ArtOfTest, Inc.

ArtOfTest, Inc.

AUTOMATION DESIGN CANVAS 2.0 USER GUIDE

CONTENT

1 Introducing Automation Design Canvas

Automation Design Canvas^{TM} (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://www.artoftest.com/products/webaii.aspx and http://www.artoftest.com/support/webaii/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^{TM} 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 faster and 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://www.artoftest.com/products/design-canvas.aspx and http://www.artoftest.com/community/videos.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.
- **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 and record the scenario once in Design Canvas and an automated test is generated. Then with Quick Execution, run the recorded test against Internet Explorer® or Mozilla Firefox® or Safari® 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.
- **Modularize Testing:** Using the "Test as a Step" feature you can modularize your test scripts into smaller testing components that are then integrated into larger test scripts.

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 back and forth 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, properties, 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 0 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 you 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 expression 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 16 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 19 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 20 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, MbUnit, XUnit as well as Visual Studio Unit tests.

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 shows you which part of the find expression failed. This helps you quickly determine how it needs to be updated to find the right element. Also Design Canvas includes live validation for verification updates and ‘element find expression’ updates hence you no longer need to repeatedly run tests over and over to validate your changes. Both of these features save test maintenance time. Section 21 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 **Error! Reference source not found.** 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 lot of time compared to maintaining more cumbersome VS Web Tests. Section 23.7 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 How to videos	http://www.artoftest.com/community/videos.aspx
WebAii Reference	http://www.artoftest.com/support/webaii/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://www.artoftest.com/community/blogs.aspx and http://www.artoftest.com/community/forums.aspx 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
- Safari 3.0 and above

2.2 Installing Automation Design Canvas

2.2.1 Prerequisites

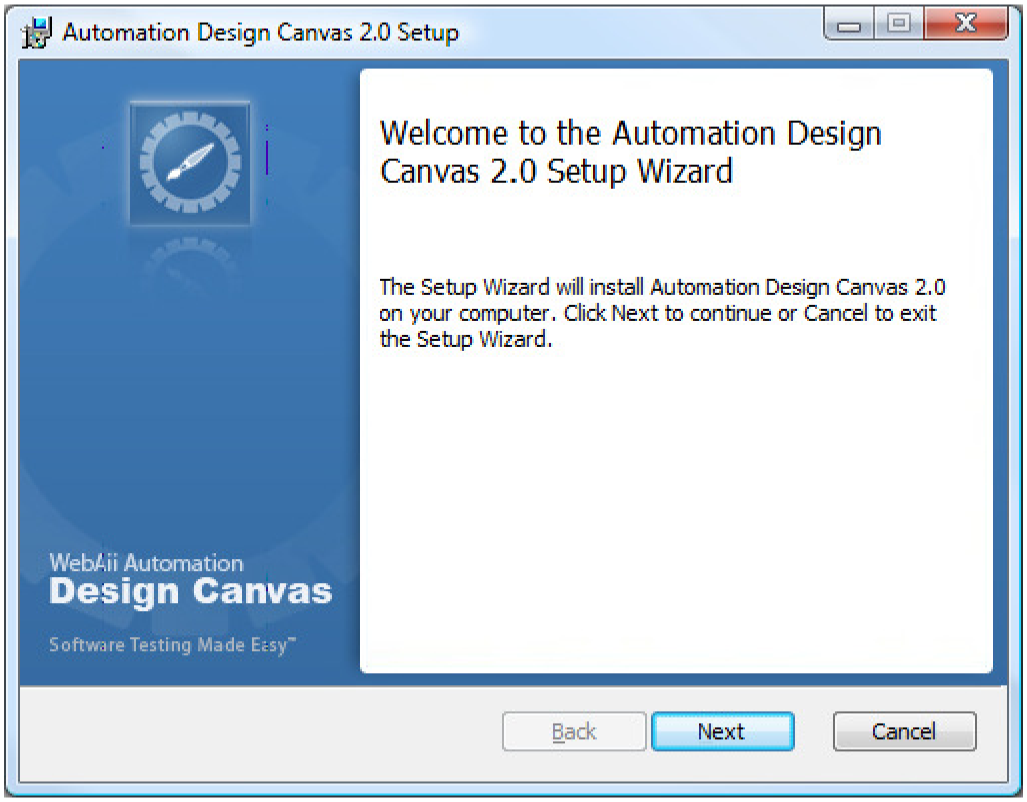
Automation Design Canvas requires you to first install Visual Studio 2008 Professional SP1 (see http://www.microsoft.com/downloads/details.aspx?FamilyId=FBEE1648-7106-44A7-9649-6D9F6D58056E&displaylang=en to download and install SP1 for Visual Studio 2008) edition or Visual Studio 2008 Team System SP1 edition. Installing Design Canvas is simple and does not require preplanning. Simply run the Automation Design Canvas install file and follow the prompts.

The installation program performs these steps:

- Install WebAii 2.0 framework if not already installed.
- Installs Automation Design Canvas.
- Registers the Visual Studio Test templates used by Automation Design Canvas with Visual Studio

Note: If you have the Beta version of Automation Design Canvas or WebAii 2.0 framework installed, please uninstall them before installing Automation Design Canvas.

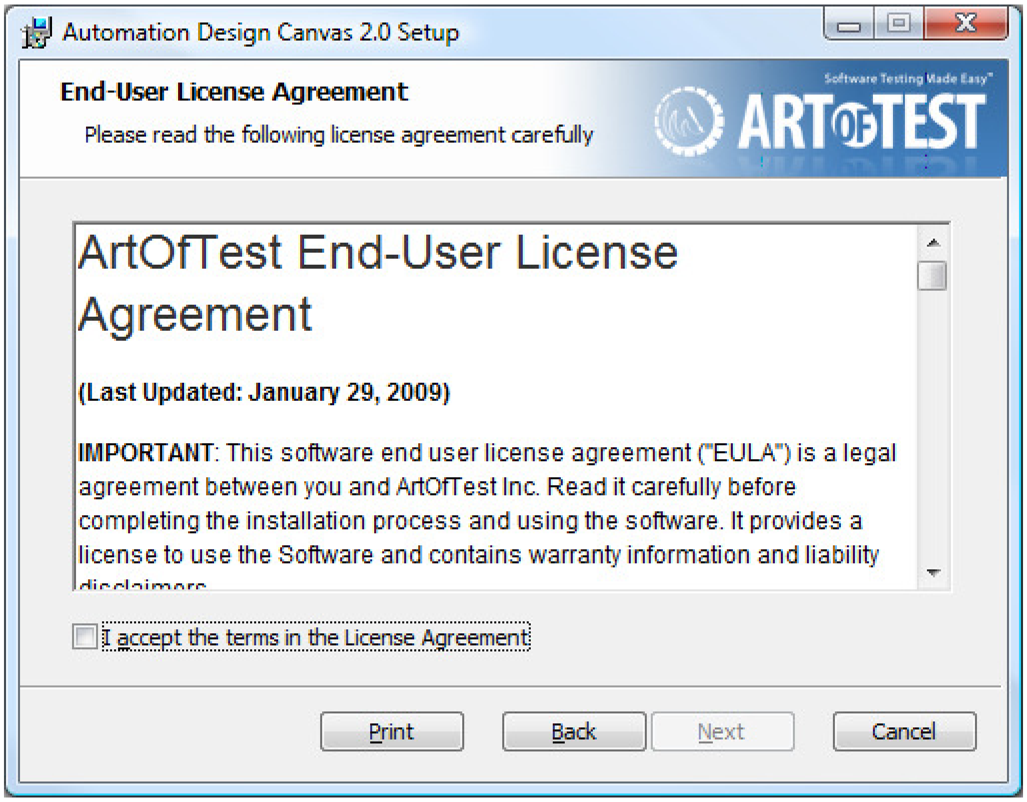
2.2.2 Installing Automation Design Canvas 2.0


Automation Design Canvas 2.0 Setup
WebAii Automation
Design Canvas
Software Testing Made Easy™
Back
Next
Cancel

Welcome to the Automation Design Canvas 2.0 Setup Wizard

The Setup Wizard will install Automation Design Canvas 2.0 on your computer. Click Next to continue or Cancel to exit the Setup Wizard.

Click **Next** to display the License Agreement.


Automation Design Canvas 2.0 Setup
Software Testing Made Easy™
ARTOFTEST
I accept the terms in the License Agreement
Print
Back
Next
Cancel

End-User License Agreement

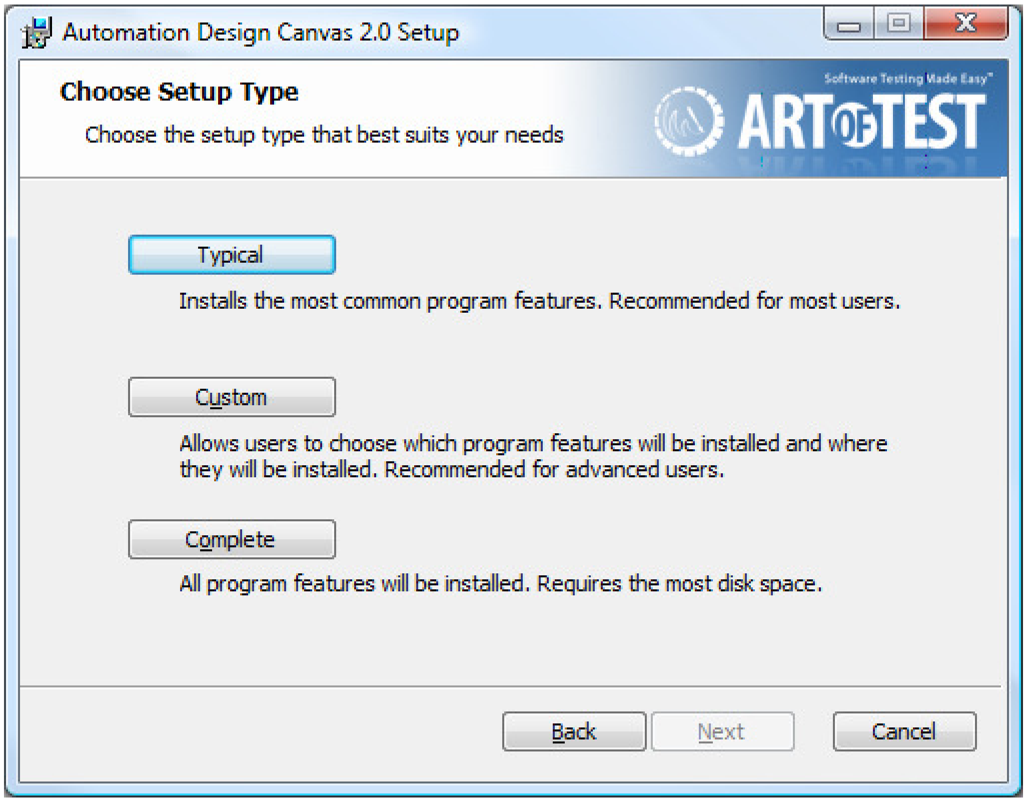
Please read the following license agreement carefully

ArtOfTest End-User License Agreement

(Last Updated: January 29, 2009)

IMPORTANT: This software end user license agreement ("EULA") is a legal agreement between you and ArtOfTest Inc. Read it carefully before completing the installation process and using the software. It provides a license to use the Software and contains warranty information and liability

Read and accept the license terms and click **Next**.


Automation Design Canvas 2.0 Setup

Choose Setup Type

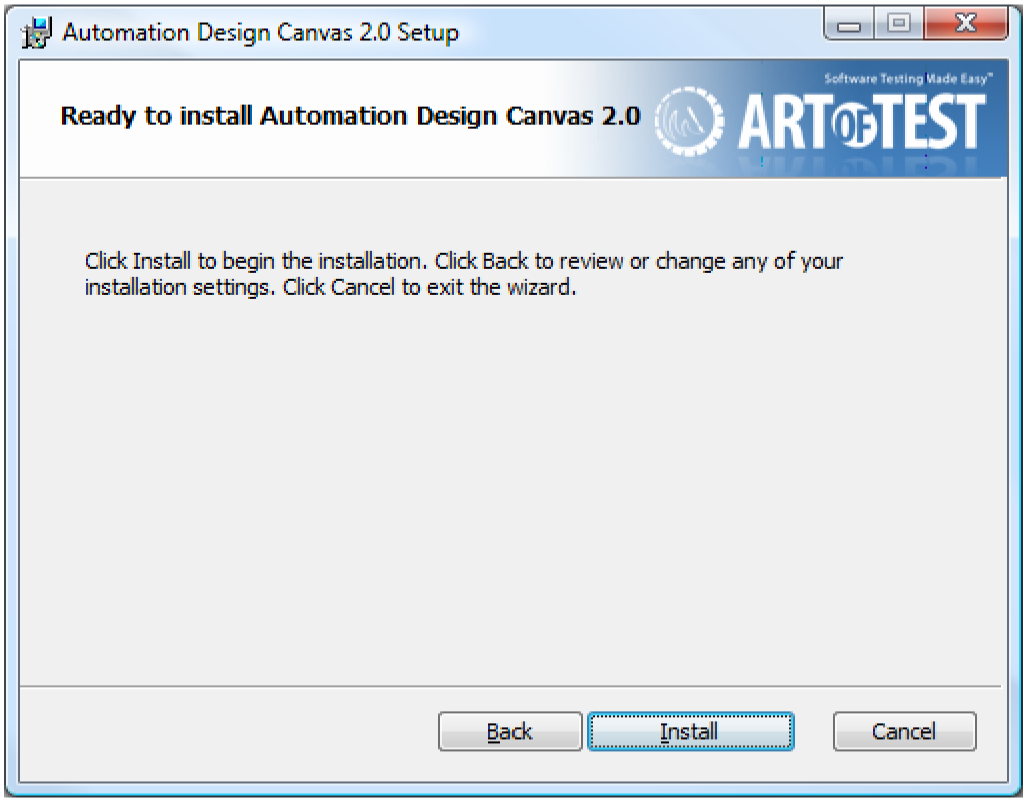
Choose the setup type that best suits your needs

Installs the most common program features. Recommended for most users.

Allows users to choose which program features will be installed and where they will be installed. Recommended for advanced users.

All program features will be installed. Requires the most disk space.

Selecting **Typical** will install the most commonly used features. Selecting **Custom** will allow you to choose what features to install and the install location. Selecting **Complete** will install all features in the default location. After choosing the type of install you are ready for the install to begin.

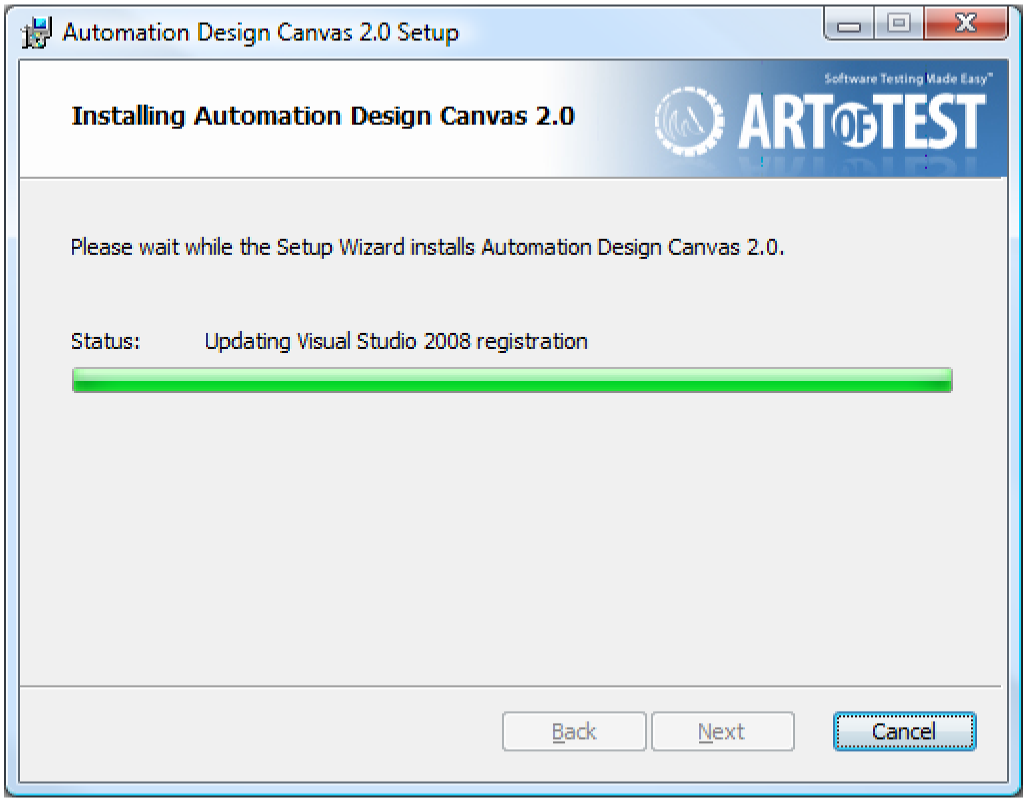

Automation Design Canvas 2.0 Setup

Ready to install Automation Design Canvas 2.0

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

Click **Install**. The installer copies the required files to C:\Program Files\ArtOfTest\Design Canvas 2.0.

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.



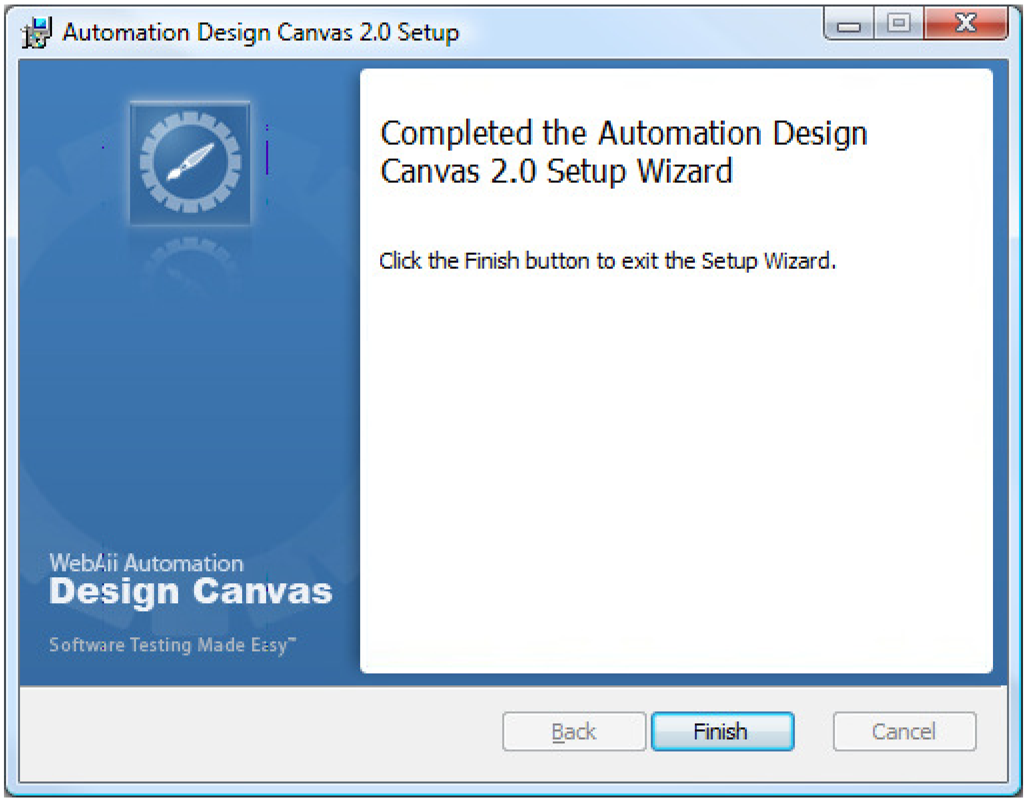
Automation Design Canvas 2.0 Setup

Installing Automation Design Canvas 2.0

Please wait while the Setup Wizard installs Automation Design Canvas 2.0.

Status: Updating Visual Studio 2008 registration

After installation is complete you will see this confirmation:



Automation Design Canvas 2.0 Setup

Completed the Automation Design Canvas 2.0 Setup Wizard

Click the Finish button to exit the Setup Wizard.

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

2.3 Configuring Your Browser for Test Automation

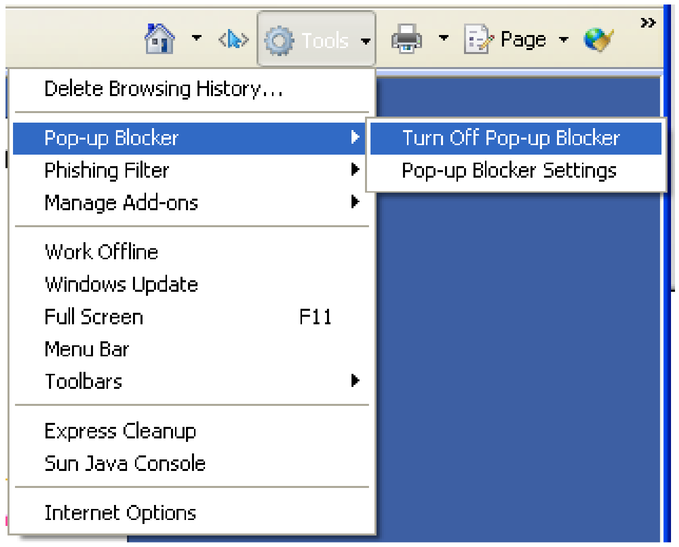
The default browser configurations are great for normal everyday standard use but these settings present a few problems for smooth browser-driven test automation. Features such as popup blockers, IE's "gold bar", and security alert dialogs get in the way of fully unattended browser-driven test automation. This section describes how to change your browser configuration settings for a smoother test automation experience. We first describe how to configure IE 7/8 settings followed by Firefox 2.0 and 3.x settings and then Safari 3.x and 4.0 settings.

2.3.1 Configuring Internet Explorer

2.3.1.1 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:

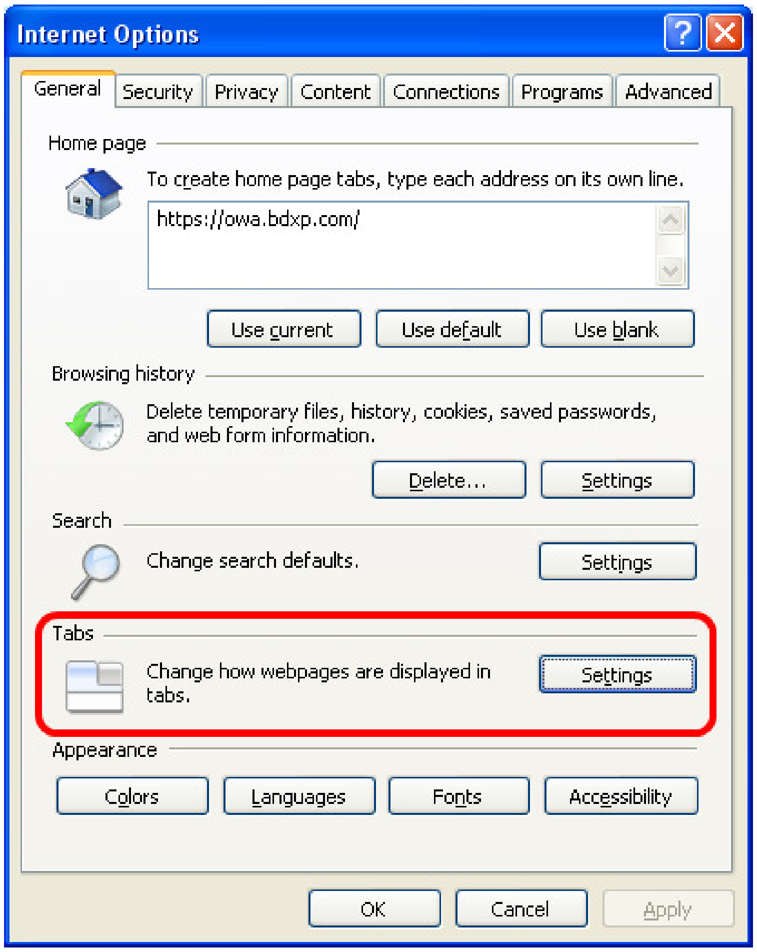
- In IE, click **Tools** then highlight Pop-up Blocker by moving the cursor over it.
- Click Turn Off Pop-up Blocker.



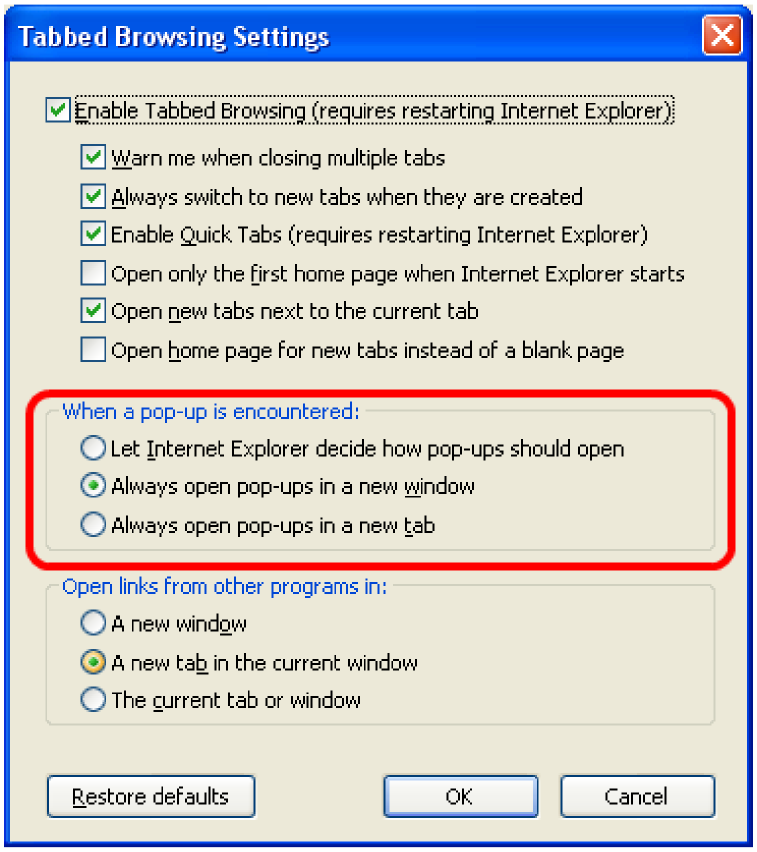
2.3.1.2 Opening Pop-ups in a New Window

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

- 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".



Tabbed Browsing Settings

- [✓] Enable Tabbed Browsing (requires restarting Internet Explorer)
  - [✓] Warn me when closing multiple tabs
  - [✓] Always switch to new tabs when they are created
  - [✓] Enable Quick Tabs (requires restarting Internet Explorer)
  - [ ] Open only the first home page when Internet Explorer starts
  - [✓] Open new tabs next to the current tab
  - [ ] Open home page for new tabs instead of a blank page

When a pop-up is encountered:
- ○ Let Internet Explorer decide how pop-ups should open
- ● Always open pop-ups in a new window
- ○ Always open pop-ups in a new tab

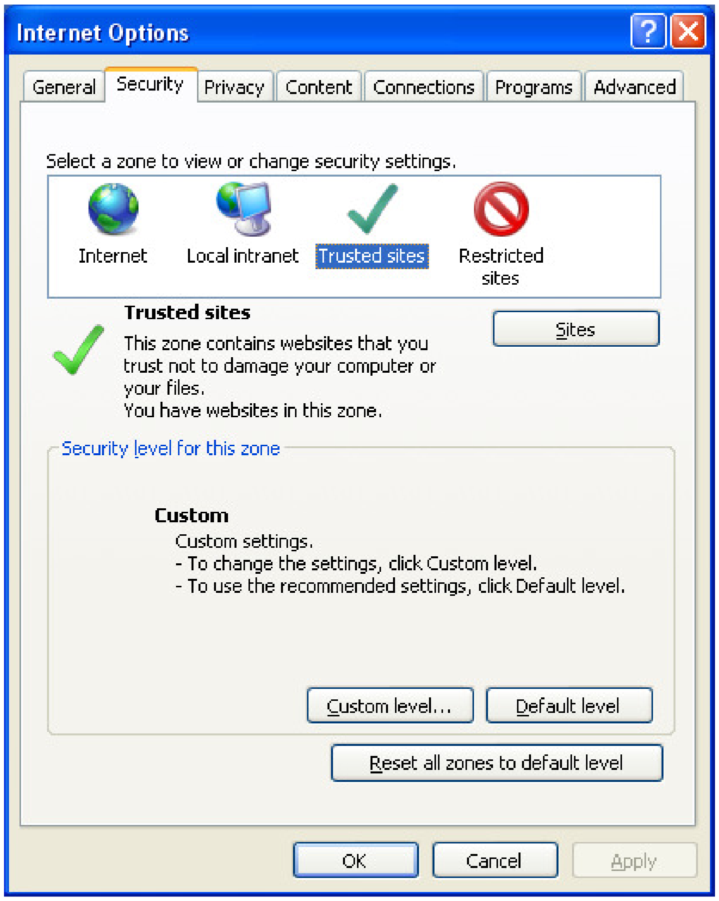
Open links from other programs in:
- ○ A new window
- ● A new tab in the current window
- ○ The current tab or window

2.3.1.3 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.



Select a zone to view or change security settings.

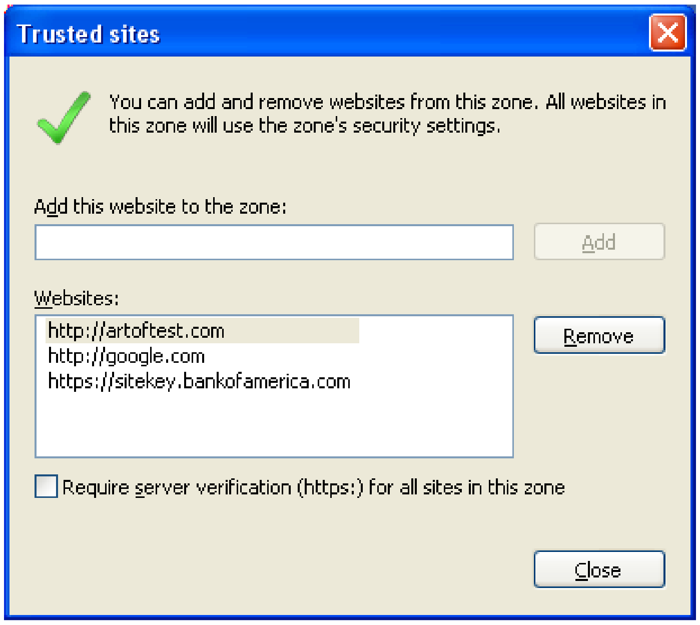
Trusted sites

This zone contains websites that you trust not to damage your computer or your files.
You have websites in this zone.

Security level for this zone

Custom

Custom settings.
- To change the settings, click Custom level.
- To use the recommended settings, click Default level.


Trusted sites

You can add and remove websites from this zone. All websites in this zone will use the zone's security settings.

Add this website to the zone:

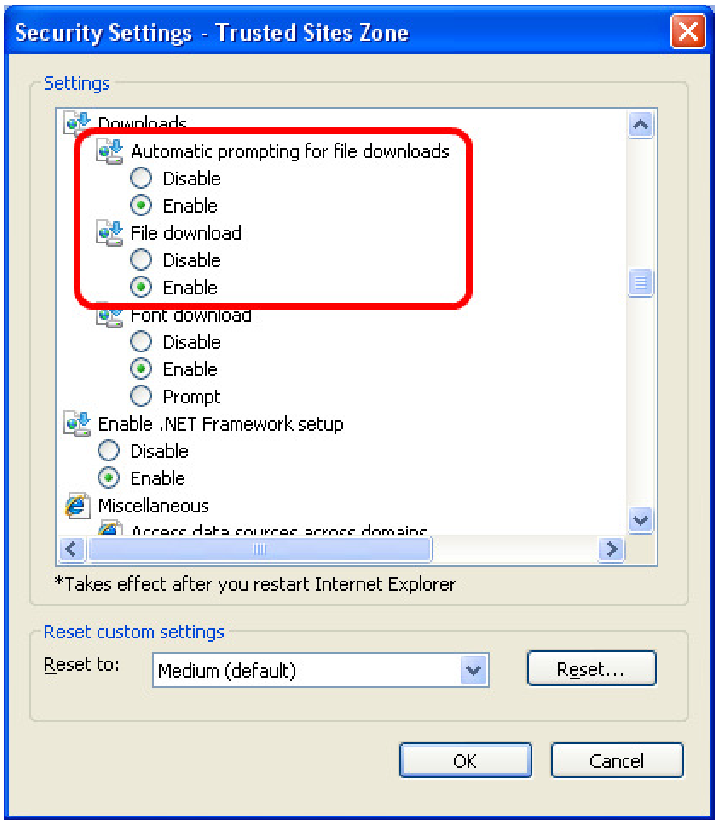
Websites:
http://artoftest.com
http://google.com
https://sitekey.bankofamerica.com

Require server verification (https:) for all sites in this zone

2.3.1.4 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.


Security Settings - Trusted Sites Zone

Settings

Downloads
- Automatic prompting for file downloads
  - ( ) Disable
  - (•) Enable
- File download
  - ( ) Disable
  - (•) Enable
- Font download
  - ( ) Disable
  - (•) Enable
  - ( ) Prompt

Enable .NET Framework setup
- ( ) Disable
- (•) Enable

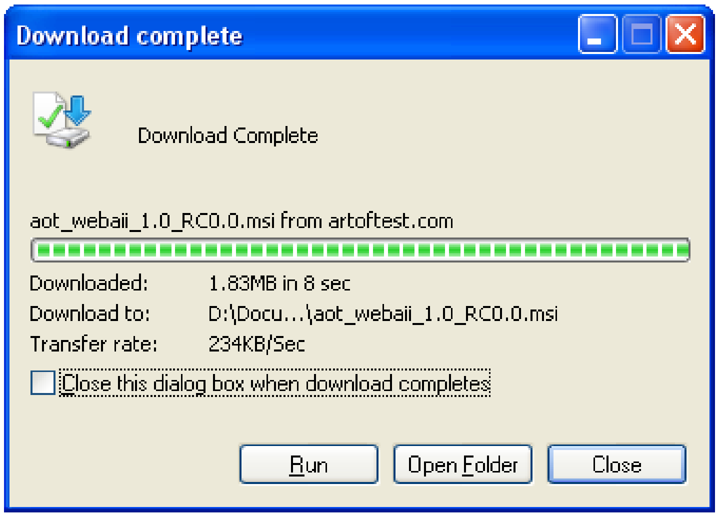
Miscellaneous
- Access data sources across domains

*Takes effect after you restart Internet Explorer

Reset custom settings

Reset to: Medium (default)

- 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.


Download complete

Download Complete

aot_webaii_1.0_RC0.0.msi from artoftest.com

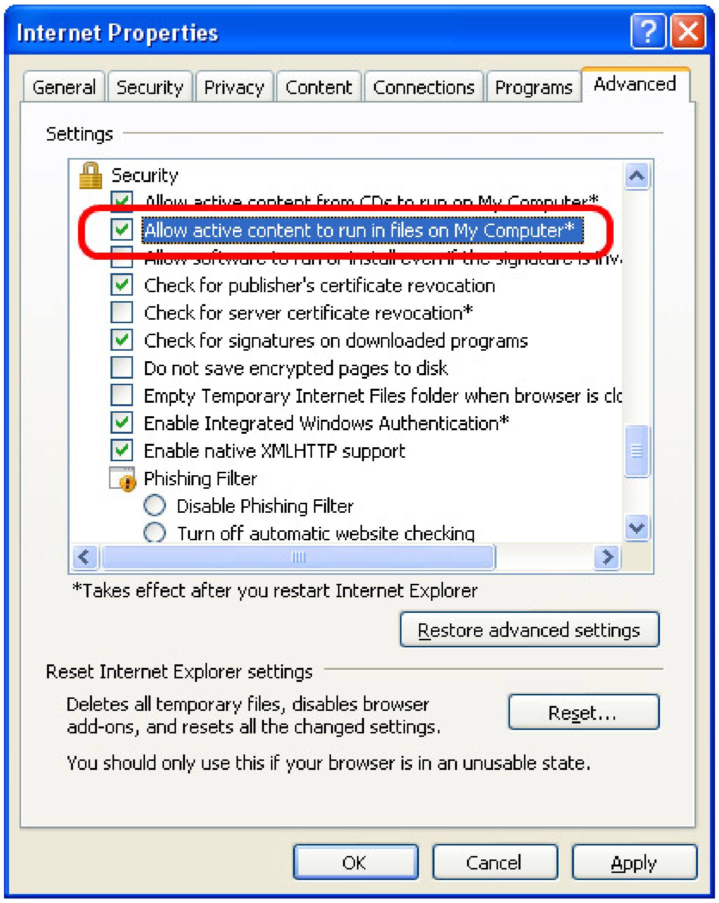
Downloaded: 1.83MB in 8 sec
Download to: D:\Docu...\aot_webaii_1.0_RC0.0.msi
Transfer rate: 234KB/Sec

[ ] Close this dialog box when download completes

2.3.1.5 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"


Internet Properties

General | Security | Privacy | Content | Connections | Programs | Advanced

Settings

Security
- [✓] Allow active content from CDs to run on My Computer*
- [✓] Allow active content to run in files on My Computer*
- [ ] Allow software to run or install even if the signature is inv
- [✓] Check for publisher's certificate revocation
- [ ] Check for server certificate revocation*
- [✓] Check for signatures on downloaded programs
- [ ] Do not save encrypted pages to disk
- [ ] Empty Temporary Internet Files folder when browser is clo
- [✓] Enable Integrated Windows Authentication*
- [✓] Enable native XMLHTTP support

Phishing Filter
- ( ) Disable Phishing Filter
- ( ) Turn off automatic website checking

*Takes effect after you restart Internet Explorer

Restore advanced settings

Reset Internet Explorer settings

Deletes all temporary files, disables browser add-ons, and resets all the changed settings.

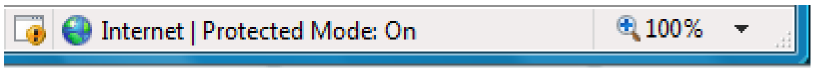
Reset...

You should only use this if your browser is in an unusable state.

OK | Cancel | Apply

2.3.1.6 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:


Internet | Protected Mode: On

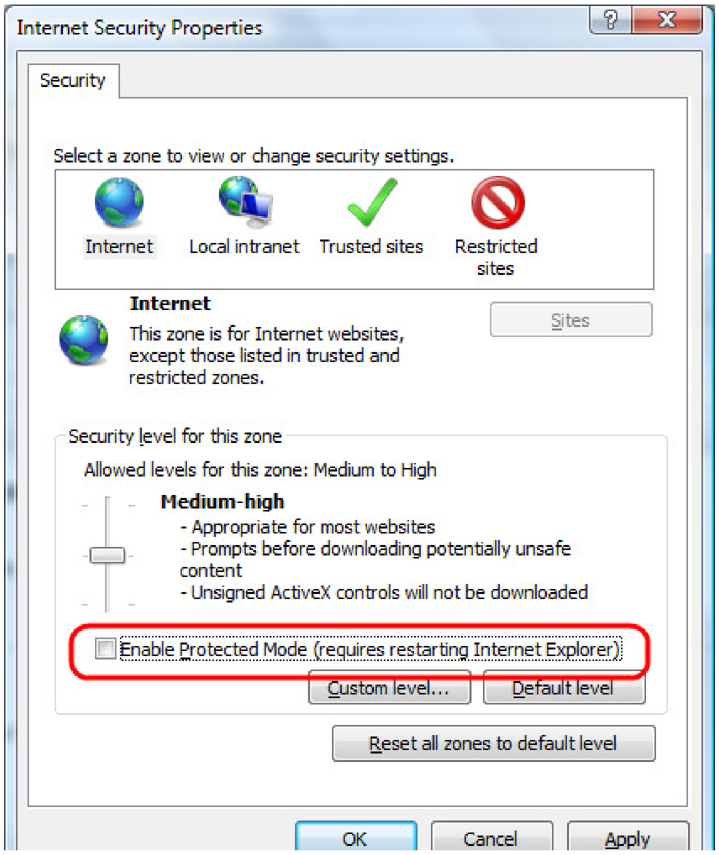
100%

There are two ways to turn off protected mode:

2.3.1.7 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.


Internet Security Properties

Security

Select a zone to view or change security settings.

Internet

This zone is for Internet websites, except those listed in trusted and restricted zones.

Security level for this zone

Allowed levels for this zone: Medium to High

Medium-high

- Appropriate for most websites
- Prompts before downloading potentially unsafe content
- Unsigned ActiveX controls will not be downloaded

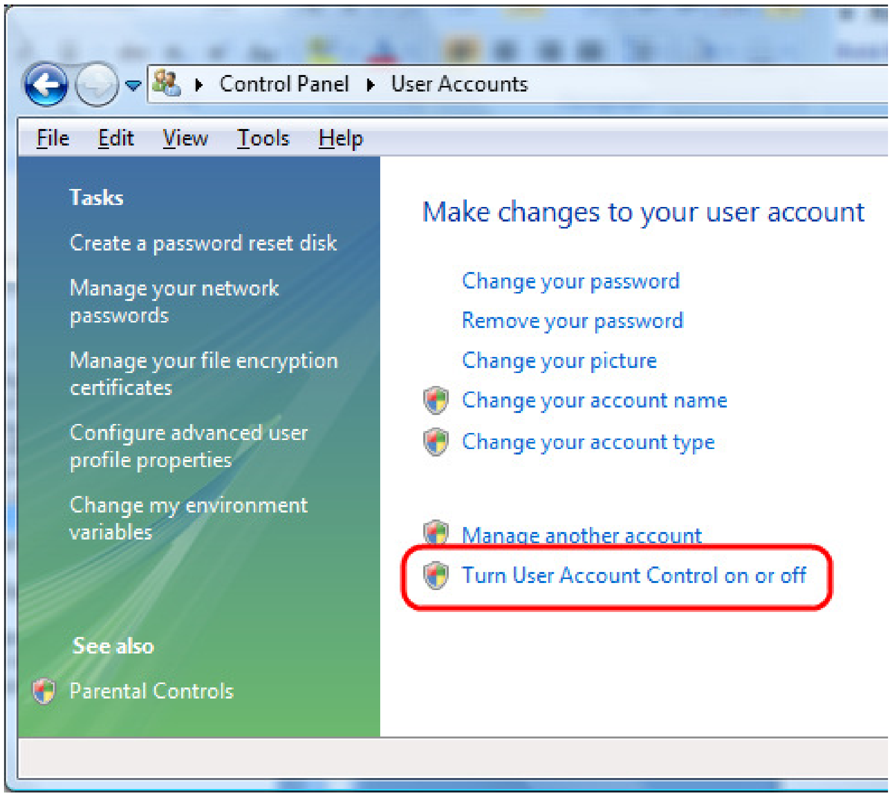
[ ] Enable Protected Mode (requires restarting Internet Explorer)

- Close and restart Internet Explorer

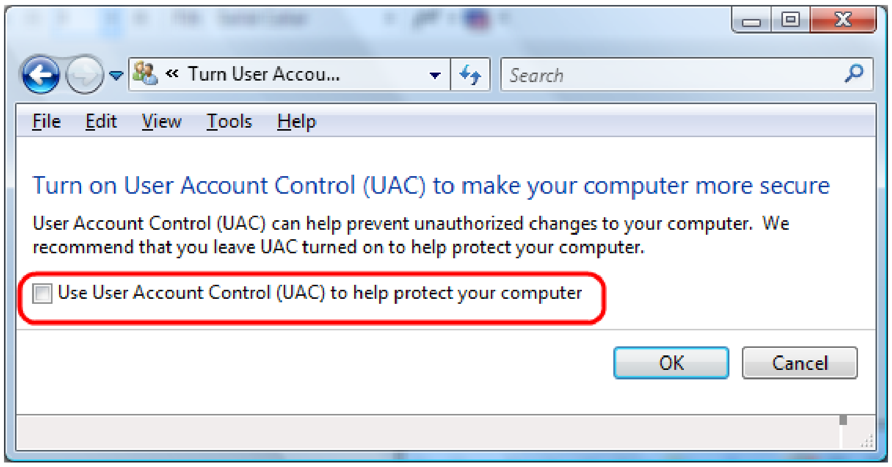
2.3.1.8 How to disable User Account Control

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.


Control Panel ▸ User Accounts
File Edit View Tools Help
Tasks
Create a password reset disk
Manage your network passwords
Manage your file encryption certificates
Configure advanced user profile properties
Change my environment variables
See also
Parental Controls
Make changes to your user account
Change your password
Remove your password
Change your picture
Change your account name
Change your account type
Manage another account
Turn User Account Control on or off

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



Turn on User Account Control (UAC) to make your computer more secure

User Account Control (UAC) can help prevent unauthorized changes to your computer. We recommend that you leave UAC turned on to help protect your computer.

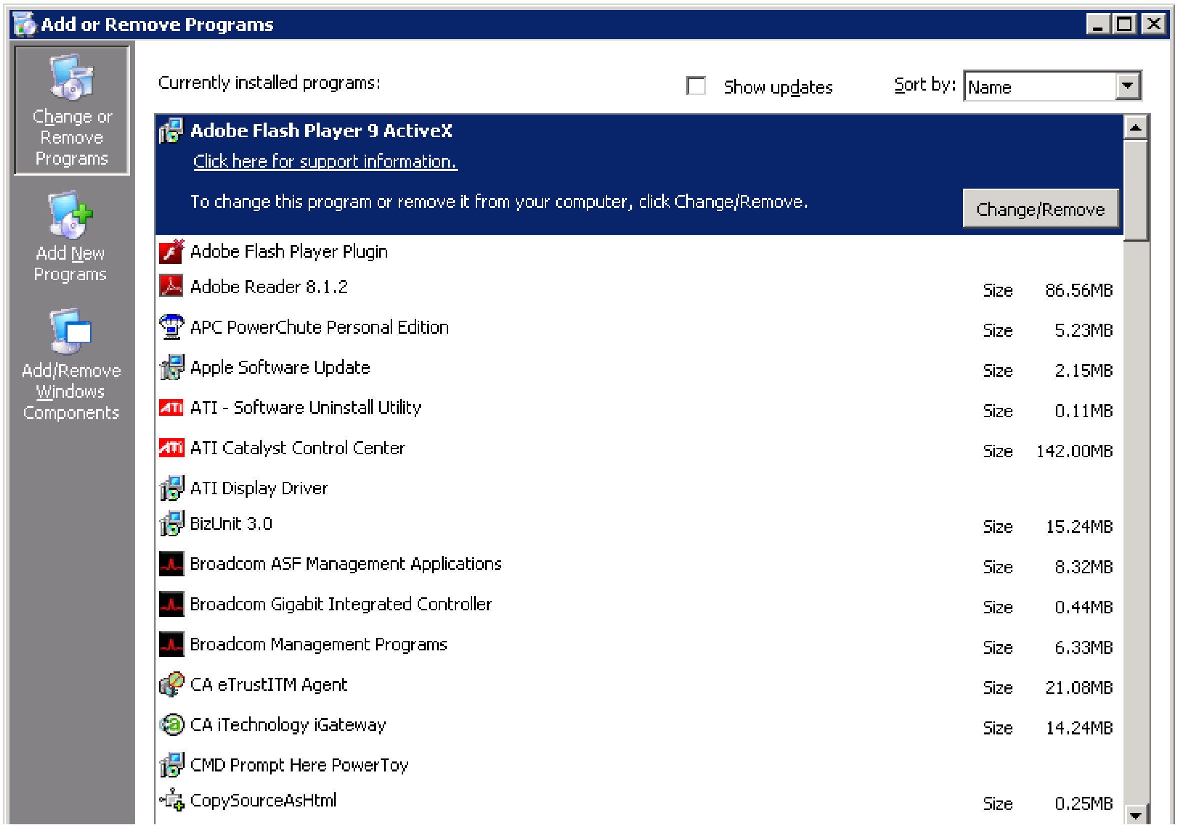
☐ Use User Account Control (UAC) to help protect your computer

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

2.3.1.9 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.

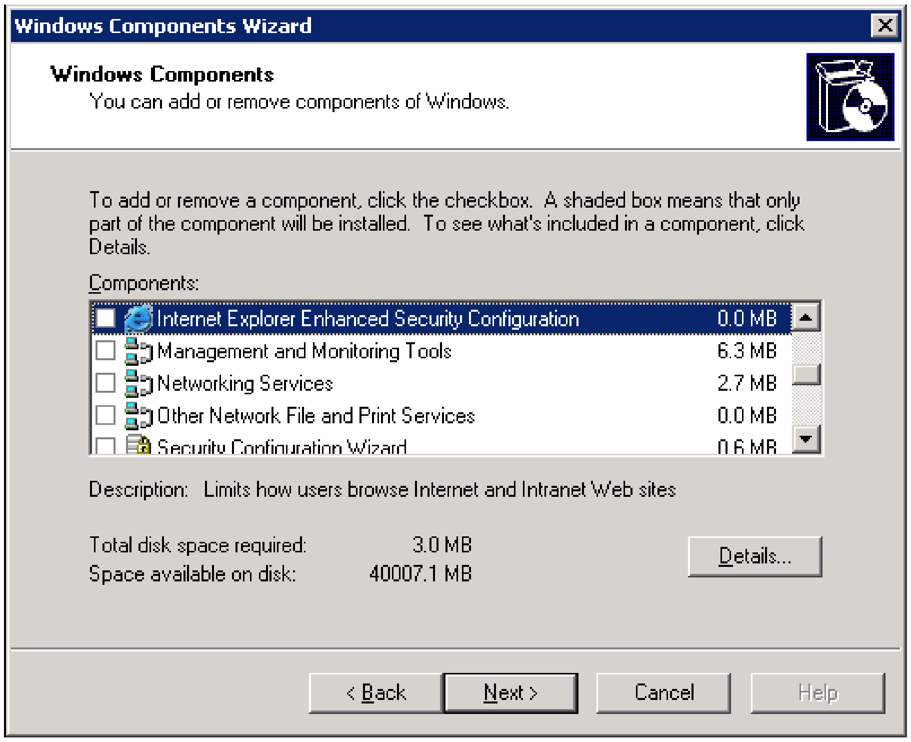


Currently installed programs:

Adobe Flash Player 9 ActiveX
Click here for support information.
To change this program or remove it from your computer, click Change/Remove.

Adobe Flash Player Plugin		
Adobe Reader 8.1.2	Size	86.56MB
APC PowerChute Personal Edition	Size	5.23MB
Apple Software Update	Size	2.15MB
ATI - Software Uninstall Utility	Size	0.11MB
ATI Catalyst Control Center	Size	142.00MB
ATI Display Driver		
BizUnit 3.0	Size	15.24MB
Broadcom ASF Management Applications	Size	8.32MB
Broadcom Gigabit Integrated Controller	Size	0.44MB
Broadcom Management Programs	Size	6.33MB
CA eTrustITM Agent	Size	21.08MB
CA iTechnology iGateway	Size	14.24MB
CMD Prompt Here PowerToy		
CopySourceAsHtml	Size	0.25MB

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



Windows Components
You can add or remove components of Windows.

To add or remove a component, click the checkbox. A shaded box means that only part of the component will be installed. To see what's included in a component, click Details.

Components:

☐ Internet Explorer Enhanced Security Configuration	0.0 MB
☐ Management and Monitoring Tools	6.3 MB
☐ Networking Services	2.7 MB
☐ Other Network File and Print Services	0.0 MB
☐ Security Configuration Wizard	0.6 MB

Description: Limits how users browse Internet and Intranet Web sites

Total disk space required: 3.0 MB
Space available on disk: 40007.1 MB

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

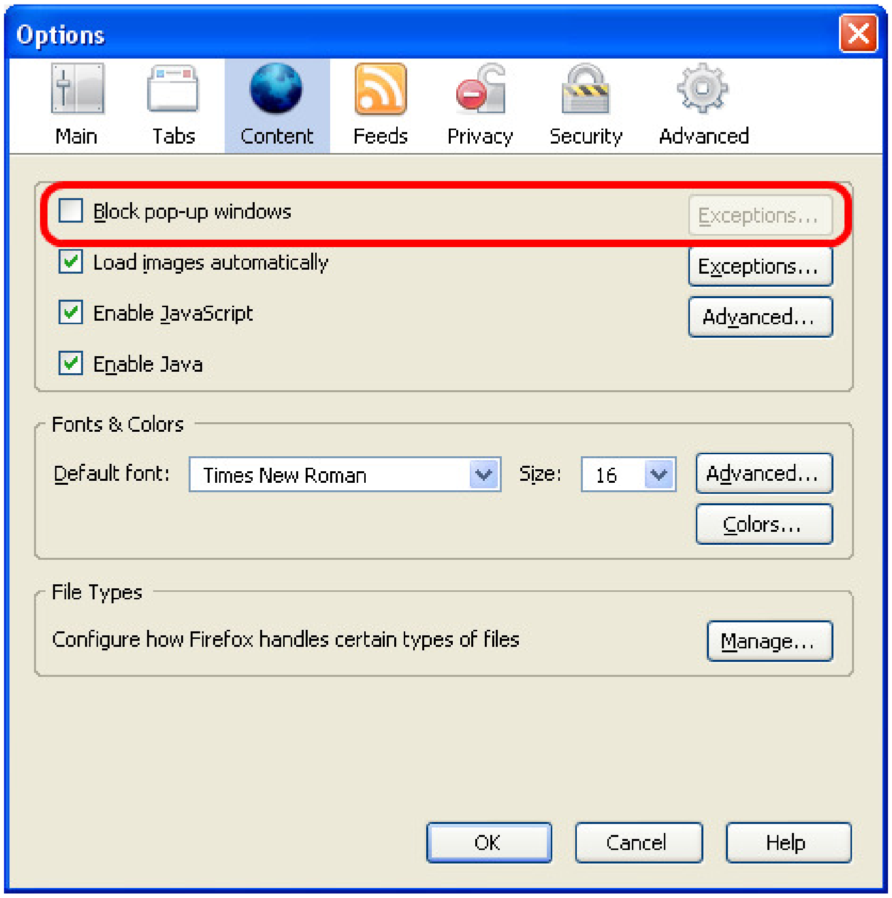
- Click **Finish**.

2.3.2 Configuring Firefox

2.3.2.1 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.



