements" your generated unit test will actually use the elements defined in your Project.cs file.

B.3.2.2 AttributeFindParam

When set to "AttributeFindParam" a hardcoded FindParam attribute to the test method declaration like this:

```
[TestMethod(), FindParam("WebaiiProduct", FindType.TagIndex, "a:3")]
```

With the FindParam attribute set the generated code refers to elements like this:

```
HtmlAnchor WebaiiProduct = Elements["WebaiiProduct "].As<HtmlAnchor>();
```

The advantage to this approach is that your unit test is not dependent on Project.cs file. It is wholly self contained and standalone.

B.3.3 URL History

This drop down simply lists all the URLs you have visited in the Recording Surface. Selecting one from the list does nothing.

B.3.4 Clear

This button will clear your URL history. Note: Be certain you really want to do this. There is no warning and once cleared there's no undo.

B.3.5 Storyboard Image Quality

This setting controls how much the snapshot of the recorded step will be compressed at the time of recording prior to being added to the storyboard. It is a percentage up to 100%. 100% means the snapshot will not be compressed; it will be stored with full resolution in the storyboard. A setting of 50 will compress the image 50% before it is placed in the storyboard. The more you compress the image the more disk space you save but at the same time the stored image gets more and more blurry.

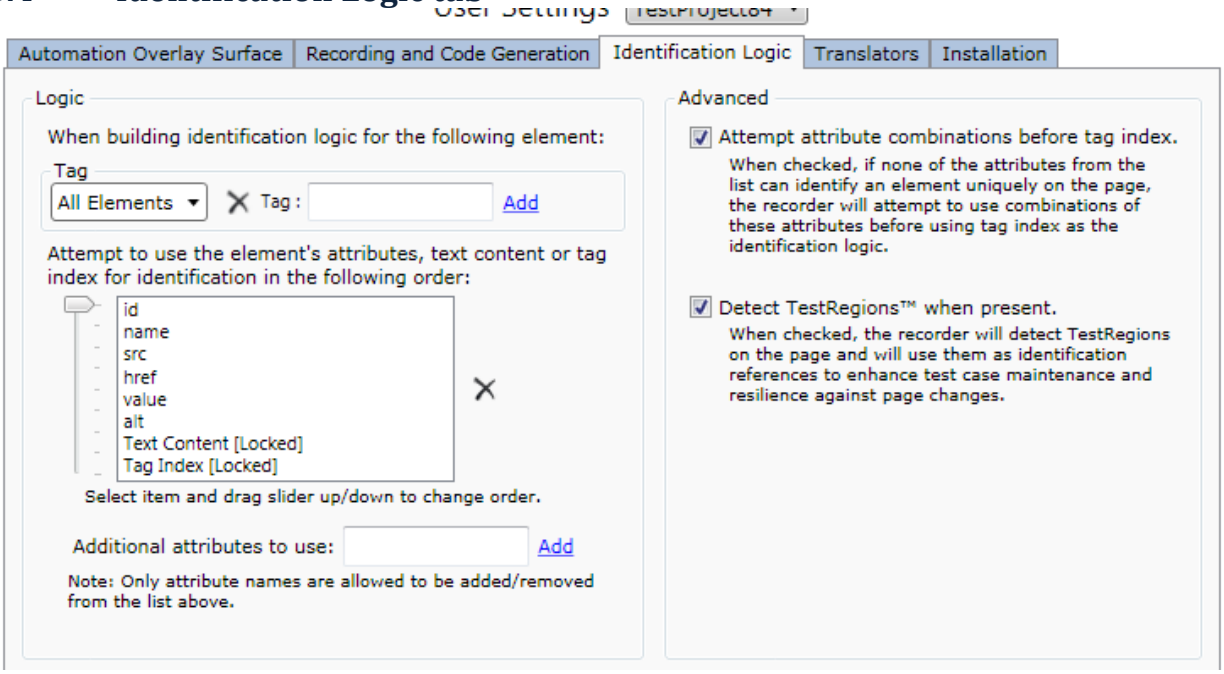
B.3.6 Use Absolute Drag Drop

When Absolute Drag Drop is checked Automation Design Canvas will record drag & drop operations using the actual pixel drop point for the drop target. When unchecked Design Canvas will try to find an HTML element closest to the drop point and use it for the drop target calculation.

B.3.7 Verbose Mode

When Verbose Mode is checked Design Canvas will display errors and exceptions in the output window.