☐ Block pop-up windows
☑ Load images automatically
☑ Enable JavaScript
☑ Enable Java

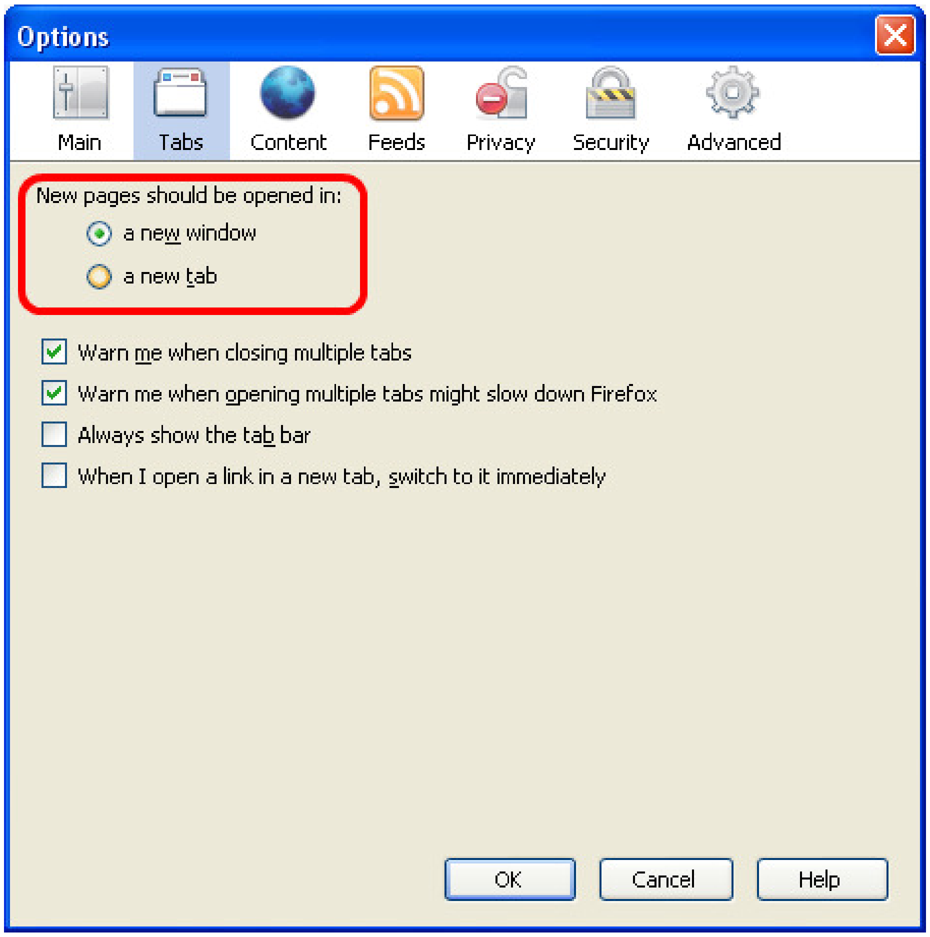
Fonts & Colors
Default font: Times New Roman Size: 16

File Types
Configure how Firefox handles certain types of files

2.3.2.2 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**.

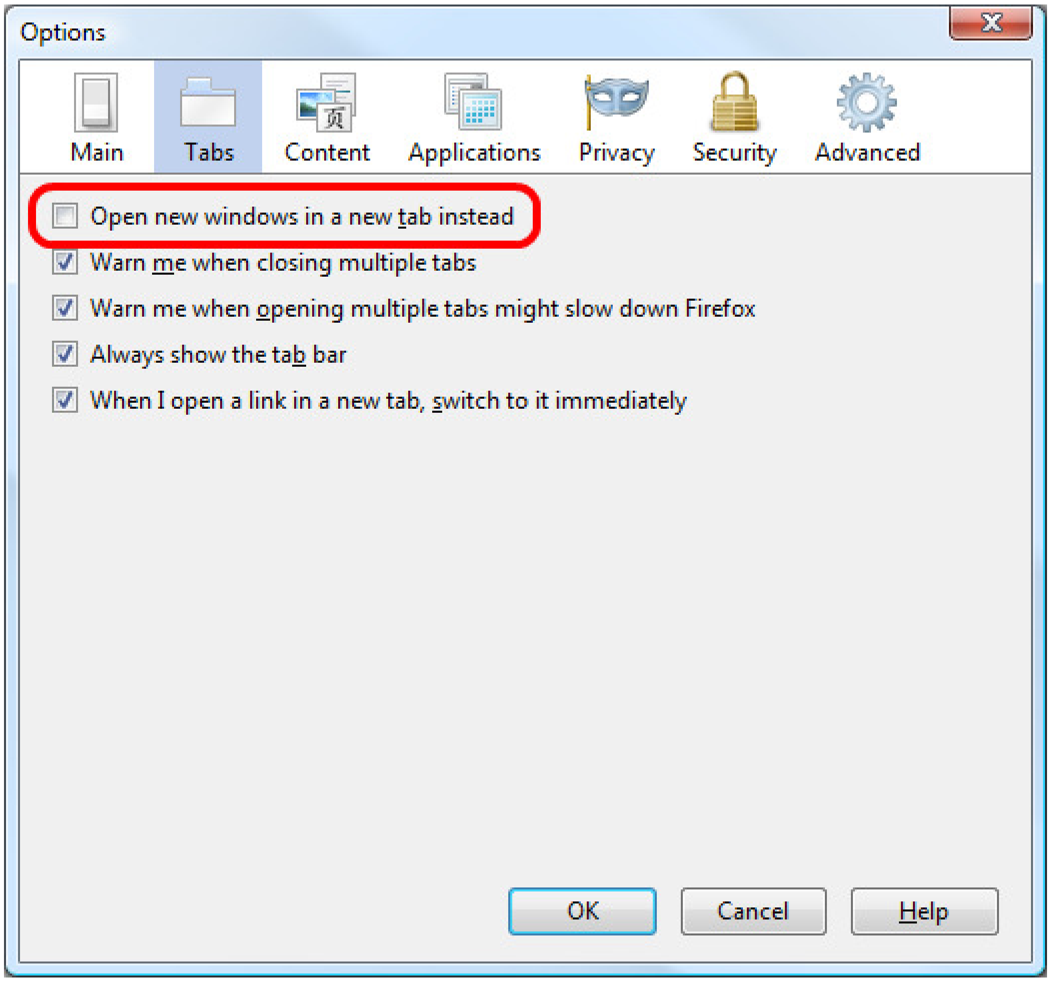


New pages should be opened in:

- [✓ a new window]
- a new tab

- [✓] Warn me when closing multiple tabs
- [✓] Warn me when opening multiple tabs might slow down Firefox
- [ ] Always show the tab bar
- [ ] When I open a link in a new tab, switch to it immediately

Firefox 3.0.x



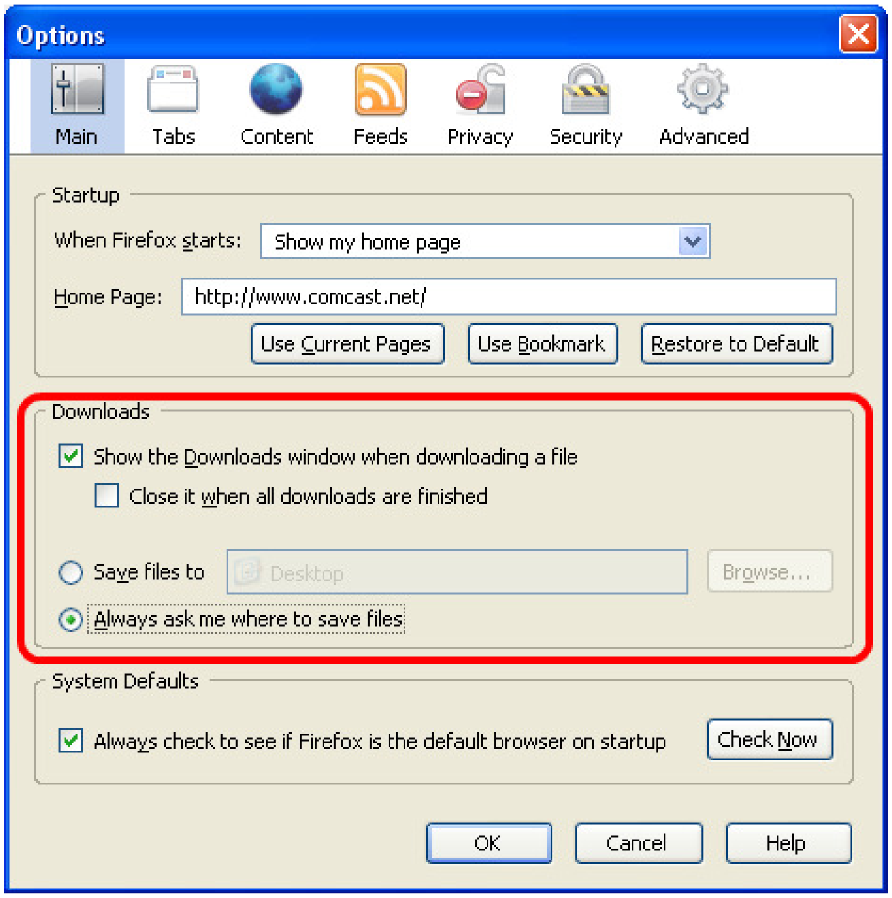
- [ ] Open new windows in a new tab instead
- [x] Warn me when closing multiple tabs
- [x] Warn me when opening multiple tabs might slow down Firefox
- [x] Always show the tab bar
- [x] When I open a link in a new tab, switch to it immediately

Firefox 3.5.x

2.3.2.3 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**".



Startup

When Firefox starts: Show my home page

Home Page: http://www.comcast.net/

Downloads

[✓] Show the Downloads window when downloading a file

[ ] Close it when all downloads are finished

( ) Save files to Desktop

(●) Always ask me where to save files

System Defaults

[✓] Always check to see if Firefox is the default browser on startup

2.4 Setting up Your Automation Design Canvas Windows

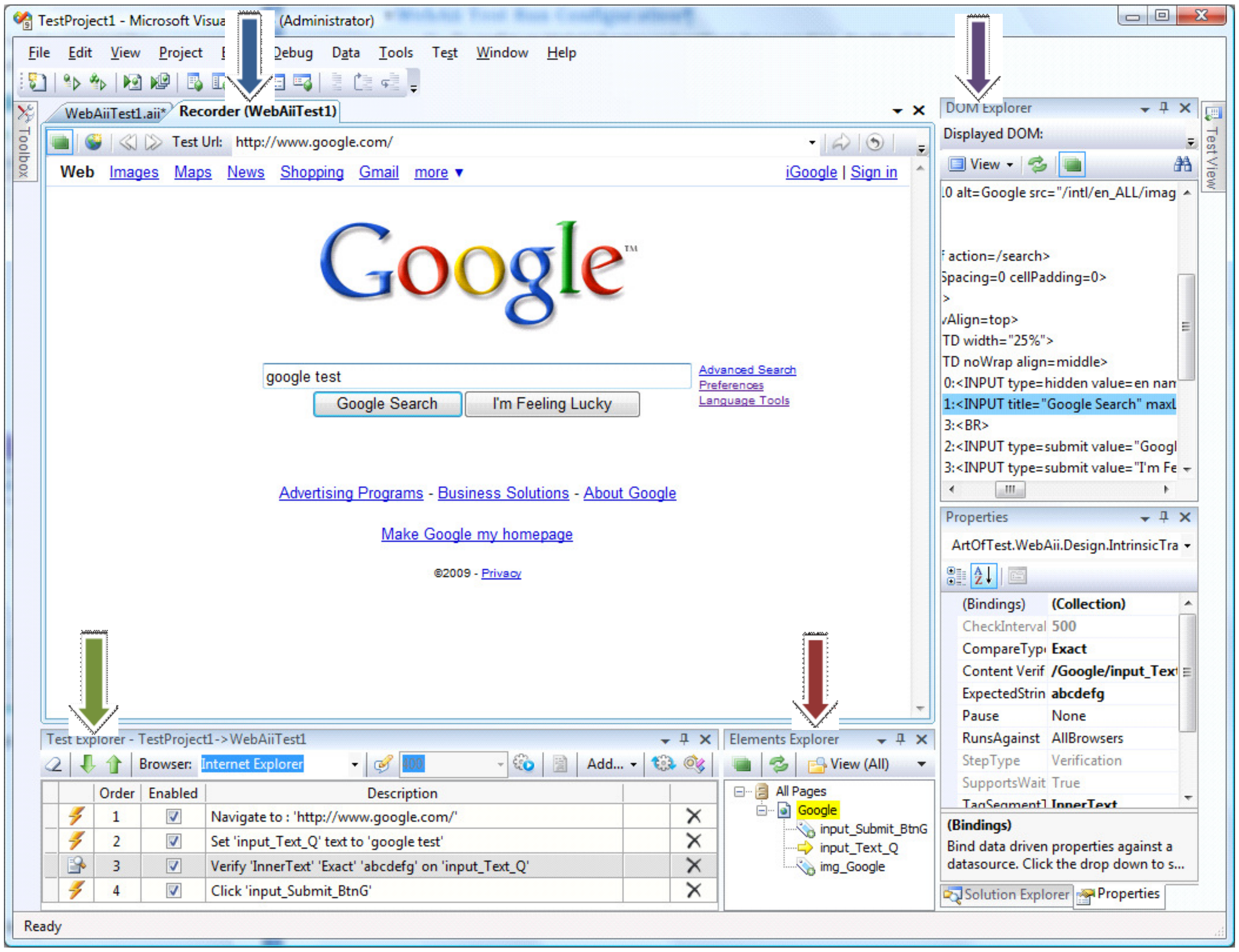
Automation Design Canvas provides the following main windows:

- **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.

2.4.1 Suggested Design Canvas Window Layout

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



```
.0 alt=Google src="/intl/en_ALL/imag
 action=/search>
Spacing=0 cellPadding=0>
>
vAlign=top>
TD width="25%">
TD noWrap align=middle>
0:<INPUT type=hidden value=en nan
1:<INPUT title="Google Search" maxL
3:<BR>
2:<INPUT type=submit value="Googl
3:<INPUT type=submit value="I'm Fe
```

Order	Enabled	Description
1	✓	Navigate to : 'http://www.google.com/'
2	✓	Set 'input_Text_Q' text to 'google test'
3	✓	Verify 'InnerText' 'Exact' 'abcdefg' on 'input_Text_Q'
4	✓	Click 'input_Submit_BtnG'

(Bindings)	(Collection)
CheckInterval	500
CompareTyp	Exact
Content Verif	/Google/input_Text
ExpectedStrin	abcdefg
Pause	None
RunsAgainst	AllBrowsers
StepType	Verification
SupportsWait	True
TagSegment1	InnerText

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

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

2.4.2 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 last selected becomes the active test in the **Recording Surface**.

Section 5 illustrates a typical Recording Surface window.

2.4.3 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 9 describes how to use **Test Explorer** window.

2.4.4 DOM Explorer

The **DOM Explorer** is a dock-able floating window that 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 in the webpage. You can also select an element from this window to perform an action on that you cannot select in the **Recording Surface**. Section 10 explains how to use the **DOM Explorer** window.

2.4.5 Elements Explorer

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

- Change the friendly name of the element
- Change the way an element is found (i.e. change its Find Expression)
- Highlight the selected element in the **Recording Surface** to see where it is located on the webpage

Section 0 describes how to use **Elements Explorer** window.

2.4.6 Storyboard

The storyboard appears on the Storyboard 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 6.2.1 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 answers to these questions:

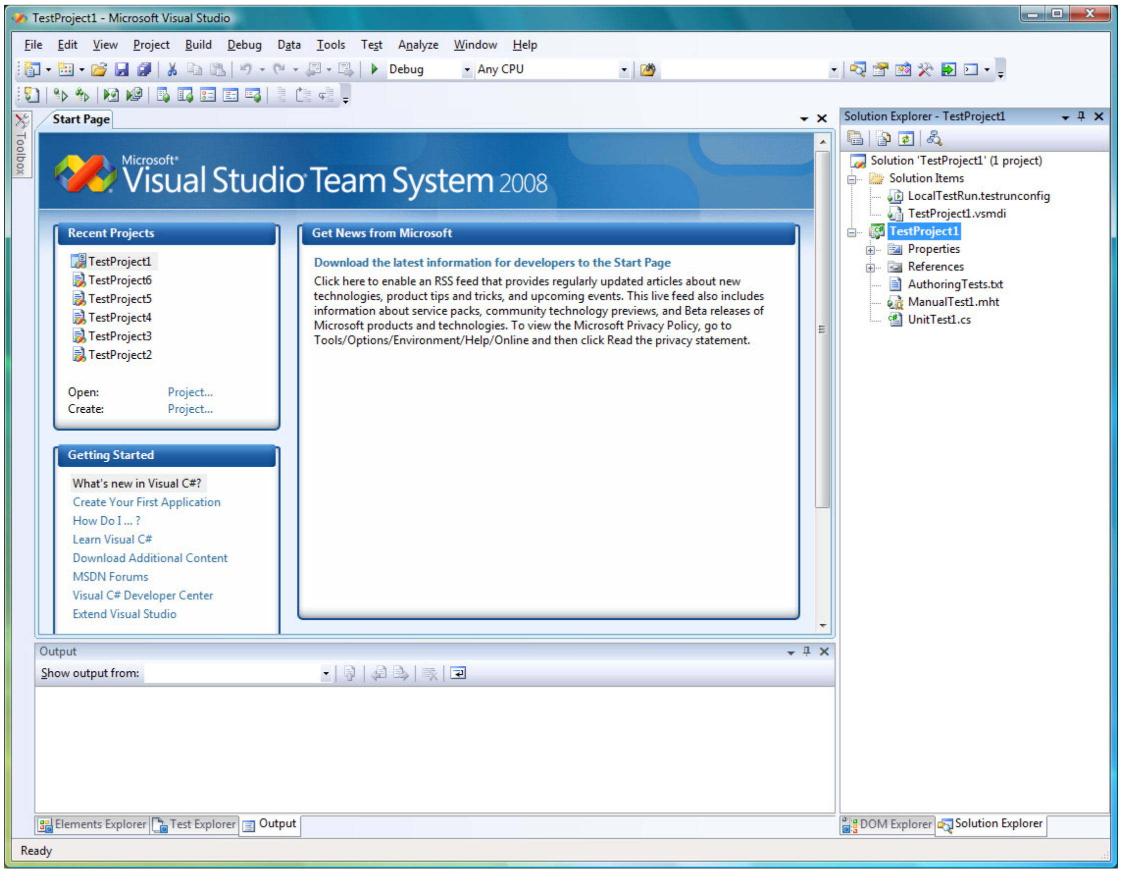
Question	Description
Web application name	
Site addresses of your web applications to be tested	
Determine which browsers you will be testing against	
Detailed listing of functions and features to be tested	
Documented test cases and scenarios if applicable	
Location of or documented data points to use in testing each field on each page	
Listing of expected results if not already included in the test cases	
Will your test involve downloading or other native Win32 windows?	

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:

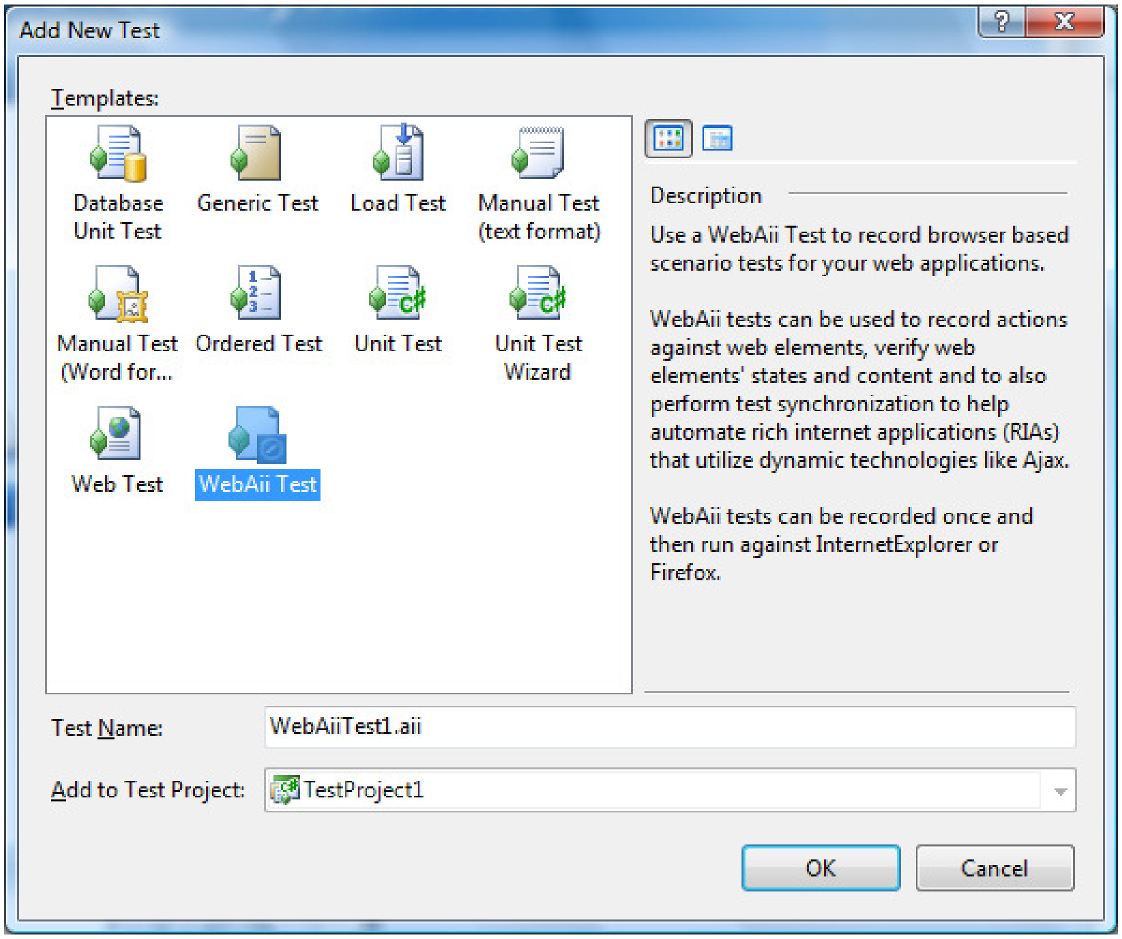
- Open Visual Studio and create a new Visual C# or Visual Basic test project.



Download the latest information for developers to the Start Page

Click here to enable an RSS feed that provides regularly updated articles about new technologies, product tips and tricks, and upcoming events. This live feed also includes information about service packs, community technology previews, and Beta releases of Microsoft products and technologies. To view the Microsoft Privacy Policy, go to Tools/Options/Environment/Help/Online and then click Read the privacy statement.

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



Description

Use a WebAii Test to record browser based scenario tests for your web applications.

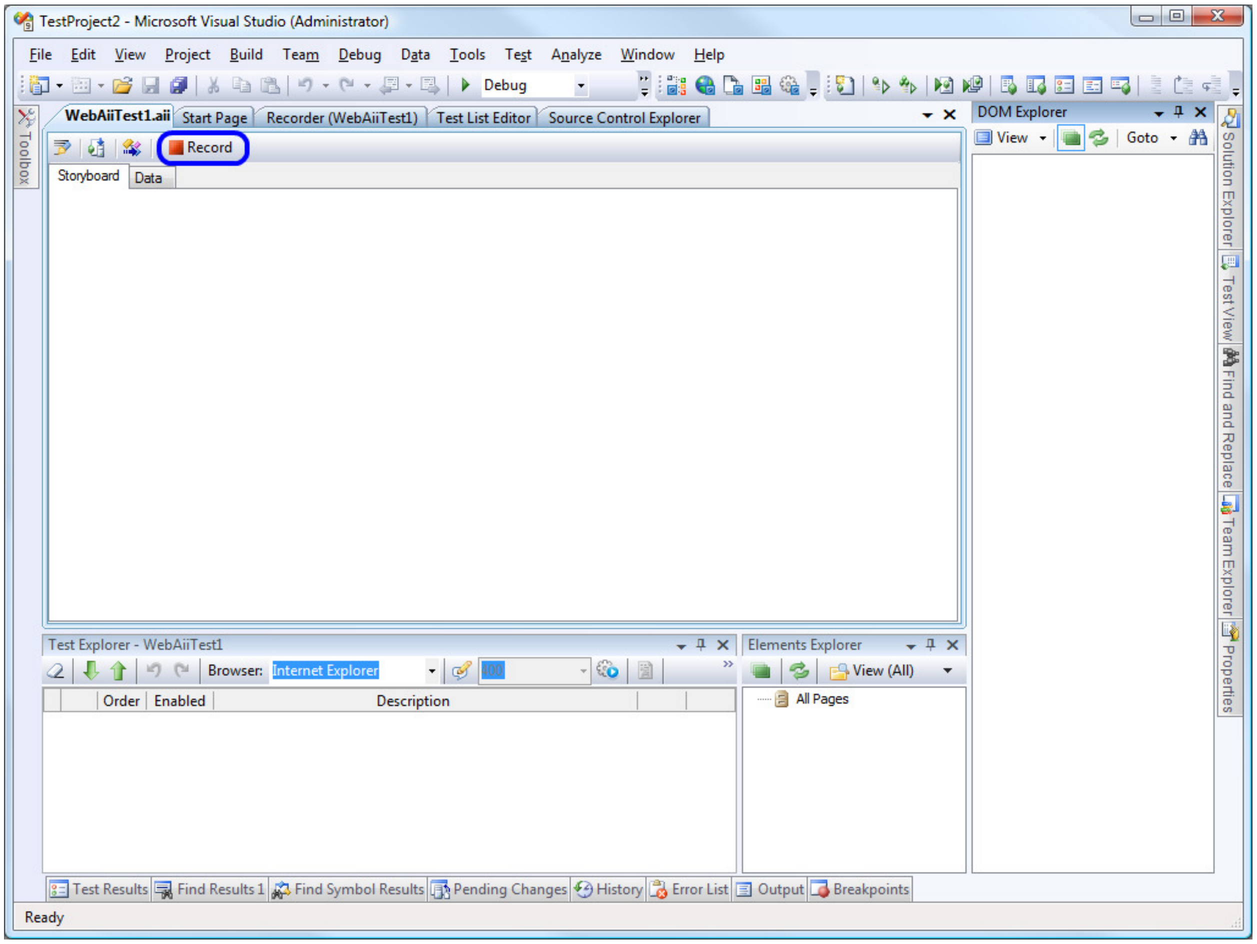
WebAii tests can be used to record actions against web elements, verify web elements' states and content and to also perform test synchronization to help automate rich internet applications (RIAs) that utilize dynamic technologies like Ajax.

WebAii tests can be recorded once and then run against InternetExplorer or Firefox.

Test Name: WebAiiTest1.aii

Add to Test Project: TestProject1

- 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.

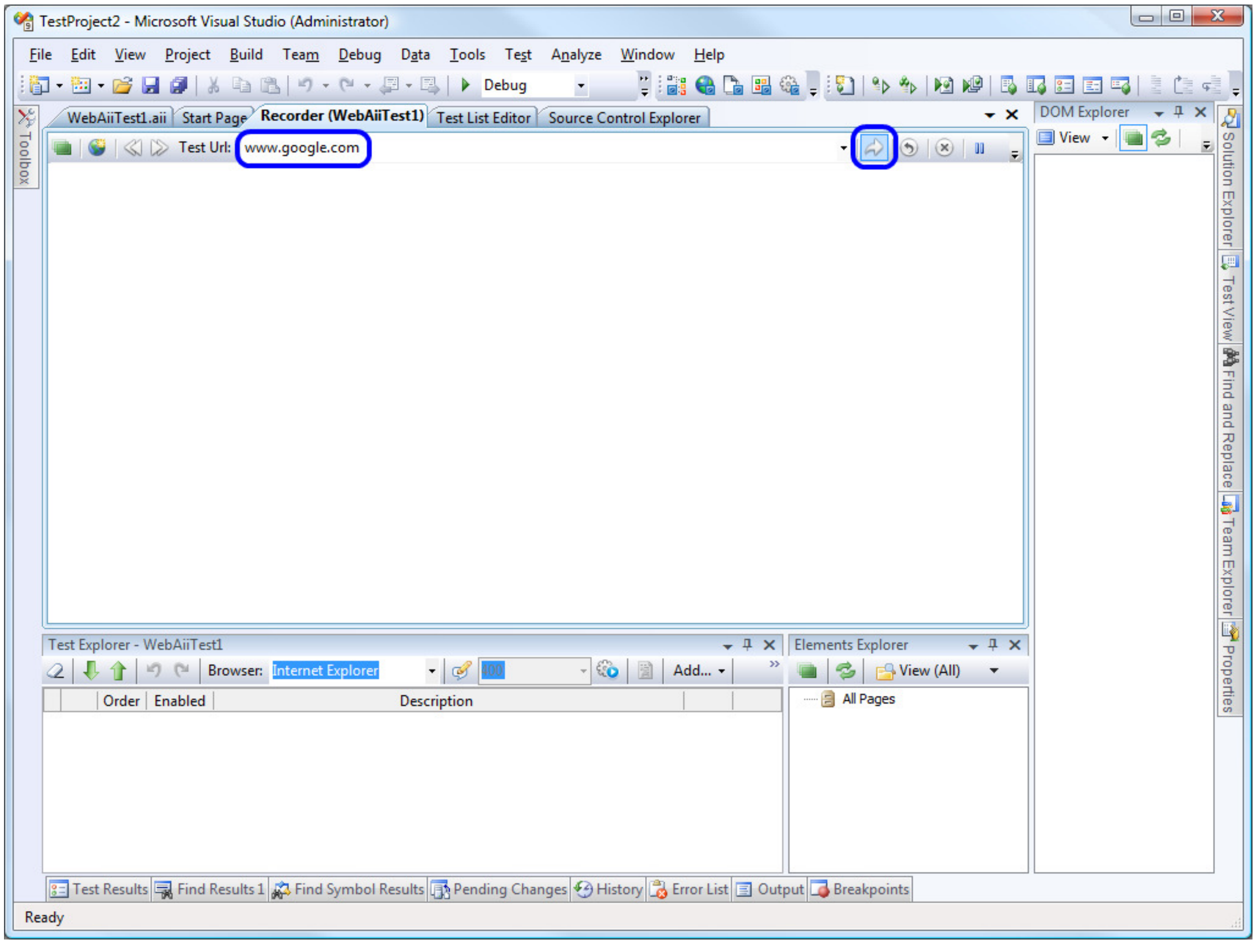


If this is your first time loading a WebAii Test see "Suggested Window Layout" in section 2.4.1.

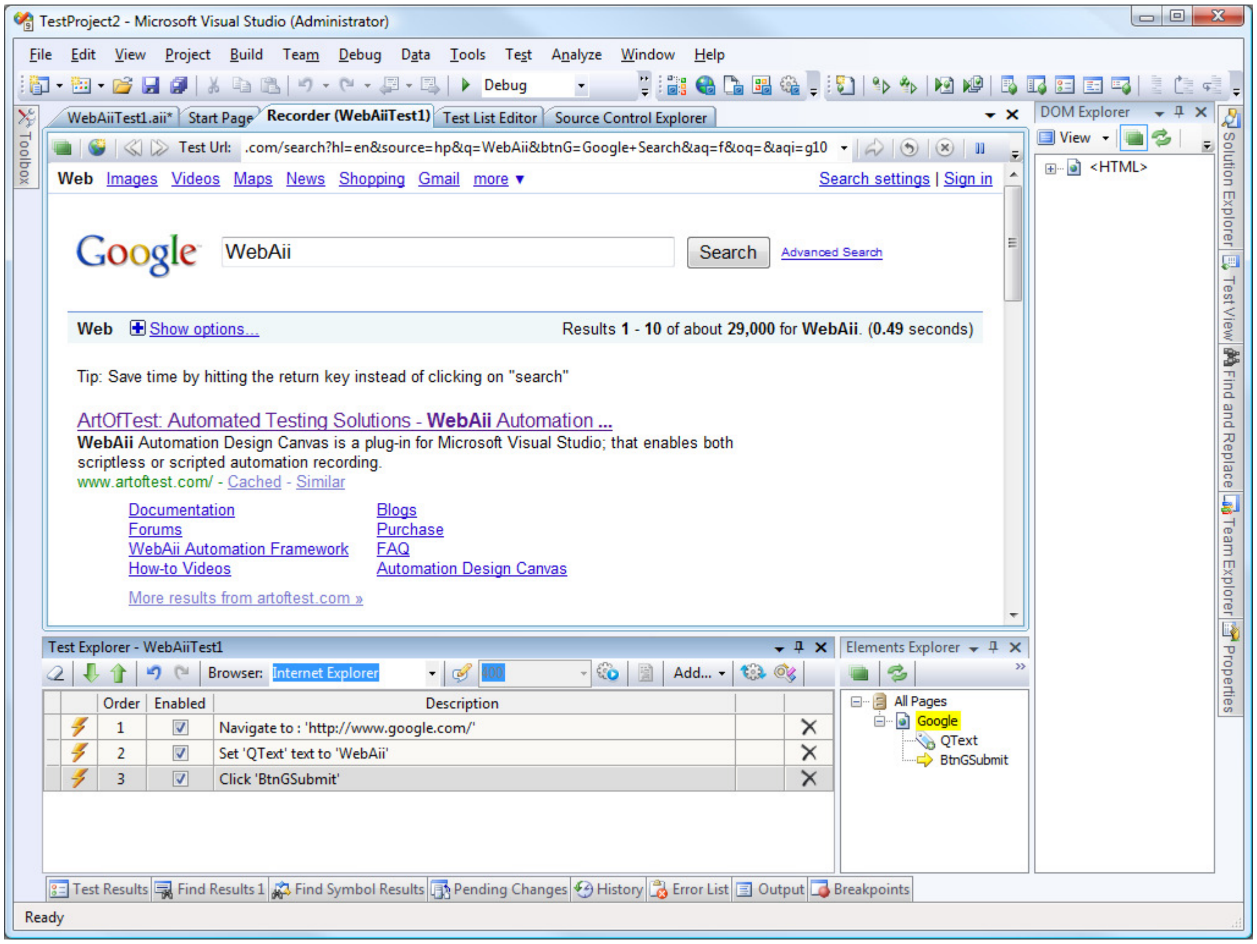
4.1.1 Recording a Test

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.


TestProject2 - Microsoft Visual Studio (Administrator)
File Edit View Project Build Team Debug Data Tools Test Analyze Window Help
WebAiiTest1.aii* Start Page Recorder (WebAiiTest1) Test List Editor Source Control Explorer
DOM Explorer
Test Explorer - WebAiiTest1
Elements Explorer
Ready

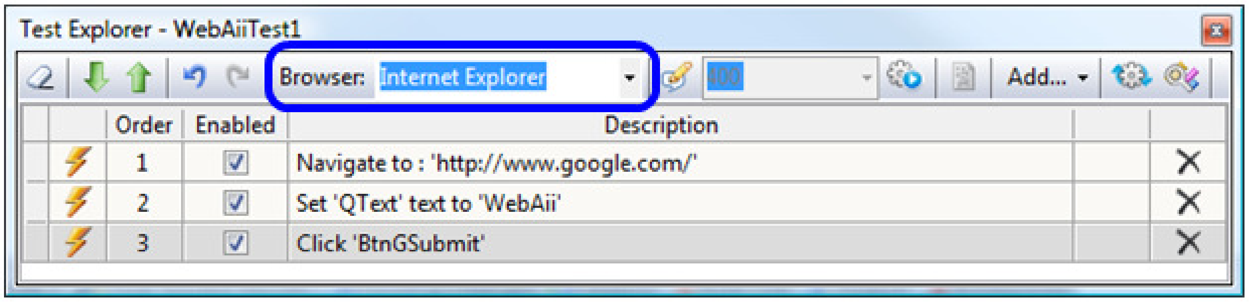
Test Url: .com/search?hl=en&source=hp&q=WebAii&btnG=Google+Search&aq=f&oq=&aqi=g10

Browser: Internet Explorer

Order	Enabled	Description
1	☑	Navigate to : 'http://www.google.com/'
2	☑	Set 'QText' text to 'WebAii'
3	☑	Click 'BtnGSubmit'

All Pages
Google
QText
BtnGSubmit

- 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:



Browser: Internet Explorer

Order	Enabled	Description
1	☑	Navigate to : 'http://www.google.com/'
2	☑	Set 'QText' text to 'WebAii'
3	☑	Click 'BtnGSubmit'

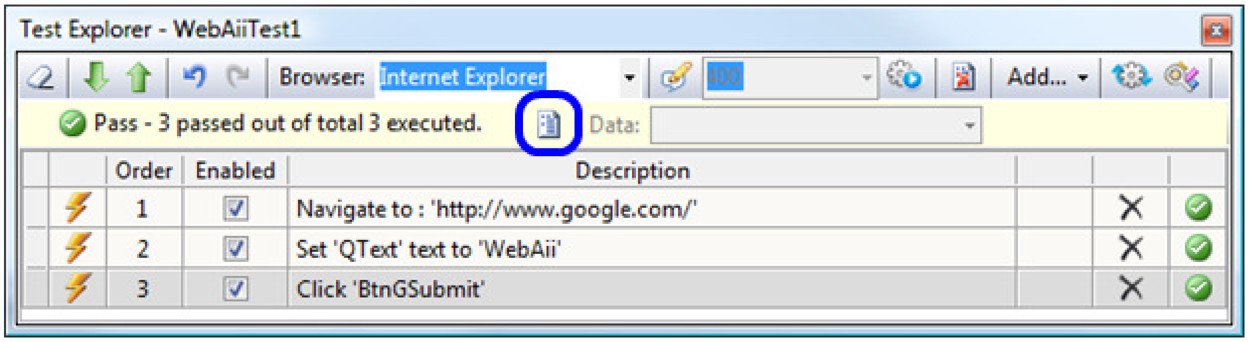
That’s how easy it is to create your first WebAii test! Save and build your project.

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.

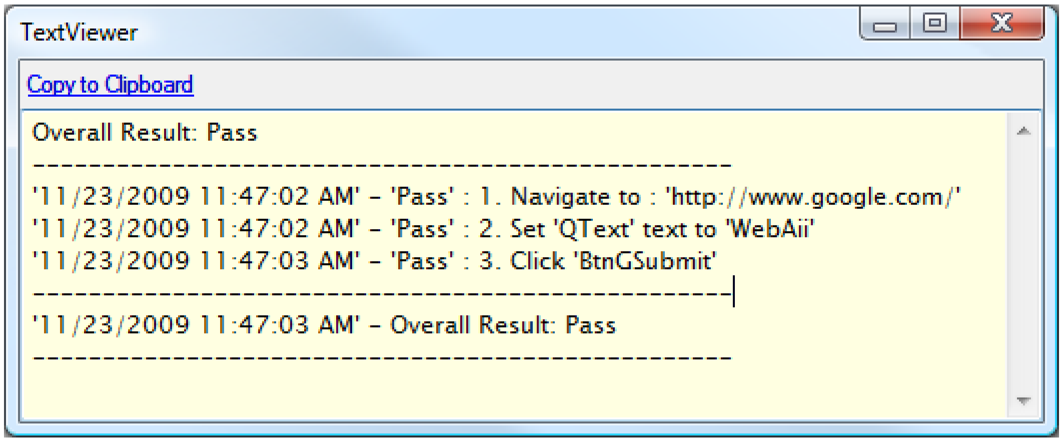
- An Internet Explorer window will open and the steps of your test automatically execute in the browser window. When the test completes the browser window automatically close and a summary of the test results is displayed in the Test Explorer window.



Pass - 3 passed out of total 3 executed.

Order	Enabled	Description
1	☑	Navigate to : 'http://www.google.com/'
2	☑	Set 'QText' text to 'WebAii'
3	☑	Click 'BtnGSubmit'

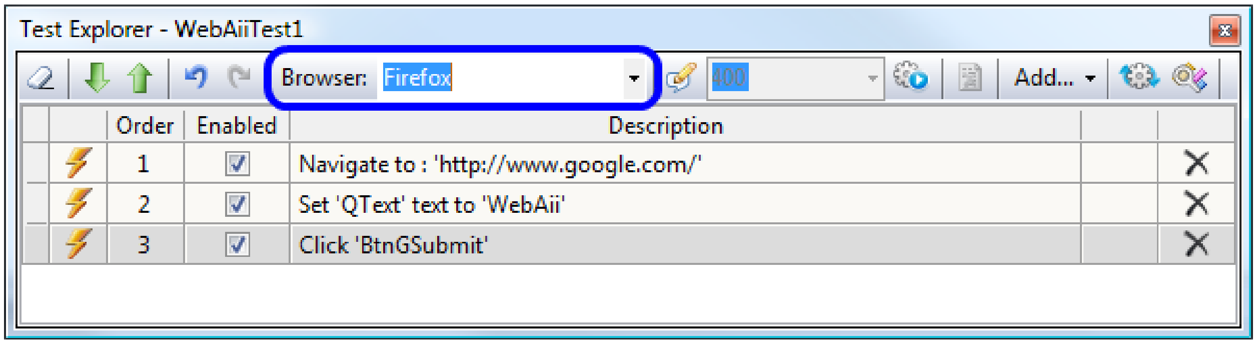
- To see detailed test results click the "View test log" button shown outlined in blue in the Test Explorer window. A TextViewer window opens displaying the log of the test run.



```
Overall Result: Pass
------------------------------------------------------
'11/23/2009 11:47:02 AM' - 'Pass' : 1. Navigate to : 'http://www.google.com/'
'11/23/2009 11:47:02 AM' - 'Pass' : 2. Set 'QText' text to 'WebAii'
'11/23/2009 11:47:03 AM' - 'Pass' : 3. Click 'BtnGSubmit'
------------------------------------------------------
'11/23/2009 11:47:03 AM' - Overall Result: Pass
------------------------------------------------------
```

4.1.3 Running Your Test Using Quick Execute (Firefox)

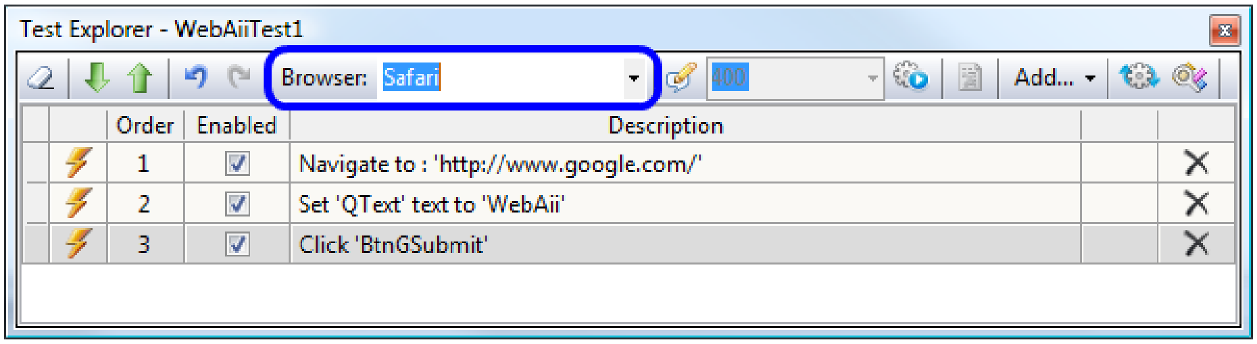
Quick executing your test in Firefox is the same as running it in IE. Follow the procedure described in section 4.1.2 except select Firefox as the browser in Test Explorer instead of the default Internet Explorer.



Order	Enabled	Description
1	☑	Navigate to : 'http://www.google.com/'
2	☑	Set 'QText' text to 'WebAii'
3	☑	Click 'BtnGSubmit'

4.1.4 Running Your Test Using Quick Execute (Safari)

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


Test Explorer - WebAiiTest1

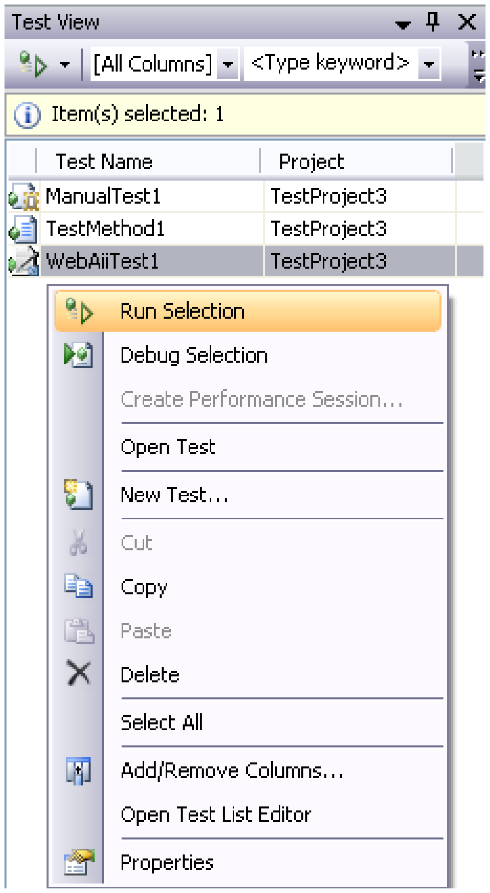
Browser: Safari

Order	Enabled	Description
1	☑	Navigate to : 'http://www.google.com/'
2	☑	Set 'QText' text to 'WebAii'
3	☑	Click 'BtnGSubmit'

4.1.5 Running Your Test in Visual Studio Test Manager

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

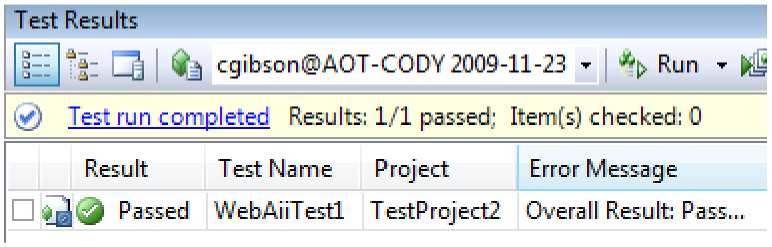
- Open the "Test View" tool window. This can be found by clicking **Test** in the main menu then **Windows** then **Test View**.
- Right click on your WebAii Test and click **Run Selection**.


Test View

Item(s) selected: 1

Test Name	Project
ManualTest1	TestProject3
TestMethod1	TestProject3
WebAiiTest1	TestProject3

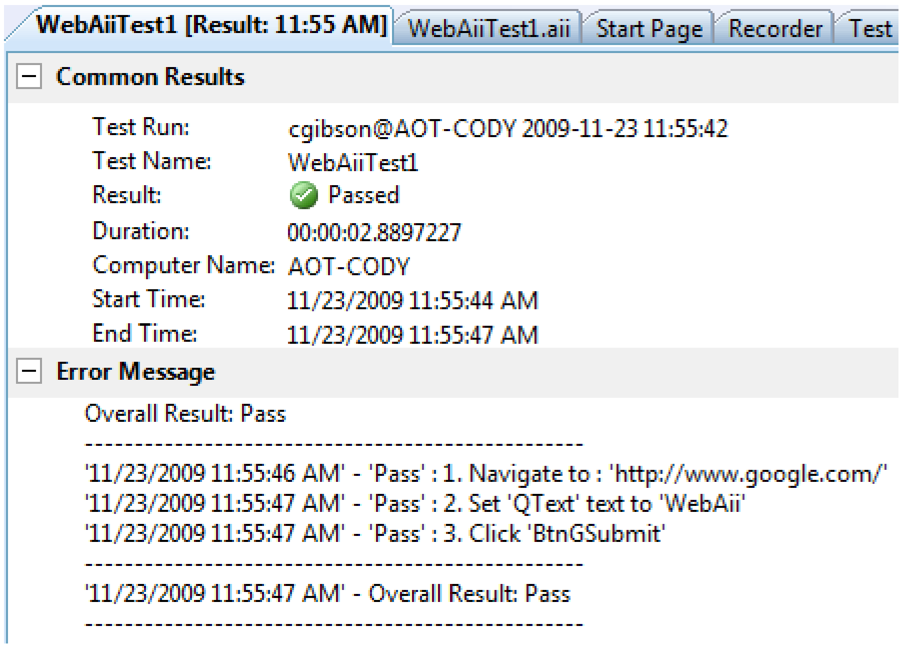
- An Internet Explorer window opens and the steps of your test automatically execute in that browser window. When the test completes the browser window automatically closes.



Test run completed Results: 1/1 passed; Item(s) checked: 0

Result	Test Name	Project	Error Message
Passed	WebAiiTest1	TestProject2	Overall Result: Pass...

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



Common Results

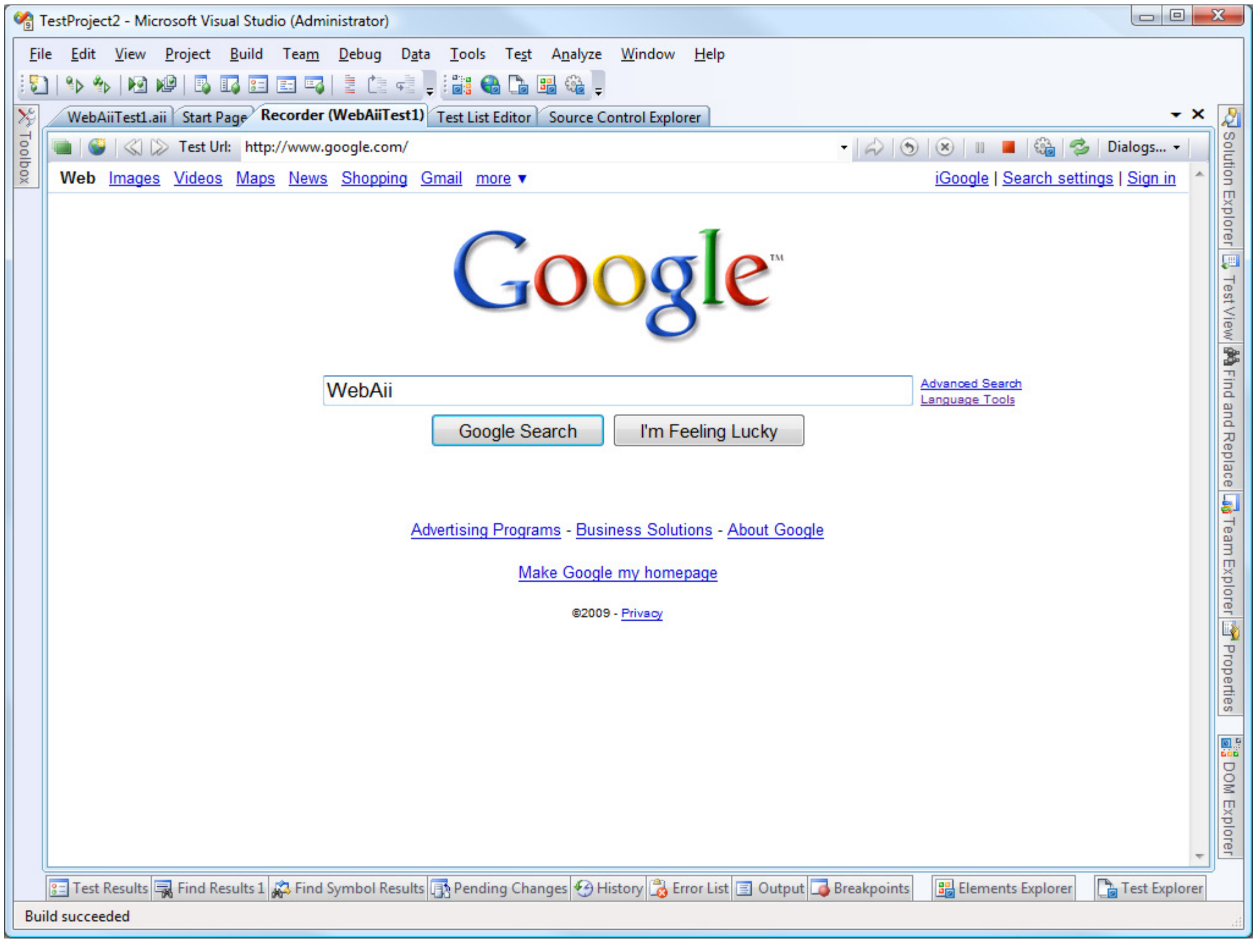
Test Run:	cgibson@AOT-CODY 2009-11-23 11:55:42
Test Name:	WebAiiTest1
Result:	Passed
Duration:	00:00:02.8897227
Computer Name:	AOT-CODY
Start Time:	11/23/2009 11:55:44 AM
End Time:	11/23/2009 11:55:47 AM

Error Message

```
Overall Result: Pass
------------------------------------------------
'11/23/2009 11:55:46 AM' - 'Pass' : 1. Navigate to : 'http://www.google.com/'
'11/23/2009 11:55:47 AM' - 'Pass' : 2. Set 'QText' text to 'WebAii'
'11/23/2009 11:55:47 AM' - 'Pass' : 3. Click 'BtnGSubmit'
------------------------------------------------
'11/23/2009 11:55:47 AM' - Overall Result: Pass
------------------------------------------------
```

5 Design Canvas Test Recorder

The Design Canvas Recording Surface window has an integrated control called the Automation Overlay Surface™ (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. Most actions performed against a web page can be recorded by just performing the action directly on the Recording Surface.


TestProject2 - Microsoft Visual Studio (Administrator)
File Edit View Project Build Team Debug Data Tools Test Analyze Window Help
WebAiiTest1.aii Start Page Recorder (WebAiiTest1) Test List Editor Source Control Explorer
Test Url: http://www.google.com/
Dialogs...
Web Images Videos Maps News Shopping Gmail more
iGoogle | Search settings | Sign in
WebAii
Google Search
I'm Feeling Lucky
Advanced Search
Language Tools
Advertising Programs - Business Solutions - About Google
Make Google my homepage
©2009 - Privacy
Toolbox
Solution Explorer
Test View
Find and Replace
Team Explorer
Properties
DOM Explorer
Test Results Find Results 1 Find Symbol Results Pending Changes History Error List Output Breakpoints Elements Explorer Test Explorer
Build succeeded

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

5.1 Test Url

Test Url: http://www.google.com/preferences?hl=en

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 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 in 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.

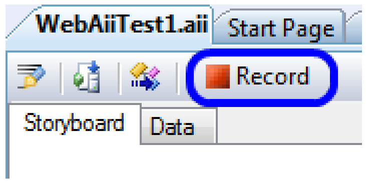
5.2.4 Back

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 turns on recording. 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 Automation Design Canvas Settings

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

5.5 Launch External IE Browser for Recording

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

There are some test automation scenarios where you will need to start with the external IE browser. For example if you're dealing with pop-up windows that rely on logon information being entered in the main window. Logon information isn't communicated between popup windows and the recording surface contained in Visual Studio.

5.6 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 handlers for the most common Win32 dialogs displayed by the browser. Section 5.6.1 describes dialog handlers in more detail.

5.6.1 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 on 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 to your test.

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.

5.6.1.1 JavaScript Alert dialogs

The JavaScript alert dialog handler is used as the standard JavaScript security alert dialog for Internet Explorer, Firefox and Safari. 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 the 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.

5.6.1.2 File upload dialogs

The file upload dialog handler handles the standard file upload dialogs for Internet Explorer, Firefox and Safari. 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 then when you run your test the file upload dialog will open and 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 really go away.
- 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 the 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.

5.6.1.3 File download dialogs

The file download dialog handler handles the standard file download dialogs for Internet Explorer, Firefox and Safari. 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 path. If you don't then when you run your test 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 really away.
- 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 the 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.

5.6.1.4 Logon dialogs

The logon dialog handler handles the standard logon dialog for Internet Explorer, Firefox and Safari. 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 the 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.

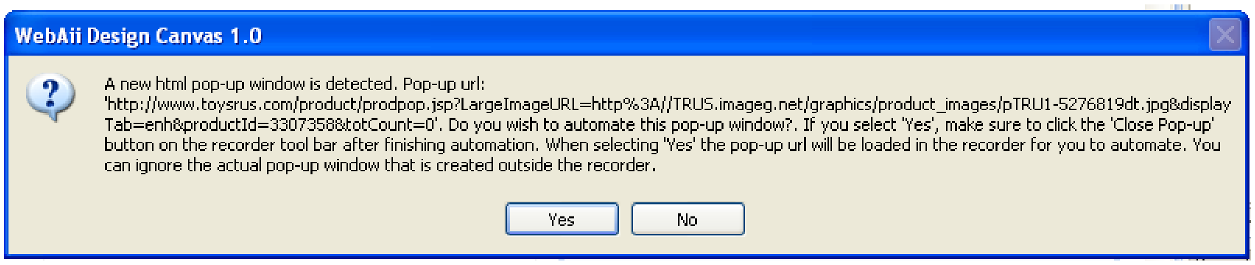
5.6.1.5 Generic dialog handler

The generic dialog handler is a special handler 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 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 the 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.

5.6.2 Handling HTML Pop-ups


WebAii Design Canvas 1.0

A new html pop-up window is detected. Pop-up url: 'http://www.toysrus.com/product/prodpop.jsp?LargeImageURL=http%3A//TRUS.imageg.net/graphics/product_images/pTRU1-5276819dt.jpg&displayTab=enh&productId=3307358&totCount=0'. Do you wish to automate this pop-up window?. If you select 'Yes', make sure to click the 'Close Pop-up' button on the recorder tool bar after finishing automation. When selecting 'Yes' the pop-up url will be loaded in the recorder for you to automate. You can ignore the actual pop-up window that is created outside the recorder.

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 we put significant effort into Design Canvas 1.1, and carried over this support in Design Canvas 2,0, 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.

5.6.2.1 Changes in 1.1

Prior to V1.1, 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.

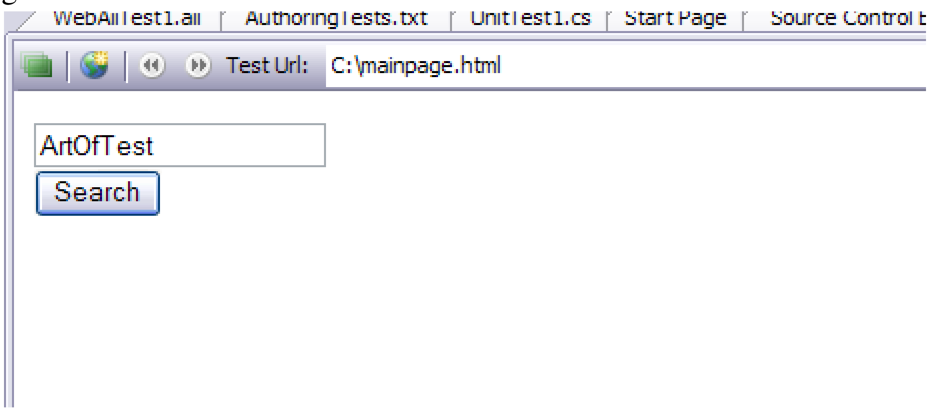
The biggest fundamental enhancement to Design Canvas V1.1, and carried over to V2.0, is the ability to record against an independent instance of Internet Explorer in place of an instance of IE that is hosted inside of Visual Studio. This enhancement enables recording from several instances of IE simultaneously which makes the handling and recording of pop-up scenarios much easier.

We no longer attempt to intercept the HTML pop-up. Instead we simply allow the pop-up to open naturally and attach a recorder to each instance as it opens. 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.

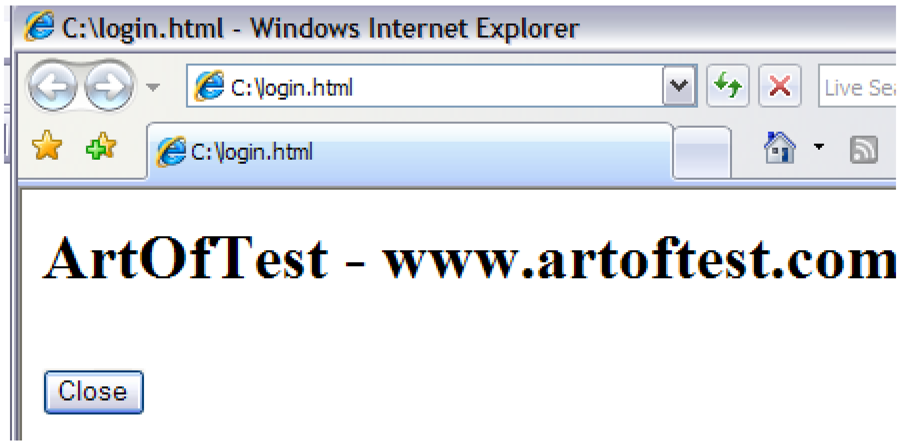
5.6.2.2 Handling HTML pop-up recording in V1.1 and V2.0.

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

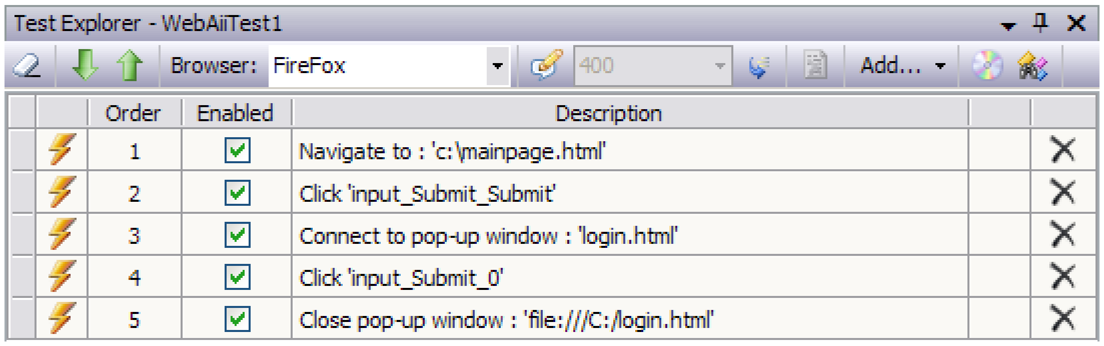


Clicking on the "Search" button launches the html pop-up (**login.html**)



ArtOfTest - www.artoftest.com

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

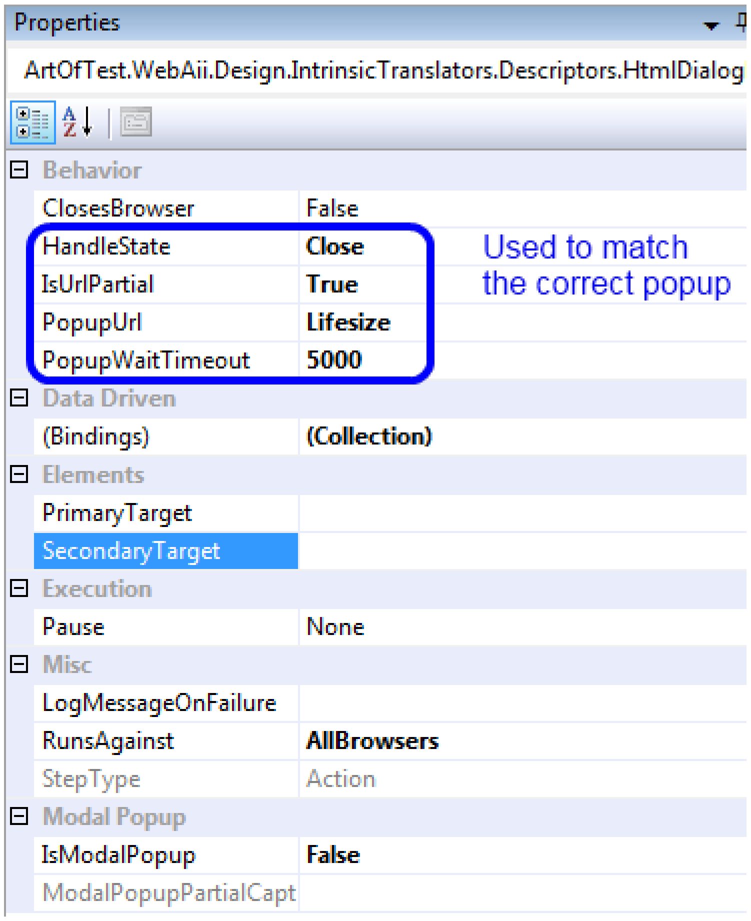


	Order	Enabled	Description		
⚡	1	✓	Navigate to : 'c:\mainpage.html'		✕
⚡	2	✓	Click 'input_Submit_Submit'		✕
⚡	3	✓	Connect to pop-up window : 'login.html'		✕
⚡	4	✓	Click 'input_Submit_0'		✕
⚡	5	✓	Close pop-up window : 'file:///C:/login.html'		✕

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:

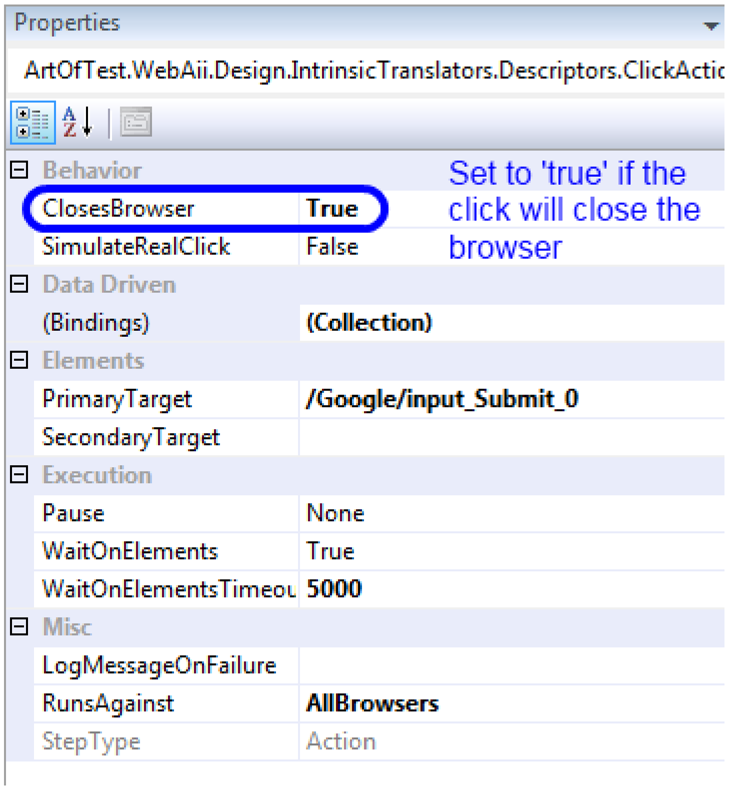
- **Click 'input_Submit_Submit':** Clicks the button that launches the pop-up. In most cases there is no need to make any changes to the properties of this step.
- **Connect to the pop-up window: 'login.html'**. Make sure the Popup URL captured in the properties (as shown below) will match the popup


Properties

ArtOfTest.WebAii.Design.IntrinsicTranslators.Descriptors.HtmlDialog

Behavior	
ClosesBrowser	False
HandleState	Close
IsUrlPartial	True
PopupUrl	Lifesize
PopupWaitTimeout	5000
Data Driven	
(Bindings)	(Collection)
Elements	
PrimaryTarget	
SecondaryTarget	
Execution	
Pause	None
Misc	
LogMessageOnFailure	
RunsAgainst	AllBrowsers
StepType	Action
Modal Popup	
IsModalPopup	False
ModalPopupPartialCapt	

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


Properties

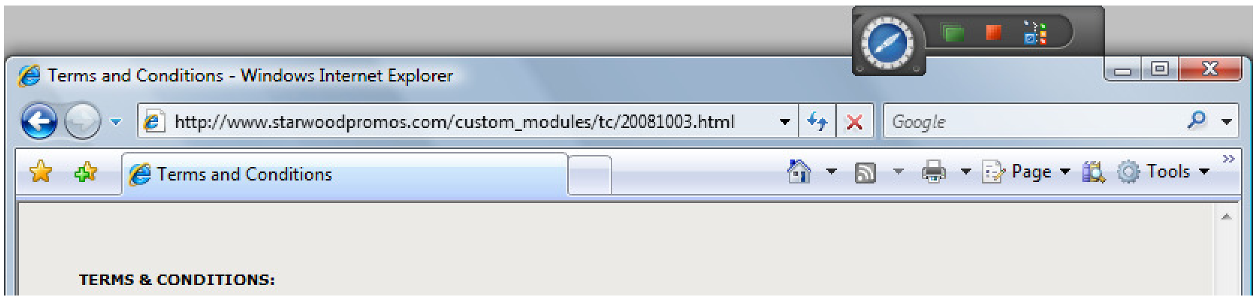
ArtOfTest.WebAii.Design.IntrinsicTranslators.Descriptors.ClickActic

Behavior	
ClosesBrowser	True
SimulateRealClick	False
Data Driven	
(Bindings)	(Collection)
Elements	
PrimaryTarget	/Google/input_Submit_0
SecondaryTarget	
Execution	
Pause	None
WaitOnElements	True
WaitOnElementsTimeou	5000
Misc	
LogMessageOnFailure	
RunsAgainst	AllBrowsers
StepType	Action

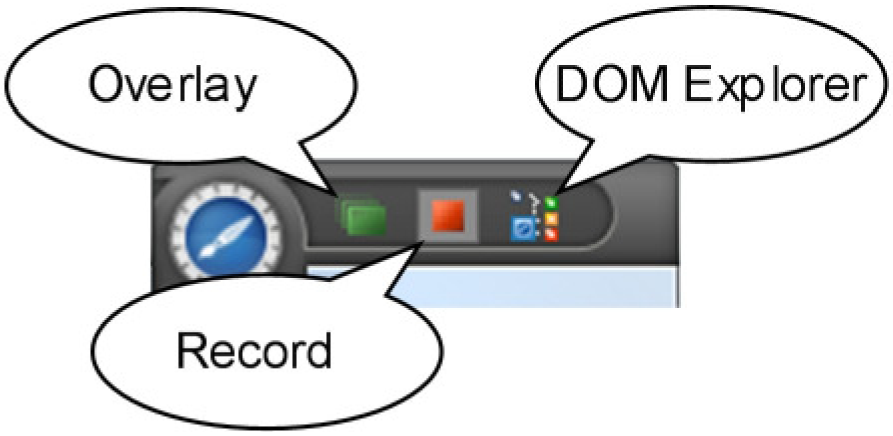
- In this scenario, this click will close the browser so we set '**ClosesBrowser**' = true.
- **Close pop-up window.** This step will close the pop-up window.

5.6.2.3 Recording From External 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.

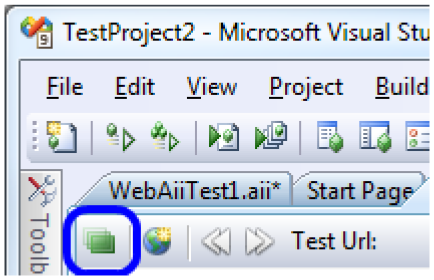

Terms and Conditions - Windows Internet Explorer
http://www.starwoodpromos.com/custom_modules/tc/20081003.html
Google
Terms and Conditions
Page
Tools
TERMS & CONDITIONS:

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

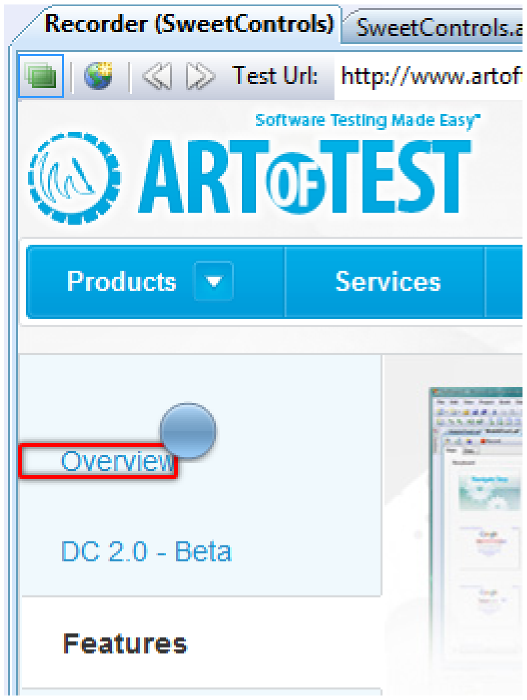


6 Automation Overlay Surface

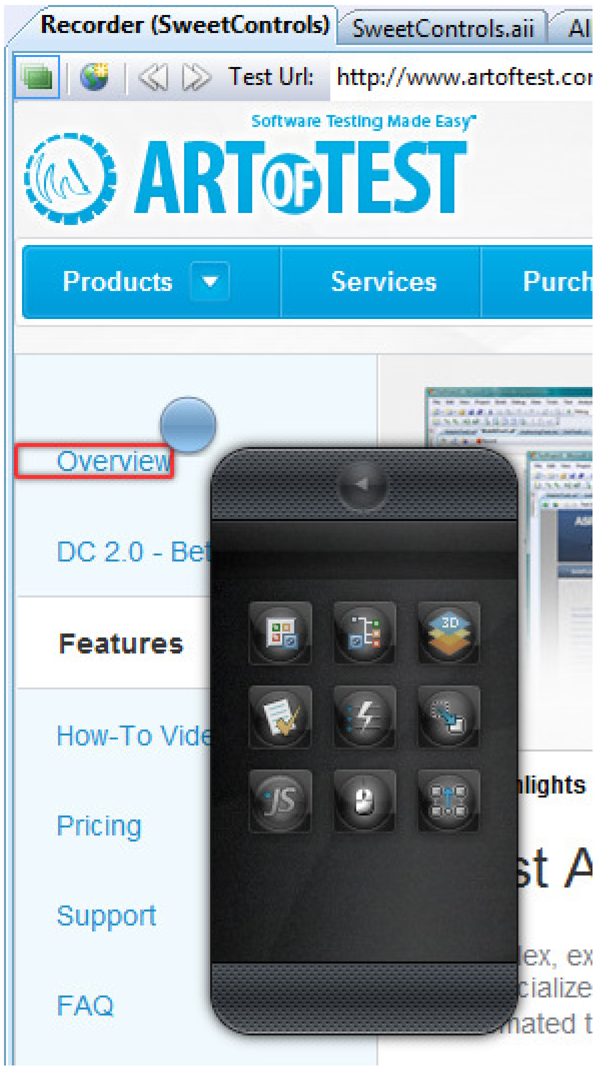
Automation Overlay Surface™ (AOS) enables you to visually highlight an element, locate it in the DOM tree, and automatically generate common verification or synchronization tasks against it. These generated tasks are automatically added to your test as steps. For example, you can verify an item is selected 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 blue outline:



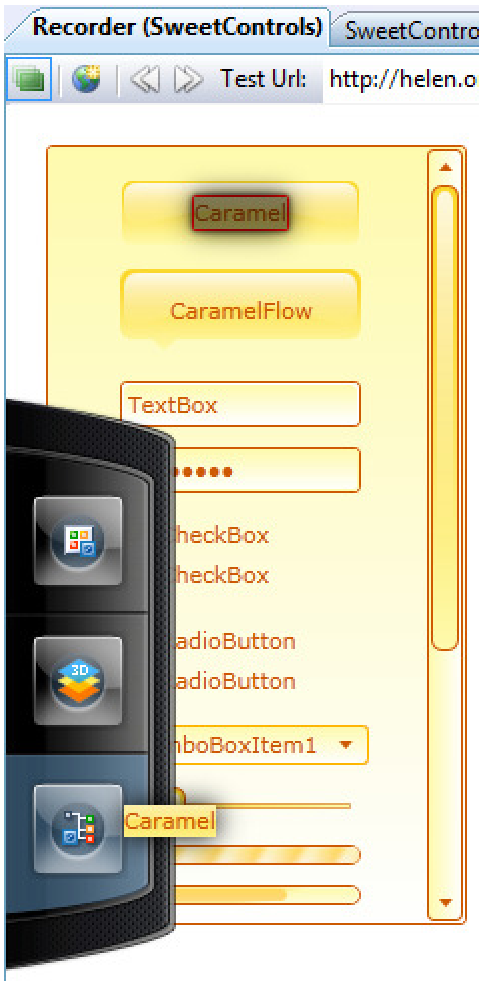
Once enabled hovering the mouse over any element in the Recording Surface will highlight that element by drawing a rectangle around it. After an element is highlighted, wait one second and a little blue nub will appear as shown below:



If you click on the blue nub you get an element menu. The Element Menu is described in detail in section 6.2.



If you drag the blue nub the pop-out menu will open:



The pop-out menu contains three buttons

6.1 Pop-out Menu

6.1.1 Add Element to Project

Dragging the blue nub to the top button will add the highlighted element to Elements Explorer.

6.1.2 View in 3D

Dragging the blue nub to the middle button will open the 3D viewer focused on the highlighted element. The 3D viewer is detailed in section 16.

6.1.3 Locate in DOM

Dragging the blue nub to the bottom button will locate and highlight the element in DOM Explorer that's highlighted in the recording surface.

6.2 Element Menu

The element menu is a collection of tools for either examining or acting on the highlighted element. It contains 9 buttons which are described below:

6.2.1 Add Element to Project Elements

Clicking this button will make Design Canvas automatically determine the best Find Expression to use for locating the element on the web page and then adds it to Elements Explorer. Once added you can refer to it by the friendly name in custom code.

6.2.2 Locate Element in DOM Explorer

Clicking this button will highlight the element in DOM Explorer.

6.2.3 Open Element in 3D Viewer

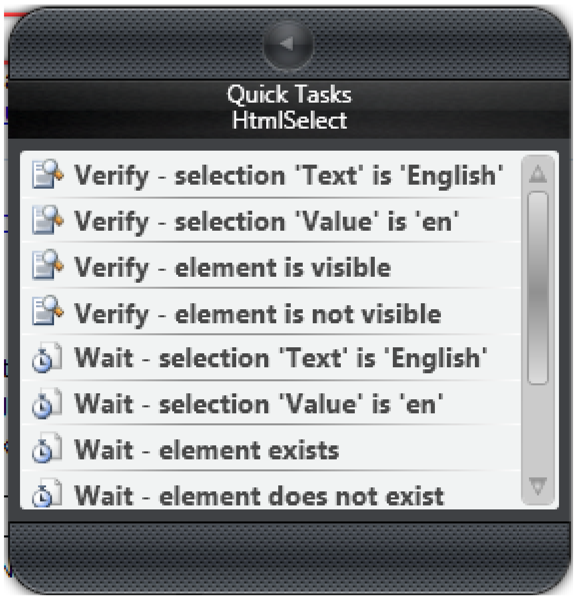
Clicking this button will activate the **3D Viewer** with this element as the starting element. The 3D viewer is described in detail in section ???.

6.2.4 Open Sentence Verification Builder

Clicking this button will open the **Sentence Verification Builder** for this element. The Sentence Verification Builder is described in detail in section 15.

6.2.5 Quick Tasks

Clicking this button opens the **Quick Tasks** menu for the element. The list of available quick tasks depends on the type of element selected. Only tasks that are appropriate for the type of element selected are included in the list. Double clicking any quick task item in the list will add that task as a test step in Test Explorer.



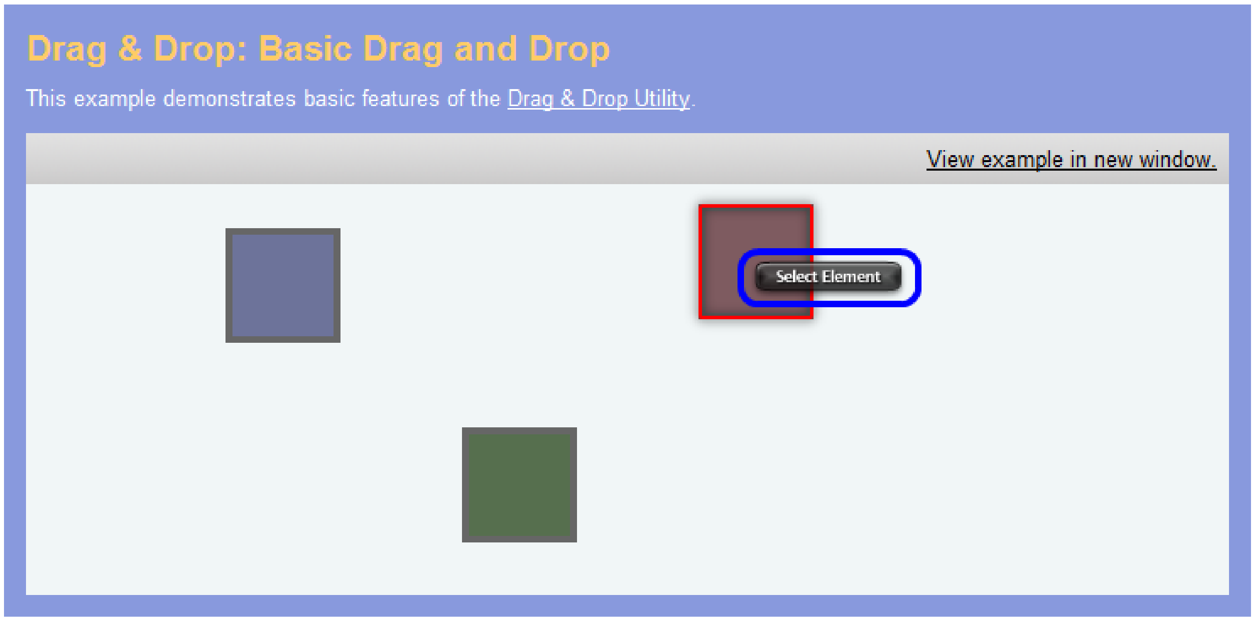
Quick Tasks
HtmlSelect

- Verify - selection 'Text' is 'English'
- Verify - selection 'Value' is 'en'
- Verify - element is visible
- Verify - element is not visible
- Wait - selection 'Text' is 'English'
- Wait - selection 'Value' is 'en'
- Wait - element exists
- Wait - element does not exist

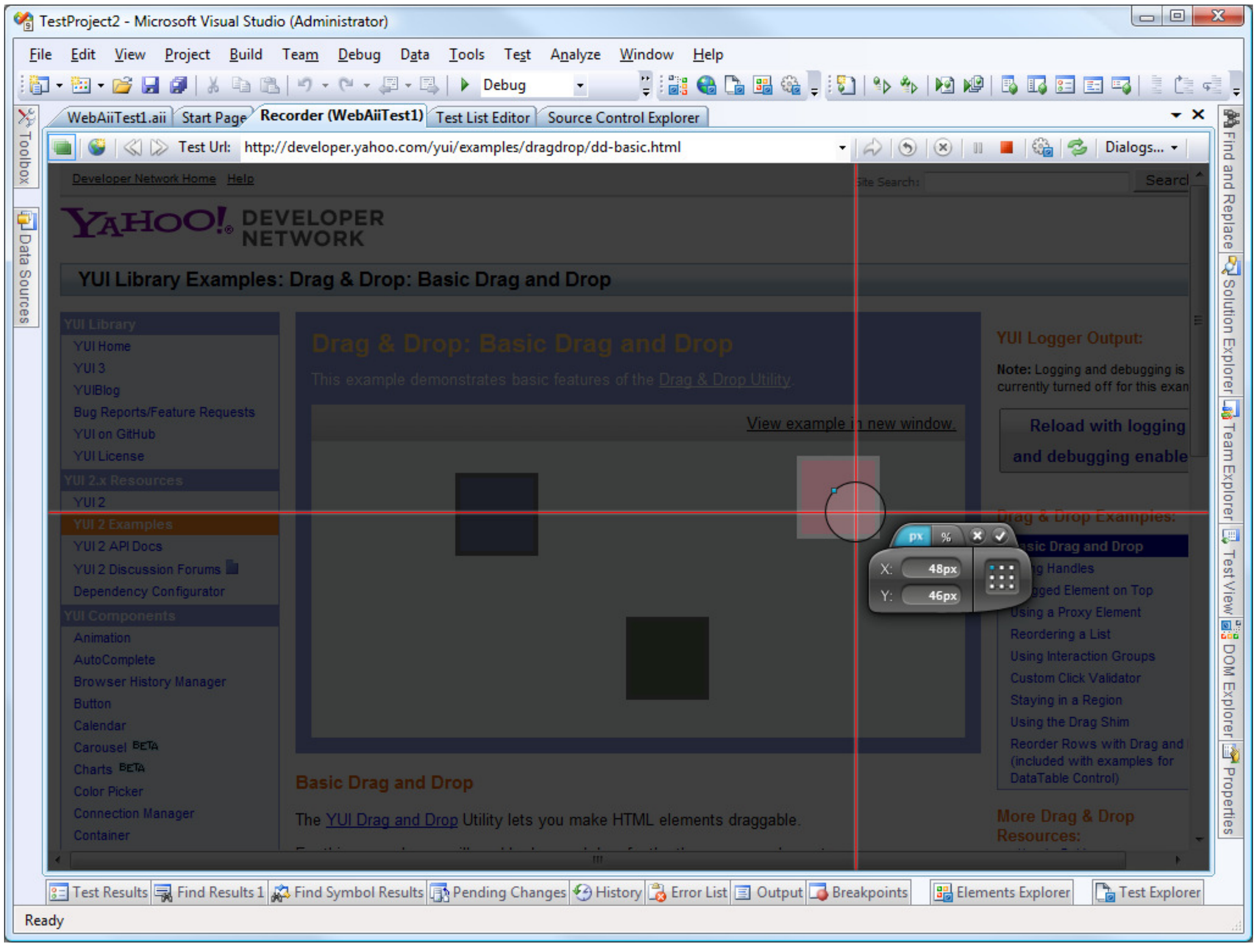
6.2.6 Drag & Drop

Clicking this button will launch the **Drag & Drop** recording wizard. There are 4 steps to the wizard:

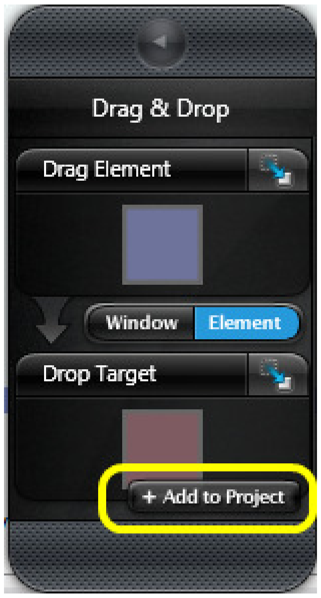
1. Choose whether the drop targat with be another element of the browsers window. If you choose "**Window**", the drop location will be an offset relative to the upper left corner of the browsers window. If you choose "**Element**" the drop location will be an offset relative to the upper left corner of the element you select in step 2.
2. This step only appears if you selected "**Element**" in step 1. Select the target element for the drop operation by:
   a. Hovering the mouse over the element.
   b. Wait one second for "**Select Element**" to appear as highlighted in blue below.
   c. Click "**Select Element**".


Drag & Drop: Basic Drag and Drop
This example demonstrates basic features of the Drag & Drop Utility.
View example in new window.
Select Element

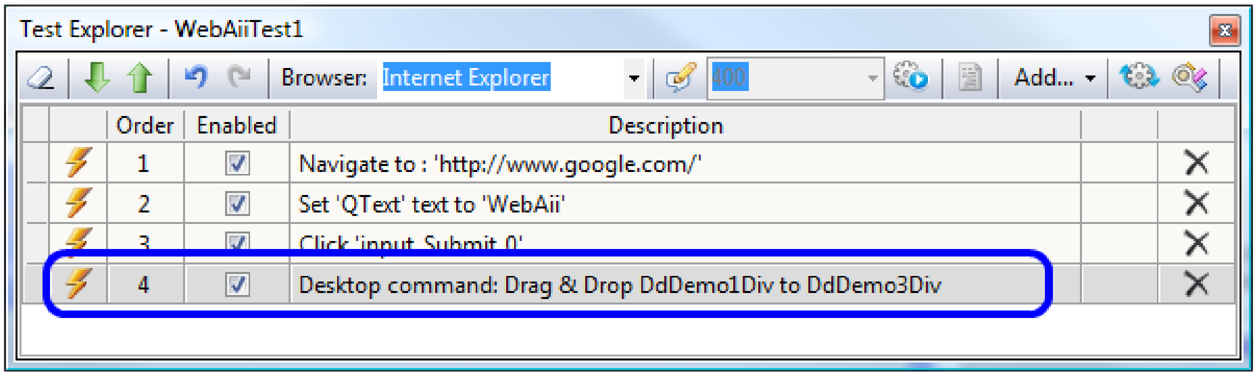
3. Select the drop point offset by dragging the crosshairs to the desired location and then clicking the checkmark. Optionally you can switch between using percent mode or absolute pixel mode.


TestProject2 - Microsoft Visual Studio (Administrator)
File Edit View Project Build Team Debug Data Tools Test Analyze Window Help
Debug
WebAiiTest1.aii
Start Page
Recorder (WebAiiTest1)
Test List Editor
Source Control Explorer
Test Url: http://developer.yahoo.com/yui/examples/dragdrop/dd-basic.html
Dialogs...
YUI Library Examples: Drag & Drop: Basic Drag and Drop
px
%
X: 48px
Y: 46px
Test Results
Find Results 1
Find Symbol Results
Pending Changes
History
Error List
Output
Breakpoints
Elements Explorer
Test Explorer
Ready

4. When all the settings for the Drag & Drop step are set right, click “**Add to Project**” as highlighted here in yellow



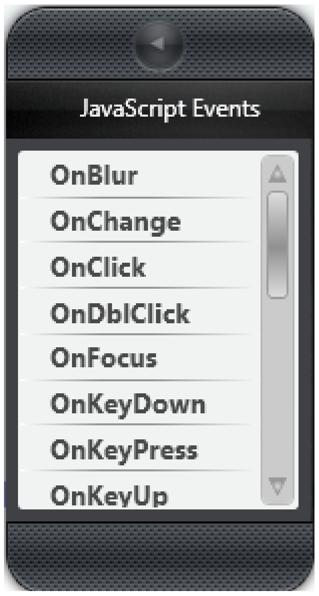
and a new Drag & Drop step will be added to Test Explorer with those settings as shown below.



Order	Enabled	Description
1	✓	Navigate to : 'http://www.google.com/'
2	✓	Set 'QText' text to 'WebAii'
3	✓	Click 'input_Submit_0'
4	✓	Desktop command: Drag & Drop DdDemo1Div to DdDemo3Div

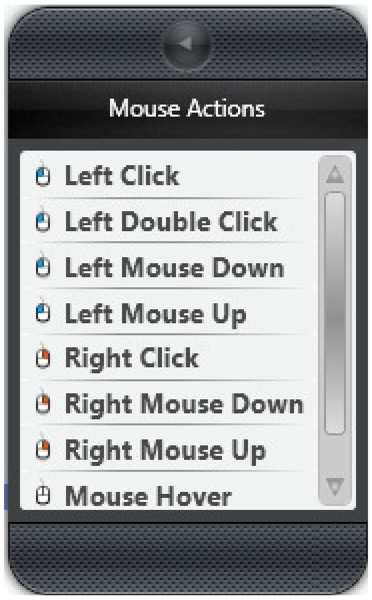
6.2.7 JavaScript Events

Clicking this button opens the **JavaScript Events** list. Double clicking an item in this list adds an "**Invoke JavaScript Event**" step to Test Explorer. The test will invoke the event handler for the event you selected for this element.



6.2.8 Mouse Actions

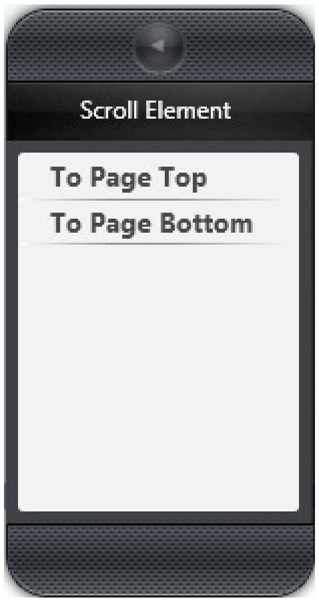
Clicking this button opens the **Mouse Actions** list. Double clicking one of the items in this list adds that particular mouse action as a step in Test Explorer. The location of the mouse action defaults to the center of this element. You can change this if needed in the Properties window after the step has been added.



6.2.9 Scroll Element

Clicking on this button opens the **Scroll Element** list. Double clicking one of the items in this list adds that particular scroll to visible action as a step in Test Explorer. Selecting "**To Page Top**" will scroll this element until the top of the element meets the top of the web page.

Selecting "**To Page Bottom**" will scroll this element until the bottom of the element meets the bottom of the web page.



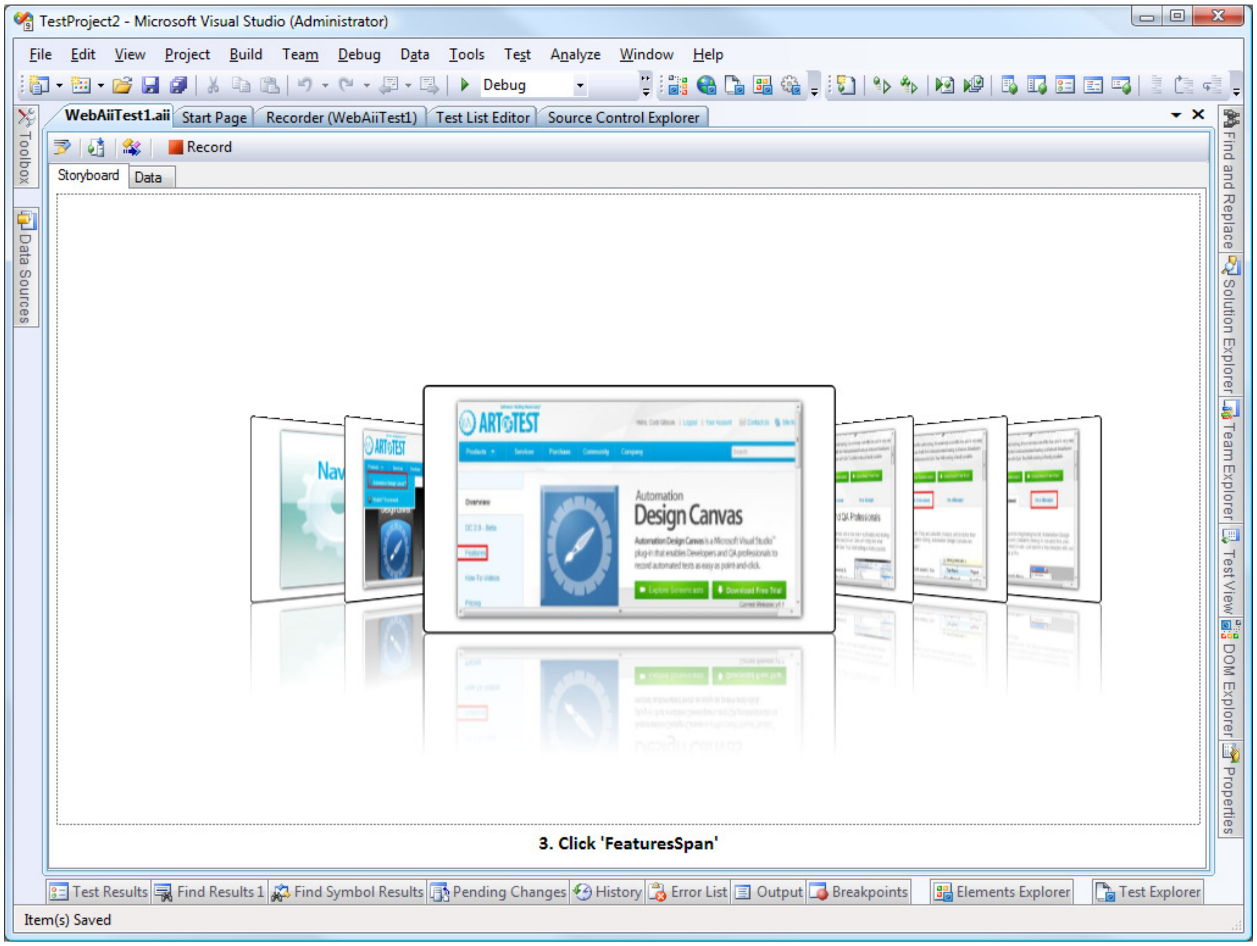
7 Storyboard Tab

Clicking the **Storyboard** tab in a WebAii test document window brings up the storyboard.

7.1 Storyboard

The storyboard is a series of screenshots 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:

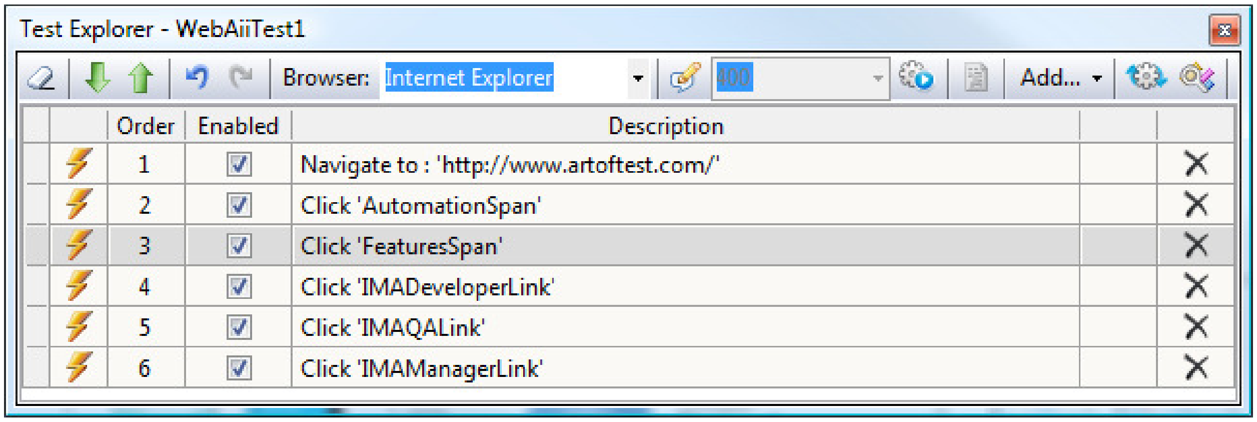

TestProject2 - Microsoft Visual Studio (Administrator)
File Edit View Project Build Team Debug Data Tools Test Analyze Window Help
Debug
WebAiiTest1.aii Start Page Recorder (WebAiiTest1) Test List Editor Source Control Explorer
Record
Storyboard Data
3. Click 'FeaturesSpan'
Test Results Find Results 1 Find Symbol Results Pending Changes History Error List Output Breakpoints Elements Explorer Test Explorer
Item(s) Saved
Toolbox
Data Sources
Find and Replace
Solution Explorer
Team Explorer
Test View
DOM Explorer
Properties

Clicking one of the images to the left or right of center scrolls the storyboard and brings that image forward. Clicking on the center image enlages it to mostly fill the screen.

8 Data tab

The data tab is used to create built-in data arrays for data-driven testing. Section 19.1 describes how to use the data tab to create and use your own data array.

9 Test Explorer Window


Test Explorer - WebAiiTest1

Browser: Internet Explorer 400 Add...

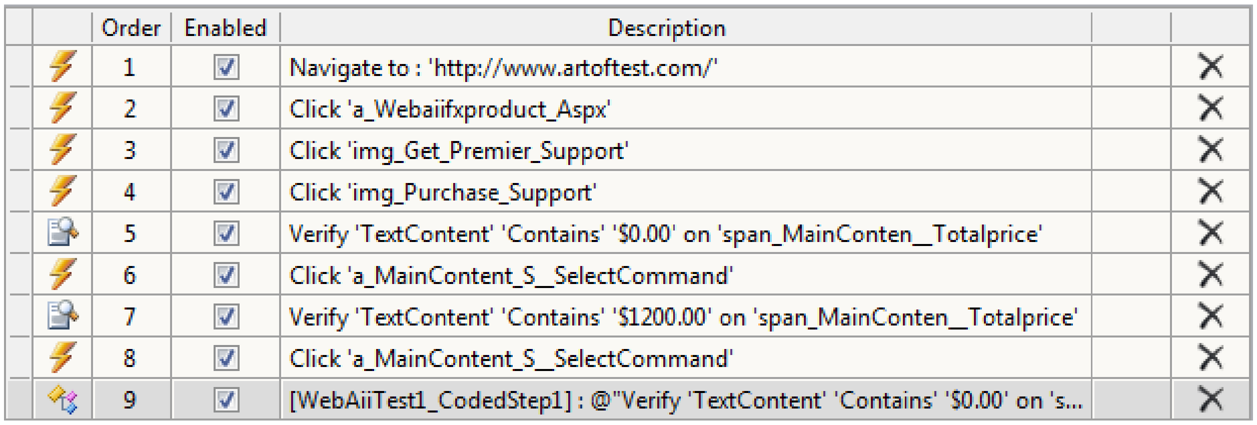
	Order	Enabled	Description		
⚡	1	✓	Navigate to : 'http://www.artoftest.com/'		✕
⚡	2	✓	Click 'AutomationSpan'		✕
⚡	3	✓	Click 'FeaturesSpan'		✕
⚡	4	✓	Click 'IMADeveloperLink'		✕
⚡	5	✓	Click 'IMAQALink'		✕
⚡	6	✓	Click 'IMAManagerLink'		✕

Test Explorer 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 a .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, modify, etc.).

9.1 Steps

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



	Order	Enabled	Description		
⚡	1	✓	Navigate to : 'http://www.artoftest.com/'		✕
⚡	2	✓	Click 'a_Webaiifxproduct_Aspx'		✕
⚡	3	✓	Click 'img_Get_Premier_Support'		✕
⚡	4	✓	Click 'img_Purchase_Support'		✕
🔍	5	✓	Verify 'TextContent' 'Contains' '$0.00' on 'span_MainConten__Totalprice'		✕
⚡	6	✓	Click 'a_MainContent_S__SelectCommand'		✕
🔍	7	✓	Verify 'TextContent' 'Contains' '$1200.00' on 'span_MainConten__Totalprice'		✕
⚡	8	✓	Click 'a_MainContent_S__SelectCommand'		✕
✦	9	✓	[WebAiiTest1_CodedStep1] : @"Verify 'TextContent' 'Contains' '$0.00' on 's...		✕

9.1.1 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 9.1.4).

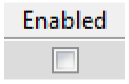
- This icon represents a code behind step. Section **Error! Reference source not found.** describes code behind steps.

9.1.2 Reordering test steps

There are two methods for reordering test steps:

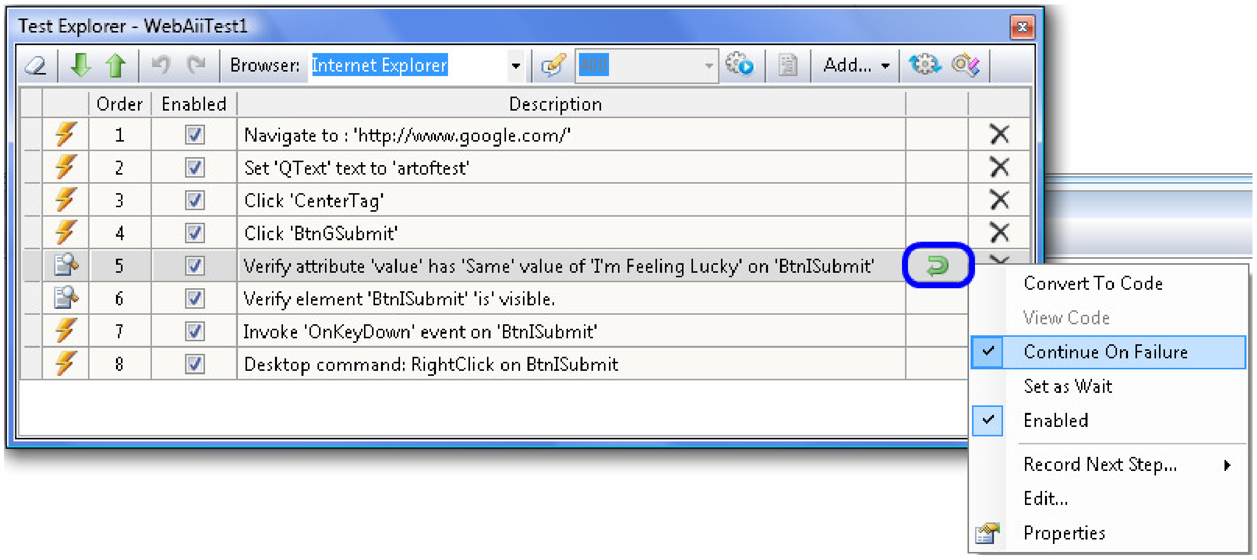
- You can use the “**Move selected step up/down**” arrows at the top of the window . These move the currently selected test step either up or down in the sequence. They do not move multiple steps if you have more than one test step highlighted.
- You can drag and drop the test step that needs to be moved to its proper location. This method will only move one step at a time. If 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 step in the proper location.

9.1.3 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 simply uncheck the “**Enabled**” checkbox next to the step you want disabled.

9.1.4 Continue On Failure



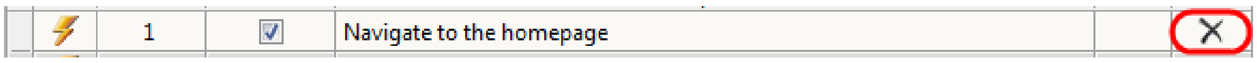
Order	Enabled	Description
1	✓	Navigate to : 'http://www.google.com/'
2	✓	Set 'QText' text to 'artoftest'
3	✓	Click 'CenterTag'
4	✓	Click 'BtnGSubmit'
5	✓	Verify attribute 'value' has 'Same' value of 'I'm Feeling Lucky' on 'BtnISubmit'
6	✓	Verify element 'BtnISubmit' 'is' visible.
7	✓	Invoke 'OnKeyDown' event on 'BtnISubmit'
8	✓	Desktop command: RightClick on BtnISubmit

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. The test will still be marked as having failed when it runs however you will be able to see results for the test steps following it.

To mark a test step “**Continue On Failure**” right click on the step and select “**Continue On Failure**” from the context menu. 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 circled in blue.

To turn off “**Continue On Failure**” repeat the same steps to toggle the setting.

9.1.5 Deleting

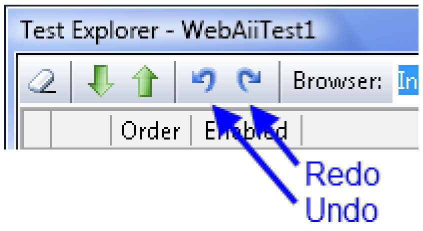

1 ✓ Navigate to the homepage

Test steps can be deleted by clicking the ”**delete**” icon for the test step.

9.1.6 Clear all steps

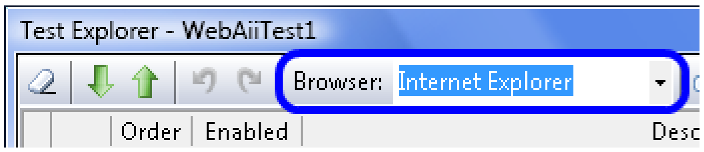
Clicking the “Clear all steps” icon deletes all of the steps in the test.

9.2 Undo & Redo



Test explorer has Undo and Redo capability. For example, if you accidentally delete a step you can click the Undo button, as shown above, to restore it. Clicking the Redo button after an Undo will Redo the delete.

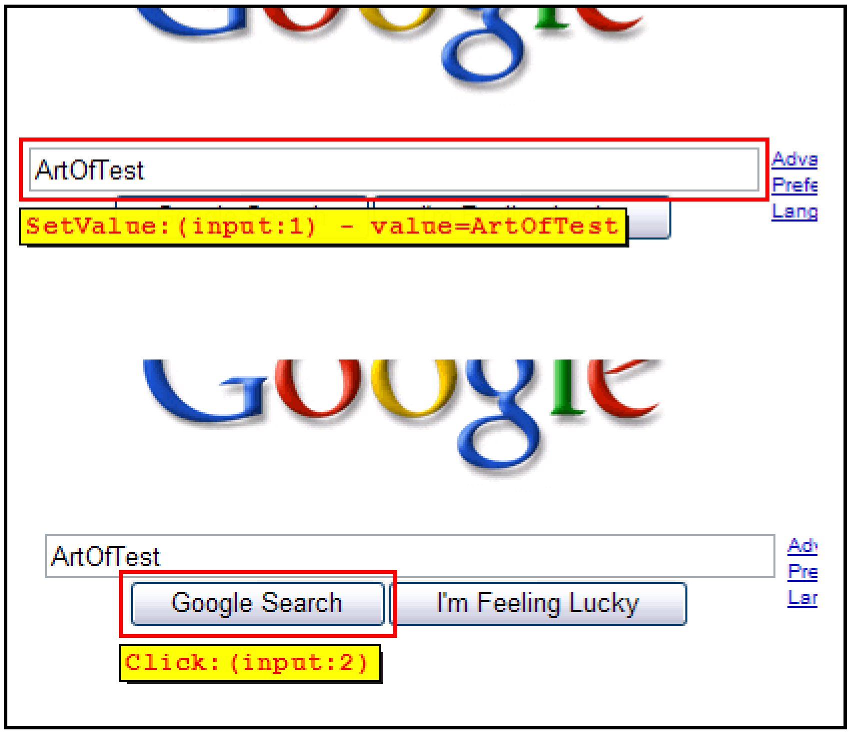
9.3 Selecting browser for quick execute



Automation Design Canvas supports running tests in Internet Explorer, Firefox and (as of Design Canvas 2.0) Safari. Which browser is used by Quick Execute is controlled by the selection in this dropdown.

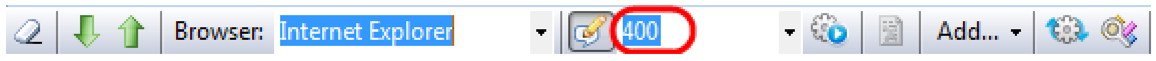
9.4 Enable annotation

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:



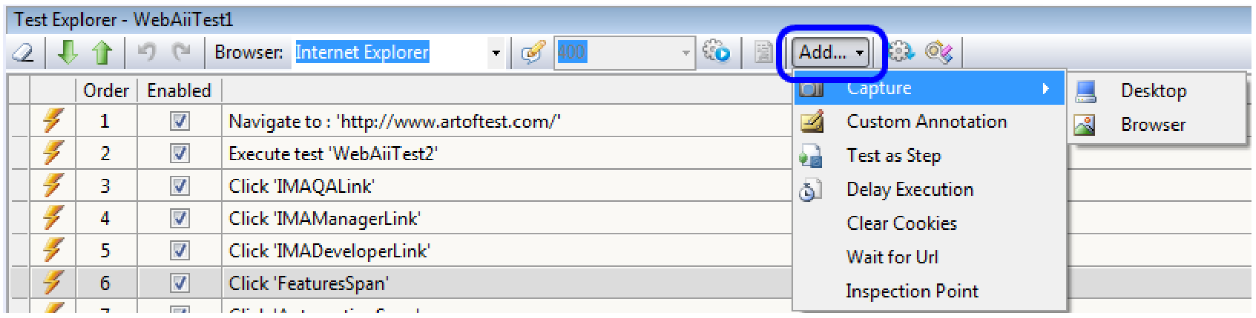
This icon located in the Test Explorer toolbar enables and disables annotation during quick execution. This can help you observe test execution. Section 23.6.1 describes how to enable annotation during test execution by Visual Studio.

9.5 Execution Delay



This setting controls the execution delay (in milliseconds) between each test step during quick execution. This can help you observe test execution. Section 23.6.1 describes how to set execution delay during Visual Studio test execution.

9.6 Add Additional Steps Dropdown


Test Explorer - WebAiiTest1

Browser: Internet Explorer

Order	Enabled	
1	✓	Navigate to : 'http://www.artoftest.com/'
2	✓	Execute test 'WebAiiTest2'
3	✓	Click 'IMAQALink'
4	✓	Click 'IMAManagerLink'
5	✓	Click 'IMADeveloperLink'
6	✓	Click 'FeaturesSpan'

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.