B.4 Identification Logic tab



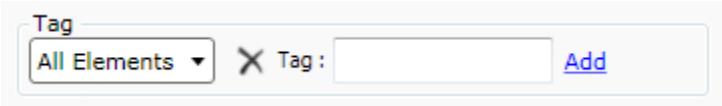
You can fully customize how Automation Design Canvas auto-generates the FindParam's used to find HTML elements on web page's. On this tab are all the settings used to control Automation Design Canvas's intelligent HTML element recording identification scheme.

B.4.1 Tag

The used recording scheme can have multiple attribute priorities. For example, during recording you may want to have Automation Design Canvas use the href attribute with all elements that have the <a> tag and for it to use the Name attribute with all elements that have the <img> tag and for all other elements have it use the ID attribute in the FindParam. This may sound complicated at first but hopefully the following sections will help clear up this concept of having multiple attribute priority schemes within a recording scheme.

To implement the example above you start by adding attribute priority schemes within the current recording scheme for the <a> tag and the <img> tag as follows:

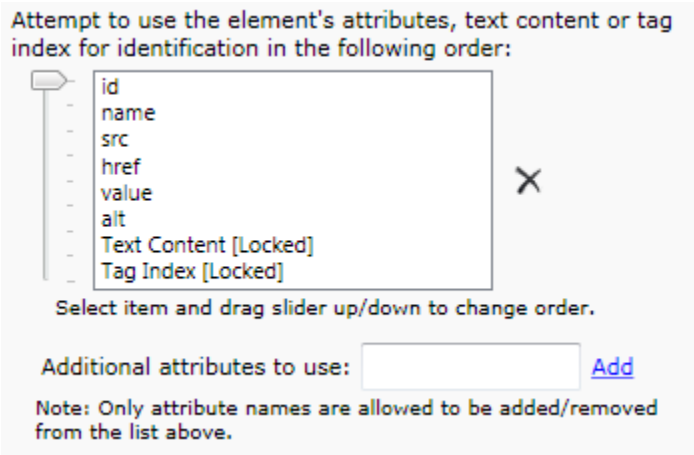
- Enter "a" in the text box for the <a> tag.
- Click Add Tag.
- Repeat the above steps for the <img> tag, entering "img" instead of "a". You can continue doing this for as many different tags you want to create a custom attribute priority scheme.



Now that you have created attribute priority schemes for these tags you can set the attribute priorities as explained in the following sections.

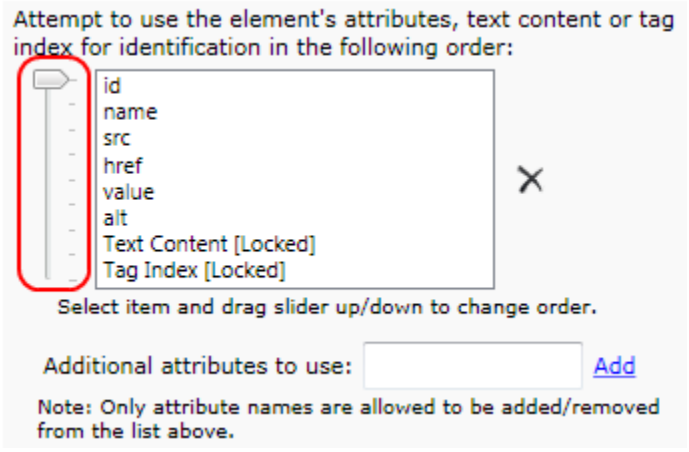
The “All Elements” entry in the dropdown is the default/last resort priority scheme that Automation Design Canvas will use when the tag of the element being recorded does not match one from the list.

B.4.2 Attribute priority for selected tag group



The controls in this group are used to control/adjust the attribute priority scheme for the tag currently selected in the tag dropdown.