9.6.1 Capture Desktop

A "**Capture Desktop**" step takes a screenshot of your desktop at that point in time and stores it. This can be useful for troubleshooting, verifying localization, and other tasks that require a snapshot for 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 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 something like this:

```
"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.

9.6.2 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 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 something like this: "C:\Documents and Settings\testuser\My Documents\Visual Studio 2008\Projects\AoT_WebsiteTests\TestResults\testuer_TESTPC 2008-06-13 14_34_38\Out".

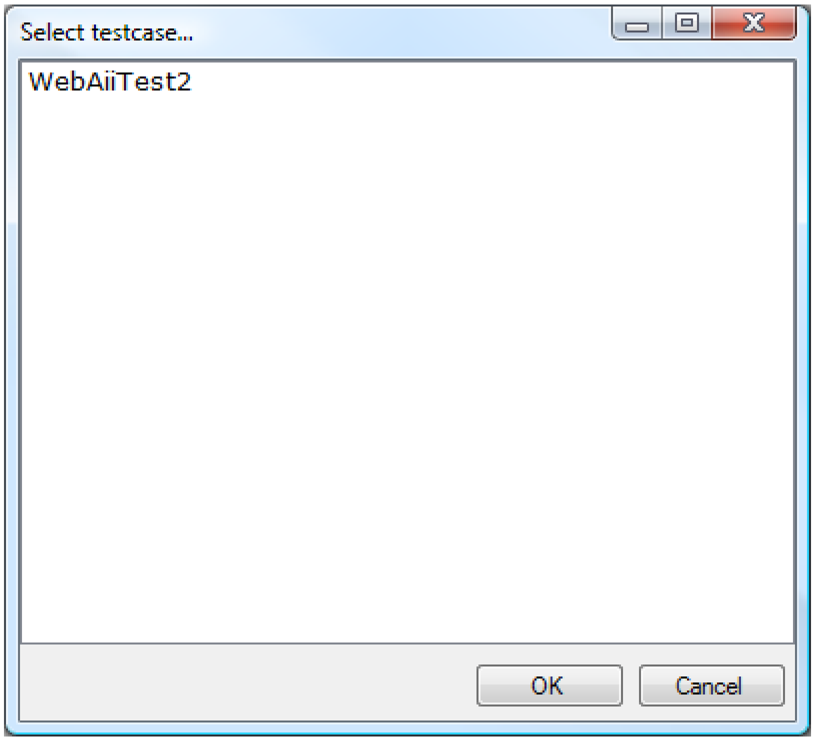
Note: Capture Browser is only performed when the test is run from Test View or Test List Editor. It is not performed when it is run using Quick Execute.

9.6.3 Custom Annotation

In addition to the automation annotation described in section 9.4 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 (top left, top center, etc.).
- The length of time (in milliseconds) to display the text.

9.6.4 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 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.

9.6.5 Delay Execution

The "**Delay Execution**" step causes your test to simply sit and wait for a fixed amount of time. After adding this step to your test, you can modify the amount of delay in the properties window. The default delay is 250 milliseconds.

9.6.6 Clear Cookies

A "**Clear Cookies**" step clears the browsers entire cookie cache. It is not selective which cookies will be deleted. All cookies for the active browser will be cleared unconditionally. This is useful in cases where logon information is stored in cookies and you want the test to start with a clean slate (not already logged in or not using a saved userid, etc.).

9.6.7 Wait for Url

The "**Wait for Url**" step suspends test execution until a particular URL is loaded in the browsers address bar. This is useful when you're dealing with redirection of a web page. Adding this step will make your test wait until the redirection to the final URL is finished.

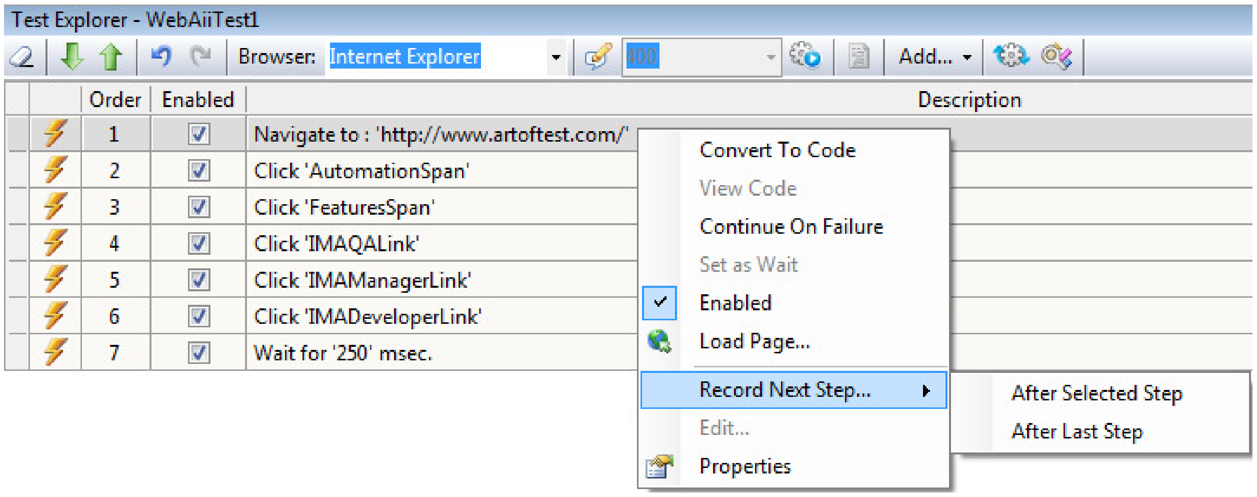
After adding this step to your test you can modify two properties:

- The URL to wait for.
- The amount of time (in milliseconds) to wait for the URL to appear. If this time is exceeded, the test will abort and report failure.

9.6.8 Inspection Point

An "**Inspection Point**" step suspends test execution and displays the DOM in a DOM Explorer window at that point in time. This allows you to examine the DOM tree which can aid in test development and troubleshooting. Test execution automatically resumes when you close this DOM Explorer window.

9.7 Test Explorer Context menu



Test Explorer - WebAiiTest1

Browser: Internet Explorer | 400 | Add...

	Order	Enabled	Description
	1	✓	Navigate to : 'http://www.artoftest.com/'
	2	✓	Click 'AutomationSpan'
	3	✓	Click 'FeaturesSpan'
	4	✓	Click 'IMAQALink'
	5	✓	Click 'IMAManagerLink'
	6	✓	Click 'IMADeveloperLink'
	7	✓	Wait for '250' msec.

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.

9.7.1 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 the WebAii test a new code behind file will be created, added to the project and attached to the 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 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 conditional execution.

9.7.2 View Code

When you right click on a step that refers to a code behind method and select “**View Code**”, 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.

9.7.3 Continue On Failure

On occasion you may want the test to continue even 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.

9.7.4 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 match. Instead WebAii will wait up to 10 seconds (the default setting) for the condition to match. The default 10 second timeout can be changed 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.

9.7.5 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 simply uncheck the “**Enabled**” check mark in the context menu.

9.7.6 Load Page…

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

9.7.7 Record Next Step

Using this menu item you can control where the next recorded step will be added to the test. Options are:

- After Selected Step – the next recorded step will be inserted into the test immediately following the currently selected step.
- After Last Step – the next recorded step will be appended to the end of the test.

9.7.8 Edit This Step

This menu item is enabled only for "**Verify**" steps. Clicking on it will open the "**Sentence Verification Builder**" (described in detail in section 15) and initialize it with the parameters of that verification step. Now you can edit the parameters as needed and validate them (assuming the page containing that element is currently loaded in the recorder, otherwise validation is disabled TODO VERIFY THIS IS STILL TRUE AFTER BUG FIXES).

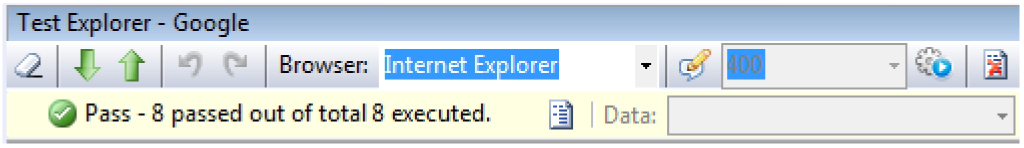
9.7.9 Properties

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.

9.8 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 under the normal VS testing framework. Test results will not be stored in the standard VS test results folder. This is intended to be used 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 suite run.

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

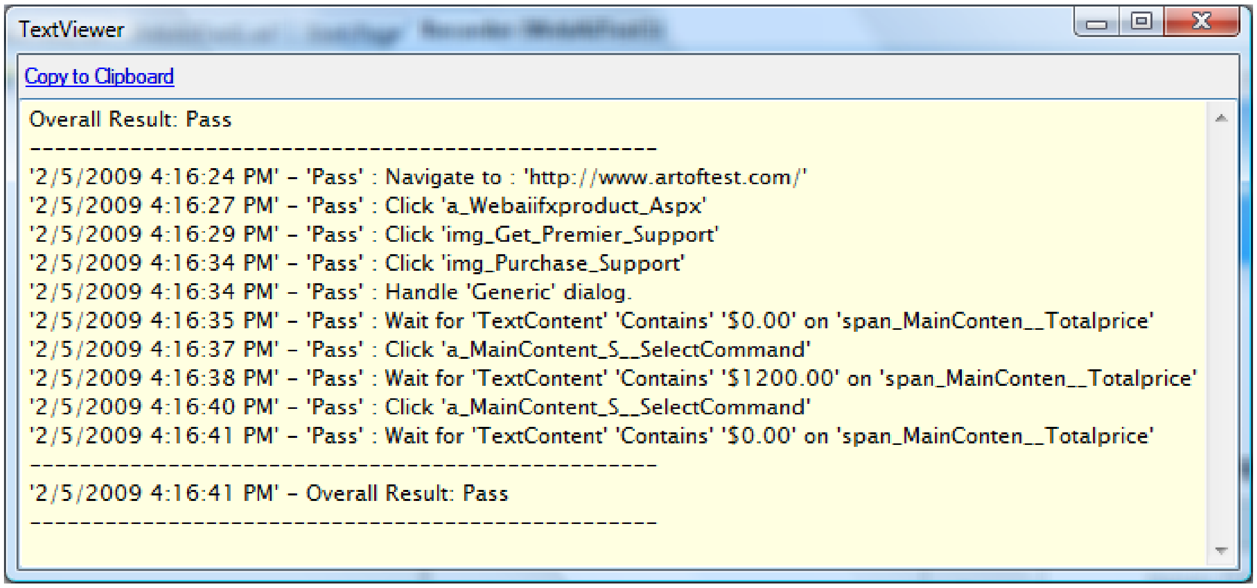


Pass - 8 passed out of total 8 executed.

The "Data" dropdown shown above is only displayed for data-driven tests. See Section 19, Data Driven Testing for details.

9.8.1 Viewing Test 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:

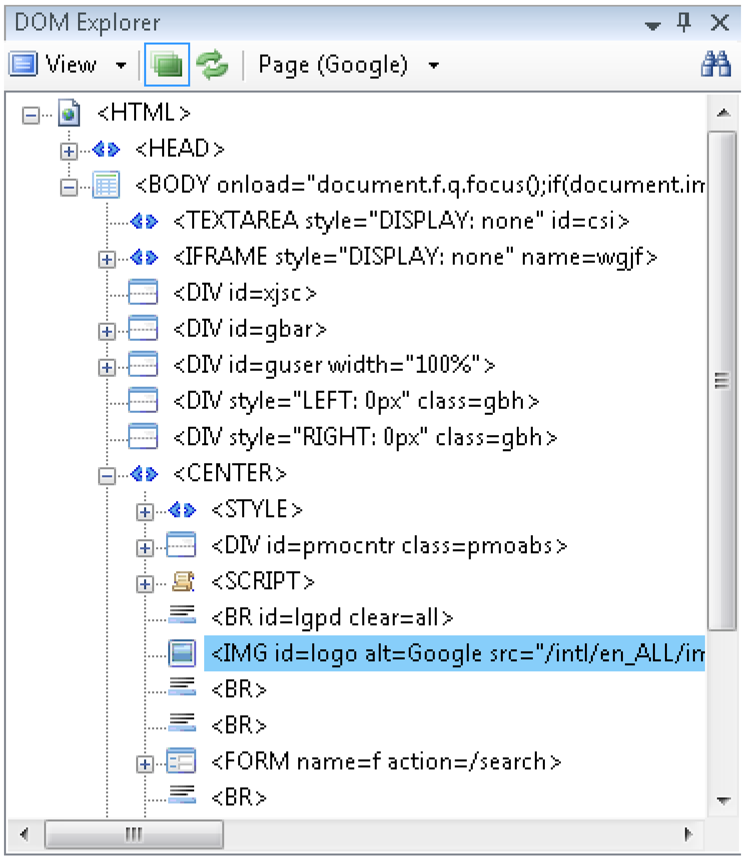


```
Overall Result: Pass
------------------------------------------------------
'2/5/2009 4:16:24 PM' - 'Pass' : Navigate to : 'http://www.artoftest.com/'
'2/5/2009 4:16:27 PM' - 'Pass' : Click 'a_Webaiifxproduct_Aspx'
'2/5/2009 4:16:29 PM' - 'Pass' : Click 'img_Get_Premier_Support'
'2/5/2009 4:16:34 PM' - 'Pass' : Click 'img_Purchase_Support'
'2/5/2009 4:16:34 PM' - 'Pass' : Handle 'Generic' dialog.
'2/5/2009 4:16:35 PM' - 'Pass' : Wait for 'TextContent' 'Contains' '$0.00' on 'span_MainConten__Totalprice'
'2/5/2009 4:16:37 PM' - 'Pass' : Click 'a_MainContent_S__SelectCommand'
'2/5/2009 4:16:38 PM' - 'Pass' : Wait for 'TextContent' 'Contains' '$1200.00' on 'span_MainConten__Totalprice'
'2/5/2009 4:16:40 PM' - 'Pass' : Click 'a_MainContent_S__SelectCommand'
'2/5/2009 4:16:41 PM' - 'Pass' : Wait for 'TextContent' 'Contains' '$0.00' on 'span_MainConten__Totalprice'
------------------------------------------------------
'2/5/2009 4:16:41 PM' - Overall Result: Pass
------------------------------------------------------
```

9.8.2 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.

10 DOM Explorer



```
<HTML>
  <HEAD>
  <BODY onload="document.f.q.focus();if(document.in
    <TEXTAREA style="DISPLAY: none" id=csi>
    <IFRAME style="DISPLAY: none" name=wgjf>
    <DIV id=xjsc>
    <DIV id=gbar>
    <DIV id=guser width="100%">
    <DIV style="LEFT: 0px" class=gbh>
    <DIV style="RIGHT: 0px" class=gbh>
    <CENTER>
      <STYLE>
      <DIV id=pmocntr class=pmoabs>
      <SCRIPT>
      <BR id=lgpd clear=all>
      <IMG id=logo alt=Google src="/intl/en_ALL/in
      <BR>
      <BR>
      <FORM name=f action=/search>
      <BR>
```

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.

10.1 DOM Explorer: View

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

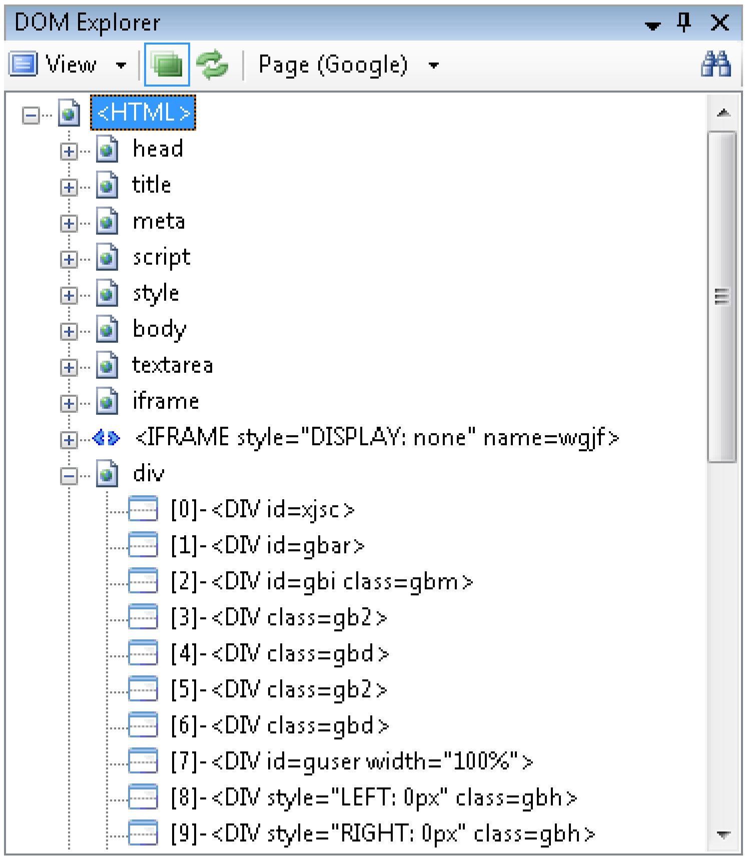
10.1.1 DOM Explorer –View Menu: Hierarchal

The Hierarchal 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 is a <HEAD> and <BODY> node. 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.

10.1.2 DOM Explorer –View Menu: TagName

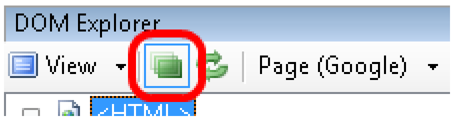
The TagName mode setting in the View menu displays the DOM as a tree view showing all the different element tag types along with their tag index for the elements that appear on the web page currently loaded in the Recording Surface. Each different tag type is displayed as a root node in the tree. It 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:



```
<HTML>
  head
  title
  meta
  script
  style
  body
  textarea
  iframe
  <IFRAME style="DISPLAY: none" name=wgjf>
  div
    [0]-<DIV id=xjsc>
    [1]-<DIV id=gbar>
    [2]-<DIV id=gbi class=gbm>
    [3]-<DIV class=gb2>
    [4]-<DIV class=gbd>
    [5]-<DIV class=gb2>
    [6]-<DIV class=gbd>
    [7]-<DIV id=guser width="100%">
    [8]-<DIV style="LEFT: 0px" class=gbh>
    [9]-<DIV style="RIGHT: 0px" class=gbh>
```

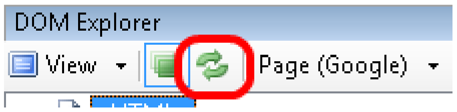
10.2 DOM Explorer: Highlight selected element



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 the element in the Recording Surface. 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.

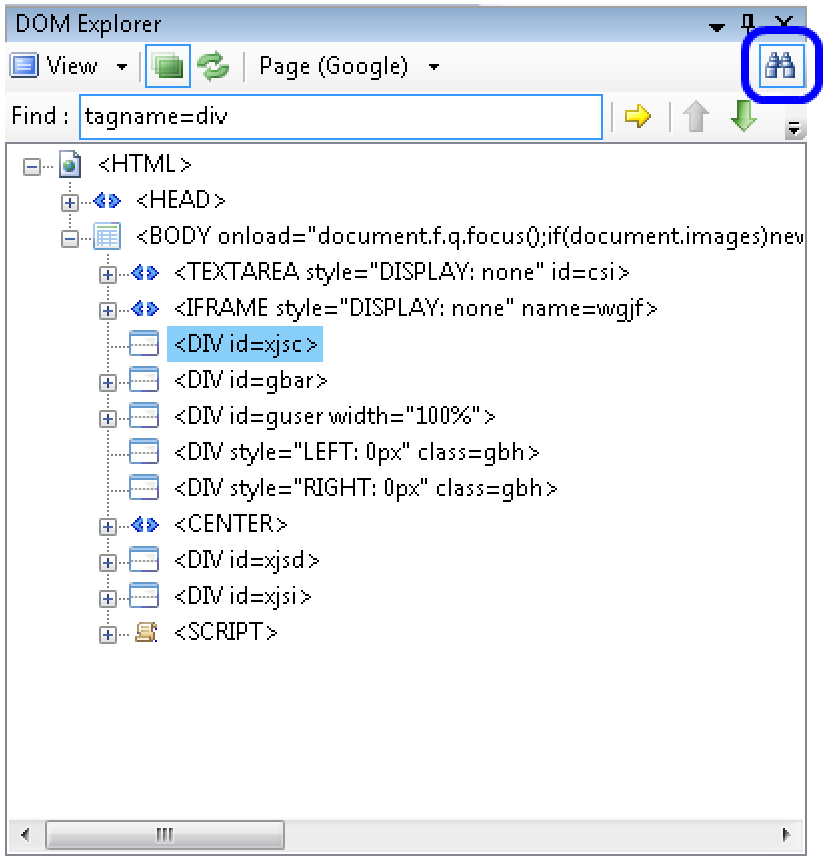
10.3 DOM Explorer: Refresh



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.

10.4 DOM Explorer: Search

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



```
Find : tagname=div
<HTML>
  <HEAD>
  <BODY onload="document.f.q.focus();if(document.images)nev
    <TEXTAREA style="DISPLAY: none" id=csi>
    <IFRAME style="DISPLAY: none" name=wgjf>
    <DIV id=xjsc>
    <DIV id=gbar>
    <DIV id=guser width="100%">
    <DIV style="LEFT: 0px" class=gbh>
    <DIV style="RIGHT: 0px" class=gbh>
    <CENTER>
    <DIV id=xjsd>
    <DIV id=xjsi>
    <SCRIPT>
```

Enter a "**Find Expression**" into the textbox to define what to search for. Find Expressions are documented online at http://www.artoftest.com/support/webaii/topicsindex.aspx?topic=byexpression. Click the

“**Evaluate Expression**” button to begin the search. The first matching element will automatically be highlighted in DOM Explorer. Clicking the “**Next Result**” and “**Previous Result**” buttons will move the current selection forward or backwards to the next element matching the find expression you entered.

10.5 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.

10.5.1 DOM Explorer – Context Menu: Goto

The “**Goto**” menu item offers a shortcut to major sections of the DOM tree. The DOM root will always appear first in the submenu. If you have one (or more) Silverlight applications contained in the webpage, those applications will also appear in the submenu. Selecting one of the Silverlight applications from the submenu will highlight the root element for that Silverlight application.

10.5.2 DOM Explorer – Context Menu: Lock On Surface

Selecting the “**Lock On Surface**” menu item opens the Automation Overlay Surface™ (AOS) for that element as described in section 0.

10.5.3 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 typically 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 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 easier than if you hard code their Find Expression or otherwise find the element programmatically in the browser’s DOM.

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

- You 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 Find Expression in one place, Elements Explorer, instead of having to update every occurrence where the element is used in your test.
- It’s easier for code behind methods to use element references from Elements Explorer than to use their own hard-coded methods.

10.5.4 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.

11 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:

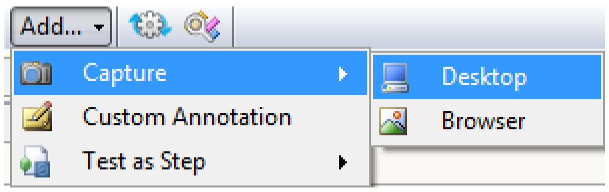
```
C:\Users\<user>\Documents\Visual Studio 2008\Projects\<project
folder>\TestResults\<user>_<computer> 2008-06-17 15_51_22\Out\Snapshot0.png
```

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.

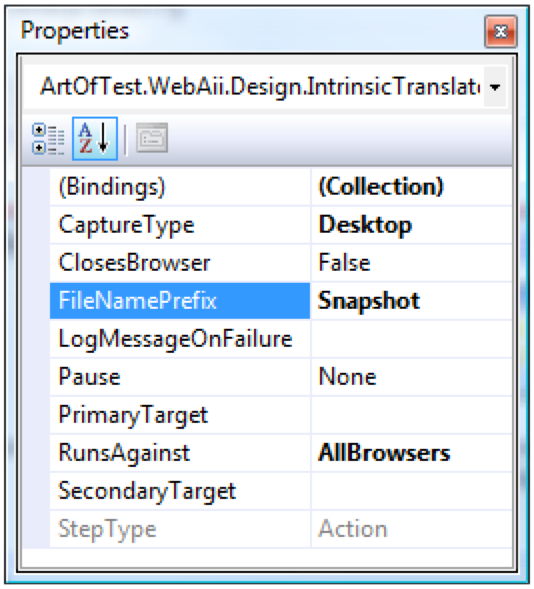
11.1 Snapshot of the desktop

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 a step in your test. 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.*


Properties

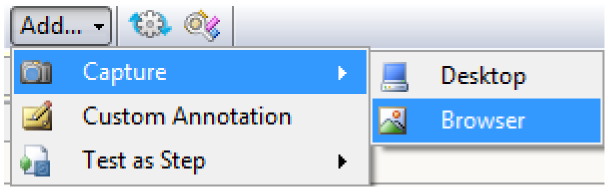
ArtOfTest.WebAii.Design.IntrinsicTranslat

(Bindings)	**(Collection)**
CaptureType	**Desktop**
ClosesBrowser	False
FileNamePrefix	**Snapshot**
LogMessageOnFailure	
Pause	None
PrimaryTarget	
RunsAgainst	**AllBrowsers**
SecondaryTarget	
StepType	Action

11.2 Snapshot of the browser

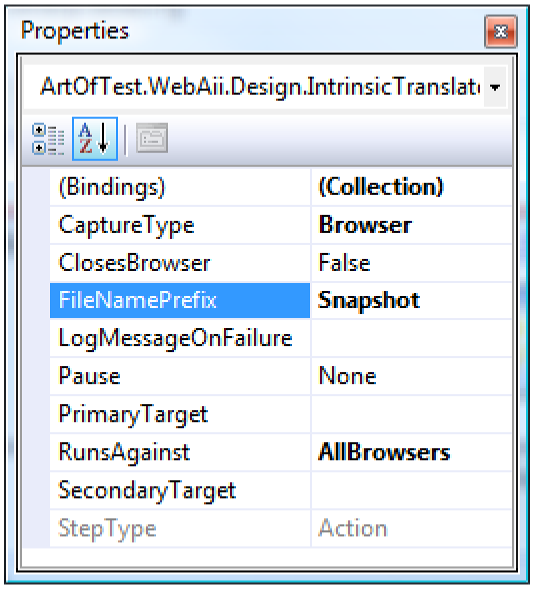
To add a "Capture Browser" step to your test:

- 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 a step in your test. 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.



Properties

ArtOfTest.WebAii.Design.IntrinsicTranslat

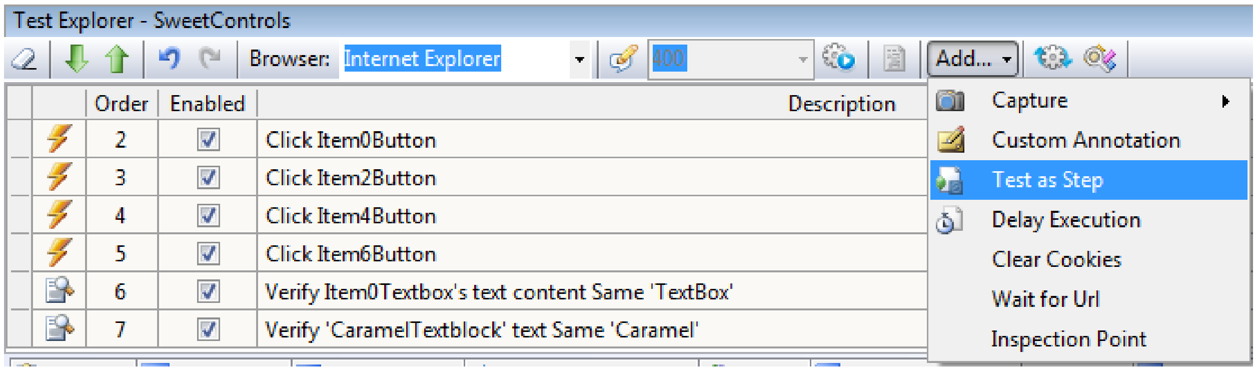
(Bindings)	(Collection)
CaptureType	Browser
ClosesBrowser	False
FileNamePrefix	Snapshot
LogMessageOnFailure	
Pause	None
PrimaryTarget	
RunsAgainst	AllBrowsers
SecondaryTarget	
StepType	Action

12 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.

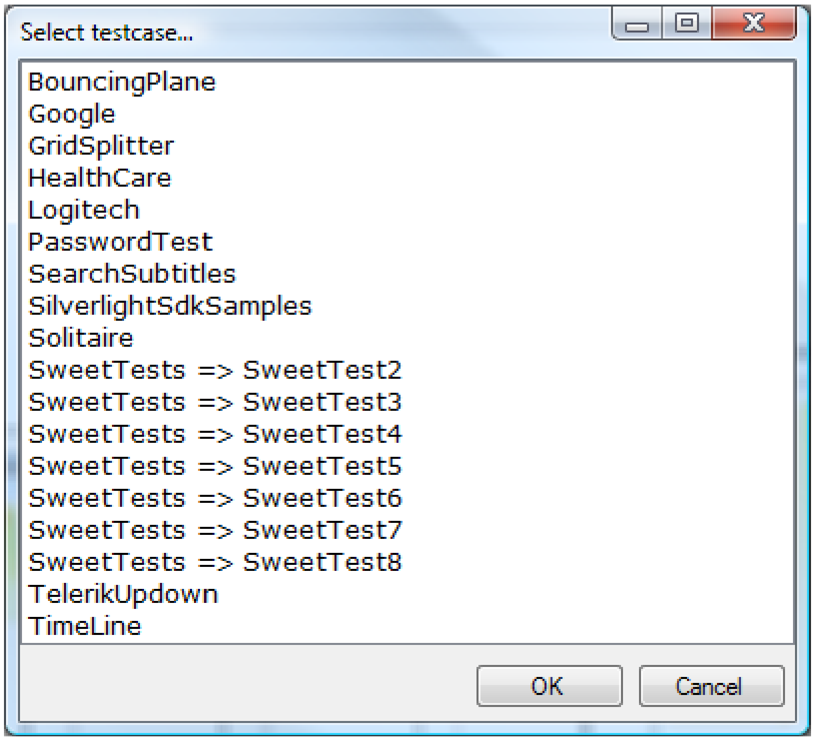


Test Explorer - SweetControls

Browser: Internet Explorer

Order	Enabled	Description
2	✓	Click Item0Button
3	✓	Click Item2Button
4	✓	Click Item4Button
5	✓	Click Item6Button
6	✓	Verify Item0Textbox's text content Same 'TextBox'
7	✓	Verify 'CaramelTextblock' text Same 'Caramel'

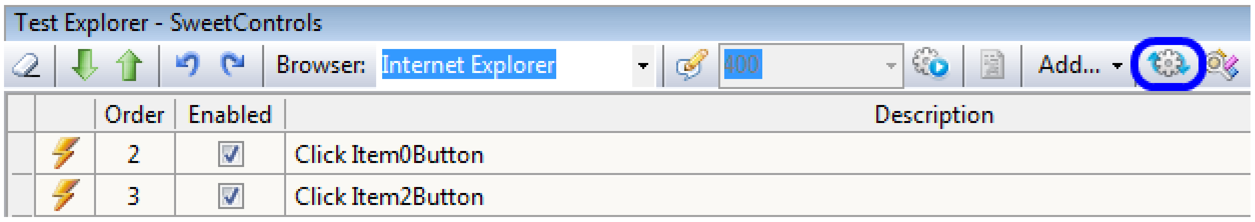
- Select the test step that has the login steps.


Select testcase...

BouncingPlane
Google
GridSplitter
HealthCare
Logitech
PasswordTest
SearchSubtitles
SilverlightSdkSamples
Solitaire
SweetTests => SweetTest2
SweetTests => SweetTest3
SweetTests => SweetTest4
SweetTests => SweetTest5
SweetTests => SweetTest6
SweetTests => SweetTest7
SweetTests => SweetTest8
TelerikUpdown
TimeLine

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

13 Re-capture Storyboard


Test Explorer - SweetControls

Browser: Internet Explorer

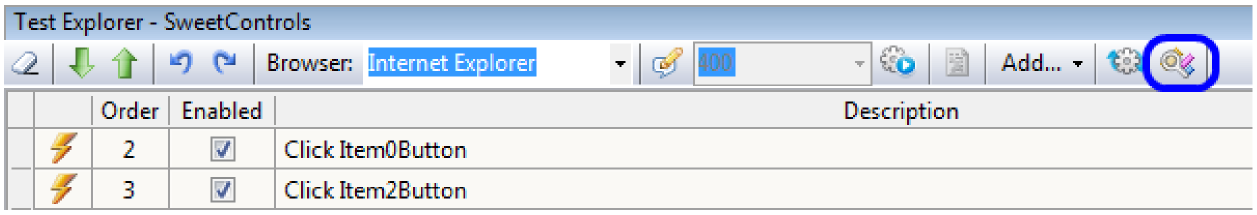
	Order	Enabled	Description
⚡	2	✓	Click Item0Button
⚡	3	✓	Click Item2Button

Clicking the "Re-capture Storyboard" button will run your test and store new screenshots of each test step in the storyboard for that test.

14 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.

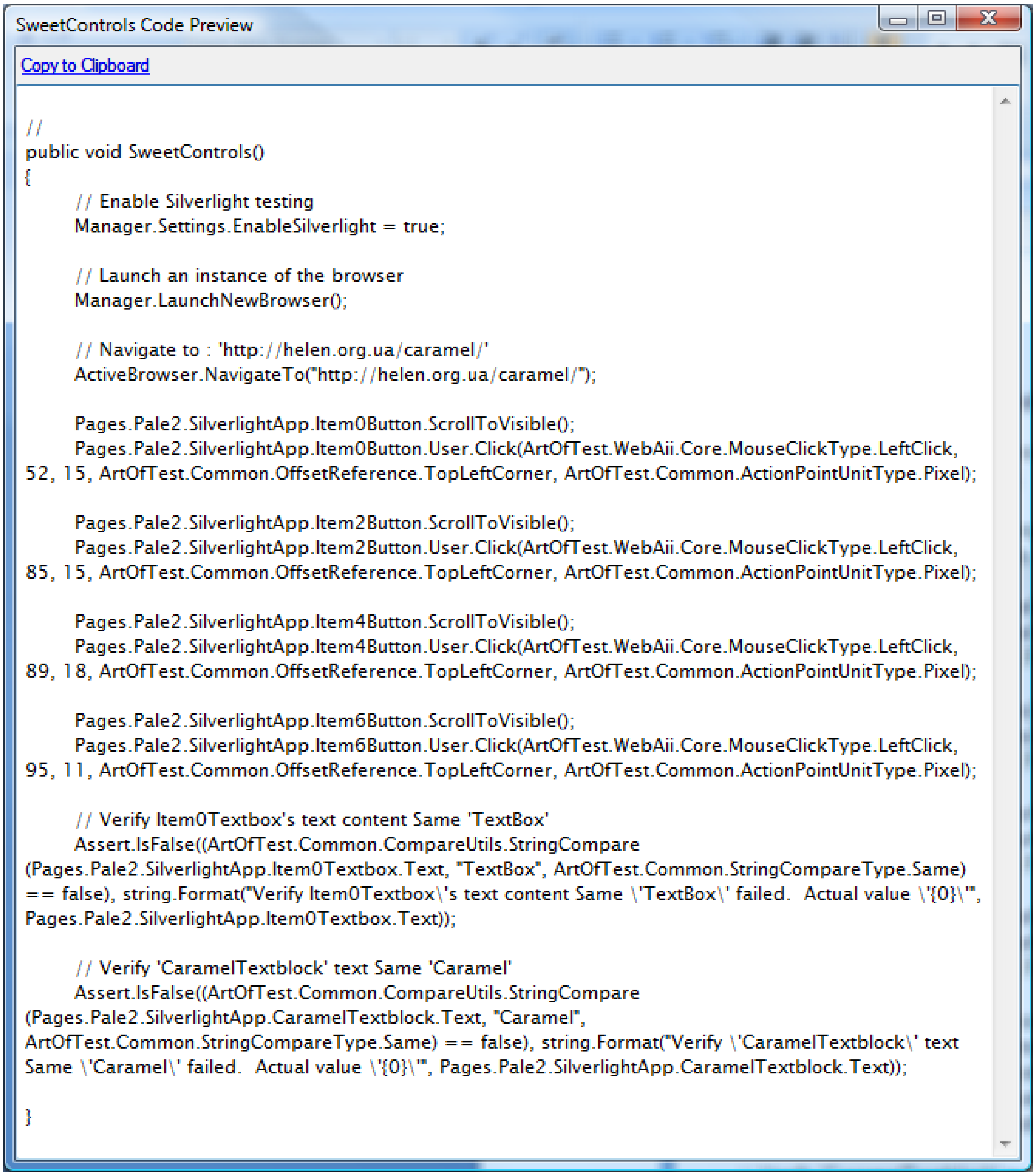
To preview the code, click the "**Preview Code**" button as shown below:


Test Explorer - SweetControls

Browser: Internet Explorer

	Order	Enabled	Description
⚡	2	✓	Click Item0Button
⚡	3	✓	Click Item2Button

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



```
//
public void SweetControls()
{
        // Enable Silverlight testing
        Manager.Settings.EnableSilverlight = true;

        // Launch an instance of the browser
        Manager.LaunchNewBrowser();

        // Navigate to : 'http://helen.org.ua/caramel/'
        ActiveBrowser.NavigateTo("http://helen.org.ua/caramel/");

        Pages.Pale2.SilverlightApp.Item0Button.ScrollToVisible();
        Pages.Pale2.SilverlightApp.Item0Button.User.Click(ArtOfTest.WebAii.Core.MouseClickType.LeftClick,
52, 15, ArtOfTest.Common.OffsetReference.TopLeftCorner, ArtOfTest.Common.ActionPointUnitType.Pixel);

        Pages.Pale2.SilverlightApp.Item2Button.ScrollToVisible();
        Pages.Pale2.SilverlightApp.Item2Button.User.Click(ArtOfTest.WebAii.Core.MouseClickType.LeftClick,
85, 15, ArtOfTest.Common.OffsetReference.TopLeftCorner, ArtOfTest.Common.ActionPointUnitType.Pixel);

        Pages.Pale2.SilverlightApp.Item4Button.ScrollToVisible();
        Pages.Pale2.SilverlightApp.Item4Button.User.Click(ArtOfTest.WebAii.Core.MouseClickType.LeftClick,
89, 18, ArtOfTest.Common.OffsetReference.TopLeftCorner, ArtOfTest.Common.ActionPointUnitType.Pixel);

        Pages.Pale2.SilverlightApp.Item6Button.ScrollToVisible();
        Pages.Pale2.SilverlightApp.Item6Button.User.Click(ArtOfTest.WebAii.Core.MouseClickType.LeftClick,
95, 11, ArtOfTest.Common.OffsetReference.TopLeftCorner, ArtOfTest.Common.ActionPointUnitType.Pixel);

        // Verify Item0Textbox's text content Same 'TextBox'
        Assert.IsFalse((ArtOfTest.Common.CompareUtils.StringCompare
(Pages.Pale2.SilverlightApp.Item0Textbox.Text, "TextBox", ArtOfTest.Common.StringCompareType.Same)
== false), string.Format("Verify Item0Textbox\'s text content Same \'TextBox\' failed.  Actual value \'{0}\'",
Pages.Pale2.SilverlightApp.Item0Textbox.Text));

        // Verify 'CaramelTextblock' text Same 'Caramel'
        Assert.IsFalse((ArtOfTest.Common.CompareUtils.StringCompare
(Pages.Pale2.SilverlightApp.CaramelTextblock.Text, "Caramel",
ArtOfTest.Common.StringCompareType.Same) == false), string.Format("Verify \'CaramelTextblock\' text
Same \'Caramel\' failed.  Actual value \'{0}\'", Pages.Pale2.SilverlightApp.CaramelTextblock.Text));

}
```

15 Sentence Verification Builder

15.1 Overview

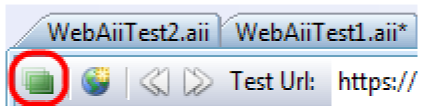
With Sentence Verification Builder you can create your own verification rules as if you were writing a sentence. To make crafting verification and synchronization as simple as possible, ArtOfTest developed an innovative Sentence Based™ user interface (UI) that guides you through crafting verifications 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 Sentence 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.

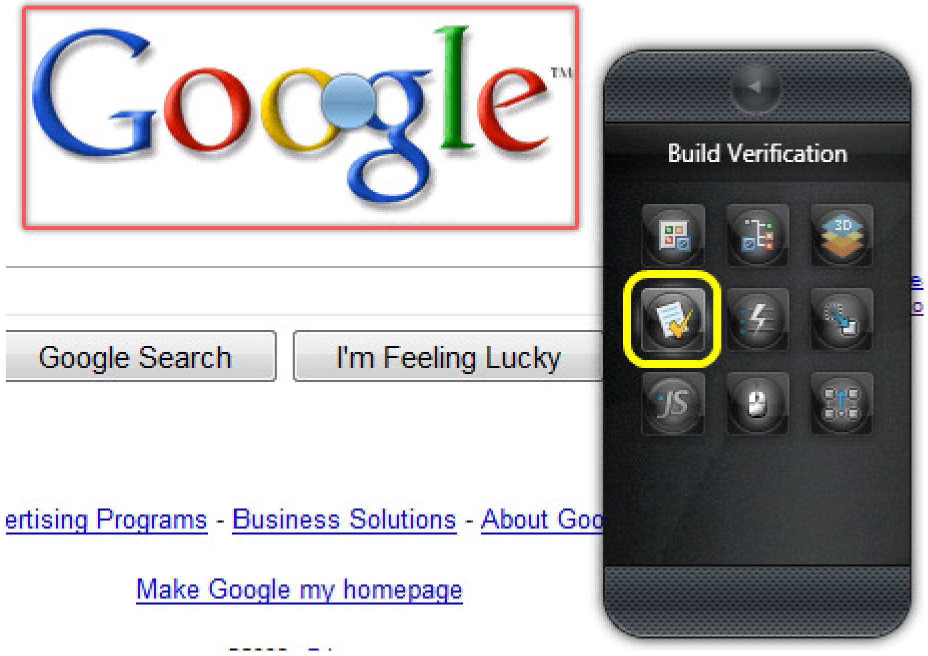
15.2 Opening Sentence Verification Builder

Follow these steps to open Sentence Verification Builder:

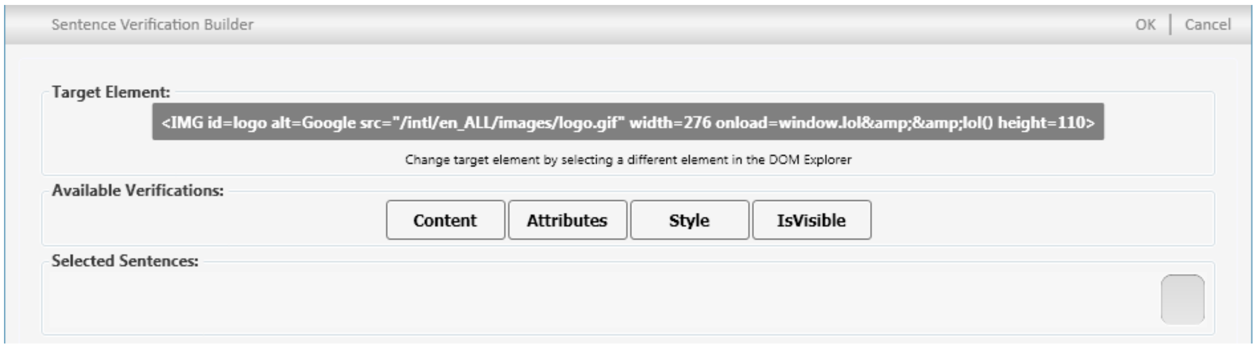
- Navigate to the web page containing the element that you want to craft a verification rule for.
- In Design Canvas Recorder, enable overlay highlighting by clicking the green overlay button on the top left-hand corner of the Recorder.



- Move the mouse over the element you want, wait one second for the little blue nub to appear then click on it. This opens the Element Menu as shown below:



- Click the “Build Verification” button from the element menu.
- Sentence Verification Builder opens with that element loaded into it.


Sentence Verification Builder
OK | Cancel
Target Element:
Change target element by selecting a different element in the DOM Explorer
Available Verifications:
Content
Attributes
Style
IsVisible
Selected Sentences:

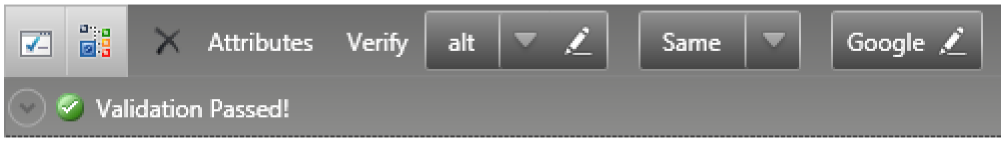
```
<IMG id=logo alt=Google src="/intl/en_ALL/images/logo.gif" width=276 onload=window.lol&amp;&amp;lol() height=110>
```

Another way to open Sentence Verification Builder is to use DOM Explorer. This is the method you must use when you are not able to highlight the element you really want in the browser window (e.g. when it is hidden behind another element, such as a <table> element).

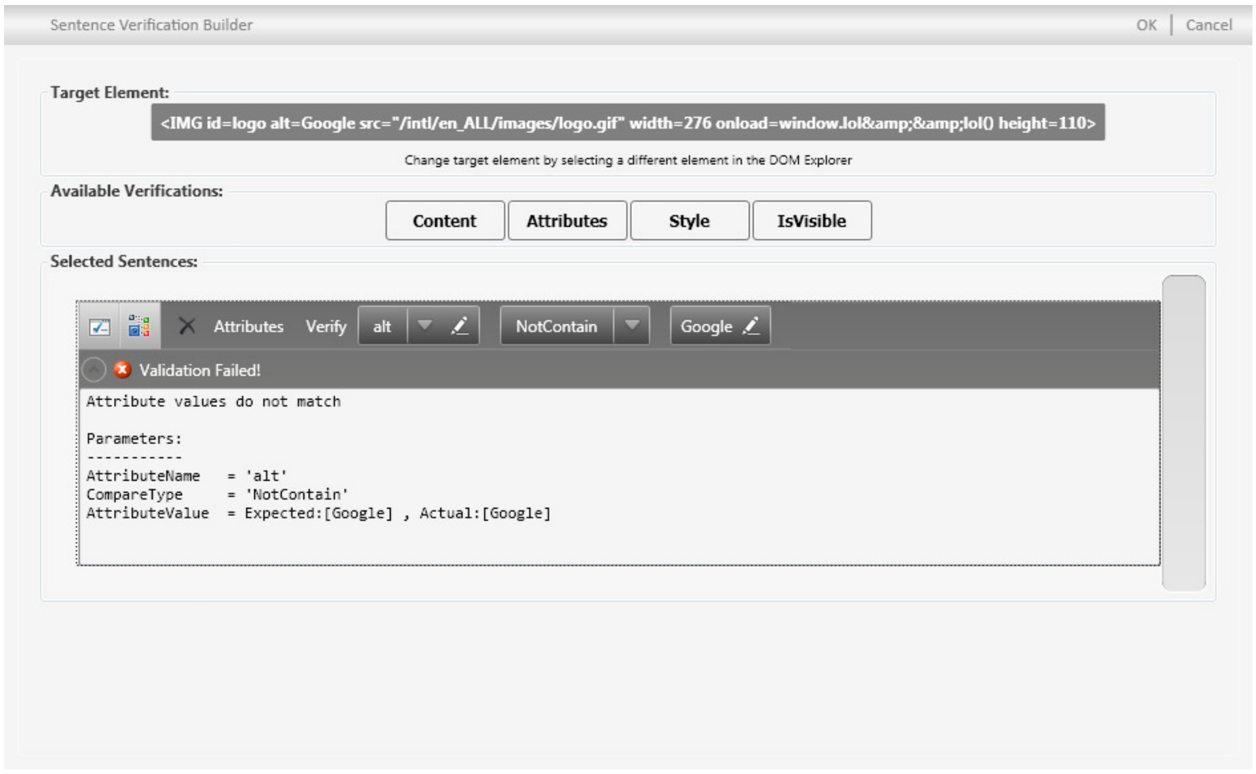
- In the DOM Explorer window drill down to the element that you want to craft a verification rule for. Sometimes a convenient shortcut is to right click on an element in the browser window that is close to the element you want (for example, a table cell element when you want to get to the parent table 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 of the DOM tree.
- Right click on the element you want and select “**Lock On Surface”**. This will open the Element Menu.
- Click the “Build Verification” button from the element menu.
- Sentence Verification Builder opens with that element loaded in it.

Each verification rule that you craft 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:



<IMG id=logo alt=Google src="/intl/en_ALL/images/logo.gif" width=276 onload=window.lol&amp;&amp;lol() height=110>

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

```
Attribute values do not match

Parameters:
-----------
AttributeName   = 'alt'
CompareType     = 'NotContain'
AttributeValue  = Expected:[Google] , Actual:[Google]
```

- – Highlights the element in the DOM Explorer window. Useful if you need to refer to the full HTML of the element as you are crafting 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 have not changed by the delete.

Clicking **OK** closes Sentence 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 9.7.4.

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

15.3 Verification Builder: Verifying Different Types of Elements

With sentence based verification you can verify the following features of HTML elements:

- Element content
- Element attribute
- Element style
- Element visibility
- Verifying a select dropdown
- Verifying checkboxes and radio buttons
- Verifying tables

You can also verify the following features of XAML (i.e. Silverlight) elements:

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

15.3.1.1 String types

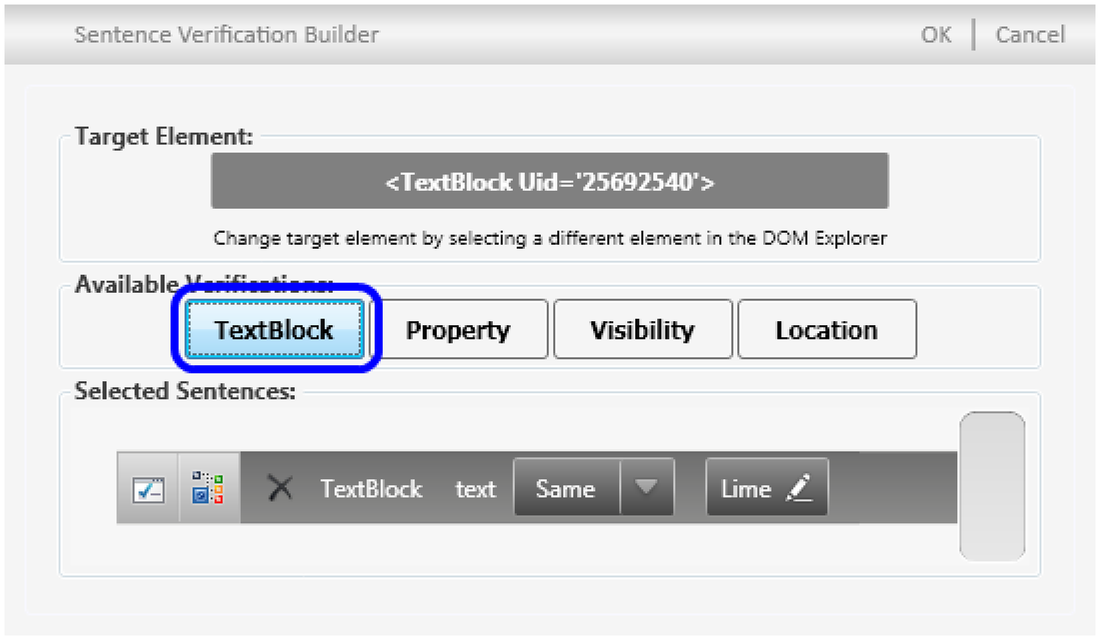
For string types you can have:

- Equal – Verify the property matches exactly with the expected value.
- Not Equal – Verify the property is different than the specified value.
- Same – Verify the property matches the expected value, ignoring case.
- Not Same – Verify the property is different than the specified value, ignoring case.
- Contains – Verify the property contains the expected value. Used mostly for ByText verifications.
- NotContains – Verify the property does *not* contain the expected value. Used mostly for ByText verifications.
- StartsWith – Verify the property starts with the expected value. Used mostly for ByText verifications.
- NotStartsWith – Verify the property does *not* starts with the expected value. Used mostly for ByText verifications.
- EndsWith – Verify the property ends with the expected value. Used mostly for ByText verifications.
- NotEndsWith – Verify the property does *not* end with the expected value. Used mostly for ByText verifications.
- Verifying element visibility
- Verifying element location

- Verifying the content of text blocks and text box controls
- Verifying Password and/or Password Masking Character

15.3.2 Verifying the content of text blocks and text box controls

Using Sentence Verification Builder you can create a verification rule to test the text content of Silverlight text block's and text boxes. Start by clicking on "**TextBox/TextBlock**" in the available verifications section.


Sentence Verification Builder
OK
Cancel
Target Element:
<TextBlock Uid='25692540'>
Change target element by selecting a different element in the DOM Explorer
Available Verifications:
TextBlock
Property
Visibility
Location
Selected Sentences:
TextBlock
text
Same
Lime

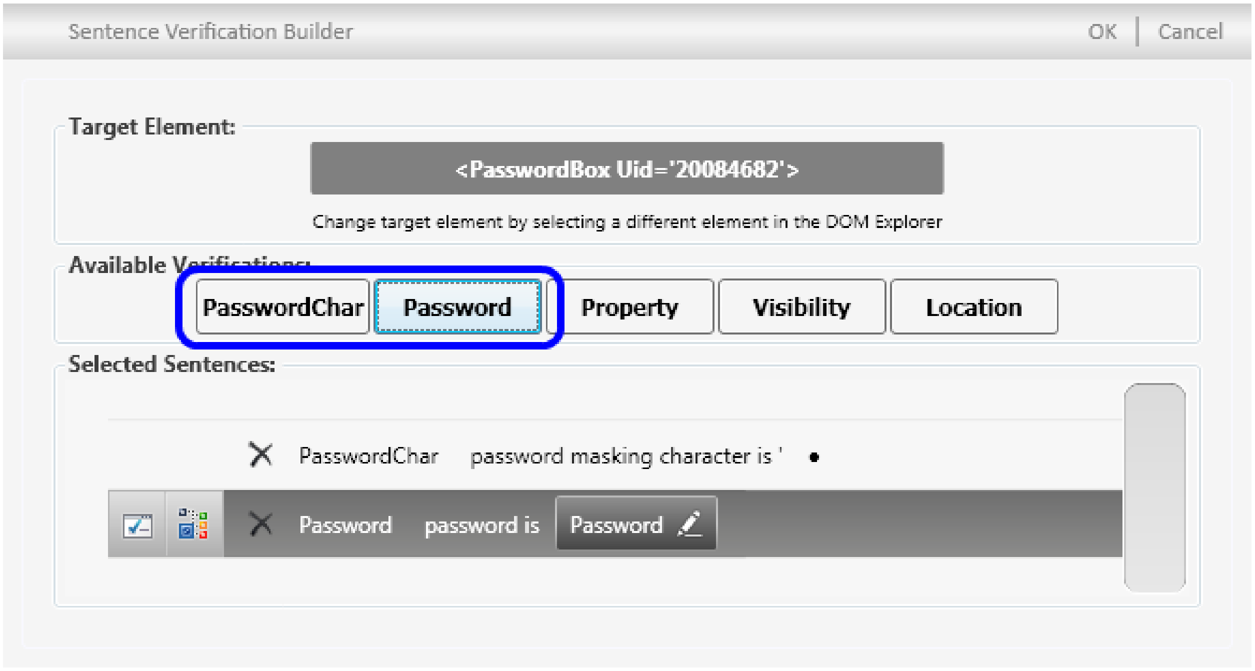
The comparison type may be one of:

- Equal – Verify the text matches exactly with the expected value.
- Not Equal – Verify the text is different than the specified value.
- Same – Verify the text matches the expected value, ignoring case.
- Not Same – Verify the text is different than the specified value, ignoring case.
- Contains – Verify the text contains the expected value. Used mostly for ByText verifications.
- NotContains – Verify the text does *not* contain the expected value. Used mostly for ByText verifications.
- StartsWith – Verify the text starts with the expected value. Used mostly for ByText verifications.
- NotStartsWith – Verify the text does *not* starts with the expected value. Used mostly for ByText verifications.
- EndsWith – Verify the text ends with the expected value. Used mostly for ByText verifications.
- NotEndsWith – Verify the text does *not* end with the expected value. Used mostly for ByText verifications.

The expected value is automatically filled with the current text contained in the TextBox/TextBlock. You may change it if needed.

15.3.3 Verifying Password and/or Password Masking Character

Using Sentence Verification Builder you can create a verification rule to test the text content or masking character of a Silverlight password input box. Start by clicking on either "**PasswordChar**" or "**Password**" in the available verifications section.


Sentence Verification Builder
OK
Cancel
Target Element:
<PasswordBox Uid='20084682'>
Change target element by selecting a different element in the DOM Explorer
Available Verifications:
PasswordChar
Password
Property
Visibility
Location
Selected Sentences:
PasswordChar password masking character is ' •
Password password is Password

Note how the current masking character and/or password are automatically filled in the expected value field. You can change the expected value if needed.

- Verifying the checked state of checkboxes and radio buttons
- Verifying the current selection in Listboxes
- Verifying the current selection in ComboBoxes

The current value of the selected index is automatically filled in the expected value. You can change it to a different integer value if you wish.

- Verifying the URL of a HyperlinkButton

The comparison type can be one of:

- Equal – Verify the URL matches exactly with the expected value.
- Not Equal – Verify the URL is different than the specified value.
- Same – Verify the URL matches the expected value, ignoring case.
- Not Same – Verify the URL is different than the specified value, ignoring case.
- Contains – Verify the URL contains the expected value.
- NotContains – Verify the URL does *not* contain the expected value.

- StartsWith – Verify the URL starts with the expected value.
- NotStartsWith – Verify the URL does *not* starts with the expected value.
- EndsWith – Verify the URL ends with the expected value.
- NotEndsWith – Verify the URL does *not* end with the expected value.

The current value of the URL for the HyperlinkButton is automatically filled in the expected value. You can change it to a different URL if needed.

- Verifying the slider value of a Slider control

The comparison type can be one of:

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

The current value of the Slider is automatically filled in the expected value. You can change it to a different integer value if needed.

- Verifying the watermark in Datepickers

The comparison type can be one of:

- Equal – Verify the watermark matches exactly with the expected value.
- Not Equal – Verify the watermark is different than the specified value.
- Same – Verify the watermark matches the expected value, ignoring case.
- Not Same – Verify the watermark is different than the specified value, ignoring case.
- Contains – Verify the watermark contains the expected value.
- NotContains – Verify the watermark does *not* contain the expected value.
- StartsWith – Verify the watermark starts with the expected value.
- NotStartsWith – Verify the watermark does *not* starts with the expected value.
- EndsWith – Verify the watermark ends with the expected value.
- NotEndsWith – Verify the watermark does *not* end with the expected value.

The current value of the watermark is automatically filled in the expected value. You can change it to a different string if needed.

- Verifying the date selection in Calenders

The comparison type can be one of:

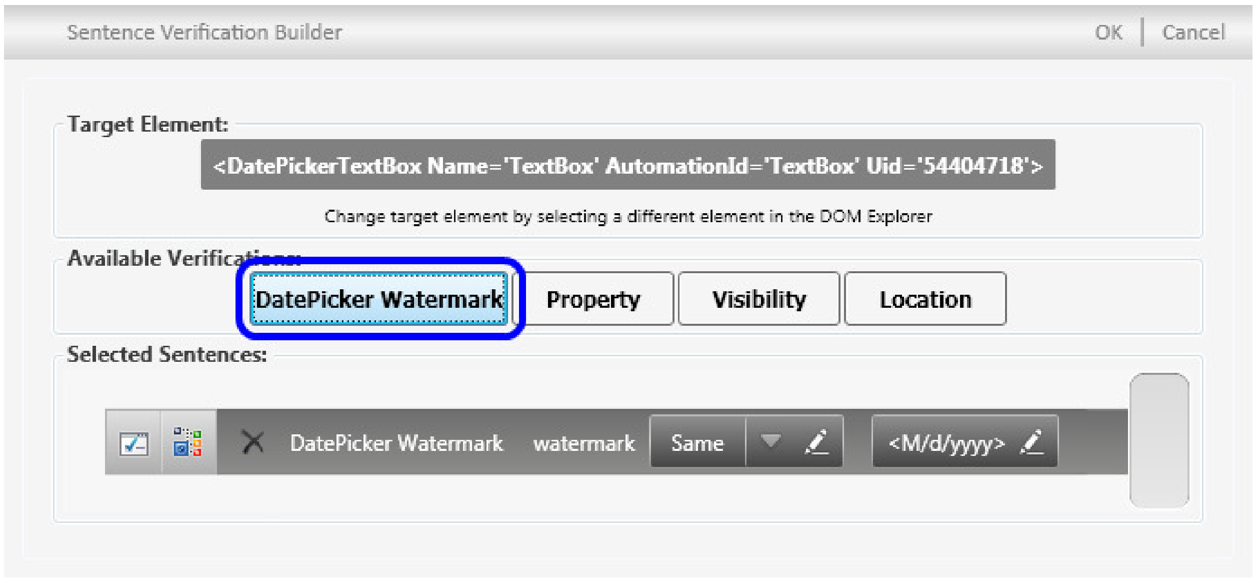
- Equals – Verify that the Slider's current value matches exactly with the expected value.
- NotEqual – Verify that the Slider's current value is not equal to the expected value.

- GreaterThan – Verify that the Slider's current value is greater than the expected value.
- GreaterThanOrEqual – Verify that the Slider's current value is greater than or equal to the expected value.
- LessThan – Verify that the Slider's current value is less than the expected value.
- LessThanOrEqual – Verify that the Slider's current value is less than or equal to the expected value.

The current value of the Slider is automatically filled in the expected value. You can change it to a different integer value if needed.

15.3.4 Verifying the watermark in Datepickers

Using Sentence Verification Builder you can create an element verification rule to test the current watermark of a Silverlight DatePicker control. Start by clicking on "**DatePicker**" in the available verifications section.


Sentence Verification Builder
OK
Cancel
Target Element:
<DatePickerTextBox Name='TextBox' AutomationId='TextBox' Uid='54404718'>
Change target element by selecting a different element in the DOM Explorer
Available Verifications:
DatePicker Watermark
Property
Visibility
Location
Selected Sentences:
DatePicker Watermark
watermark
Same
<M/d/yyyy>

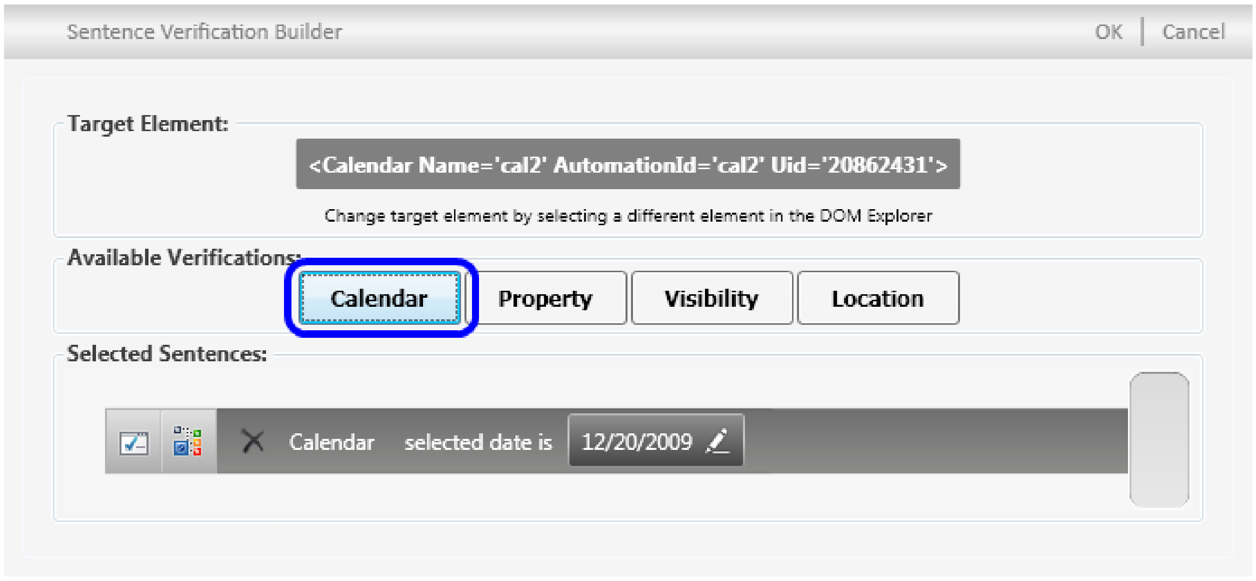
The comparison type can be one of:

- Equal – Verify the watermark matches exactly with the expected value.
- Not Equal – Verify the watermark is different than the specified value.
- Same – Verify the watermark matches the expected value, ignoring case.
- Not Same – Verify the watermark is different than the specified value, ignoring case.
- Contains – Verify the watermark contains the expected value.
- NotContains – Verify the watermark does *not* contain the expected value.
- StartsWith – Verify the watermark starts with the expected value.
- NotStartsWith – Verify the watermark does *not* starts with the expected value.
- EndsWith – Verify the watermark ends with the expected value.
- NotEndsWith – Verify the watermark does *not* end with the expected value.

The current value of the watermark is automatically filled in the expected value. You can change it to a different string if needed.

15.3.5 Verifying the date selection in Calenders

Using Sentence Verification Builder you can create an element verification rule to test which date is selected in a Silverlight Calendar control. Start by clicking on “Calendar” in the available verifications section.



<Calendar Name='cal2' AutomationId='cal2' Uid='20862431'>

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

Calendar selected date is 12/20/2009

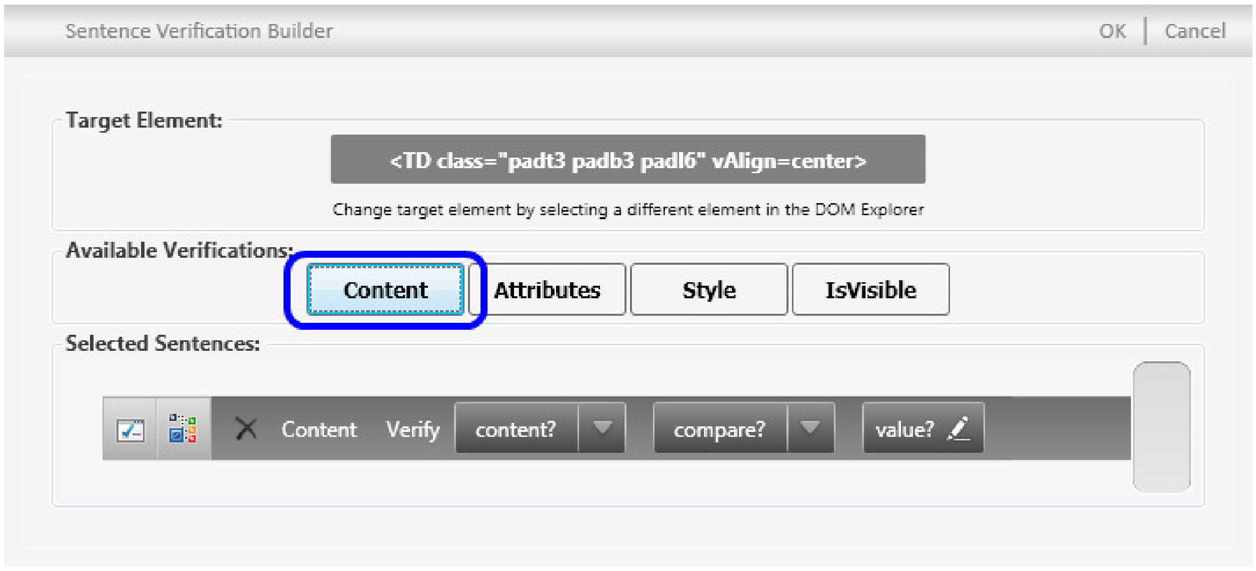
The currently selected date is automatically filled in the expected value. You can change it to a different date if needed.

NOTE: Multiple date selection is not currently supported. Only the first date selected in a multi-select calendar control will be used.

- Verifying the tab selection of a tabbed control

15.3.6 Element content

All elements have the ability to verify element content, both HTML and text content. Start by clicking “**Content**” in the list of Available Verifications. A new content verification sentence is added to the Selected Sentences section:


Sentence Verification Builder
OK
Cancel
Target Element:
<TD class="padt3 padb3 padl6" vAlign=center>
Change target element by selecting a different element in the DOM Explorer
Available Verifications:
Content
Attributes
Style
IsVisible
Selected Sentences:
Content
Verify
content?
compare?
value?

Now you just need to specify the three options for a content verification:

15.3.6.1 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, “<div id="div2">”. The same care must be taken for this type of compare as for InnerText.
- OuterMarkup – The outer markup of a tag. For example, “<div id="div4">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.

15.3.6.2 Compare

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

- Exact – Verify the content matches exactly with the expected value.
- Same – Verify the content matches the expected value, ignoring case.

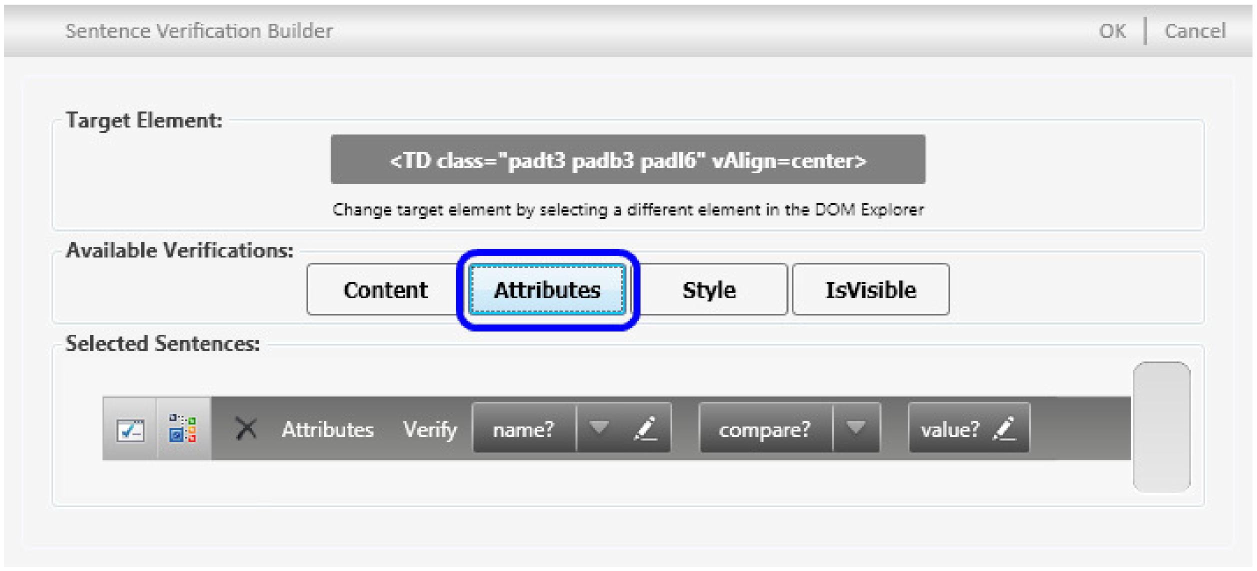
- Contains – Verify the content contains the expected value. Used mostly for those attributes having string values rather than numeric values.
- NotContain – Verify the content does *not* contain the expected value. Used mostly for those attributes having string values rather than numeric values.
- StartsWith – Verify the content starts with the expected value. Used mostly for those attributes having string values rather than numeric values.
- EndsWith – Verify the content ends with the expected value. Used mostly for those attributes having string values rather than numeric values.
- RegEx – Verify the content 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.

15.3.6.3 Value

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

15.3.7 Element attribute

Using Sentence Verification Builder you can create a verification rule that evaluates any attribute set on the element. Start by clicking "**Attributes**" in the list of Available Verifications. A new Attribute verification sentence will be added to the Selected Sentences section:


Sentence Verification Builder
OK
Cancel
Target Element:
<TD class="padt3 padb3 padl6" vAlign=center>
Change target element by selecting a different element in the DOM Explorer
Available Verifications:
Content
Attributes
Style
IsVisible
Selected Sentences:
Attributes
Verify
name?
compare?
value?

Now you just need to specify the three options that go along with an attribute verification:

15.3.7.1 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.

15.3.7.2 Compare

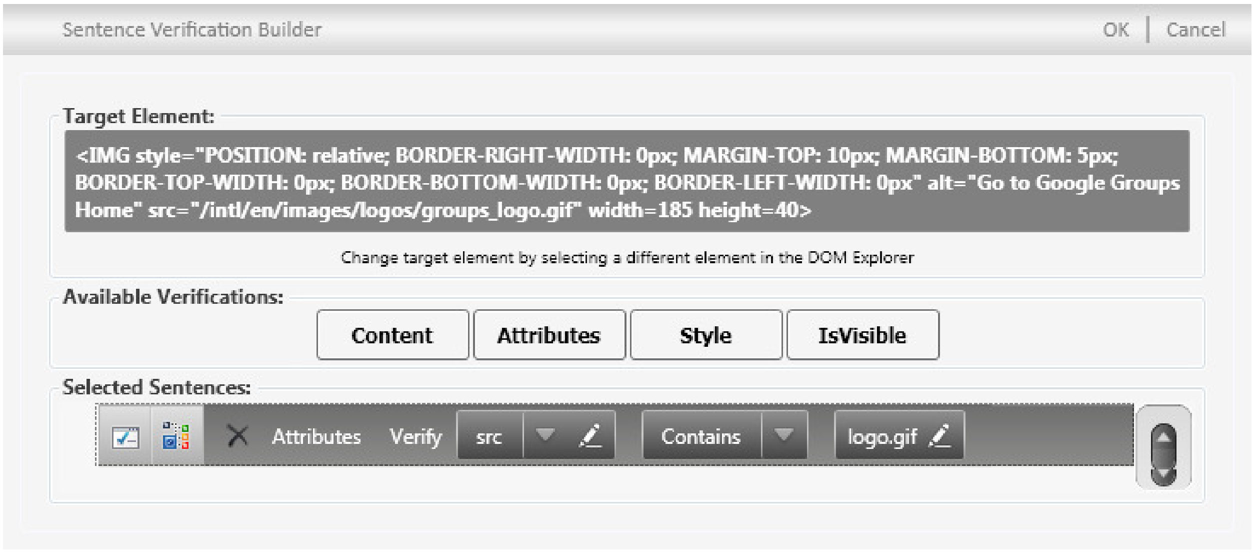
Compare – Select which compare type to be performed. You can select from:

- Exact – Verify the attribute value matches exactly with the expected value.
- Same – Verify the attribute value matches the expected value, ignoring case.
- 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 ends 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.

15.3.7.3 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”:

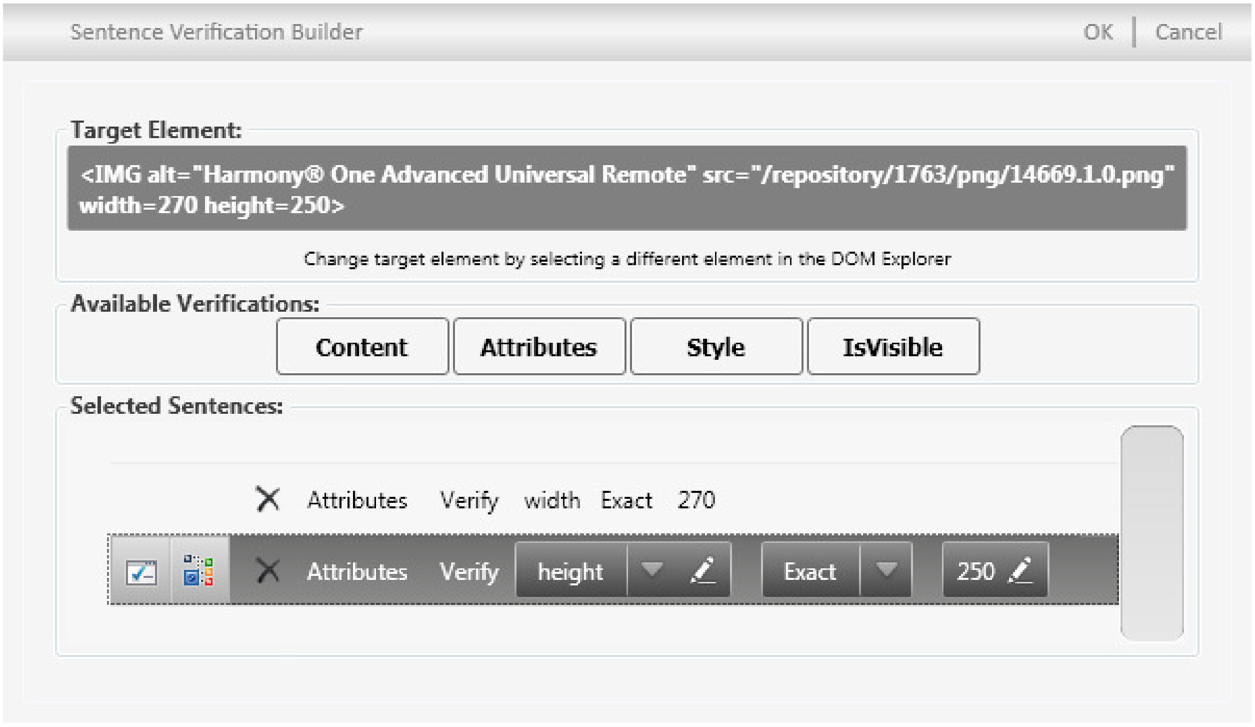


<IMG style="POSITION: relative; BORDER-RIGHT-WIDTH: 0px; MARGIN-TOP: 10px; MARGIN-BOTTOM: 5px; BORDER-TOP-WIDTH: 0px; BORDER-BOTTOM-WIDTH: 0px; BORDER-LEFT-WIDTH: 0px" alt="Go to Google Groups Home" src="/intl/en/images/logos/groups_logo.gif" width=185 height=40>

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

Attributes Verify src Contains logo.gif

- This verifies that the width attribute equals 270 and that the height attribute equals 250. Both element verification rules will examine the same <IMG> element. When saved the two verification steps are added to Test Explorer, one for each element verification rule:



<IMG alt="Harmony® One Advanced Universal Remote" src="/repository/1763/png/14669.1.0.png" width=270 height=250>

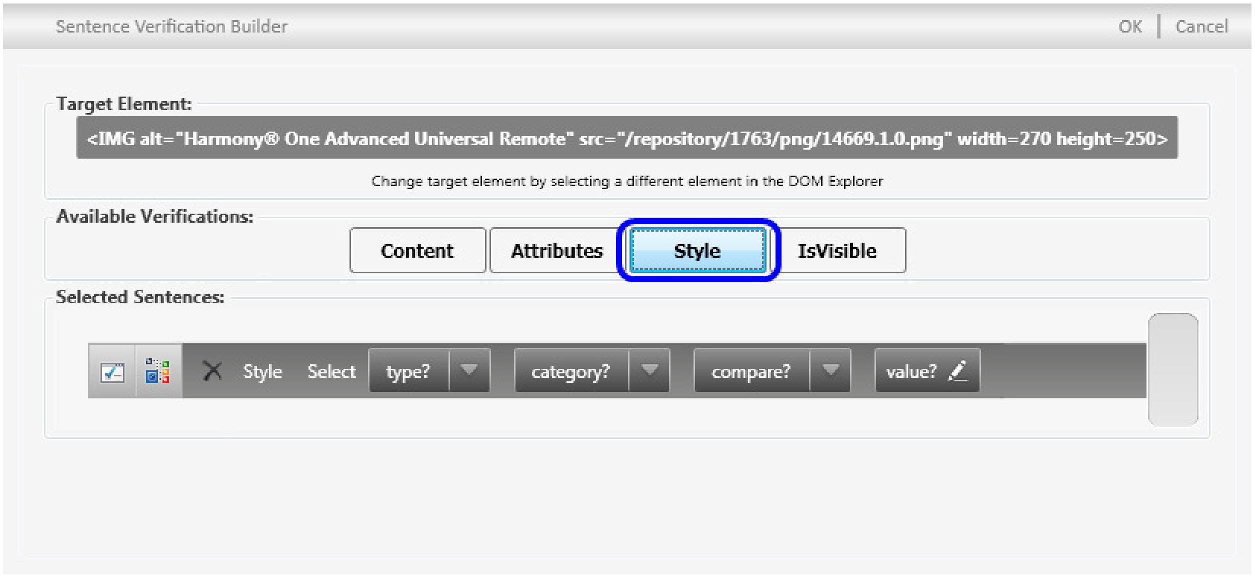
Change target element by selecting a different element in the DOM Explorer

Attributes Verify width Exact 270
Attributes Verify height Exact 250

15.3.8 Element style

Using Sentence Verification 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 clicking "**Style**" from the list of available verifications:



<IMG alt="Harmony® One Advanced Universal Remote" src="/repository/1763/png/14669.1.0.png" width=270 height=250>

Now you simply need to specify the four options that go along with a style verification:

15.3.8.1 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.

15.3.8.2 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

15.3.8.3 Compare

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

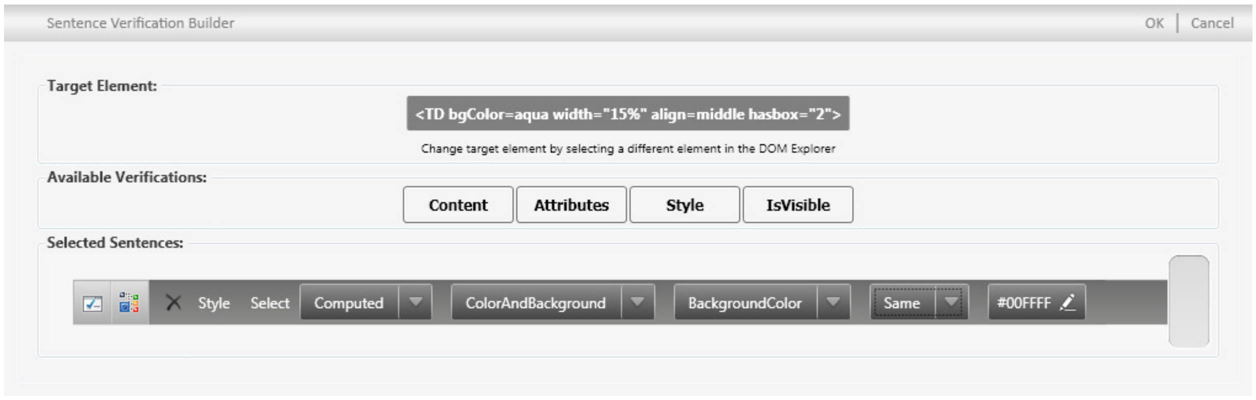
- Exact – Verify the style value matches exactly with the expected value.
- Same – Verify the style value matches the expected value, ignoring case.
- Contains – Verify the style value contains the expected value. Used mostly for those styles having string values rather than numeric values.
- NotContain – 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 ends 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.

15.3.8.4 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 "aqua" was translated from the HTML <TD bgColor=aqua width="15%" align="middle" hasbox="2"> to its hex equivalent "#00FFFF":



<TD bgColor=aqua width="15%" align=middle hasbox="2">

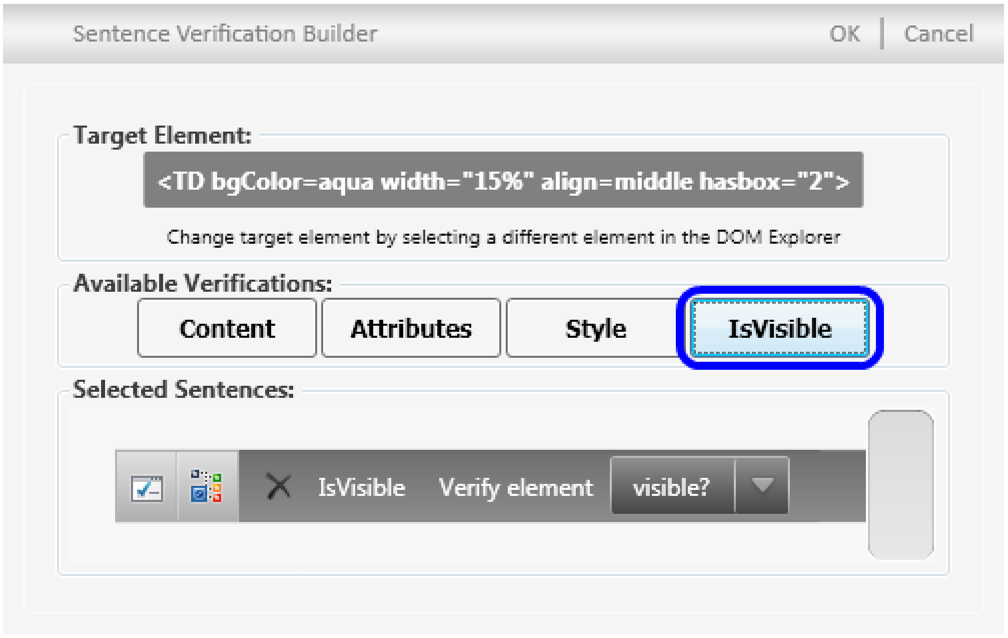
Change target element by selecting a different element in the DOM Explorer

Style Select Computed ColorAndBackground BackgroundColor Same #00FFFF

15.3.9 Element visibility

Using Sentence Verification Builder you can create a verification rule that evaluates the visibility state of the element. WebAii tests whether or not an element is visible 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 clicking IsVisible in the list of Available Verifications:



<TD bgColor=aqua width="15%" align=middle hasbox="2">

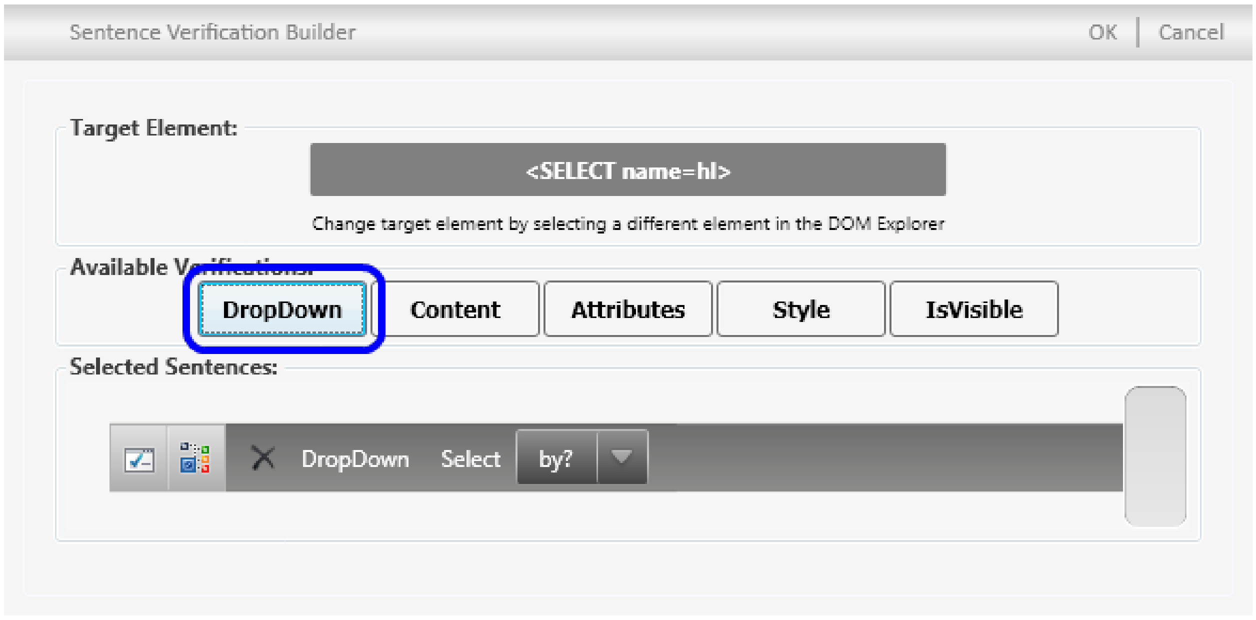
Change target element by selecting a different element in the DOM Explorer

15.3.9.1 Visible

Now all you need to specify is "**true**" or "**false**" for the "**visible**" option displayed:

15.3.10 Verifying a select dropdown element

Using Sentence Verification Builder you can create an element verification rule to validate what is currently selected for a <select> element. Start by clicking "**DropDown**" from the list of available verifications (NOTE: This button is only present when you have highlighted a <select> tag):



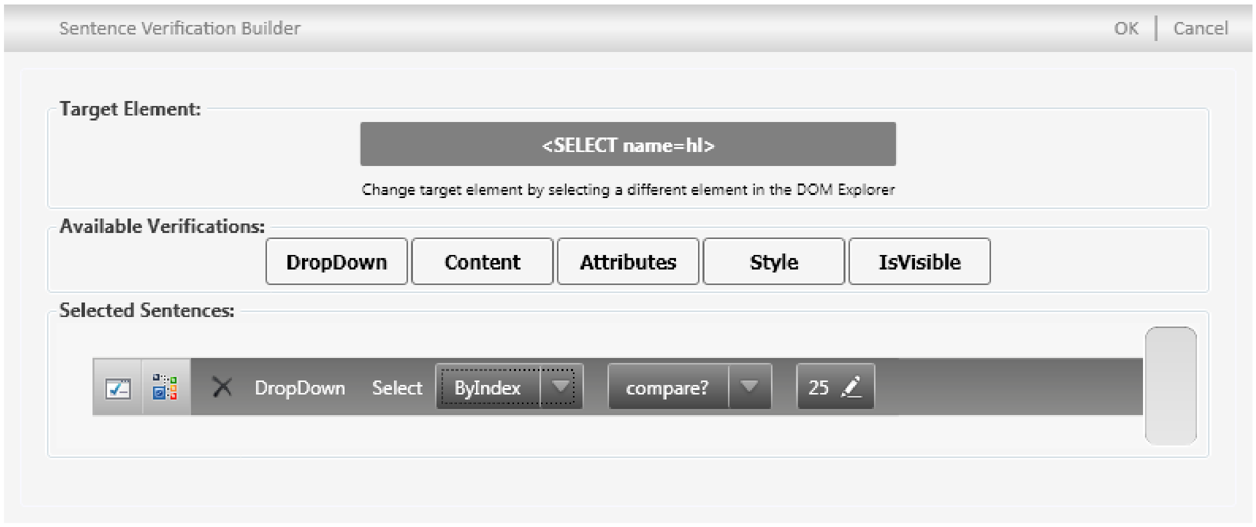
Now you need to select the type of DropDown verification:

15.3.10.1 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 index 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 the comparison Type, two additional options are displayed:


Sentence Verification Builder
OK
Cancel
Target Element:
<SELECT name=hl>
Change target element by selecting a different element in the DOM Explorer
Available Verifications:
DropDown
Content
Attributes
Style
IsVisible
Selected Sentences:
DropDown Select ByIndex compare? 25

15.3.10.2 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.

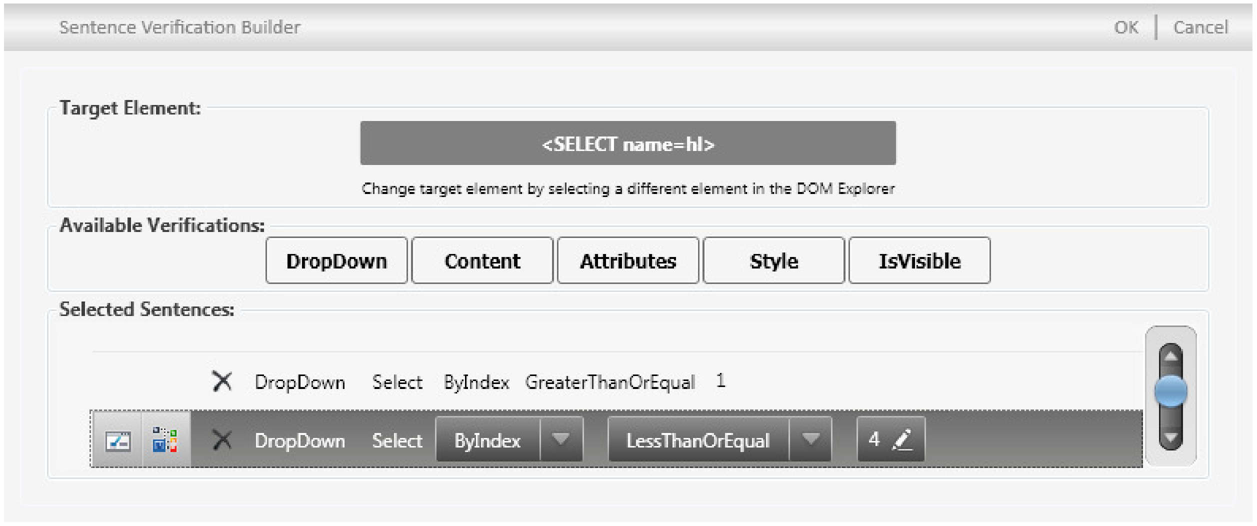
When you select ByValue or ByText then you can select from the following for the compare type to be performed:

- Exact – Verify the selection matches exactly with the expected value.
- Same – Verify the selection matches the expected value, ignoring case.
- 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 ends 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.

15.3.10.3 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, i.e. 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:



Target Element: <SELECT name=hl>

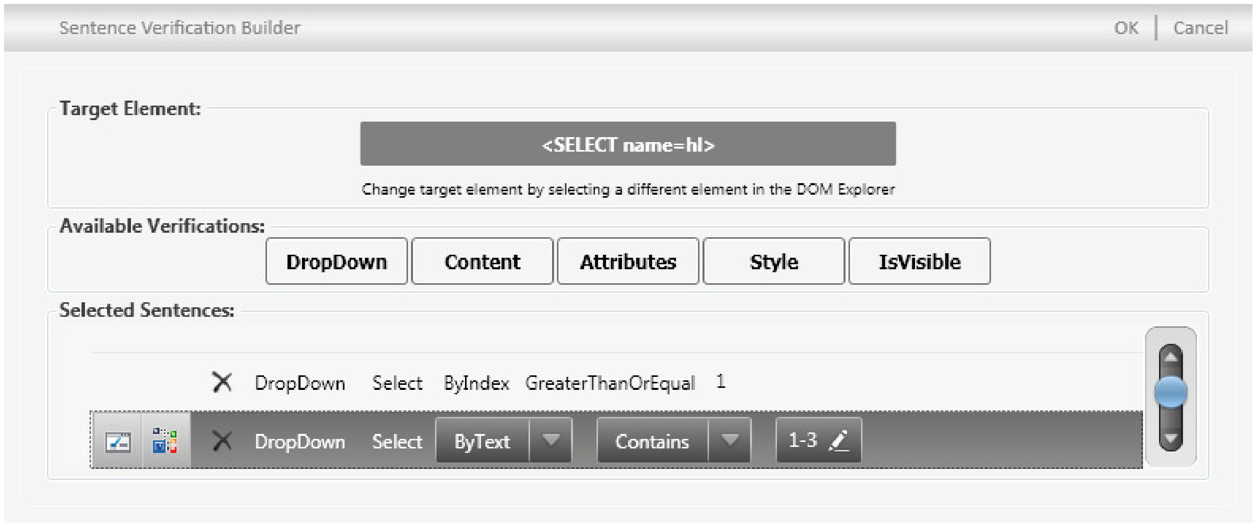
Change target element by selecting a different element in the DOM Explorer

Selected Sentences:

DropDown Select ByIndex GreaterThanOrEqual 1

DropDown Select ByIndex LessThanOrEqual 4

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



Target Element: <SELECT name=hl>

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

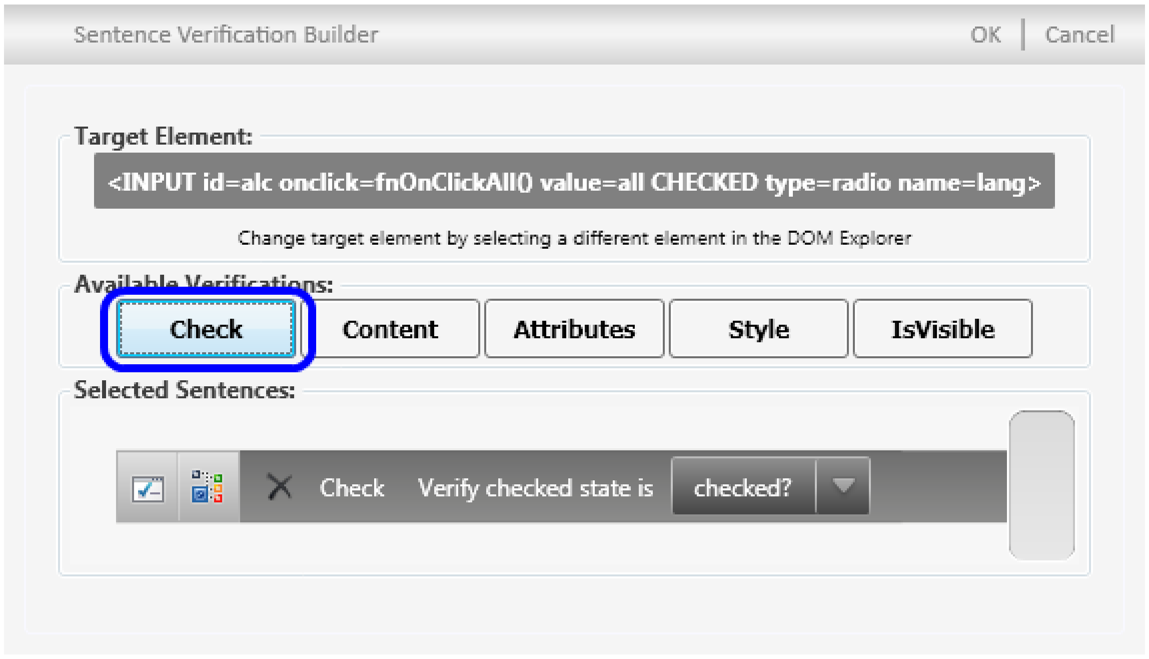
Selected Sentences:

DropDown Select ByIndex GreaterThanOrEqual 1

DropDown Select ByText Contains 1-3

15.3.11 Verifying checkboxes and radio buttons

Using Sentence 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 clicking "**Check**" in the available verifications section.



<INPUT id=alc onclick=fnOnClickAll() value=all CHECKED type=radio name=lang>

Check Verify checked state is checked?

Now all you need to do is select "**true**" or "**false**" for the checked option.

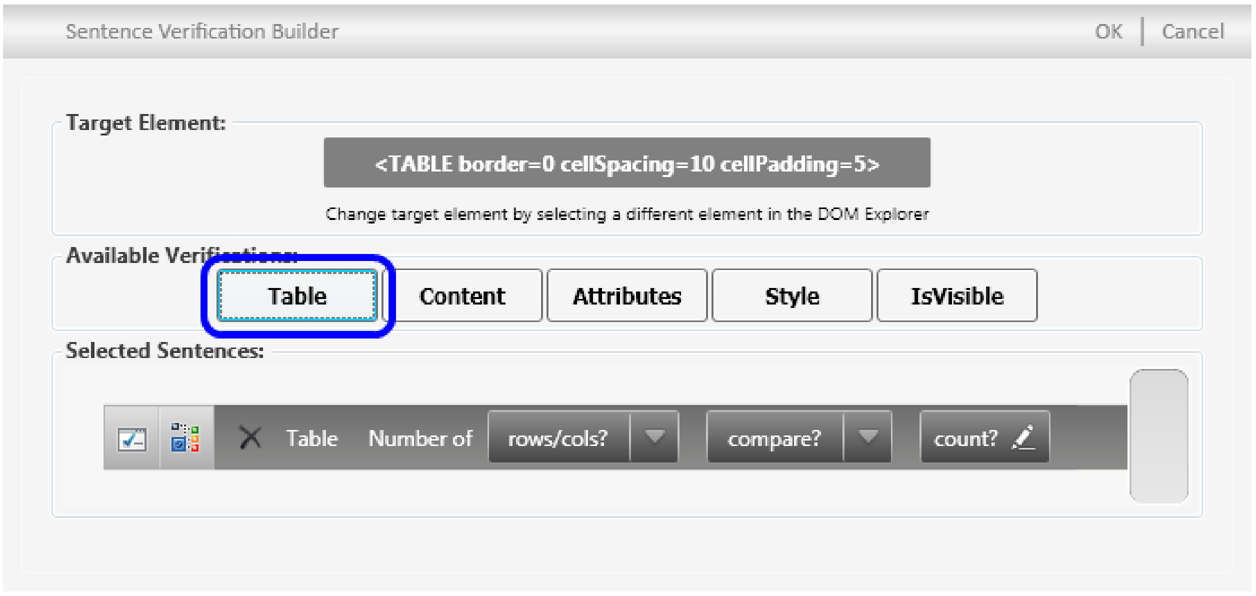
15.3.11.1 Checked

Checked – Choose either:

- True – The checkbox/radio button is checked.
- False – The checkbox/radio button is unchecked.

15.3.12 Verifying tables

Using Sentence 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 clicking “**Table**” in the available verifications section.



Sentence Verification Builder
OK
Cancel
Target Element:
<TABLE border=0 cellSpacing=10 cellPadding=5>
Change target element by selecting a different element in the DOM Explorer
Available Verifications:
Table
Content
Attributes
Style
IsVisible
Selected Sentences:
Table
Number of
rows/cols?
compare?
count?

Now all you need to do is specify the three options that go along with a table verification.

15.3.12.1 Rows/columns

Rows/Columns – Select either:

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

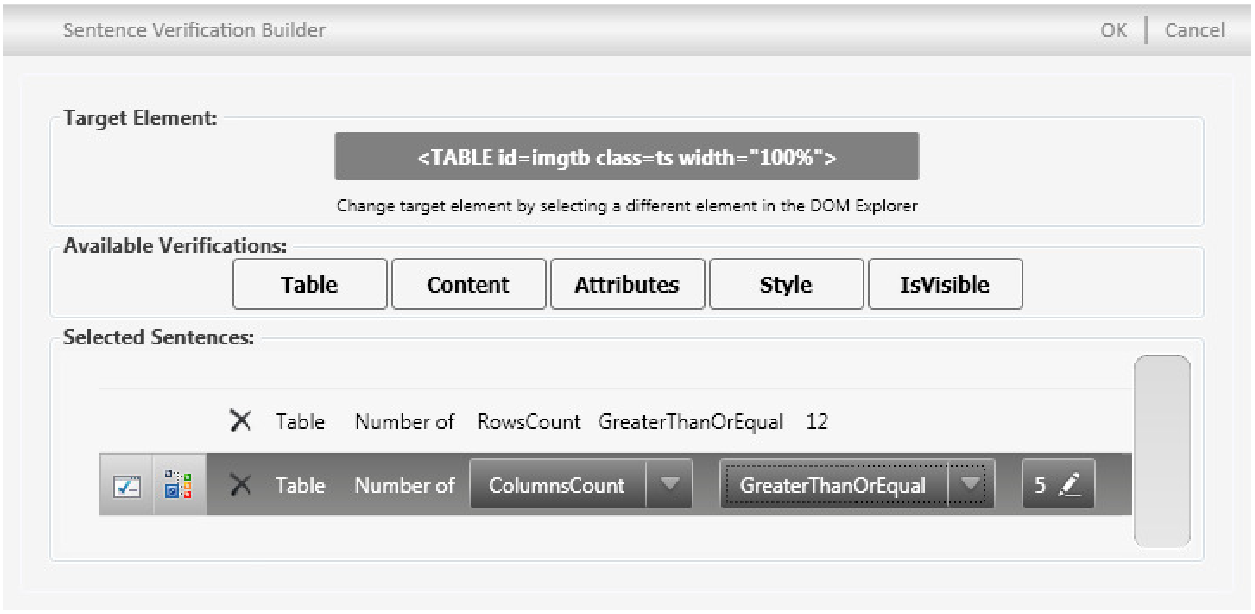
15.3.12.2 Compare

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.

15.3.12.3 Value

Value – This is the expected value to use in the comparison. The value is filled with the current count of either the number of rows or the number of columns depending on the “rows/columns” selection you made. How this value is used by the verification depends on the comparison type selected as described above. The following example shows a verification that the number of rows is at least 12 and the number of columns is at least 5.


Sentence Verification Builder
OK
Cancel
Target Element:
Change target element by selecting a different element in the DOM Explorer
Available Verifications:
Table
Content
Attributes
Style
IsVisible
Selected Sentences:

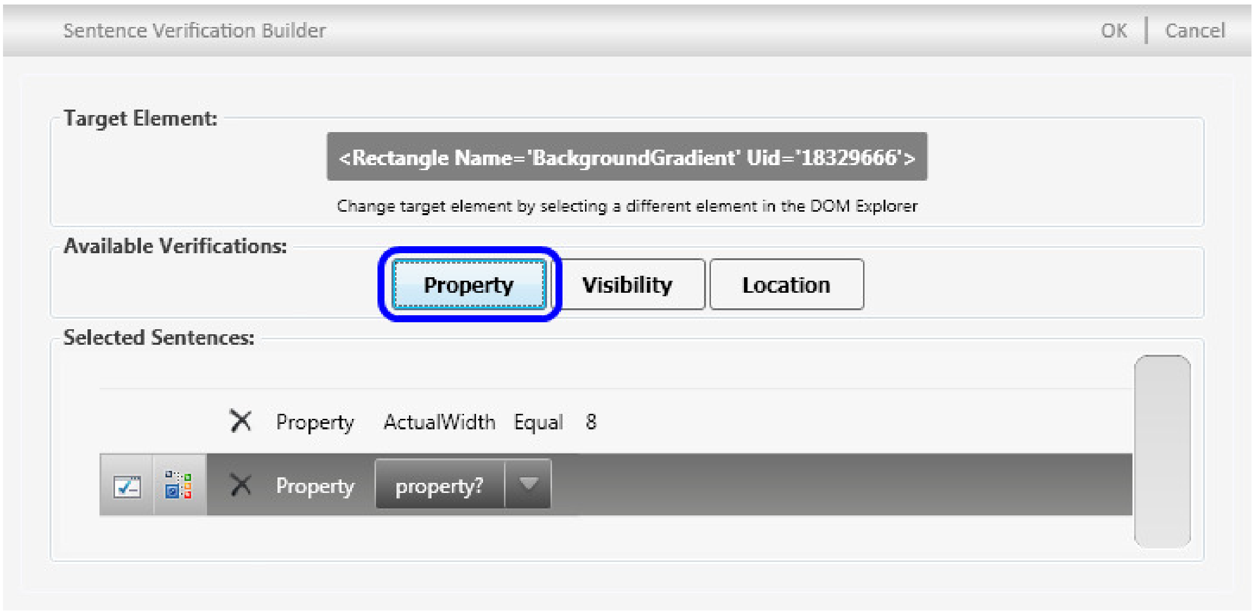
<TABLE id=imgtb class=ts width="100%">

Table Number of RowsCount GreaterThanOrEqual 12

Table Number of ColumnsCount GreaterThanOrEqual 5

15.3.13 Verifying a property of the element

Using Sentence Verification Builder you can create an element verification rule to test the value of a property of a XAML element. When you open Sentence Verification Builder Design Canvas queries the element to retrieve the list of properties available for the currently highlighted element and presents the list in the property dropdown. Start by clicking "**Property**" in the available verifications section.

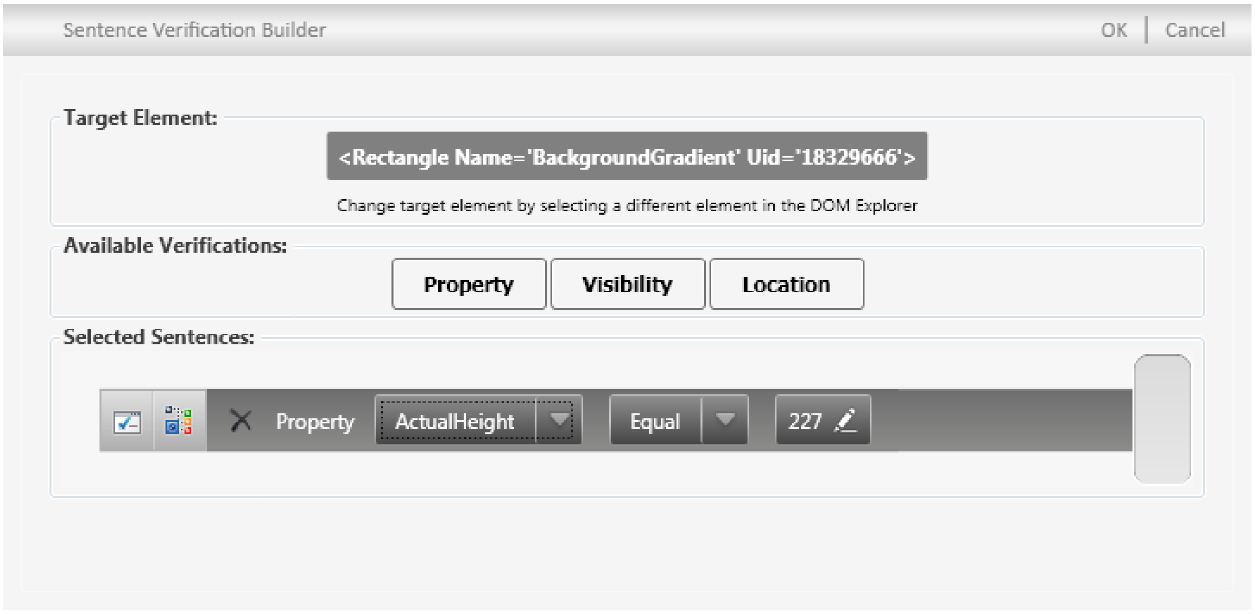

Sentence Verification Builder
OK
Cancel
Target Element:
Change target element by selecting a different element in the DOM Explorer
Available Verifications:
Property
Visibility
Location
Selected Sentences:

<Rectangle Name='BackgroundGradient' Uid='18329666'>

Property ActualWidth Equal 8

Property property?

Now you need to select which property you want to validate. After you select the name of the property, you'll get two more options: comparison type and value.



<Rectangle Name='BackgroundGradient' Uid='18329666'>

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

Which options are available in the comparison type dropdown depends on the type of value for the property you selected.

15.3.13.1 Boolean types

For Boolean types you can have "Equal" or "Not Equal"

15.3.13.2 Numeric types

For numeric types you can have:

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

15.3.13.3 String types

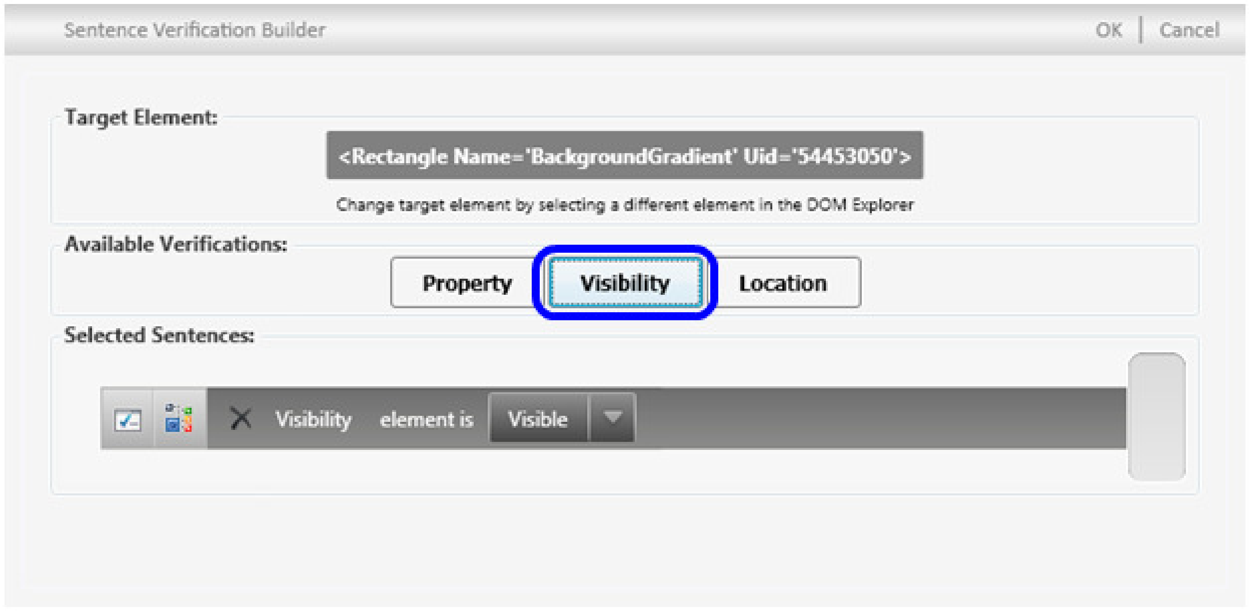
For string types you can have:

- Equal – Verify the property matches exactly with the expected value.
- Not Equal – Verify the property is different than the specified value.
- Same – Verify the property matches the expected value, ignoring case.
- Not Same – Verify the property is different than the specified value, ignoring case.

- Contains – Verify the property contains the expected value. Used mostly for ByText verifications.
- NotContains – Verify the property does *not* contain the expected value. Used mostly for ByText verifications.
- StartsWith – Verify the property starts with the expected value. Used mostly for ByText verifications.
- NotStartsWith – Verify the property does *not* starts with the expected value. Used mostly for ByText verifications.
- EndsWith – Verify the property ends with the expected value. Used mostly for ByText verifications.
- NotEndsWith – Verify the property does *not* end with the expected value. Used mostly for ByText verifications.

15.3.14 Verifying element visibility

Using Sentence Verification Builder you can create an element verification rule to test the display state of a XAML element. Start by clicking on "**Visibility**" in the available verifications section.



<Rectangle Name='BackgroundGradient' Uid='54453050'>

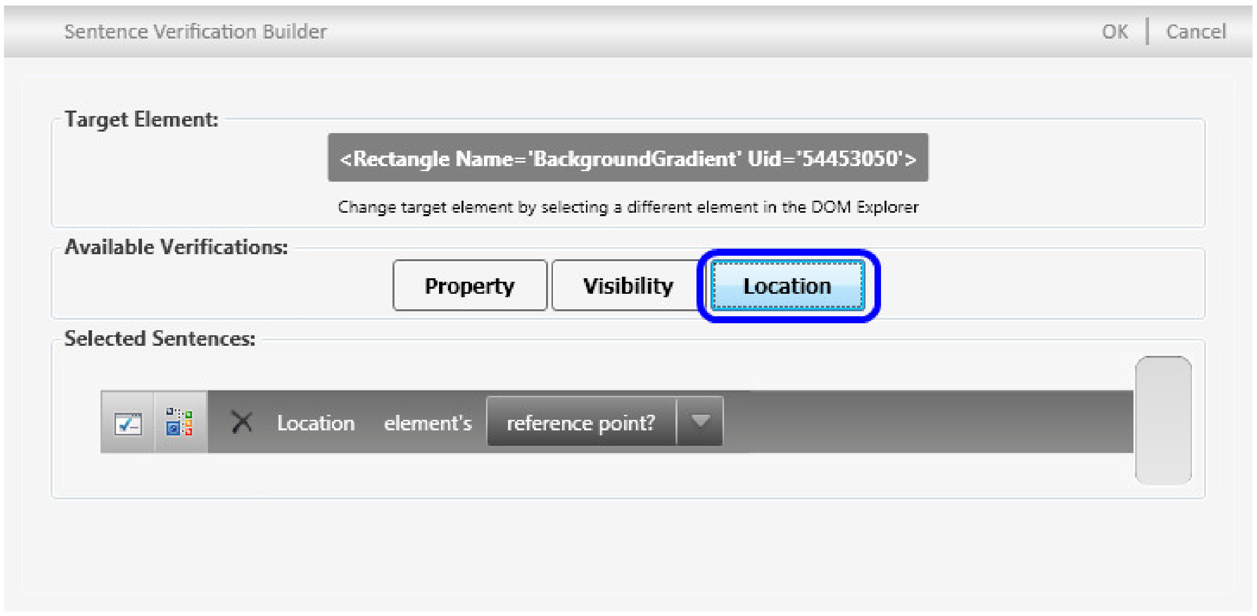
Change target element by selecting a different element in the DOM Explorer

The expected value can be either:

- Visible – Display the element.
- Collapsed – Do not display the element, and do not reserve space for it in the layout.

15.3.15 Verifying element location

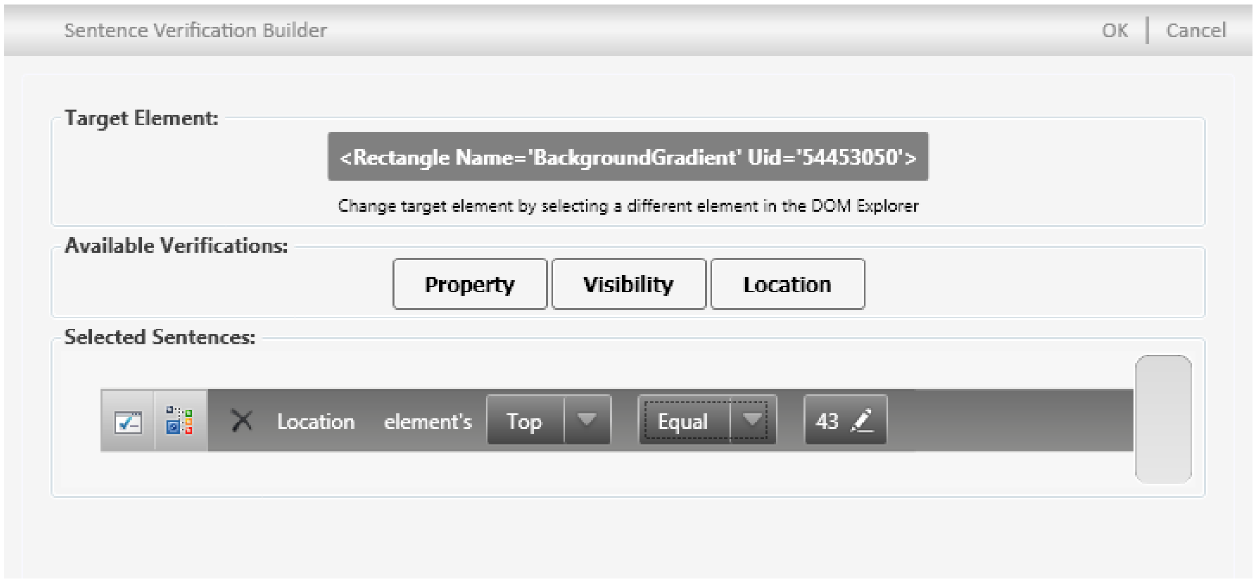
Using Sentence Verification Builder you can create an element verification rule to test the location a XAML element is being displayed at within the Silverlight's application window. Start by clicking on "**Location**" in the available verifications section.


Sentence Verification Builder
OK | Cancel
Target Element:
<Rectangle Name='BackgroundGradient' Uid='54453050'>
Change target element by selecting a different element in the DOM Explorer
Available Verifications:
Property
Visibility
Location
Selected Sentences:
Location element's reference point?

Now you need to select the reference point to be checked. It can be one of:

- Left – Verifies the left edge of the element.
- Right – Verifies the right edge of the element.
- Top – Verifies the top edge of the element.
- Bottom – Verifies the bottom edge of the element.

Once you have selected the reference point, you get two more options that you can specify.


Sentence Verification Builder
OK
Cancel
Target Element:
<Rectangle Name='BackgroundGradient' Uid='54453050'>
Change target element by selecting a different element in the DOM Explorer
Available Verifications:
Property
Visibility
Location
Selected Sentences:
Location
element's
Top
Equal
43

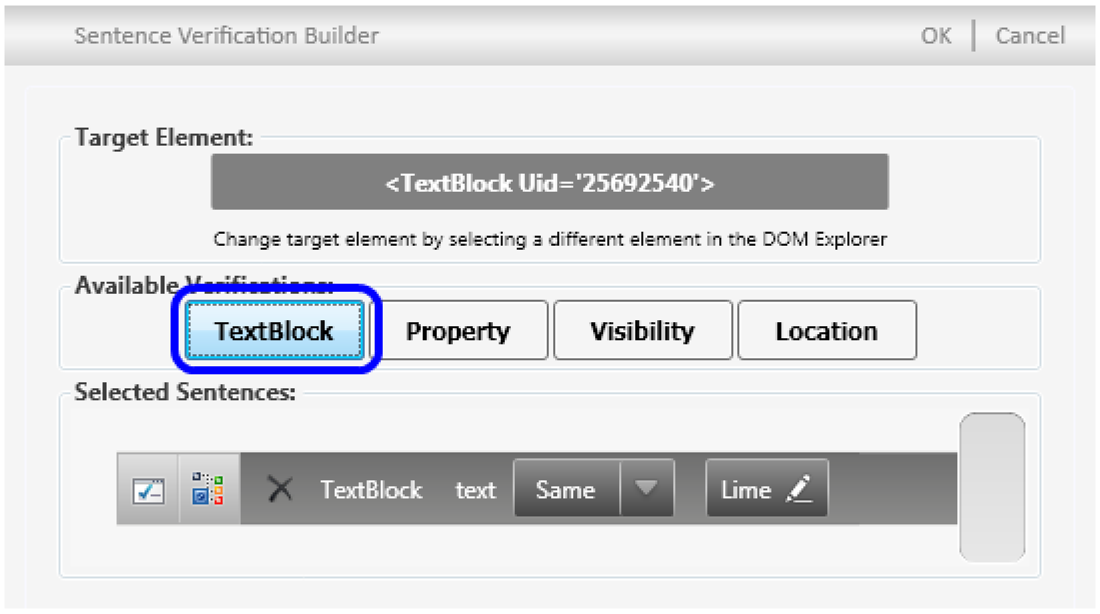
The comparison type can be one of:

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

The expected value can be any valid integer.

15.3.16Verifying the content of text blocks and text box controls

Using Sentence Verification Builder you can create a verification rule to test the text content of Silverlight text block's and text boxes. Start by clicking on "**TextBox/TextBlock**" in the available verifications section.


Sentence Verification Builder
OK
Cancel
Target Element:
<TextBlock Uid='25692540'>
Change target element by selecting a different element in the DOM Explorer
Available Verifications:
TextBlock
Property
Visibility
Location
Selected Sentences:
TextBlock
text
Same
Lime

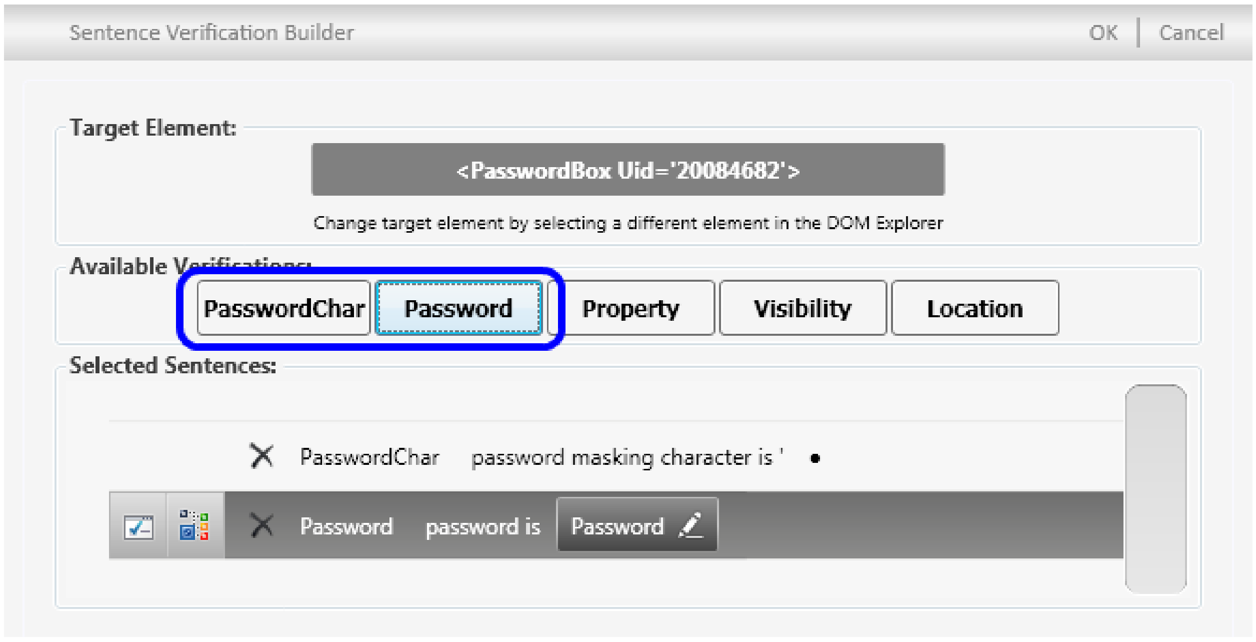
The comparison type may be one of:

- Equal – Verify the text matches exactly with the expected value.
- Not Equal – Verify the text is different than the specified value.
- Same – Verify the text matches the expected value, ignoring case.
- Not Same – Verify the text is different than the specified value, ignoring case.
- Contains – Verify the text contains the expected value. Used mostly for ByText verifications.
- NotContains – Verify the text does *not* contain the expected value. Used mostly for ByText verifications.
- StartsWith – Verify the text starts with the expected value. Used mostly for ByText verifications.
- NotStartsWith – Verify the text does *not* starts with the expected value. Used mostly for ByText verifications.
- EndsWith – Verify the text ends with the expected value. Used mostly for ByText verifications.
- NotEndsWith – Verify the text does *not* end with the expected value. Used mostly for ByText verifications.

The expected value is automatically filled with the current text contained in the TextBox/TextBlock. You may change it if needed.

15.3.17 Verifying Password and/or Password Masking Character

Using Sentence Verification Builder you can create a verification rule to test the text content or masking character of a Silverlight password input box. Start by clicking on either "**PasswordChar**" or "**Password**" in the available verifications section.

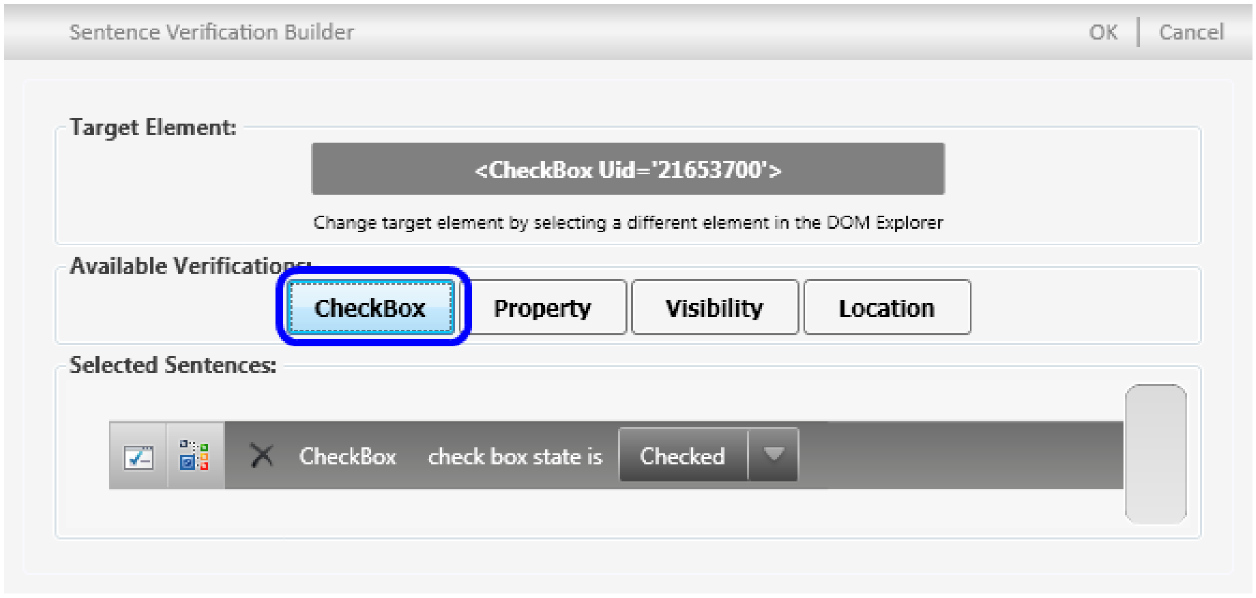


Sentence Verification Builder
OK
Cancel
Target Element:
<PasswordBox Uid='20084682'>
Change target element by selecting a different element in the DOM Explorer
Available Verifications:
PasswordChar
Password
Property
Visibility
Location
Selected Sentences:
PasswordChar password masking character is ' •
Password password is Password

Note how the current masking character and/or password are automatically filled in the expected value field. You can change the expected value if needed.

15.3.18Verifying the checked state of checkboxes and radio buttons

Using Sentence Verification Builder you can create a verification rule to test the checked state of a Silverlight radio button or checkbox. Start by clicking on "**Checked**" in the available verifications section.

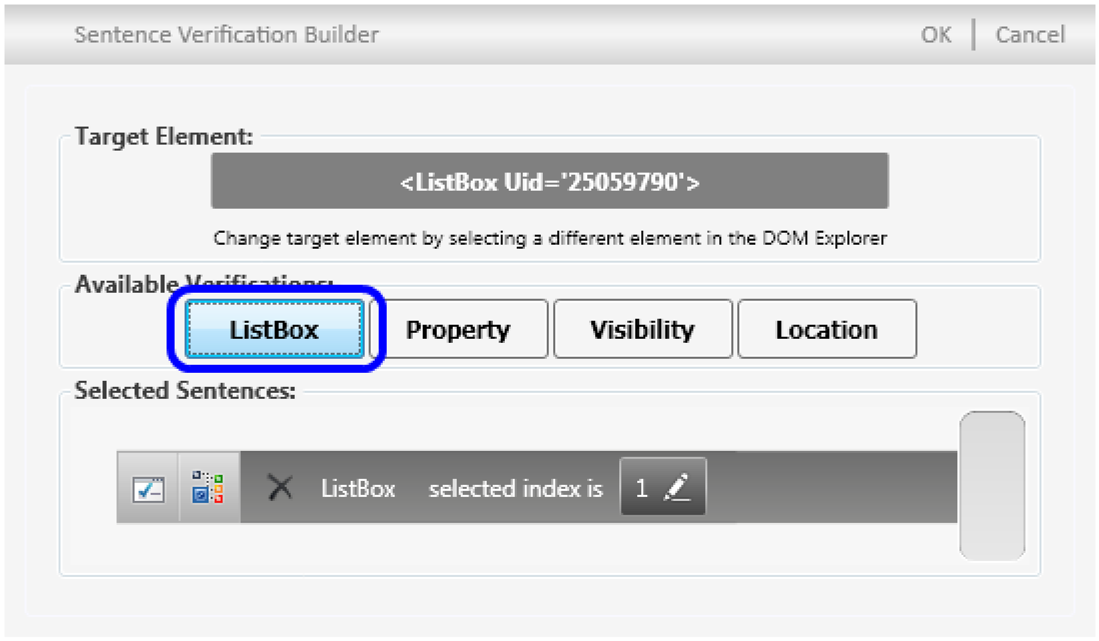


Sentence Verification Builder
OK
Cancel
Target Element:
<CheckBox Uid='21653700'>
Change target element by selecting a different element in the DOM Explorer
Available Verifications:
CheckBox
Property
Visibility
Location
Selected Sentences:
CheckBox check box state is Checked

Note how the current checked state is automatically filled in as the expected value. You can change this if needed.

15.3.19 Verifying the current selection in Listboxes

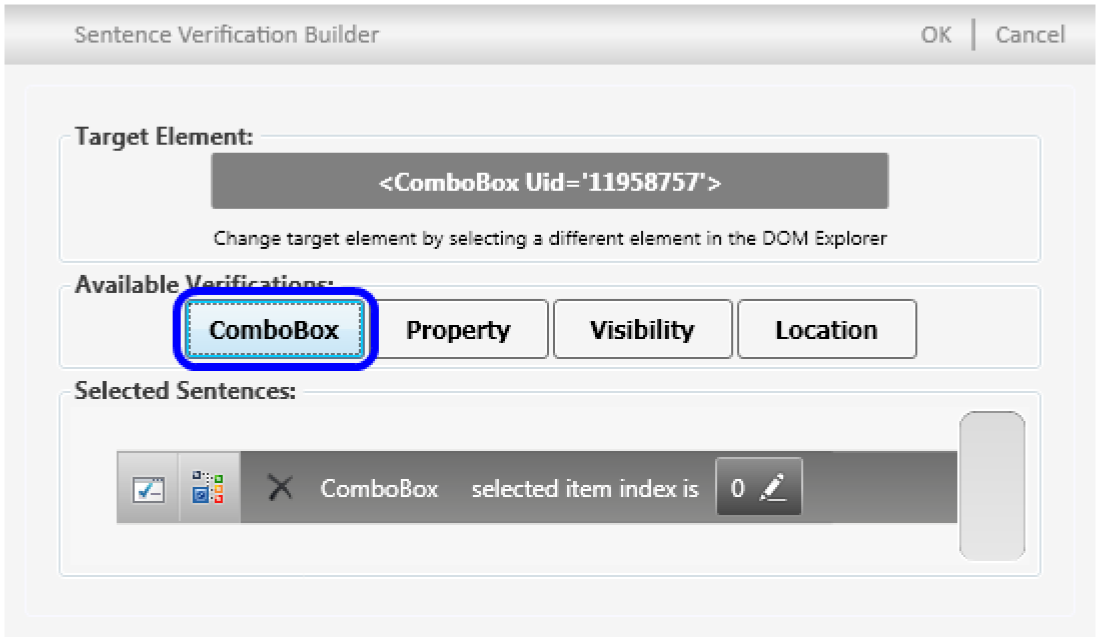
Using Sentence Verification Builder you can create an element verification rule to test the selection index of a Silverlight ListBox. Start by clicking on “**ListBox**” in the available verifications section.


Sentence Verification Builder
OK
Cancel
Target Element:
<ListBox Uid='25059790'>
Change target element by selecting a different element in the DOM Explorer
Available Verifications:
ListBox
Property
Visibility
Location
Selected Sentences:
ListBox selected index is 1

The current value of the selected index is automatically filled in the expected value. You can change it to a different integer value if you wish.

15.3.20 Verifying the current selection in ComboBoxes

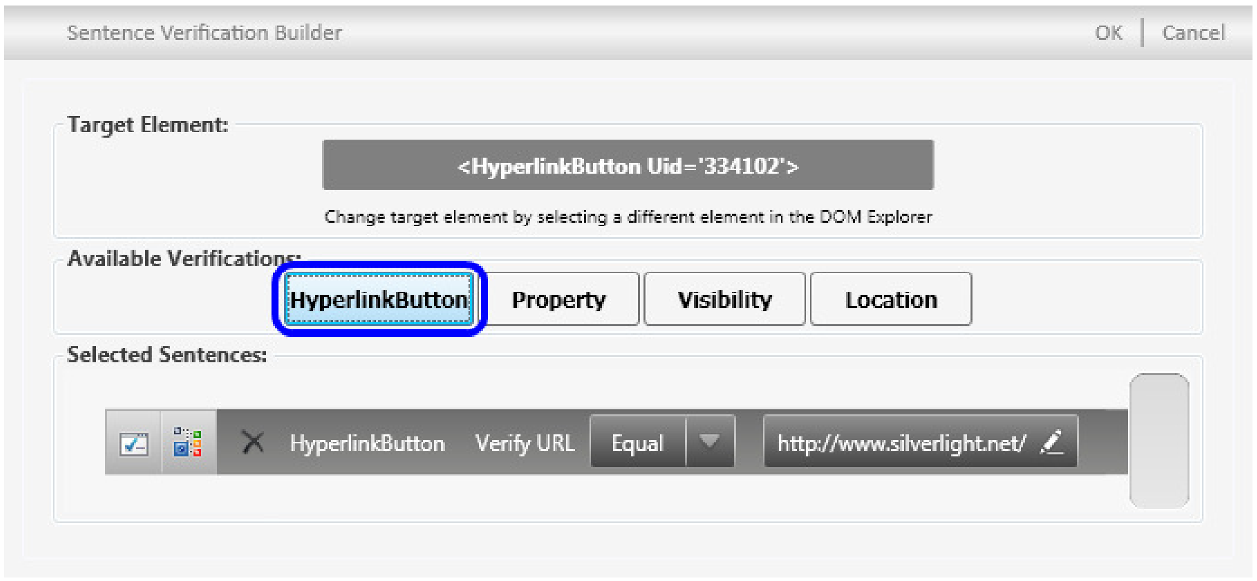
Using Sentence Verification Builder you can create an element verification rule to test the selection index of a Silverlight ComboBox. Start by clicking on "**ComboBox**" in the available verifications section.


Sentence Verification Builder
OK
Cancel
Target Element:
<ComboBox Uid='11958757'>
Change target element by selecting a different element in the DOM Explorer
Available Verifications:
ComboBox
Property
Visibility
Location
Selected Sentences:
ComboBox selected item index is 0

The current value of the selected index is automatically filled in the expected value. You can change it to a different integer value if you wish.

15.3.21 Verifying the URL of a HyperlinkButton

Using Sentence Verification Builder you can create an element verification rule to test the selection index of a Silverlight HyperlinkButton. Start by clicking on “**HyperlinkButton**” in the available verifications section.


Sentence Verification Builder
OK
Cancel
Target Element:
<HyperlinkButton Uid='334102'>
Change target element by selecting a different element in the DOM Explorer
Available Verifications:
HyperlinkButton
Property
Visibility
Location
Selected Sentences:
HyperlinkButton
Verify URL
Equal
http://www.silverlight.net/

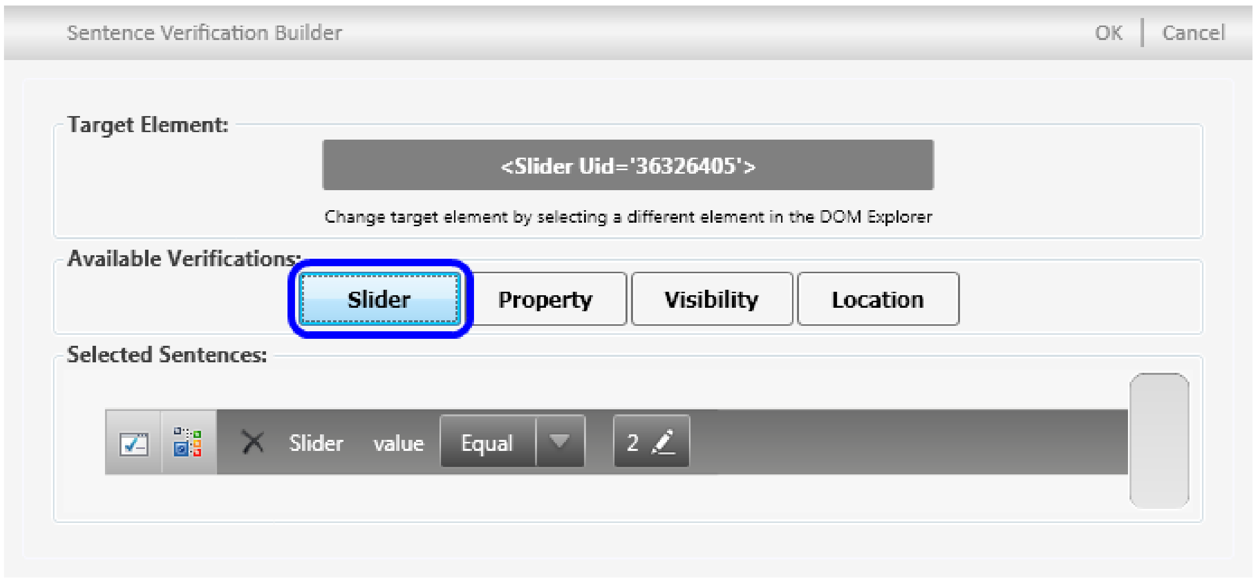
The comparison type can be one of:

- Equal – Verify the URL matches exactly with the expected value.
- Not Equal – Verify the URL is different than the specified value.
- Same – Verify the URL matches the expected value, ignoring case.
- Not Same – Verify the URL is different than the specified value, ignoring case.
- Contains – Verify the URL contains the expected value.
- NotContains – Verify the URL does *not* contain the expected value.
- StartsWith – Verify the URL starts with the expected value.
- NotStartsWith – Verify the URL does *not* starts with the expected value.
- EndsWith – Verify the URL ends with the expected value.
- NotEndsWith – Verify the URL does *not* end with the expected value.

The current value of the URL for the HyperlinkButton is automatically filled in the expected value. You can change it to a different URL if needed.

15.3.22 Verifying the slider value of a Slider control

Using Sentence Verification Builder you can create an element verification rule to test the current value of a Silverlight Slider control. Start by clicking on “**Slider**” in the available verifications section.



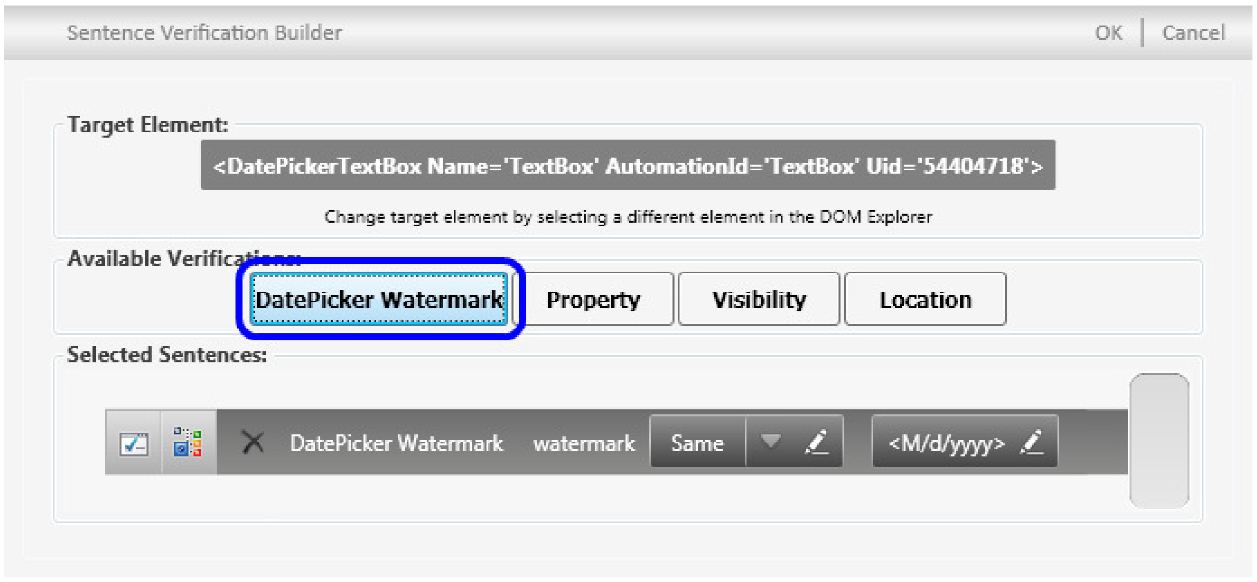
The comparison type can be one of:

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

The current value of the Slider is automatically filled in the expected value. You can change it to a different integer value if needed.

15.3.23 Verifying the watermark in Datepickers

Using Sentence Verification Builder you can create an element verification rule to test the current watermark of a Silverlight DatePicker control. Start by clicking on "**DatePicker**" in the available verifications section.



<DatePickerTextBox Name='TextBox' AutomationId='TextBox' Uid='54404718'>

DatePicker Watermark watermark Same <M/d/yyyy>

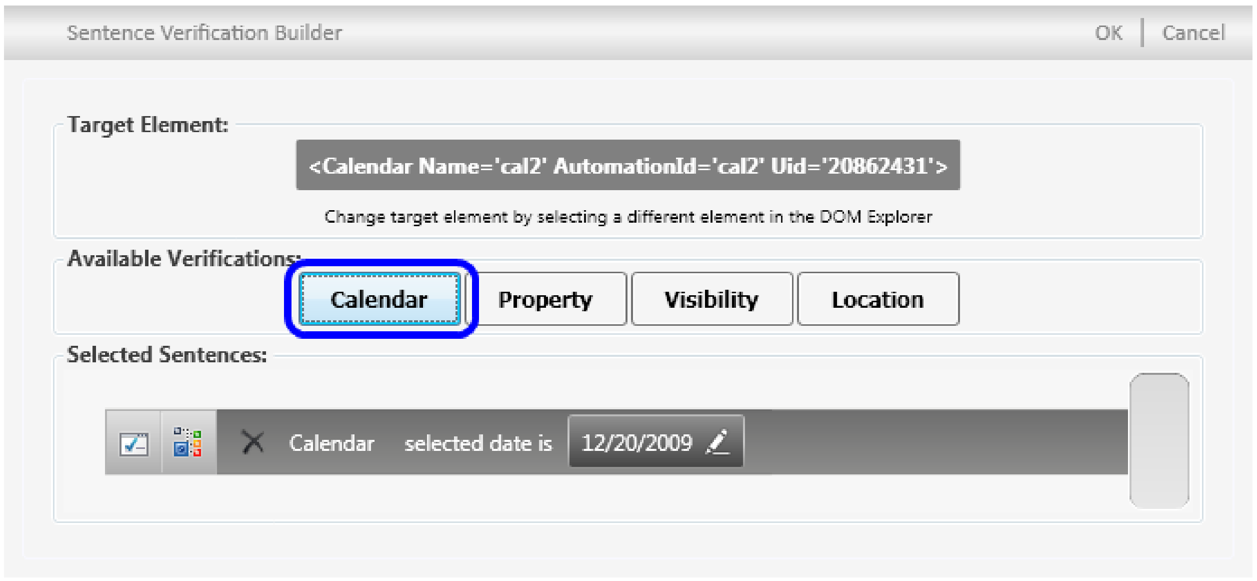
The comparison type can be one of:

- Equal – Verify the watermark matches exactly with the expected value.
- Not Equal – Verify the watermark is different than the specified value.
- Same – Verify the watermark matches the expected value, ignoring case.
- Not Same – Verify the watermark is different than the specified value, ignoring case.
- Contains – Verify the watermark contains the expected value.
- NotContains – Verify the watermark does *not* contain the expected value.
- StartsWith – Verify the watermark starts with the expected value.
- NotStartsWith – Verify the watermark does *not* starts with the expected value.
- EndsWith – Verify the watermark ends with the expected value.
- NotEndsWith – Verify the watermark does *not* end with the expected value.

The current value of the watermark is automatically filled in the expected value. You can change it to a different string if needed.

15.3.24 Verifying the date selection in Calenders

Using Sentence Verification Builder you can create an element verification rule to test which date is selected in a Silverlight Calendar control. Start by clicking on “Calendar” in the available verifications section.

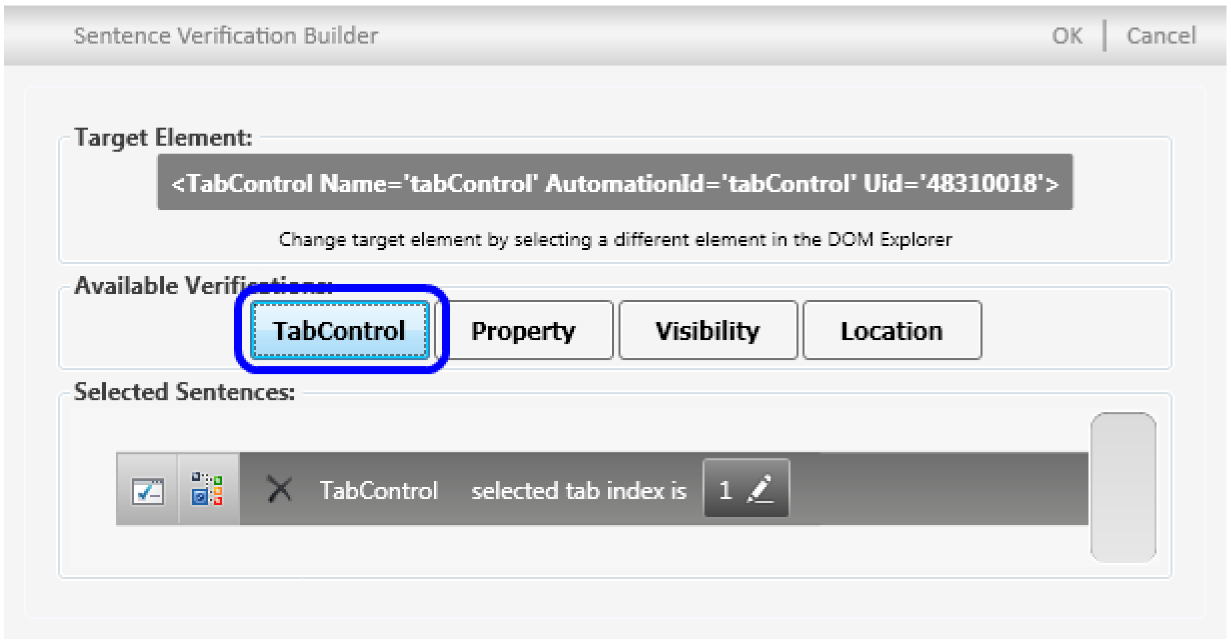

Sentence Verification Builder
OK
Cancel
Target Element:
<Calendar Name='cal2' AutomationId='cal2' Uid='20862431'>
Change target element by selecting a different element in the DOM Explorer
Available Verifications:
Calendar
Property
Visibility
Location
Selected Sentences:
Calendar selected date is 12/20/2009

The currently selected date is automatically filled in the expected value. You can change it to a different date if needed.

NOTE: Multiple date selection is not currently supported. Only the first date selected in a multi-select calendar control will be used.

15.3.25 Verifying the tab selection of a tabbed control

Using Sentence Verification Builder you can create an element verification rule to test which tab is currently selected for a Silverlight TabControl. Start by clicking on "**TabControl**" in the available verifications section.



<TabControl Name='tabControl' AutomationId='tabControl' Uid='48310018'>

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

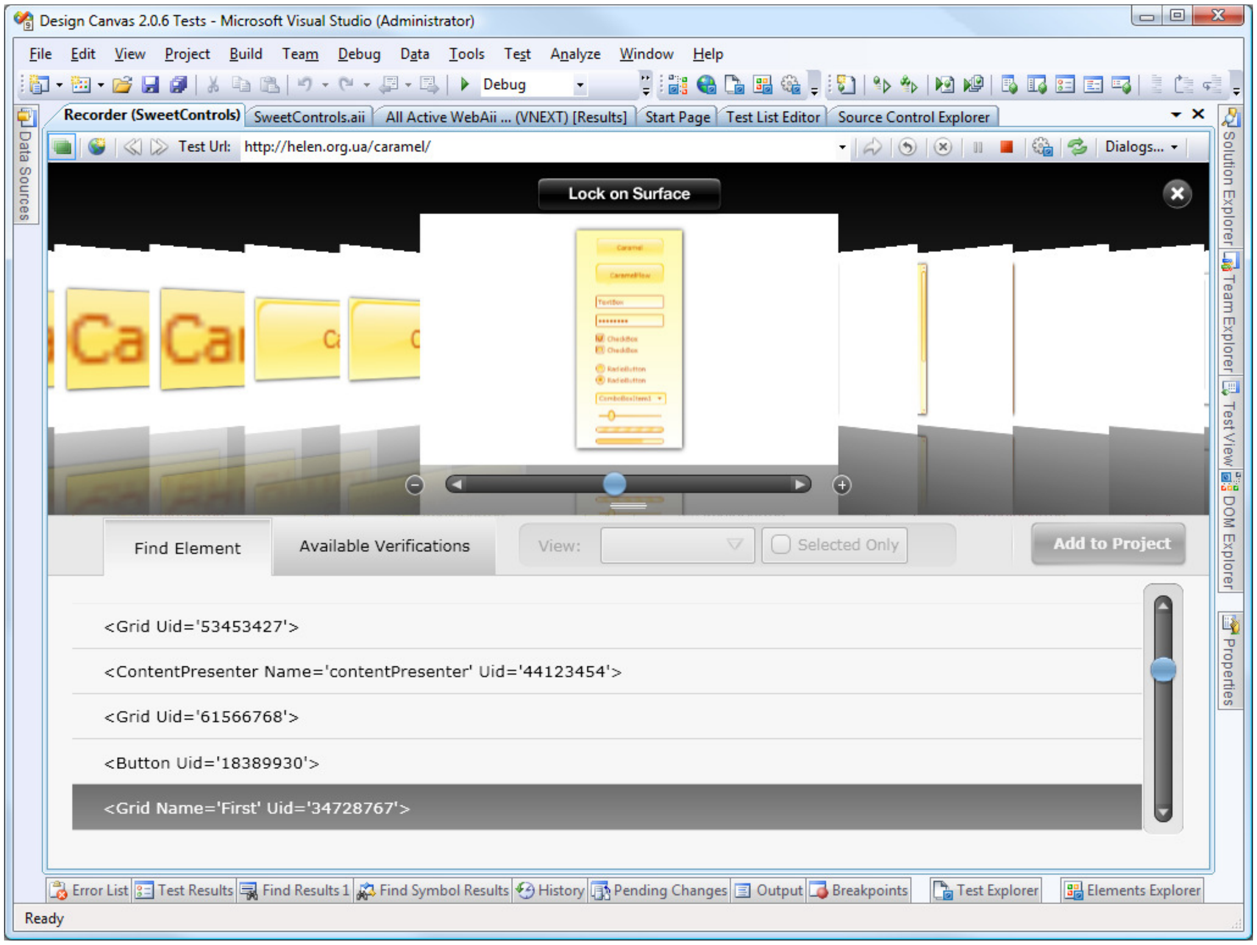
TabControl selected tab index is 1

The currently selected tab index is automatically filled in the expected value. You can change it to a different integer value if needed.

16 3D Viewer

Design Canvas 2.0 sports a new and elegant 3D viewer model. In the 3D viewer you can:

- Study the hierarchy of an element.
- Lock the Automation Overlay Surface on any element contained in the hierarchy.
- Craft verifications and add them to your test.


Design Canvas 2.0.6 Tests - Microsoft Visual Studio (Administrator)
File Edit View Project Build Team Debug Data Tools Test Analyze Window Help
Debug
Recorder (SweetControls) SweetControls.aii All Active WebAii ... (VNEXT) [Results] Start Page Test List Editor Source Control Explorer
Test Url: http://helen.org.ua/caramel/
Dialogs...
Lock on Surface
Find Element
Available Verifications
View:
Selected Only
Add to Project
Data Sources
Solution Explorer
Team Explorer
Test View
DOM Explorer
Properties
Error List Test Results Find Results 1 Find Symbol Results History Pending Changes Output Breakpoints Test Explorer Elements Explorer
Ready