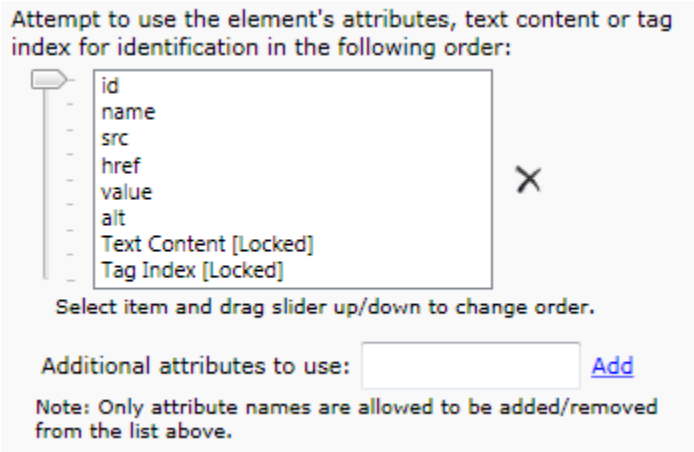
B.4.3 Slider



The up/down slider next to the attribute priority list moves the selected attribute up or down in the list (you can only select one attribute at a time). By moving it up in the list you are raising the

priority of that attribute. By moving it down in the list you are doing the opposite, lowering the priority of that attribute. The order of the attributes in this list determines the order (or priority) in which Automation Design Canvas will test them when it is looking for one that uniquely identifies the element in the list.

Let's look at an example to help make this clear. Suppose this is your attribute priority list for the <IMG> tag:

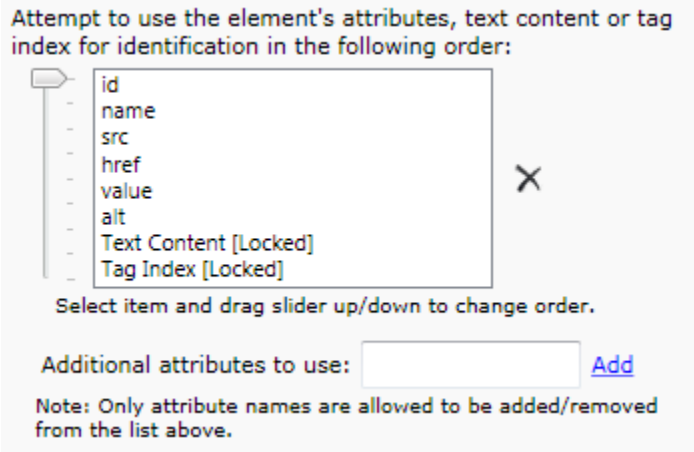


When adding an HTML <img> element to Elements Explorer during recording, Automation Design Canvas tries using the id attribute first. If the value of the id attribute for that element turns out to be unique for the entire set of elements on that web page, Automation Design Canvas will add the element and create a FindParam for it that finds the element by that id attribute value. It will not bother trying the other attributes since it found one that is unique.

If the id attribute is not unique or is not even present for that element, Design Canvas will try the next attribute in the attribute priority list which is the name attribute in this example. It will keep doing this and finally resort to using the TagIndex value as the last resort when nothing else works i.e., none of the attributes in the attribute priority list are unique for that element.

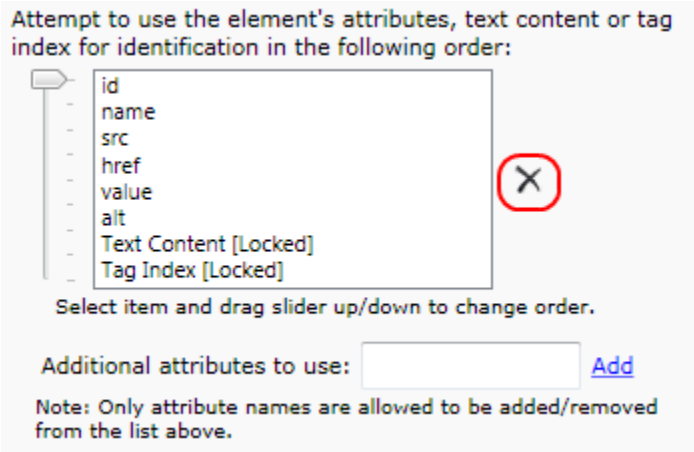
When the HTML element being added doesn't have an attribute priority list for it in the active recording scheme, Design Canvas uses the default "All Elements" attribute priority scheme.

B.4.4 Add



You can add an attribute to the attribute priority list by first typing the name of the attribute into the text box then clicking add. It will be added to the bottom of the list. Do not forget to adjust its priority once added.