```
<Grid Uid='53453427'>
<ContentPresenter Name='contentPresenter' Uid='44123454'>
<Grid Uid='61566768'>
<Button Uid='18389930'>
<Grid Name='First' Uid='34728767'>
```

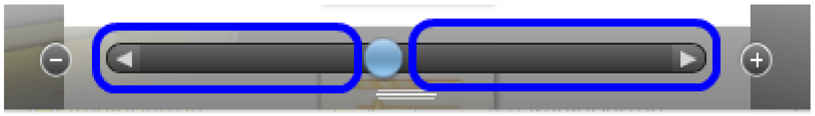
The top half of the 3D viewer shows you a graphical representation of all the elements from the element having focus in the recording surface all the way up to the root element (the <HTML> element for an HTML type of element, or the Page element for a Silverlight type of element). In other words, the left most image is the element that currently has focus. The next image is an image of that elements parent. The next image is an image of its grandparent and so on all the way up to the root element.

The bottom half of the 3D viewer shows you a vertical listing of the same set of elements. The topmost element is the element having focus in the recording surface. The next element down the list is that elements parent. The next element is its grandparent and so on all the way to the root element.

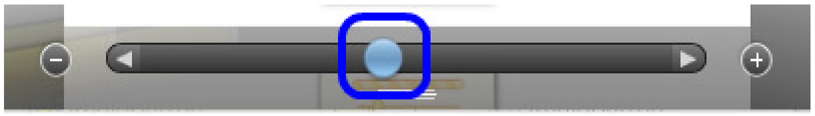
16.1 Scrolling through the Image Set

There are many ways you can scroll through the image set:

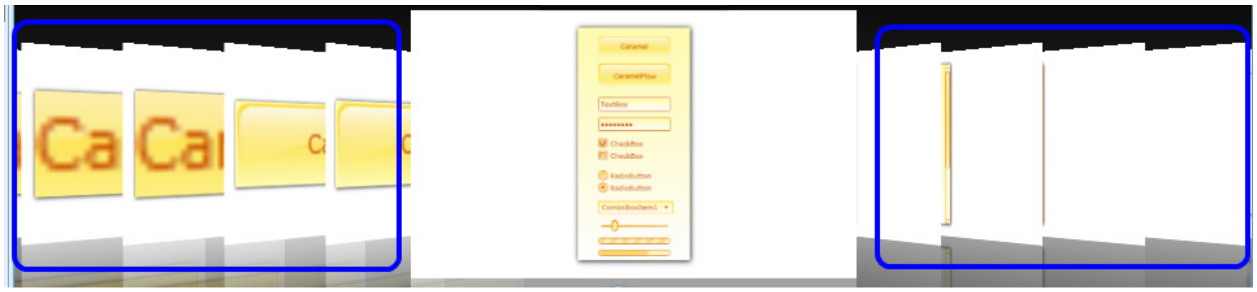
- Clicking on the left/right arrows will scroll the images, one image at a time.



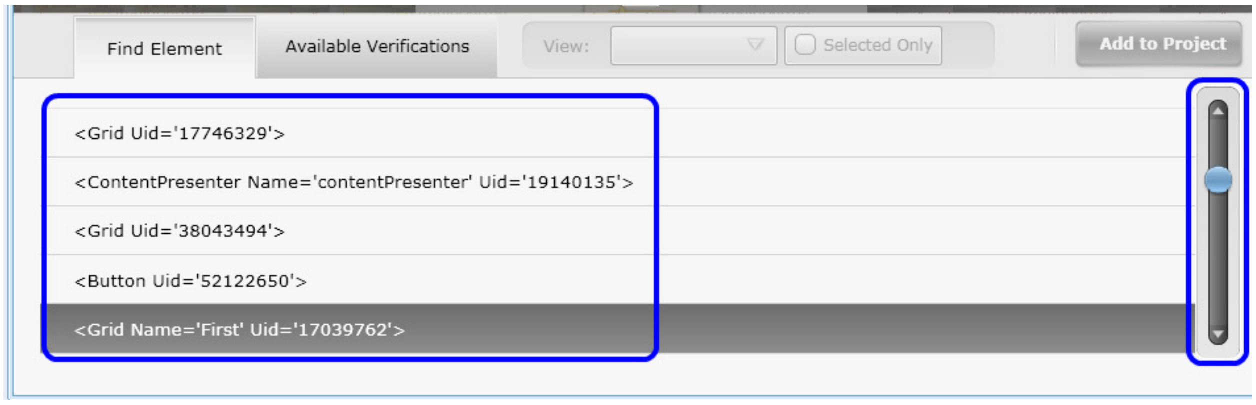
- Dragging the blue nub will scroll the images left/right.



- Clicking on a different image will bring that image to the center.

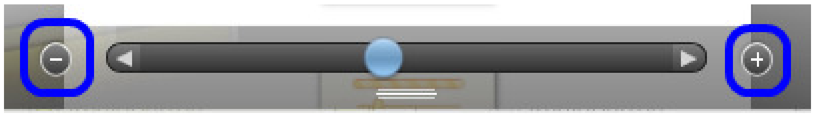


- Clicking on a different element will scroll the images to the image of that element.

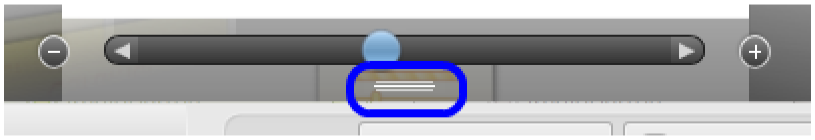


<Grid Uid='17746329'>
<ContentPresenter Name='contentPresenter' Uid='19140135'>
<Grid Uid='38043494'>
<Button Uid='52122650'>
<Grid Name='First' Uid='17039762'>

You can zoom in/out of the images by clicking + and – buttons:

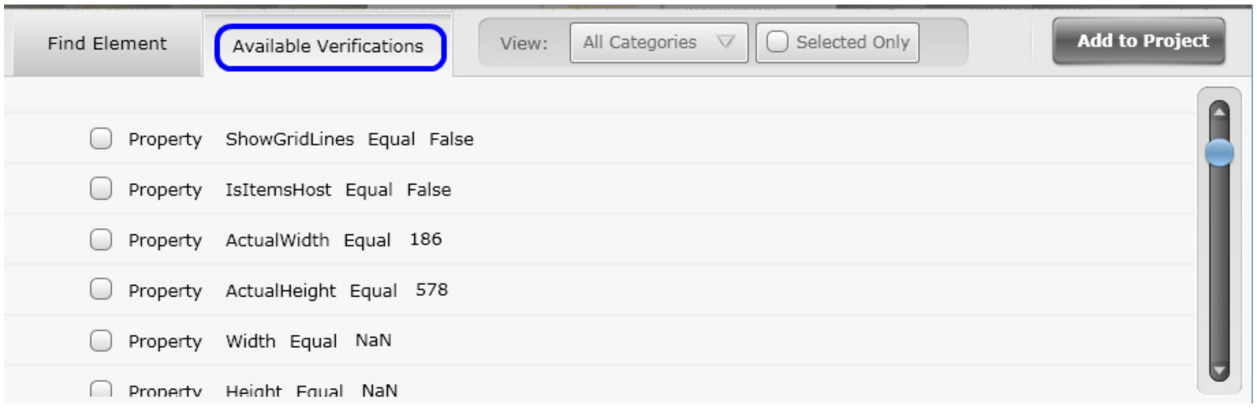


You can resize the top and bottom halves by dragging the horizontal bar separating the two.



16.2 Adding Verifications

Once you have selected the element you're interested in, click on the Available Verifications tab and Design Canvas will compile a list of all the possible verifications that can be performed on that element:



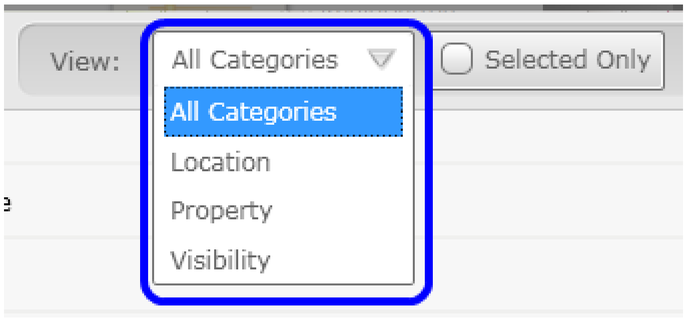
```
Property   ShowGridLines  Equal  False
Property   IsItemsHost  Equal  False
Property   ActualWidth  Equal  186
Property   ActualHeight  Equal  578
Property   Width  Equal  NaN
Property   Height  Equal  NaN
```

Here you select (and modify if needed) the verifications you would like added to your test.

16.2.1 Filtering the Verifications List

You can filter the list of verifications that are displayed by using the View filters.

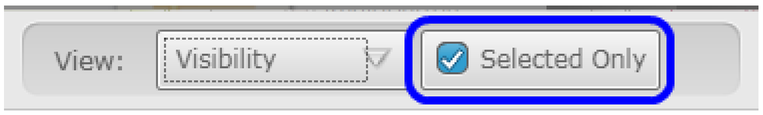
16.2.1.1 Categories Filter



If you open the categories drop down you can select from:

- All Categories – All of the possible verifications will be displayed in the list.
- Location – Only verifications relating to the elements location (left edge, right edge, top edge, bottom edge) will be displayed in the list.
- Property – Only verifications relating to the elements properties (width, height, name, opacity, etc.) will be displayed in the list.
- Visibility – Only verifications relating to the elements visibility (Visible, Collapsed, etc.) will be displayed in the list.

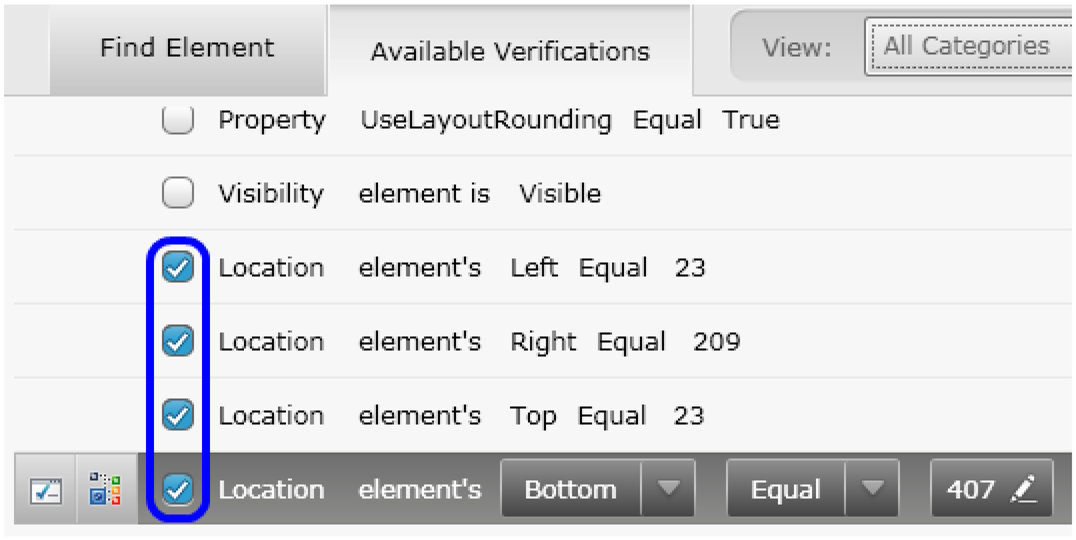
16.2.1.2 Selected Only Filter



By checking the Selected Only filter only those verifications that you have selected will be displayed in the list.

16.2.2 Selecting Verifications

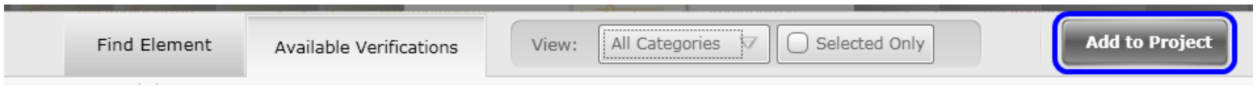
You select a verification you would like added by simply checking its checkbox.



Property UseLayoutRounding Equal True
Visibility element is Visible
Location element's Left Equal 23
Location element's Right Equal 209
Location element's Top Equal 23
Location element's Bottom Equal 407

16.2.3 Add to Project

You add the selected verifications by clicking the "**Add to Project**" button. All of the verifications that you selected (by checking their checkbox) are added as steps to the test. Each verification will be added as a separate test step in the test. Thus if you selected four verifications, four verification steps will be added to your test.

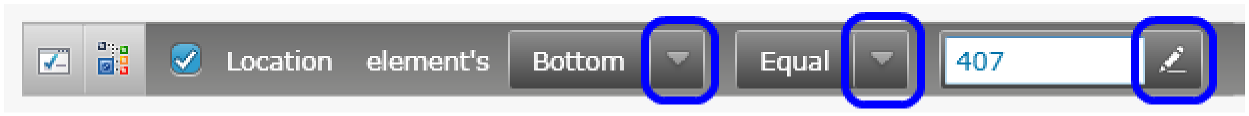


16.3 Modifying Verifications

Optionally you can modify any of the listed verifications. You can change:

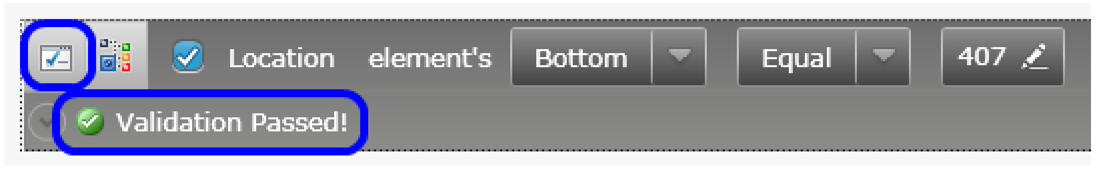
- Which attribute/property the verification will look at.
- The comparison type that will be performed.
- The value to use in the comparison.

First click on a verification you want to modify. Then click on the drop downs to change the selection. Or click on the edit button to change the value to use in the comparison.

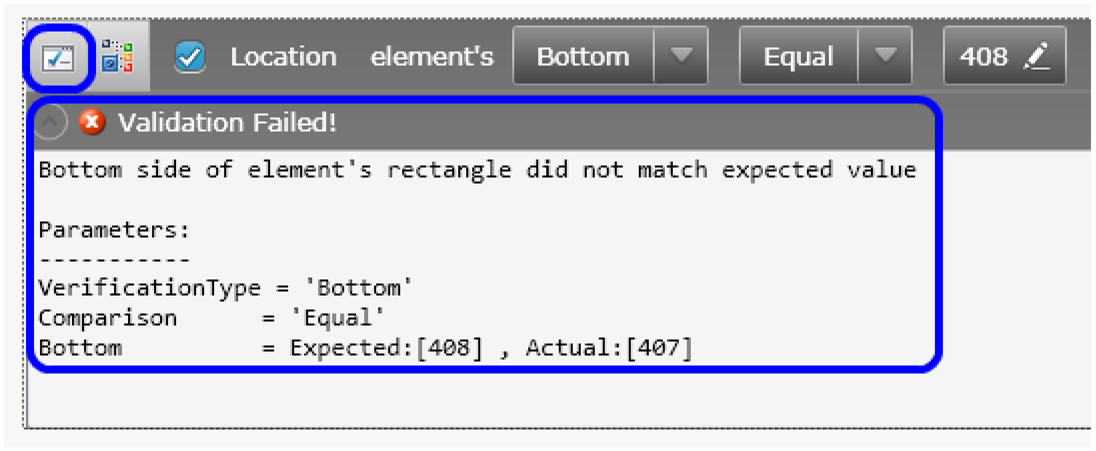


16.3.1 Validating Modified Verifications

After modifying a verification it's generally a good idea to make sure it works right. You can do this by clicking on the Validate button. If it passes validation it will look like this:



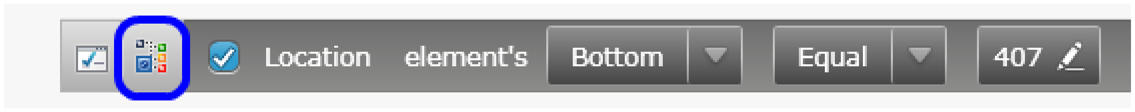
If it fails validation it will look something like this:



```
Bottom side of element's rectangle did not match expected value

Parameters:
-----------
VerificationType = 'Bottom'
Comparison       = 'Equal'
Bottom           = Expected:[408] , Actual:[407]
```

16.4 Locating Elements in DOM

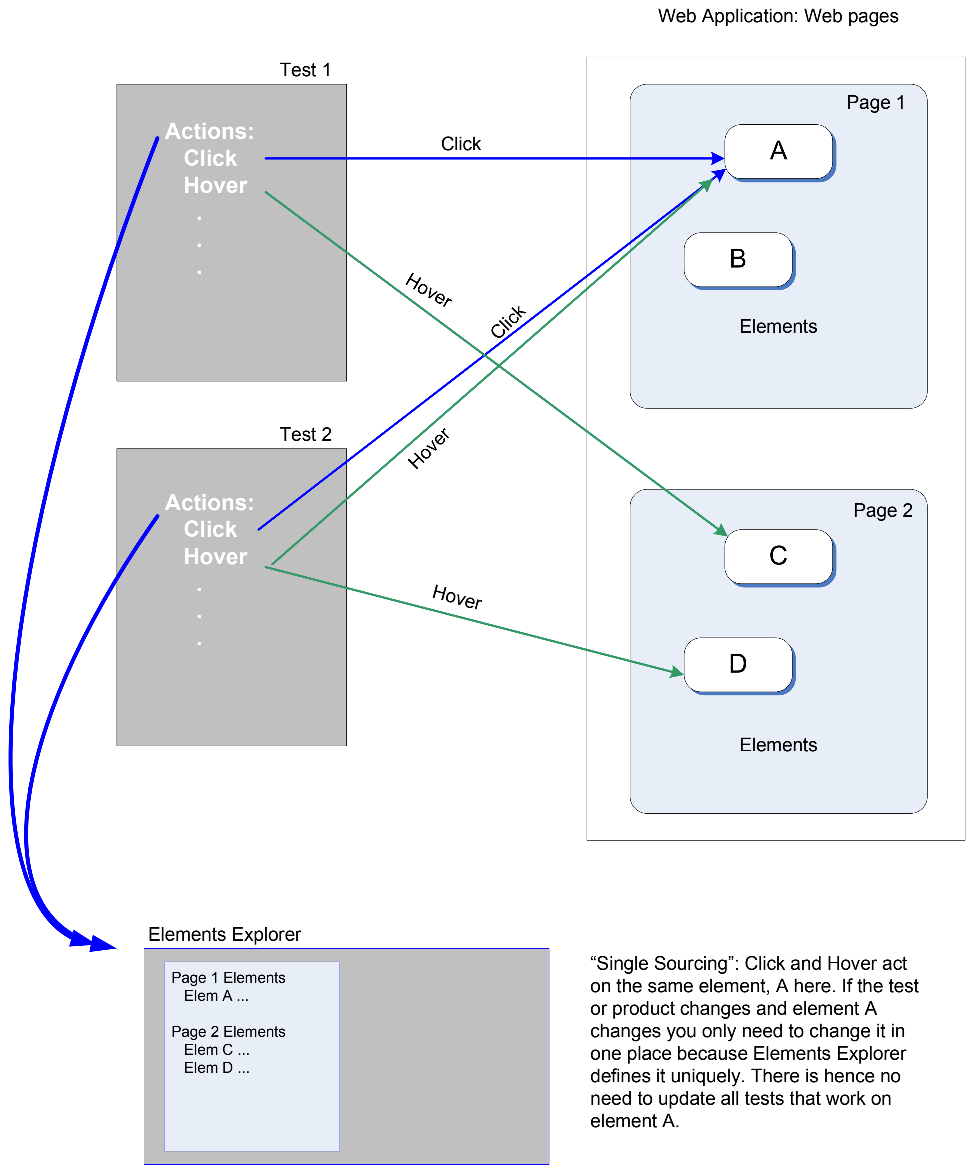
One other thing you can do on the Available Verifications tab is locate the current element in the DOM tree. Just click the "**Locate in DOM**" button and the current element will be highlighted in the DOM Explorer window.



17 Web Element Abstraction

17.1 Introducing Web Element Abstraction

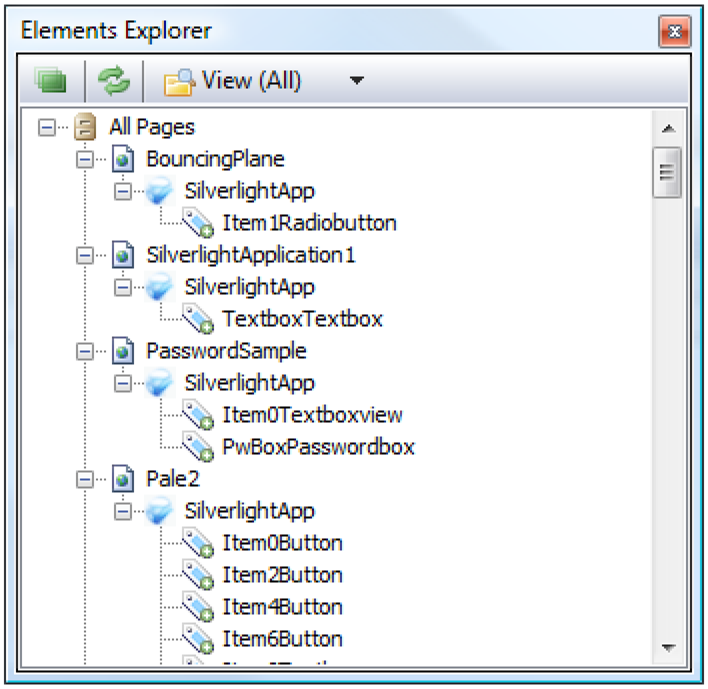
The following diagram illustrates part of the web element abstraction concept. Because elements are uniquely defined in Elements Explorer regardless of how the web page or web page tests change or how often and from where each web page element is acted on in the test, you only need to modify that element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



“Single Sourcing”: Click and Hover act on the same element, A here. If the test or product changes and element A changes you only need to change it in one place because Elements Explorer defines it uniquely. There is hence no need to update all tests that work on element A.

Automation Design Canvas: Web Element Abstraction

17.2 Elements Explorer Window



All web page elements targeted for automation in your tests are abstracted out and filed under a specific page/frame/SilverlightApp 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 expressions 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. TODO: VERIFY THIS IS STILL TRUE AFTER THE CONFIG SETTINGS REWRITE IS COMPLETE.

17.2.1 Elements Explorer: Highlighting elements on the active page

Clicking on this icon allows you to visually highlight elements in the Recording Surface. Once enabled, as you select elements in Elements Explorer by clicking on them, that element will be highlighted 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.

17.2.2 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.

17.2.3 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:

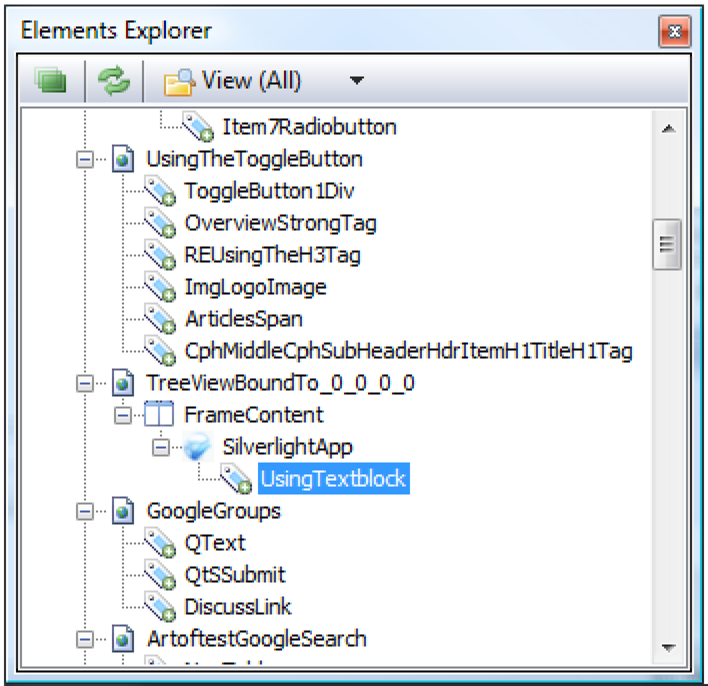
17.2.3.1 View (All)

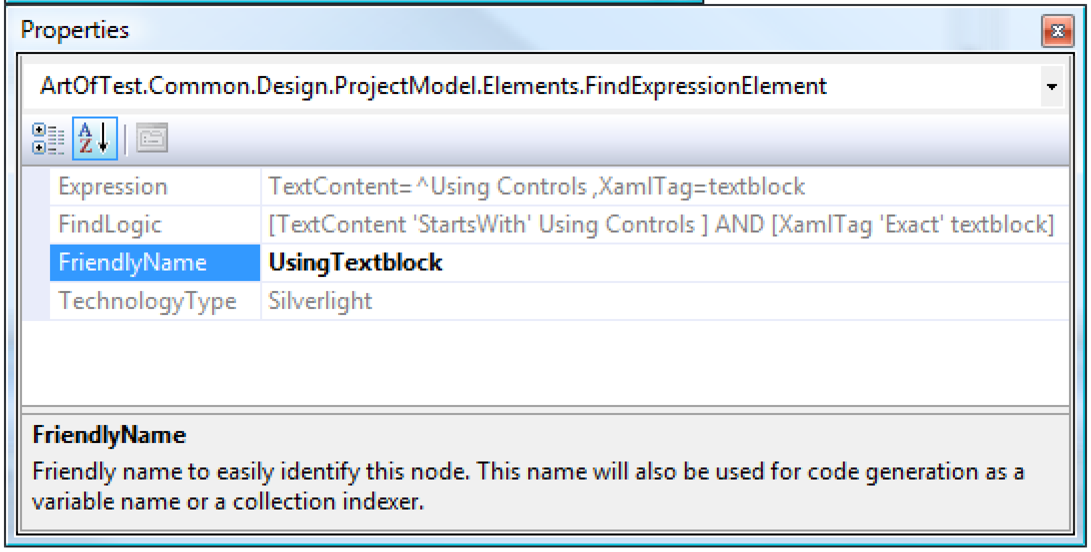
The ‘View (All)’ mode displays all elements that have been added to Elements Explorer from all web pages and frames in a hierarchical format.

17.2.3.2 View current page only

The ‘View (Current)’ page only mode displays 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.

17.2.4 Elements Explorer: Element properties




Properties

ArtOfTest.Common.Design.ProjectModel.Elements.FindExpressionElement

Expression	TextContent=^Using Controls ,XamlTag=textblock
FindLogic	[TextContent 'StartsWith' Using Controls] AND [XamlTag 'Exact' textblock]
FriendlyName	**UsingTextblock**
TechnologyType	Silverlight

FriendlyName
Friendly name to easily identify this node. This name will also be used for code generation as a variable name or a collection indexer.

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.*

17.2.4.1 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

FindExpression 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.

17.2.4.2 Element properties: Expression

This read-only property shows the find expression that will be used to locate the element on the web page. You must use the Find Expression Builder to modify it.

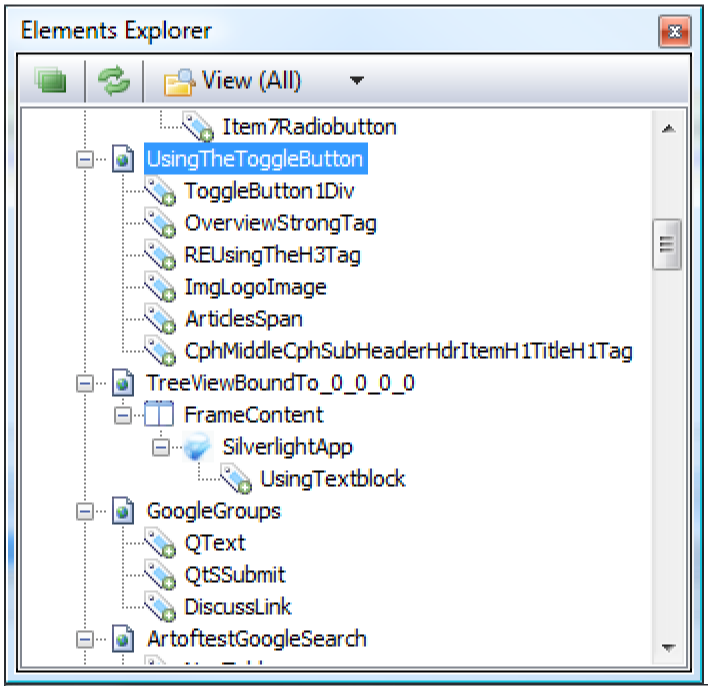
17.2.4.3 Element properties: Find Logic

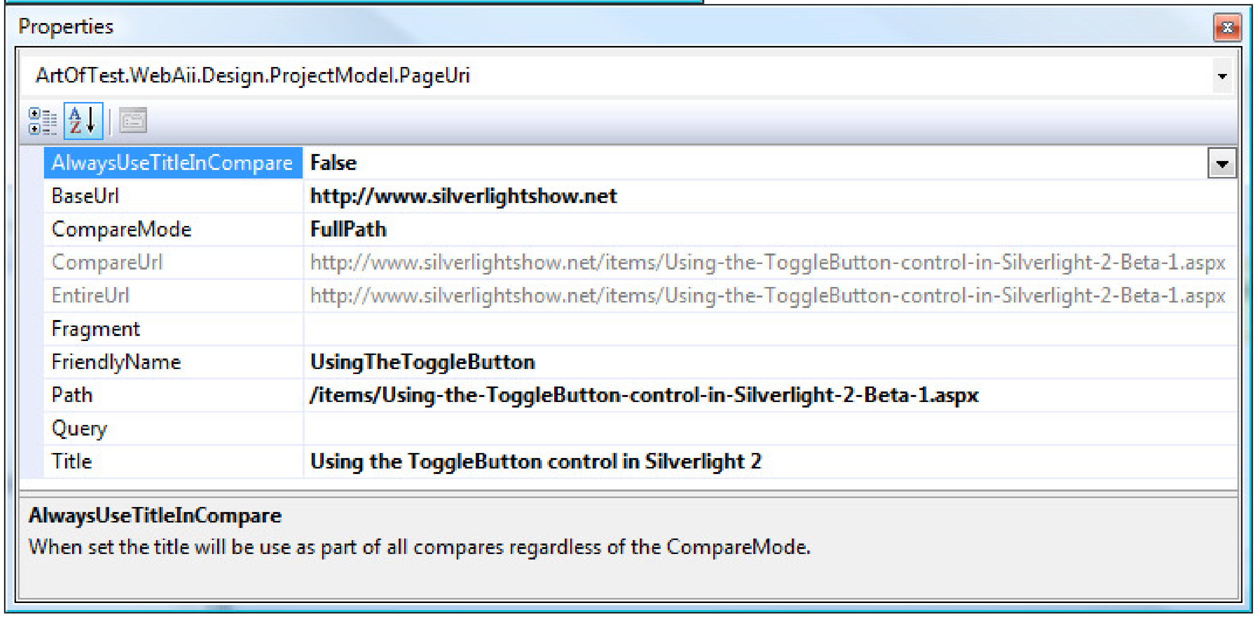
This read-only property shows how the Find Expression will be interpreted and used by the WebAii framework.

17.2.4.4 Element properties: Find Logic

This read-only property is set to either Silverlight or HTML based on the element type when it was captured to Elements Explorer.

17.2.5 Elements Explorer: Page properties





ArtOfTest.WebAii.Design.ProjectModel.PageUri

AlwaysUseTitleInCompare	False
BaseUrl	http://www.silverlightshow.net
CompareMode	FullPath
CompareUrl	http://www.silverlightshow.net/items/Using-the-ToggleButton-control-in-Silverlight-2-Beta-1.aspx
EntireUrl	http://www.silverlightshow.net/items/Using-the-ToggleButton-control-in-Silverlight-2-Beta-1.aspx
Fragment	
FriendlyName	UsingTheToggleButton
Path	/items/Using-the-ToggleButton-control-in-Silverlight-2-Beta-1.aspx
Query	
Title	Using the ToggleButton control in Silverlight 2

AlwaysUseTitleInCompare
When set the title will be use as part of all compares regardless of the CompareMode.

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

17.2.5.1 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.

17.2.5.2 Page Properties: Page properties

Page properties define the characteristics of the URL of a 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
AlwaysUseTitleInCompare	When set to "**true**" the title will always be used as part of compares regardless of the CompareMode setting.
BaseUrl	The BaseUrl property is the base part of the URL to use in comparisons. It contains the protocol and host name part of the URL.e.g. http://artoftest.com.
CompareMode	The mode used to identify this URL as explained in section 17.2.5.2.1.
CompareUrl	The portion of the URL that will be used in comparisons according to the CompareMode setting. This property is a read-only calculated property based on the other property settings.
EntireUrl	You cannot directly change the EntireUrl property. Automation Design Canvas displays the full URL which is simply the four properties BaseUrl, Path, Query, and Fragment concatenated together to form a complete URL. This property is not actually used anywhere. It is displayed only for reference and user validation when entering the first three properties.
Fragment	The fragment portion of the URL, from the # character to the end of the URL.
FriendlyName	The name displayed in Elements Explorer for this web page.
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.
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 found 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.

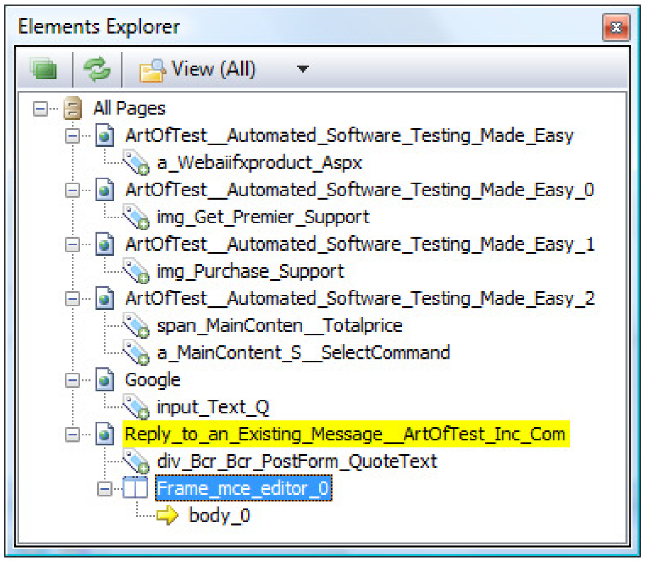
17.2.5.2.1 Page Properties – CompareUrl: CompareMode

The CompareMode property controls how WebAii determines of this page is the active page. The following are the possible settings and how they work.

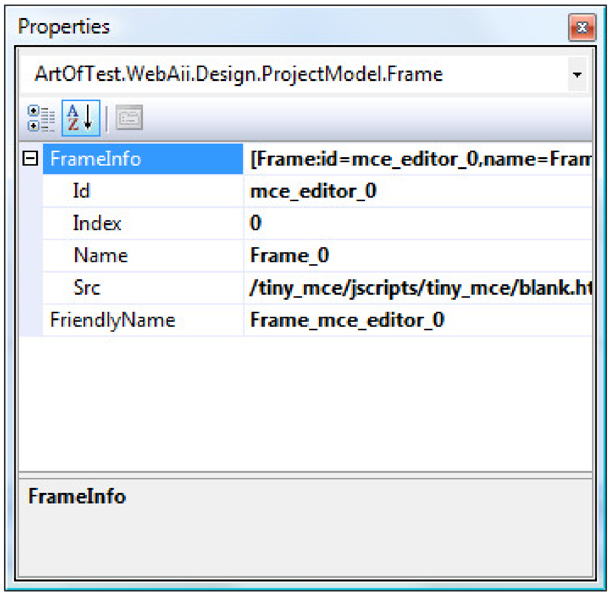
- BaseUrl – WebAii determines if this page is active by comparing the current URL in the web browser to the just the BaseUrl property.

- 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 and fragment portions 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.
- FullPathAndQueryNoFragment – WebAii determines if this page is active by comparing the current URL in the web browser to the entire URL of this web page minus the fragment portion.
- 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 and fragment portions 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 and fragment properties of this web page. The BaseUrl portion of the URL in the web browser will be ignored.
- RelativePathAndQueryNoFragment – 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 and fragment portions of the URL in the web browser will be ignored
- Title – WebAii determines if this page is active by looking at the web page title as specified in the header of the HTML.

17.2.6 Elements Explorer: Frame properties



```
All Pages
  ArtOfTest__Automated_Software_Testing_Made_Easy
    a_Webaiifxproduct_Aspx
  ArtOfTest__Automated_Software_Testing_Made_Easy_0
    img_Get_Premier_Support
  ArtOfTest__Automated_Software_Testing_Made_Easy_1
    img_Purchase_Support
  ArtOfTest__Automated_Software_Testing_Made_Easy_2
    span_MainConten__Totalprice
    a_MainContent_S__SelectCommand
  Google
    input_Text_Q
  Reply_to_an_Existing_Message__ArtOfTest_Inc_Com
    div_Bcr_Bcr_PostForm_QuoteText
    Frame_mce_editor_0
      body_0
```



FrameInfo	[Frame:id=mce_editor_0,name=Fram
Id	mce_editor_0
Index	0
Name	Frame_0
Src	/tiny_mce/jscripts/tiny_mce/blank.ht
FriendlyName	Frame_mce_editor_0

FrameInfo

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:

17.2.6.1 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.

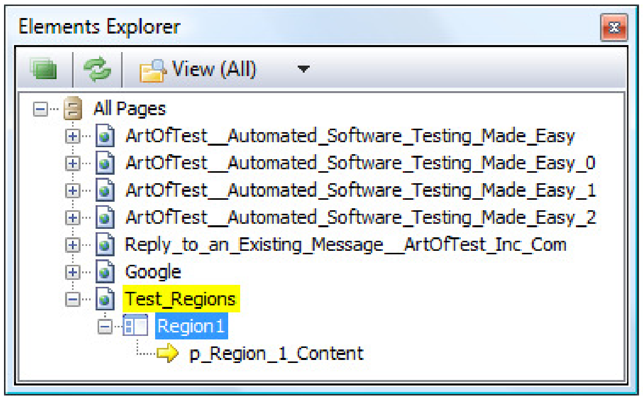
17.2.6.2 Frame Properties: FriendlyName

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.

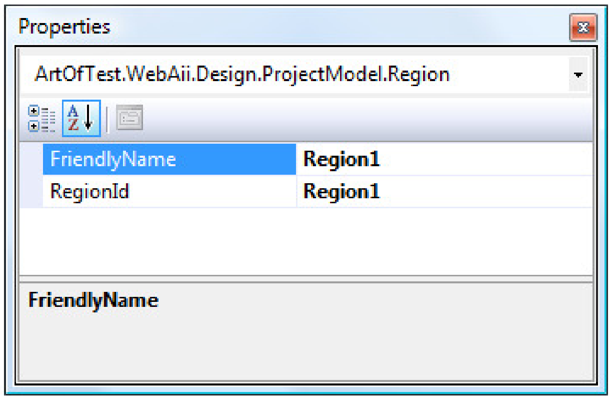
17.2.7 Elements Explorer: Silverlight App properties

When a web page contains one or more Silverlight applications then an additional "**SilverlightApp**" node is added to Elements Explorer as elements are added. This node has the same properties as any other standard element. You can edit its properties using the Find Expression builder.

17.2.8 Elements Explorer: Test Regions properties



```
All Pages
  ArtOfTest__Automated_Software_Testing_Made_Easy
  ArtOfTest__Automated_Software_Testing_Made_Easy_0
  ArtOfTest__Automated_Software_Testing_Made_Easy_1
  ArtOfTest__Automated_Software_Testing_Made_Easy_2
  Reply_to_an_Existing_Message__ArtOfTest_Inc_Com
  Google
  Test_Regions
    Region1
      p_Region_1_Content
```



ArtOfTest.WebAii.Design.ProjectModel.Region

FriendlyName	**Region1**
RegionId	**Region1**

FriendlyName

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

17.2.8.1 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.

17.2.8.2 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.

17.2.9 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.

17.2.10 Elements Explorer: Context menu

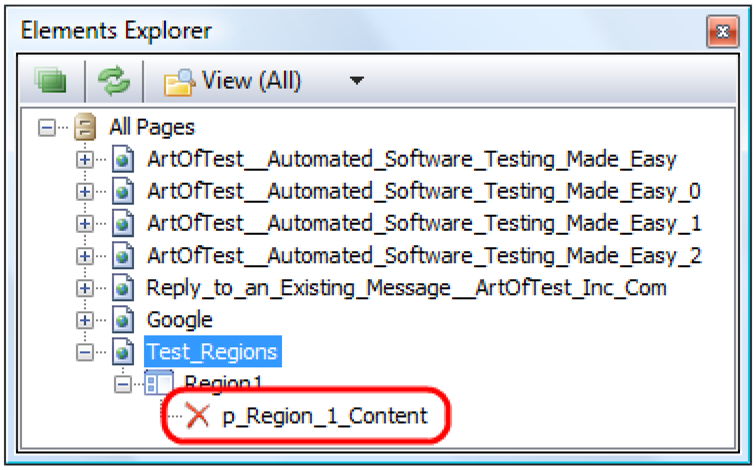
When you right click on any node in Elements Explorer a context menu is displayed. The following sections describe each menu item contained in this context menu.

17.2.10.1 Context Menu: Edit

When you right click on an element the Edit menu option will be enabled. By selecting this option the Find Expression Builder will open. This UI is described in detail in section **Error! Reference source not found.**.

17.2.10.2 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 Find Expression settings. Any elements that cannot be found on the web page will have an error icon displayed next to them as shown here:



```
All Pages
  ArtOfTest__Automated_Software_Testing_Made_Easy
  ArtOfTest__Automated_Software_Testing_Made_Easy_0
  ArtOfTest__Automated_Software_Testing_Made_Easy_1
  ArtOfTest__Automated_Software_Testing_Made_Easy_2
  Reply_to_an_Existing_Message__ArtOfTest_Inc_Com
  Google
  Test_Regions
    Region1
      p_Region_1_Content
```

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:
TODO: GET SCREENSHOT AFTER BUG 1898 IS FIXED

17.2.10.3 Context Menu: Delete

When you right click on an element that is not referenced by any tests, 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 an 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 will not be automatically deleted.

It is not absolutely necessary to remove unused elements from Elements Explorer. It may be helpful 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.

17.2.10.4 Context Menu: Locate in DOM Explorer

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 10.

17.2.10.5 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 working on a test you have started, need to perform maintenance on a test, or need to start a new test in the same project.

17.2.10.6 Context Menu: Properties

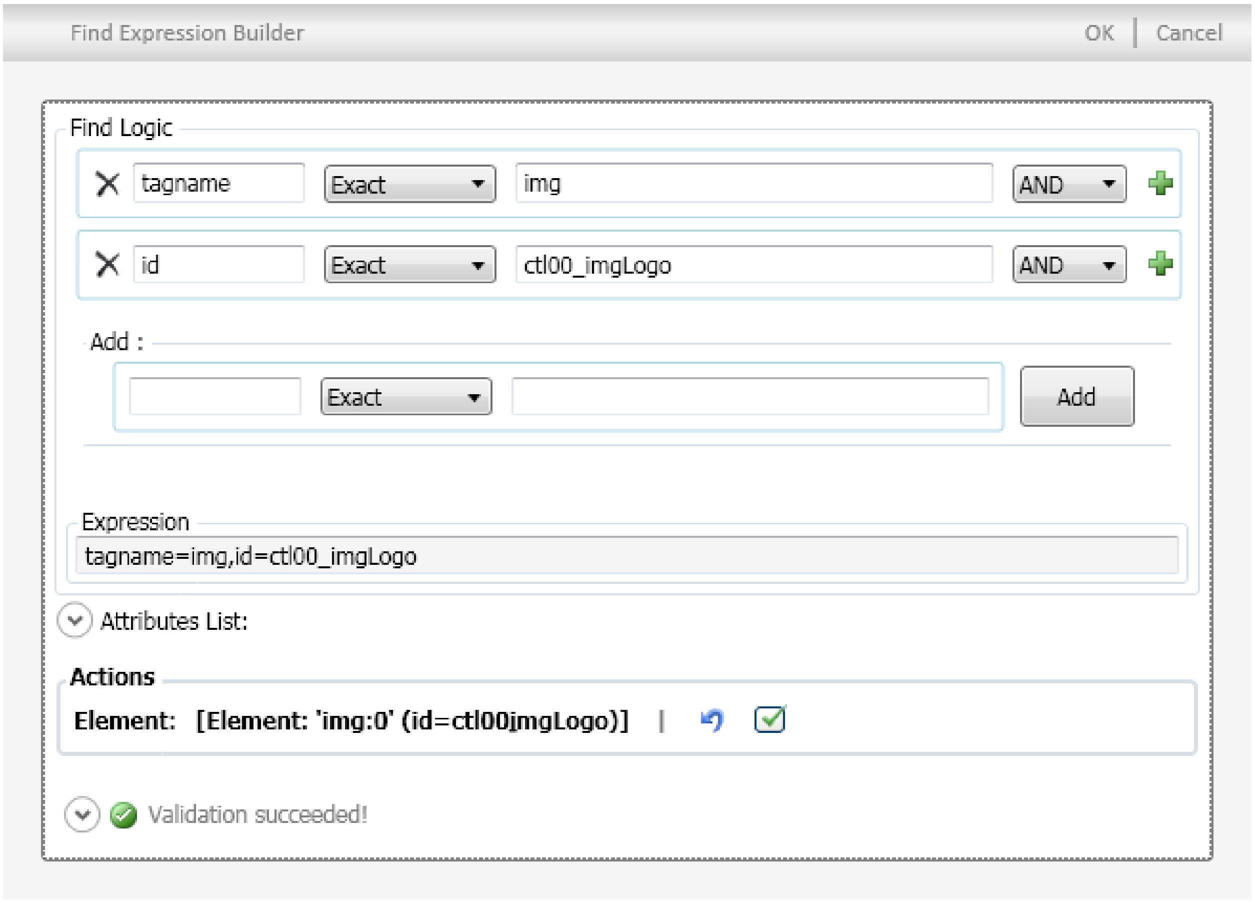
The Properties menu item is always enabled; however it only does something when you click on a web page node, a region node, a frame node, or an element node. When you select Properties, the properties for that node are displayed in the properties window allowing you to view and edit them as needed.

18 Find Expression Builder

Whenever a web page element is added to Elements Explorer, a "**Find Expression**" is generated that tells the framework how to find that element in the DOM tree. Find Expression's have many basic approaches it can use to find elements on a web page. It can also use a combination of the basic approaches. Since there can be more than one approach for locating an element, Design Canvas tries to intelligently determine which Find Expression combination works best on that particular web page for finding the element. This intelligence is configurable as described in B.4.

The Find Expression Builder is what you use to edit the Find Expression of an element. You can also test the Find Expression against the web page loaded in the Recording Surface to verify that it works.

You activate the Find Expression Builder by right clicking on one of the elements in Elements Explorer and selecting “**Edit Element**”.



tagname	Exact	img	AND
id	Exact	ctl00_imgLogo	AND

Expression
tagname=img,id=ctl00_imgLogo

Element: [Element: 'img:0' (id=ctl00imgLogo)]

Validation succeeded!

18.1 Find Logic

The “**Find Logic**” displays the logic currently being used to locate the element in the DOM tree. Each row is a find expression clause that will be used during an element search. All of the clauses must evaluate to True before Design Canvas decided it has found the right element in the DOM tree.

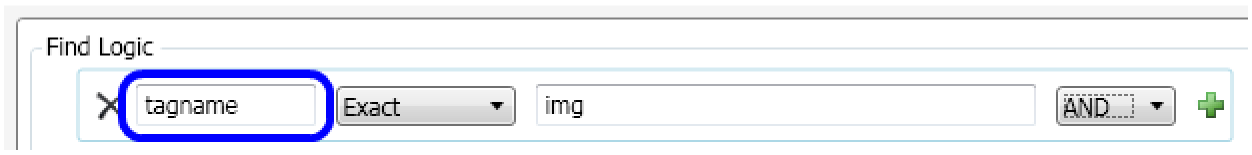
In the example above it will find an element in the DOM tree that has:

1. The tag name equals “img” and
2. Its id attribute equals “ctl00_imgLogo”

In other words it will find an element that looks something like this in the HTML code:

```
<img id=”ctl00_imgLogo”>
```

18.1.1 Element Property to Compare



The first text box identifies what element property/attribute to compare. For HTML elements the options are:

- Any valid attribute name (id, name, etc.) For example an <a> element having a particular href attribute or an <input> having a certain id attribute.
- InnerText – search for an element in which the InnerText matches the specified expression. 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 a text string. If you have multiple elements with the same text on your web page you may wind up getting the wrong element returned.
- InnerMarkup – search for an element in which the InnerMarkup matches the specified expression. For example, "<div id="div2">". The same care must be taken for this find method as for InnerText.
- OuterMarkup – search for an element in which the OuterMarkup matches the specified expression. For example, "<div id="div4">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 – search for an element in which the TextContent matches the specified expression. 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 – search for an element in which the StartTagContent matches the specified expression.
- NodeIndexPath – search for an element having the specified NodeIndexPath.
- TagName – search for an element in which the TagName matches the specified expression.
- TagIndex – search for an element at the specified zero based TagIndex. For example the 14th <a> element or the first <div> element. It is important to note that WebAii uses zero-based indexing. Thus the first <div> element has an index 0.

<u>Note</u>: There is a difference between InnerText and TextContent that is worth noting: Example:

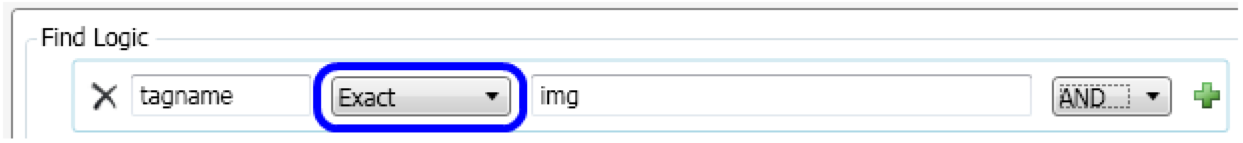
```
<div id="div1">
     Text1<div id="div2">
          Text2</div>
```

The InnerText for div1 is “Text1Text2” i.e. it is recursive. The TextContent of div1 is: “Text1” i.e. non-recursive.

For Silverlight elements the options are:

- AutomationId – search for an element having an automation ID of a specific value.
- TextContent – search for an element containing or not containing specific text in the element.
- XamlTag – search for an element of a specific type.
- Name – search for an element having a name of a specific value.
- TagIndex – the zero based index value of a specific tag.

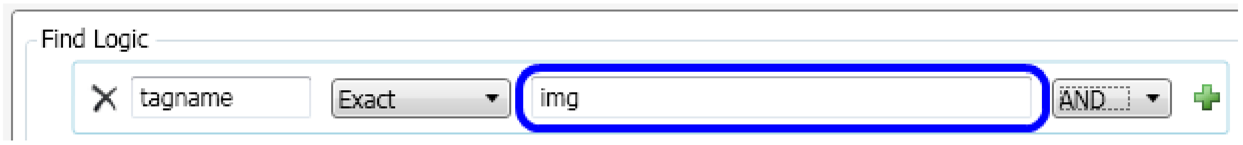
18.1.2 Comparison type



The comparison type can be one of:

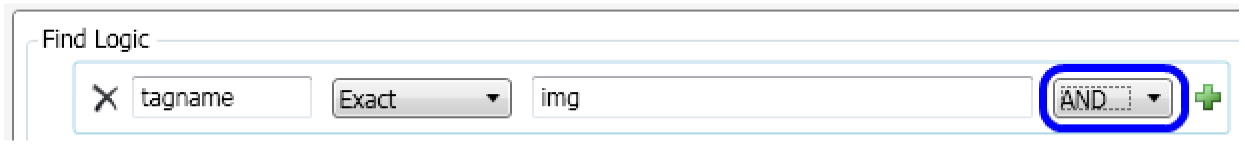
- Exact – Find the element with the property/attribute that matches exactly to the specified value.
- Contains – Find the element with the property/attribute that contains the specified value. For example, a Name property that contains “mainLogo”.
- NotContains - Find the element with the property/attribute that does *not* contains the specified value. For example, a Name property that does *not* contain “mainLogo”.
- StartsWith – Find the element with the property/attribute that starts with the specified value. For example, a Name that starts with “inputSubmit”.
- EndsWith – Find the element with the property that starts with the specified value. For example, a Name that ends with “inputSubmit”.
- Regex – Find the element with the property/attribute that 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 value to look for is used as the Regex expression in the match comparison
- Missing – Find the element that does *not* have the property/attribute. The value is ignored.
- Exists – Find the element that has the property/attribute regardless of what the value is.

18.1.3 Value



In the Value textbox enter the value you want to be used by the comparison operation.

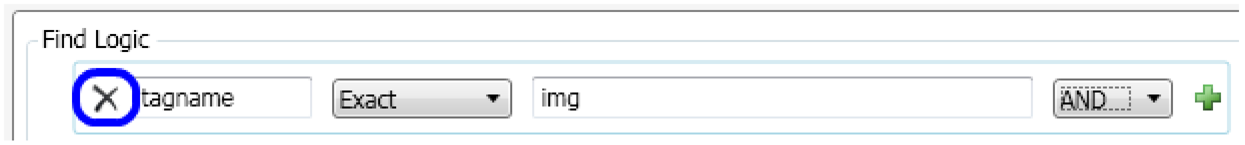
18.1.4 AND/THEN



This drop down can be set to either AND or THEN.

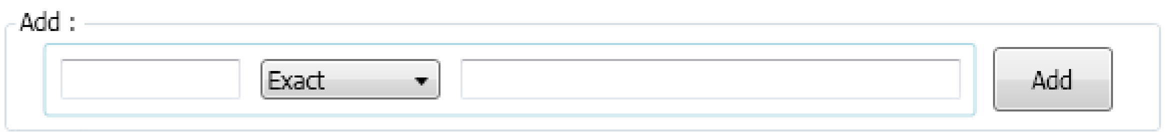
- AND – When set to AND the current find expression clause and the next find expression clause must both evaluate to True before Design Canvas decides it has found the right element.
- THEN – When set to THEN, it specifies a "chained" find expression. This means it will find an element matching the current and all previous find expression clauses in the DOM tree and then resume searching but only the children of that element. This feature is very useful when your webpage contains the same special control multiple times. Often these special controls use the same name and/or ID and can be difficult to distinguish one from another without following a specific hierarchal path.

18.1.5 Delete clause



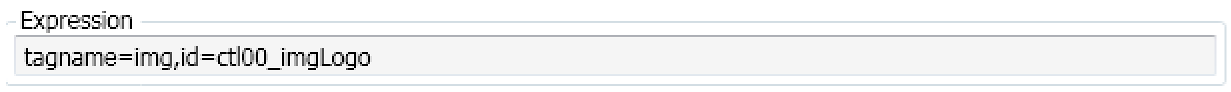
Clicking on the Delete button will delete that find expression clause. Be careful though because there is no single UNDO. If you accidentally delete a find expression clause you'll need to recreate it or Cancel (see section 18.4.2) all changes and start over.

18.2 Add



In the "**Add**" you can create a new find expression clause to be added to the list. The three options are the same as all find expression clauses as explained in sections 18.1.1 – 18.1.3. Once you have filled in the options click the "**Add**" button to add it to the list.

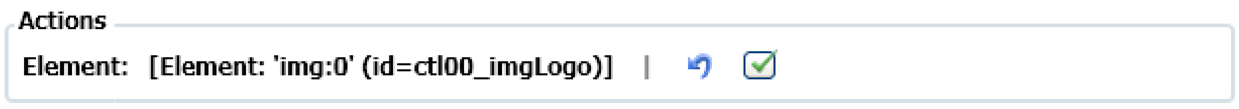
18.3 Expression



tagname=img,id=ctl00_imgLogo

The Expression section displays the Find Expression generated from all the find expression clauses you have specified. This is the Find Expression that the WebAii framework will actually use when performing an element search in the DOM tree. Find Expressions are documented at http://www.artoftest.com/support/webaii/topicsindex.aspx?topic=byexpression.

18.4 Actions

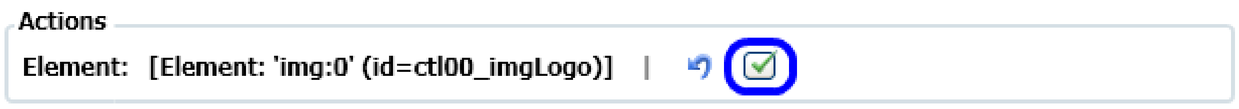


Element: [Element: 'img:0' (id=ctl00_imgLogo)]

The Actions section actually serves two purposes:

- It displays the HTML for the element found using the current set of find clauses.
- It contains the Validate and Discard Changes buttons.

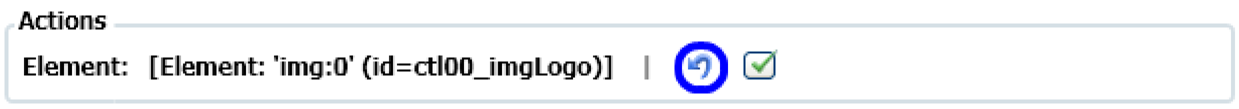
18.4.1 Validate Button



Element: [Element: 'img:0' (id=ctl00_imgLogo)]

After making changes to the find clauses you can click the Validate button to make Design Canvas perform an element search in the current DOM tree and return the element it found. This way you can verify the find clauses are finding the correct element.

18.4.2 Discard Changes Button

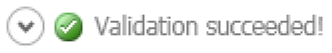


Element: [Element: 'img:0' (id=ctl00_imgLogo)]

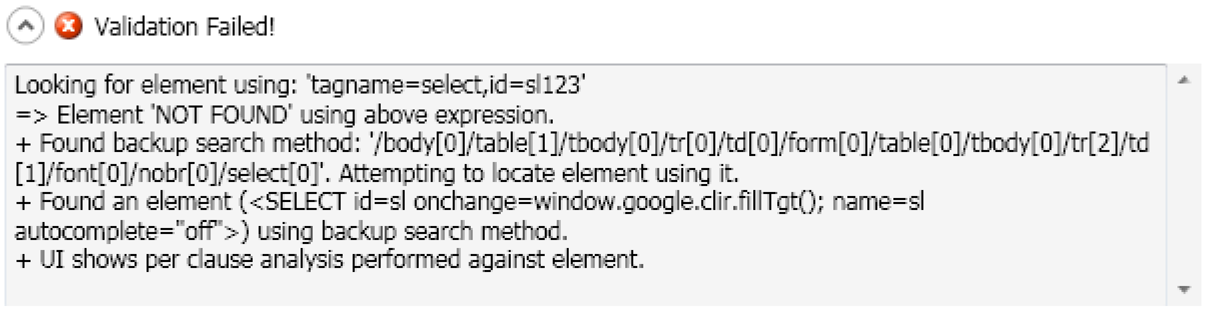
If you make a mistake (such as accidentally deleting a needed find clause) you can cancel revert back to the original find clauses by click the Discard button.

18.5 Validation succeeded/failed

After clicking the Validate button you will see either:



Or