B.4.5 Delete



You can delete an attribute from the attribute priority list by selecting the attribute from the list to be deleted from the tag list then clicking Delete. ***Note****: Be sure you want to do this. There is no warning and once deleted there is no undo.*

B.4.6 Detect Test Regions

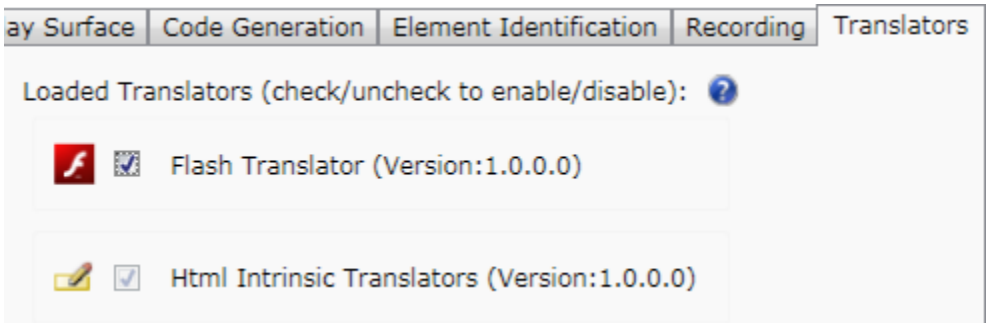
This option will craft the FindParam using a test region (if present) as part of the FindParam generation. More information about using test regions in web page's is available online at http://www.artoftest.com/Resources/WebAii/Documentation/topicsindex.aspx?topic=introtestregions.

B.4.7 Attempt attribute combinations

Automation Design Canvas tries to find a single attribute of the element that will uniquely identify that element from all other elements located on the web page, such as a unique ID or

Name. When this method fails Automation Design Canvas has the ability to try attribute combinations to uniquely identify that element, such as style and href.

B.5 Translators tab



This section is for future expansion and extensibility of Design Canvas. Design Canvas was designed with significant extensibility in mind. Currently the only thing you can do on this panel is enable or disable the Flash automation extension. The Design Canvas extensibility model is a forthcoming feature to be announced soon.

C Optimizing Test Run Performance

The default settings applied to test steps and elements are considered “safe” settings, that is, they provide the most reliable configuration for making tests pass most of the time. To increase test performance you can carefully and selectively change these settings.

C.1 ValidatePageUrl property

This property applies to elements contained in Elements Explorer. Normally when an element gets recorded this property is set to False, meaning that Automation Design Canvas will take the time to verify it is on the correct web page before acting on this element. If you can be certain you are on the right web page you can set this to False to save some time.

C.2 WaitOnElements property

This property applies to test steps. Normally when a test step is recorded this property is set to True, meaning that Design Canvas will wait for the elements used by that test step to be present on the web page before executing that test step. If you can be certain the elements are already present on the web page you can set this to False to save some time.

C.3 Focus

This property applies to drag and drop steps. By leaving this property set to False Design Canvas will not take the time to try to scroll the element into view. If you can be certain that the element will always be in view you can set this to False to save some time.

D Files used by Automation Design Canvas

These are all of the Automation Design Canvas specific files that are added to a test project containing a WebAii test. You may of course have more files than these in your test project. They are listed here only for reference.

D.1 .aii files

These files contain the steps of your test as displayed in Test Explorer. It also contains the find parameters for every element used by that test. You have one of these files for every WebAii test in your project. The contents are in XML format making it easier to be stored in a source code library and potentially have multiple changes by a development team auto-merged by the source control system. The contents however are in a proprietary format that should not be directly modified.

D.2 .aii.cs files

These files contain the source code to your code behind methods. They must be compiled by Visual Studio before your test will execute properly. You have one of these files for every WebAii test in your project using a code behind file and it will have the same name as the matching .aii file that this file is linked to.

D.3 .resx files

These files store the snapshots taken during recording as displayed by the storyboard. It also stores the browsers DOM state during recording for each step of your test. You have one of these files for every WebAii test in your project and it will have the same name as the matching .aii file that this file is linked to. The contents are in XML format making it easier to be stored in a source code library and potentially have multiple changes by a development team auto-merged by the source control system. The contents however are in a proprietary format that should not be directly modified.

D.4 Project.Elements files

This file is just a place holder. It doesn't actually hold any information.

D.5 Project.cs files

This file stores all the elements as displayed in Elements Explorer for the entire project. You have only one of these per test project that contains a WebAii test. The contents of this file are auto-generated as your project elements changes. This file contains the declarative strongly-typed element definitions that can be used for coded test, unit tests or referenced from other projects.

Note: This file is constantly being generated so any changed made manually to this file will be lost when the tool auto-generates it. You should rely on the Element Explorer editing feature to update how an element is being found.