```
Looking for element using: 'tagname=select,id=sl123'
=> Element 'NOT FOUND' using above expression.
+ Found backup search method: '/body[0]/table[1]/tbody[0]/tr[0]/td[0]/form[0]/table[0]/tbody[0]/tr[2]/td
[1]/font[0]/nobr[0]/select[0]'. Attempting to locate element using it.
+ Found an element (<SELECT id=sl onchange=window.google.clir.fillTgt(); name=sl
autocomplete="off">) using backup search method.
+ UI shows per clause analysis performed against element.
```

The error message displayed when validation fails hopefully will be useful in determining how to modify the find clauses so it can find the correct element.

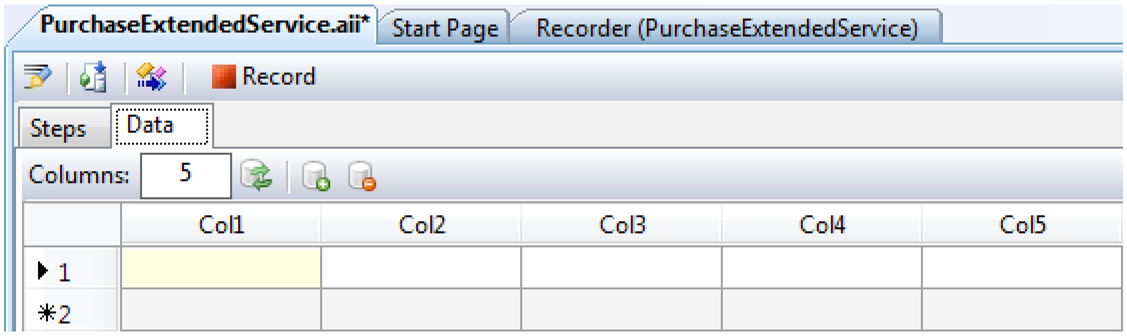
19 Data-Driven Testing

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 data 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.

19.1 Built-In Data Grid

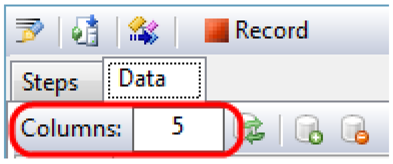
To add and manipulate the internal data grid (data array) for data-driven testing you must first open the Data tab for the .aii test as explained in section 8.



PurchaseExtendedService.aii*
Start Page
Recorder (PurchaseExtendedService)
Record
Steps
Data
Columns:
5

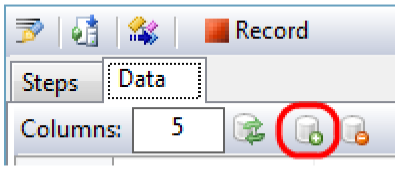
	Col1	Col2	Col3	Col4	Col5
▶ 1					
✱2					

19.1.1 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.

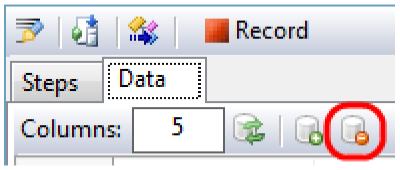
19.1.2 Create new data table



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

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

19.1.3 Remove data table

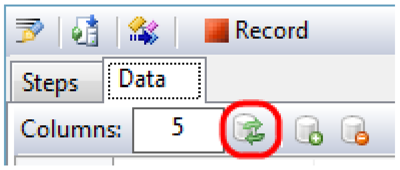


To remove the data grid from the test simply click the **Remove Data Table** icon. You will be prompted to confirm the delete.
***Note**: Once the data grid is deleted all the data it contained is immediately discarded. There is no way to undo the delete so be certain you want to discard the data grid!*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

19.1.4 Deleting columns

There are two methods for deleting columns you no longer need. Each method is detailed in the following two sections.

19.1.4.1 Update Columns



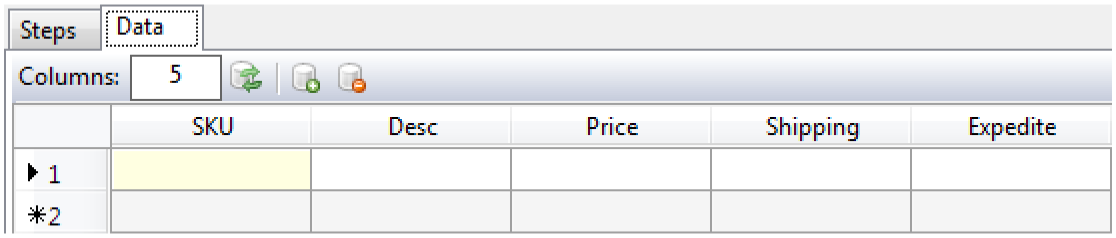
You can change the number of columns kept by your data grid. You can add or remove columns. You do not have a choice of which columns will be removed however. Enter the number of columns in the **Columns** textbox then click the **Update Columns** icon. If you are adding columns they will be immediately 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.

19.1.4.2 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.

19.1.5 Changing column names



	SKU	Desc	Price	Shipping	Expedite
▶ 1					
*2					

You have the ability to change the names of the columns. 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.

19.1.6 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 stored. 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.

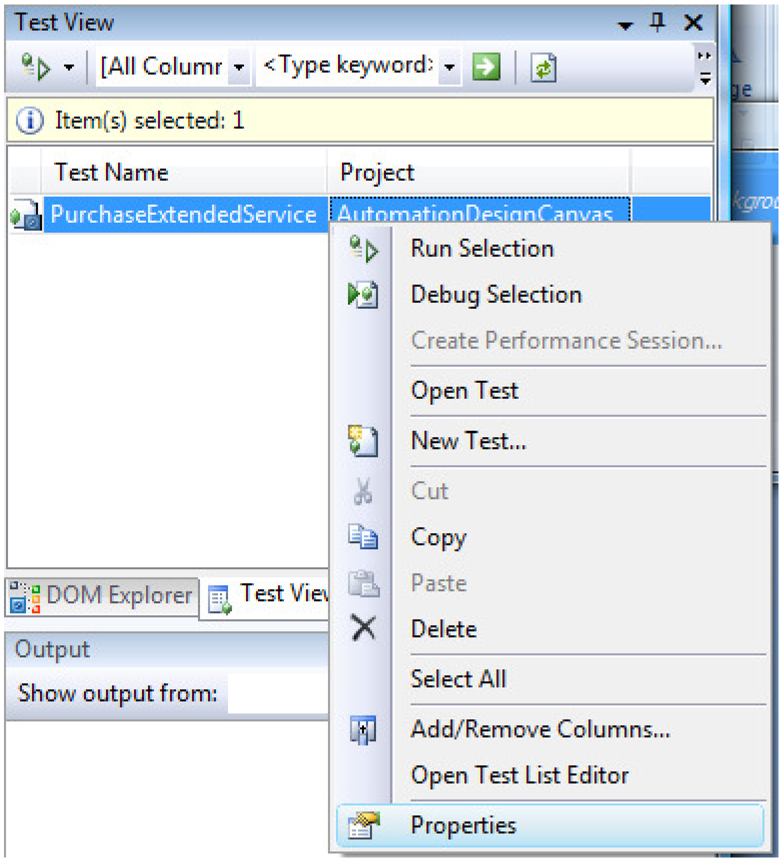
19.1.7 Deleting rows

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!*

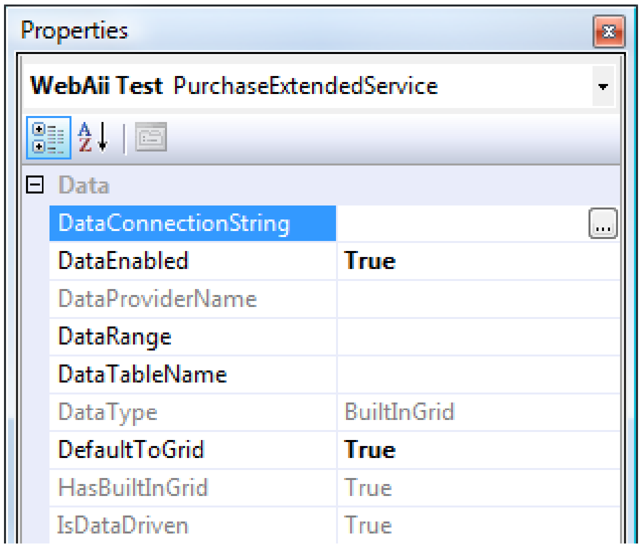
19.2 Binding test step properties to an external data source

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.

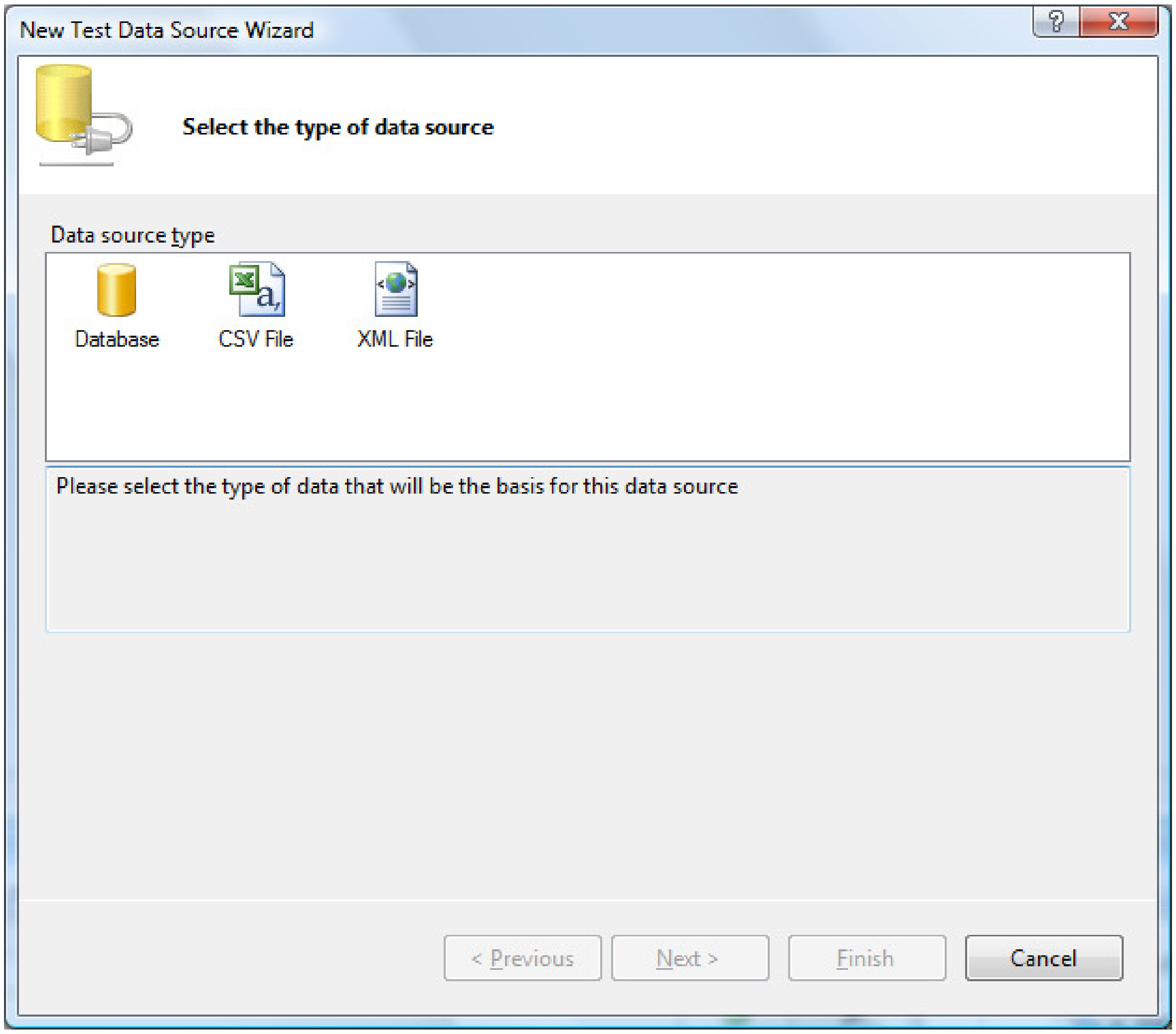


- Right click the test that you would like to start using an external data source for and select Properties. This opens the Properties window showing all the properties for that test.
- Look for the Data section in the properties window. If this section is not already expanded, then expand it by clicking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



DataConnectionString	
DataEnabled	True
DataProviderName	
DataRange	
DataTableName	
DataType	BuiltInGrid
DefaultToGrid	True
HasBuiltInGrid	True
IsDataDriven	True

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



Please select the type of data that will be the basis for this data source

- 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

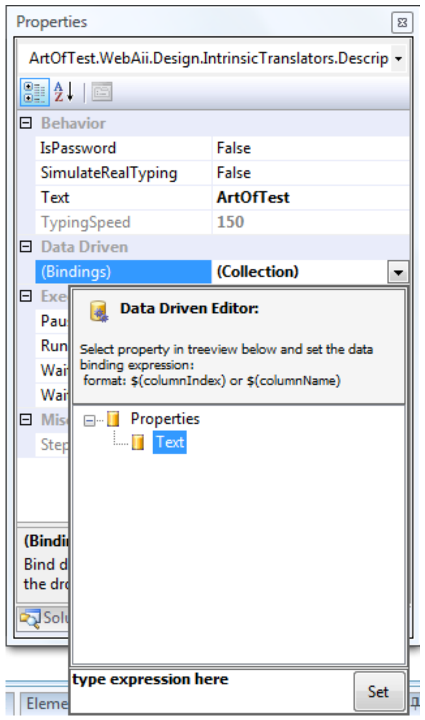
19.3 Reference the data array in a recorded step

Section 22.3 explains how to reference the data from your data grid.

19.4 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 9.7.6.
- 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.



Properties

ArtOfTest.WebAii.Design.IntrinsicTranslators.Descrip

⊟ Behavior	
IsPassword	False
SimulateRealTyping	False
Text	**ArtOfTest**
TypingSpeed	150
⊟ Data Driven	
(Bindings)	**(Collection)**
⊟ Exe	
Pau	
Run	
Wai	
Wai	
⊟ Mis	
Step	

(Bindi
Bind d
the dr

Sol

Eleme

Data Driven Editor:

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

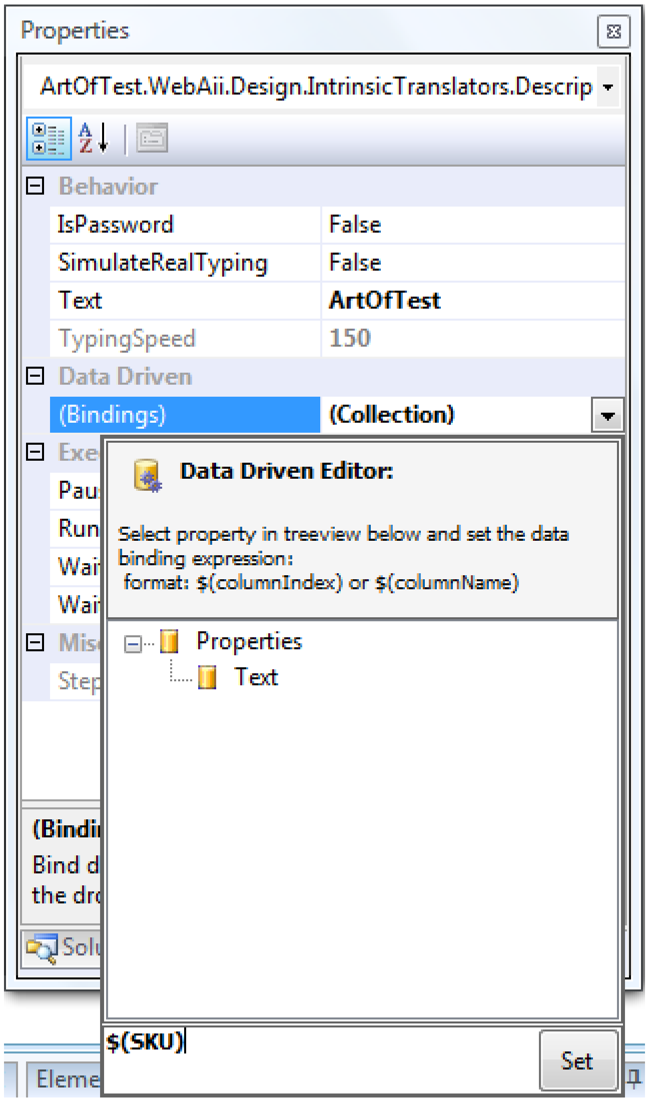
⊟ Properties
Text

type expression here

Set

- 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:
  ($0)
  or
  ($SKU)

- Use whichever form you prefer.



Properties

ArtOfTest.WebAii.Design.IntrinsicTranslators.Descrip

Behavior	
IsPassword	False
SimulateRealTyping	False
Text	**ArtOfTest**
TypingSpeed	150
Data Driven	
(Bindings)	**(Collection)**

Data Driven Editor:

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

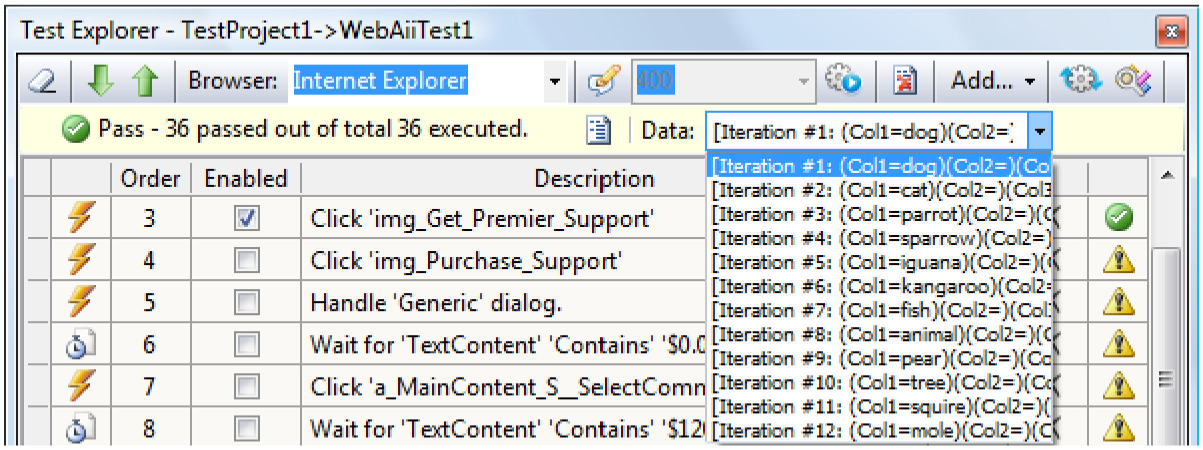
Properties
Text

$(SKU)

- 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’.

19.5 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:



Test Explorer - TestProject1->WebAiiTest1

Browser: Internet Explorer | 400 | Add...

Pass - 36 passed out of total 36 executed. | Data: [Iteration #1: (Col1=dog)(Col2=

Order	Enabled	Description	
3	✓	Click 'img_Get_Premier_Support'	✓
4		Click 'img_Purchase_Support'	⚠
5		Handle 'Generic' dialog.	⚠
6		Wait for 'TextContent' 'Contains' '$0.0	⚠
7		Click 'a_MainContent_S_SelectComn	⚠
8		Wait for 'TextContent' 'Contains' '$12(	⚠

```
[Iteration #1: (Col1=dog)(Col2=)(Co
[Iteration #2: (Col1=cat)(Col2=)(Col3
[Iteration #3: (Col1=parrot)(Col2=)(C
[Iteration #4: (Col1=sparrow)(Col2=)
[Iteration #5: (Col1=iguana)(Col2=)(
[Iteration #6: (Col1=kangaroo)(Col2=
[Iteration #7: (Col1=fish)(Col2=)(Col:
[Iteration #8: (Col1=animal)(Col2=)(C
[Iteration #9: (Col1=pear)(Col2=)(Co
[Iteration #10: (Col1=tree)(Col2=)(Co
[Iteration #11: (Col1=squire)(Col2=)(
[Iteration #12: (Col1=mole)(Col2=)(C
```

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.

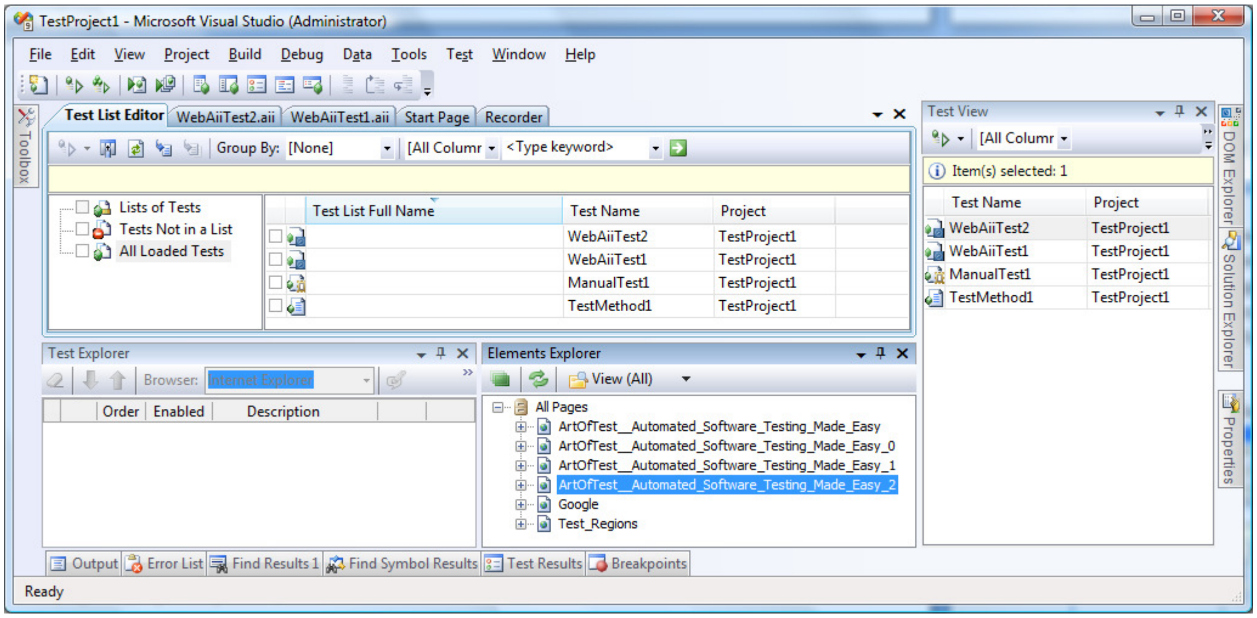
When you select one of the iterations from the iteration drop down list the pass/fail results for each test step in that iteration are displayed in Test Explorer. If one of them failed you can click on it to see the details of the failure in the debug UI. Hopefully you will have enough information to 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.

20 Visual Studio Integration

20.1 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 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:



Test List Full Name	Test Name	Project
	WebAiiTest2	TestProject1
	WebAiiTest1	TestProject1
	ManualTest1	TestProject1
	TestMethod1	TestProject1

Test Name	Project
WebAiiTest2	TestProject1
WebAiiTest1	TestProject1
ManualTest1	TestProject1
TestMethod1	TestProject1

Order	Enabled	Description

All Pages
- ArtOfTest__Automated_Software_Testing_Made_Easy
- ArtOfTest__Automated_Software_Testing_Made_Easy_0
- ArtOfTest__Automated_Software_Testing_Made_Easy_1
- ArtOfTest__Automated_Software_Testing_Made_Easy_2
- Google
- Test_Regions

21 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 Capturing:** As you record each step, Design Canvas captures the target element highlighted in a screenshot. 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.
- **Captured DOM Tree:** When a failure is detected, the DOM tree is captured and saved.

21.1 Resolving Test Step Failures

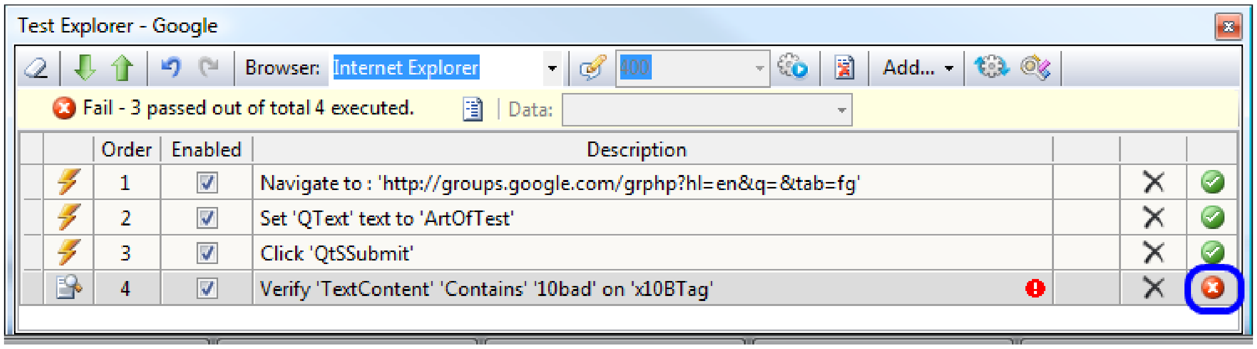
Automation Design Canvas offers built-in support through its execution engine to help solve some of the most common web automation failures and provides contextual information to help you readily resolve any issues. Typical web automation test step failures occur when:

- The test could not find an element it was looking for on a web page

  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.

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 the Step Failure Details UI.


Test Explorer - Google

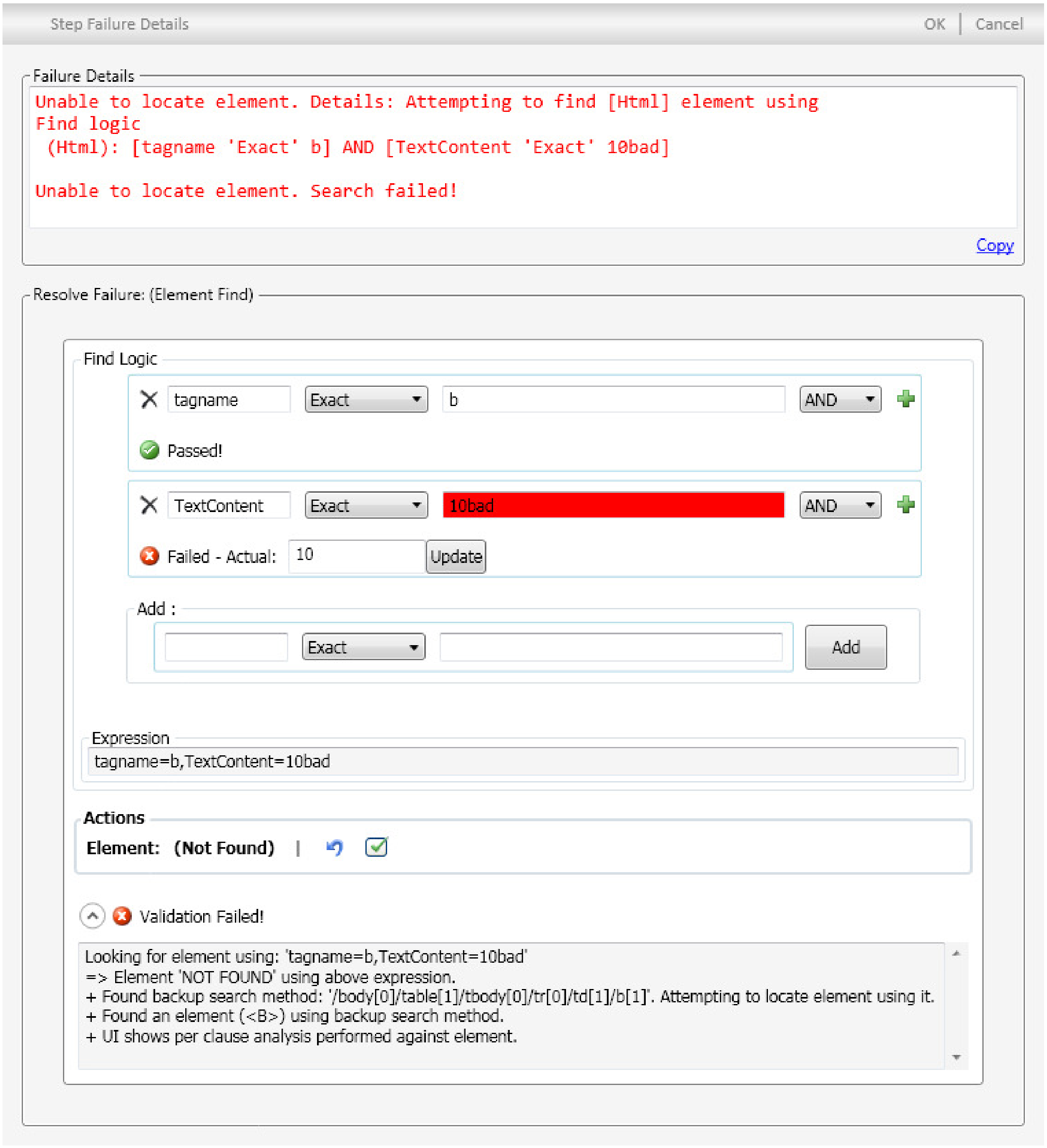
Browser: Internet Explorer 400 Add...

Fail - 3 passed out of total 4 executed. Data:

	Order	Enabled	Description			
	1	✓	Navigate to : 'http://groups.google.com/grphp?hl=en&q=&tab=fg'		✕	✓
	2	✓	Set 'QText' text to 'ArtOfTest'		✕	✓
	3	✓	Click 'QtSSubmit'		✕	✓
	4	✓	Verify 'TextContent' 'Contains' '10bad' on 'x10BTag'	!	✕	✕

21.2 Resolving element find failures

If Automation Design Canvas was unable to find the element the Step Failure Details will look something like the figure below. The top of the Element Editor shows details of the failure, for example:



```
Unable to locate element. Details: Attempting to find [Html] element using
Find logic
 (Html): [tagname 'Exact' b] AND [TextContent 'Exact' 10bad]

Unable to locate element. Search failed!
```

tagname	Exact	b	AND
Passed!			
TextContent	Exact	10bad	AND
Failed - Actual:	10	Update	

Expression
tagname=b,TextContent=10bad

Actions
Element: (Not Found)

Validation Failed!

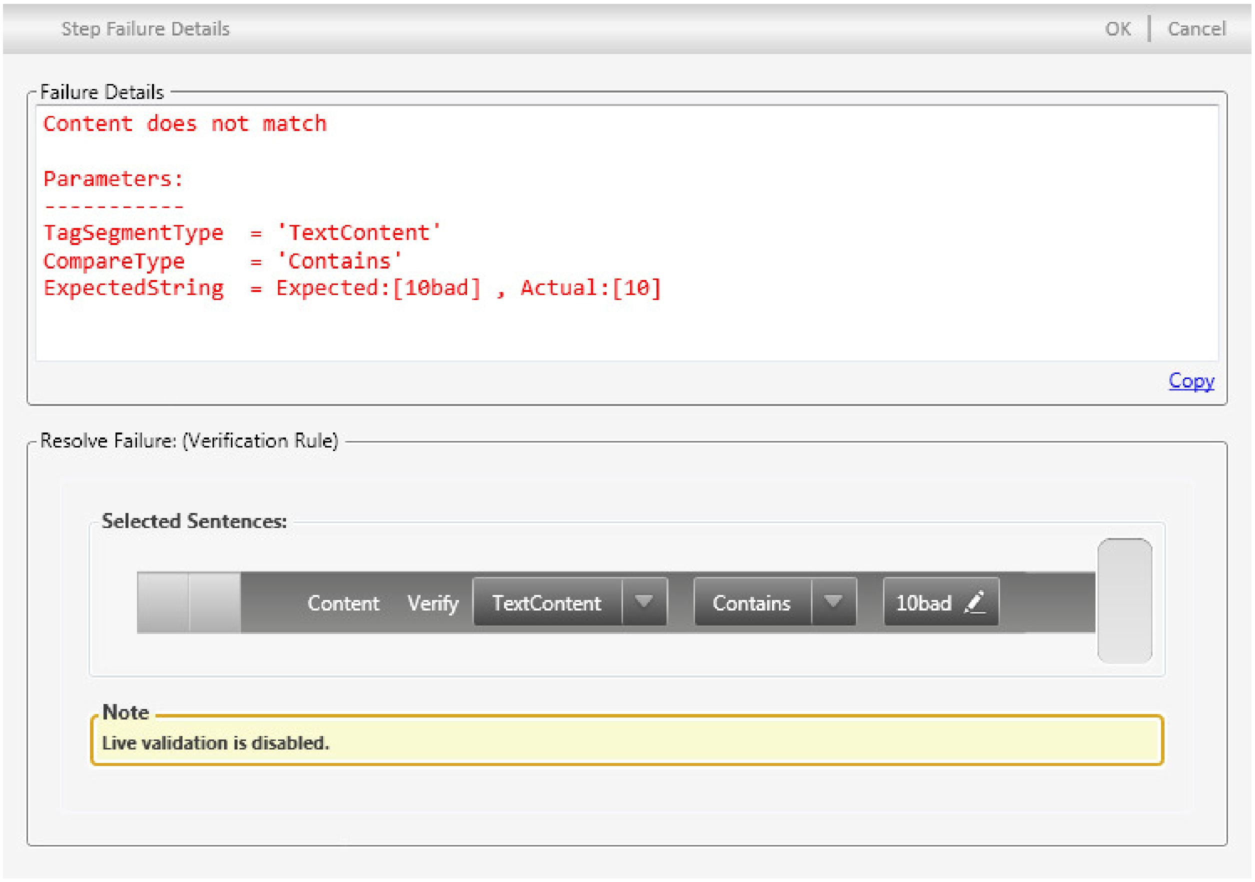
```
Looking for element using: 'tagname=b,TextContent=10bad'
=> Element 'NOT FOUND' using above expression.
+ Found backup search method: '/body[0]/table[1]/tbody[0]/tr[0]/td[1]/b[1]'. Attempting to locate element using it.
+ Found an element (<B>) using backup search method.
+ UI shows per clause analysis performed against element.
```

Here you see that Design Canvas was unable to find an element with a tagname of <b> and having text content equal to “10bad”. In this example Design Canvas was able to find an element using its backup method (a tag index path) and reports the differences. You have the option of updating the find expression clause with the suggested value.

Clicking OK will save the changes you made.

21.3 Resolving verification failures

If Automation Design Canvas found the element but could not validate some property of the element, Step Failure Details will look something like the below image. The top of Step Failure Details shows the details of the failure that Design Canvas encountered as illustrated by the following example:



```
Content does not match

Parameters:
-----------
TagSegmentType  = 'TextContent'
CompareType     = 'Contains'
ExpectedString  = Expected:[10bad] , Actual:[10]
```

Live validation is disabled.

Here you see that Design Canvas was expecting the string “10bad” but instead found the string “10”. We can proceed to change the verification rule as needed. Clicking OK saves your changes to the verification test step in the project.

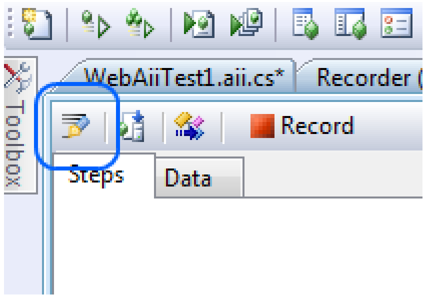
22 Test Customization: Adding a Code Behind File

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 methods to creating code behind methods:

- You can convert an existing recorded step into a code behind method. Section 9.7.1 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. Click this button to add a code-behind file to your test:



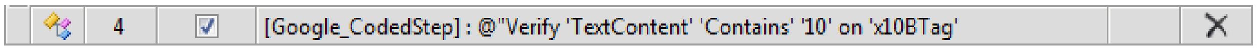
22.1 CodedStep Attribute

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 Google : BaseWebAiiTest
{
    #region [ Dynamic Pages Reference ]

    private Pages _pages;

    /// <summary>
    /// Gets the Pages object that has references
    /// to all the elements, frames or regions
    /// in this project.
    /// </summary>
    public Pages Pages
    {
        get
        {
            if (_pages == null)
            {
                _pages = new Pages(Manager.Current);
            }
            return _pages;
        }
    }

    #endregion

    [CodedStep(@"Verify 'TextContent' 'Contains' '10' on 'x10BTag'")]
    public void Google_CodedStep()
    {

        // Verify 'TextContent' 'Contains' '10' on 'x10BTag'

Pages.GoogleGroups_0.x10BTag.AssertContent().TextContent(ArtOfTest.Common.StringCompareType.Conta
ins, "10");

    }
}
```

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:


4 [Google_CodedStep] : @"Verify 'TextContent' 'Contains' '10' on 'x10BTag'

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.

22.2 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 web page name within the Pages object. Use the friendly name of the element as displayed in Elements Explorer as the element of the web page object. This returns a strongly typed WebAii 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/support/webaii/topicsindex.aspx.

22.3 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"]));
}
```

The data property code is shown above underlined and highlighted in yellow. Section 19 describes how to create a data array or connect to an external data source.

23 Using VS Web Tests and Generating VS Load Tests

23.1 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 23.3 explains how to do this.

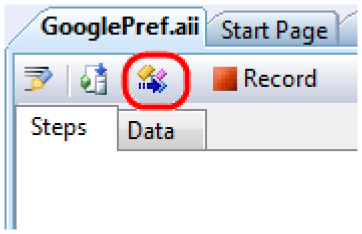
It is important to remember that once generated the new VS Web Test becomes its own independent test in your test project. Any changes to the Web Test or WebAii test are not automatically carried over from one to the other or back. The best practice is to use your WebAii test as the master source, make 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.

23.2 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.

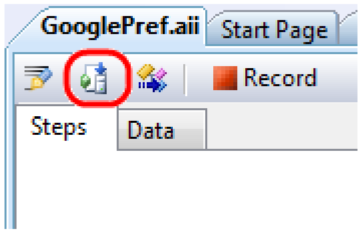
It is important to remember that once generated the new unit test becomes its own independent test in your test project. Any changes to the unit test or the WebAii test are not automatically carried over from one to the other. The best practice is to use your WebAii test as the master source, make any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

23.3 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. 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.

23.4 Debugging Your Test Using VS Debugger

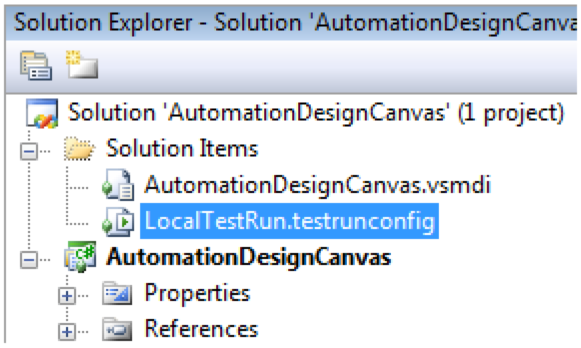
You can debug your code behind methods the same way you debug any other coded unit test with Visual Studio. You can set breakpoints in your code behind methods, and then execute them in debug mode. When execution reaches the breakpoint, the WebAii test will stop on that line of code.

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.

23.5 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:

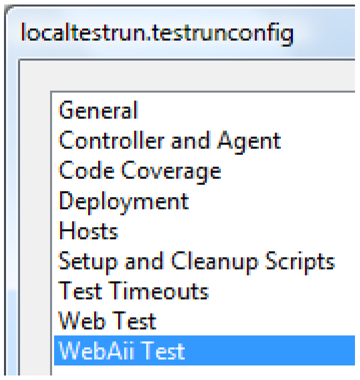
- Open the Solution Explorer window.
- Locate the test run configuration file for your project. The default is named LocalTestRun.testrunconfig.



Solution Explorer - Solution 'AutomationDesignCanva

Solution 'AutomationDesignCanvas' (1 project)
Solution Items
AutomationDesignCanvas.vsmdi
LocalTestRun.testrunconfig
AutomationDesignCanvas
Properties
References

- 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.



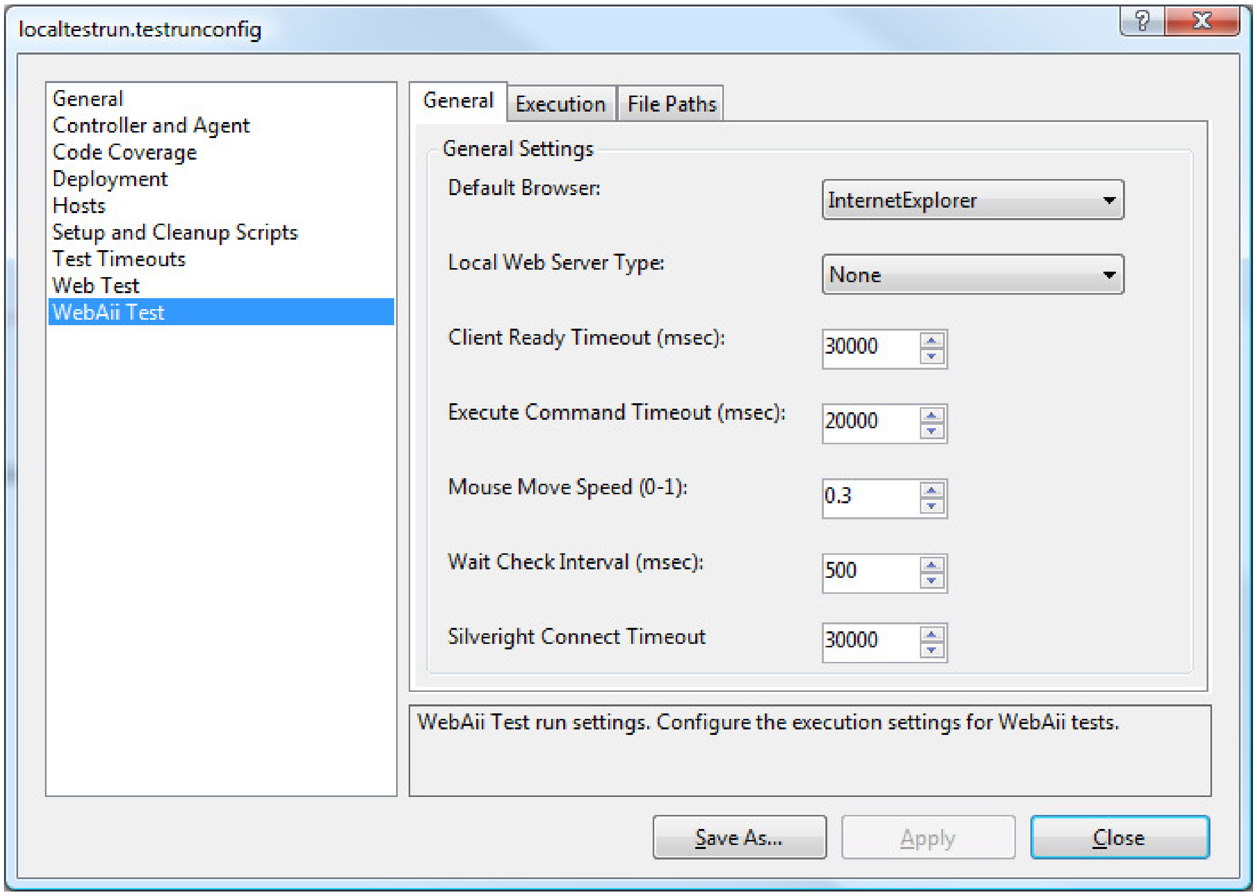
localtestrun.testrunconfig

General
Controller and Agent
Code Coverage
Deployment
Hosts
Setup and Cleanup Scripts
Test Timeouts
Web Test
WebAii Test

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

23.6 General settings tab

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


localtestrun.testrunconfig

General
Controller and Agent
Code Coverage
Deployment
Hosts
Setup and Cleanup Scripts
Test Timeouts
Web Test
WebAii Test

General | Execution | File Paths

General Settings

Default Browser:	InternetExplorer
Local Web Server Type:	None
Client Ready Timeout (msec):	30000
Execute Command Timeout (msec):	20000
Mouse Move Speed (0-1):	0.3
Wait Check Interval (msec):	500
Silveright Connect Timeout	30000

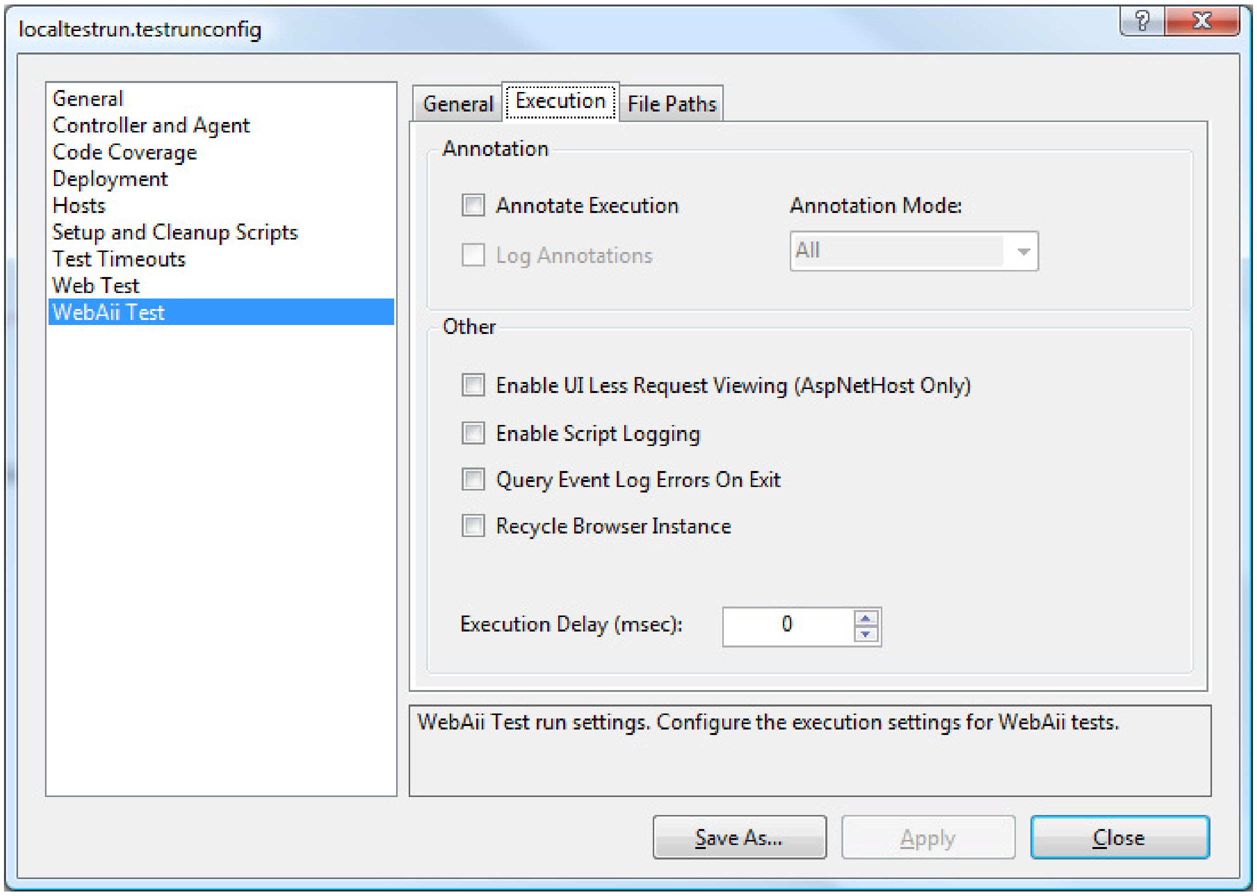
WebAii Test run settings. Configure the execution settings for WebAii tests.

- 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.
- Silverlight Connect Timeout – Sets the amount of time (in milliseconds) to wait for the WebAii framework to successfully connect to a Silverlight application contained on a webpage.

23.6.1 Execution tab

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


localtestrun.testrunconfig

General
Controller and Agent
Code Coverage
Deployment
Hosts
Setup and Cleanup Scripts
Test Timeouts
Web Test
WebAii Test

General | Execution | File Paths

Annotation

☐ Annotate Execution   Annotation Mode: All
☐ Log Annotations

Other

☐ Enable UI Less Request Viewing (AspNetHost Only)
☐ Enable Script Logging
☐ Query Event Log Errors On Exit
☐ Recycle Browser Instance

Execution Delay (msec): 0

WebAii Test run settings. Configure the execution settings for WebAii tests.

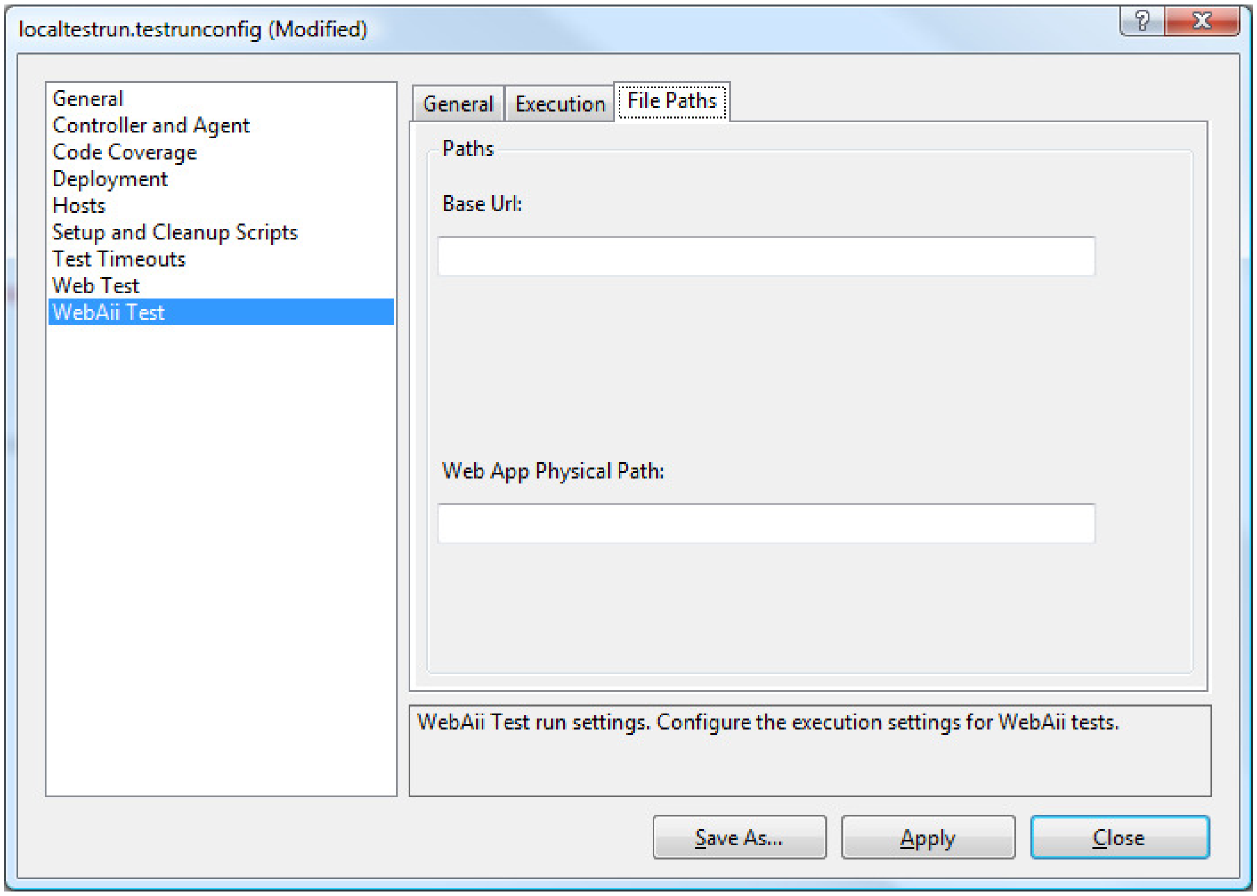
- Annotate Execution – Controls whether or not annotation is turned on during execution. The annotator is described in section 9.4.
- Log Annotations – Controls whether or not to write annotations to the WebAii test log file.
- Annotation Mode – Controls what types of annotation will be displayed in the browser. You can select from:
    - Native Only – only WebAii automatic annotations will be displayed
    - Custom Only – only custom annotation steps you added to your test will be displayed.
    - All – Both native and custom annotations 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.
- Recycle Browser Instance – Controls whether or not the one browser instance will be opened and used for all tests in a sequence versus opening and closing a new browser instance for each single test in a sequence.
- 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.

23.6.2 File Paths tab

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


localtestrun.testrunconfig (Modified)

General
Controller and Agent
Code Coverage
Deployment
Hosts
Setup and Cleanup Scripts
Test Timeouts
Web Test
WebAii Test

General | Execution | File Paths

Paths

Base Url:

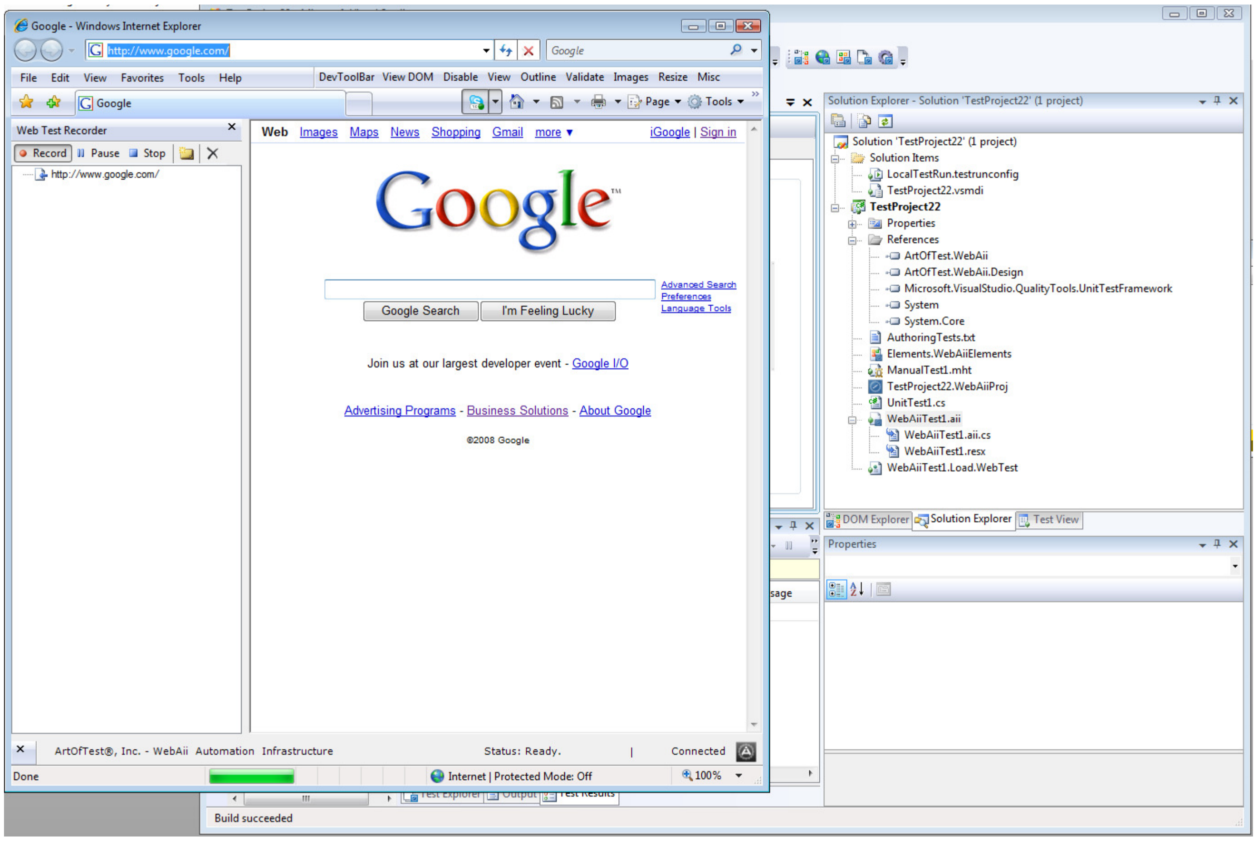
Web App Physical Path:

WebAii Test run settings. Configure the execution settings for WebAii tests.

- 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").
- Web App Physical Path – This setting is used when **Default Browser** (found on the **General** tab) is set to "**AspNetHost**". It must be the full physical path to the web application root that you are testing.

23.7 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 LogMessageOnFailure

Any text you enter here will be added to the log, but only if that step fails.

A.1.3 Pause

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.1.4 PrimaryTarget

Identifies which element that step interacts with. If the step is a Drag & Drop operation it represents the element that will be dragged. If you click the … a Project Elements Selector dialog opens allowing you to change the target element of that step.

A.1.5 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.
- SafariOnly – The step will only run when the test is running in a Safari browser window.

A.1.6 SecondaryTarget

This property is only used for Drag & Drop operations. It specifies the target element that the dragged element will be dragged to.

A.1.7 SimulateRealClick

When set to true, Click steps will be performed by moving the mouse curser and sending a mouse click event to the browser window. When set to false, a Click event will be sent directly to the target element.

Best practice states that using the Click event is generally more reliable and preferable to use than mouse clicks. Mouse clicks can fail if another window gets in the way of a target element by overlaying the browser window. When this happens all you get is a click on the overlaid window instead of on the intended target element. Also locking the computer, even the screen saver can interfere with simulating real mouse clicks.

A.1.8 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.1.9 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 BaseUrl

A.2.2 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 Automation 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 ends 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,"<div id="div2">". 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,"<div id="div4">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 ends 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 ends 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.

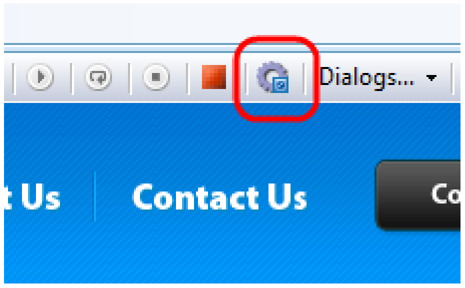
A.23.4.3 DroptargetType

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

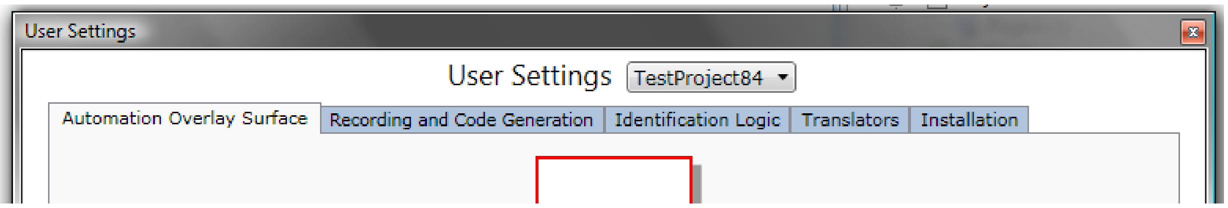
- Element – Drops onto the specified element using the specified drop offset values.
- Window – Ignores the specified drop element and uses the drop offset values to be values within the browser window.

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:

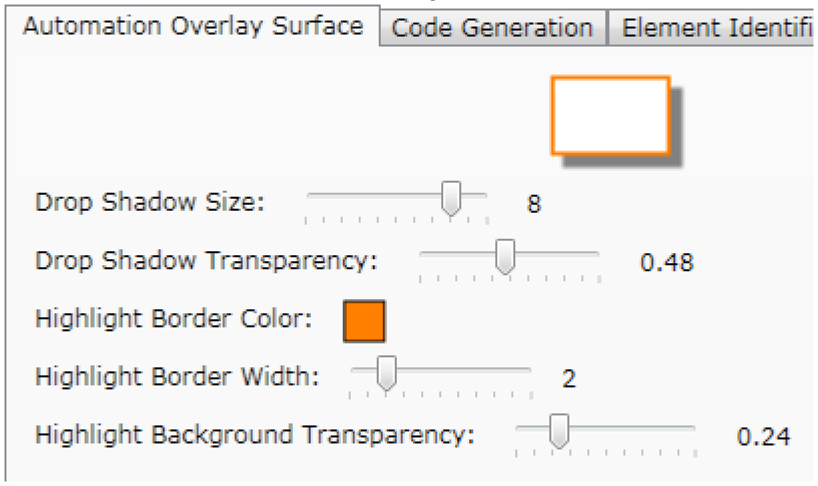


In this dialog you can change the WebAii project settings for all of the WebAii projects that are in your solution. Once the dialog opens you need to select which project you want to change the settings of. Select the project you are going to work with using the dropdown at the top of the dialog:



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

B.1 Automation Overlay Surface tab



Drop Shadow Size: 8

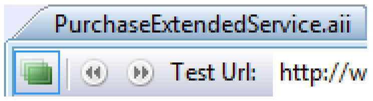
Drop Shadow Transparency: 0.48

Highlight Border Color: 🟧

Highlight Border Width: 2

Highlight Background Transparency: 0.24

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.

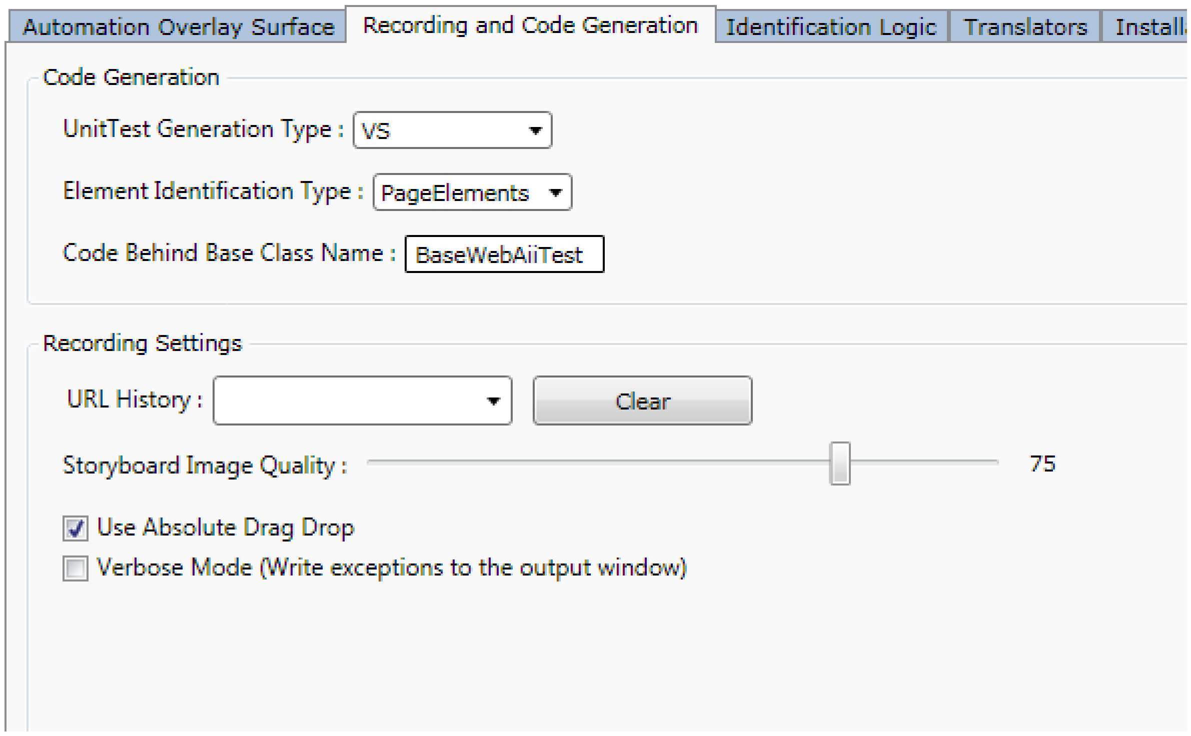
B.2.3 Highlight Border Width

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



Code Generation

UnitTest Generation Type : VS

Element Identification Type : PageElements

Code Behind Base Class Name : BaseWebAiiTest

Recording Settings

URL History :

Clear

Storyboard Image Quality : 75

[✓] Use Absolute Drag Drop

[ ] Verbose Mode (Write exceptions to the output window)

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.

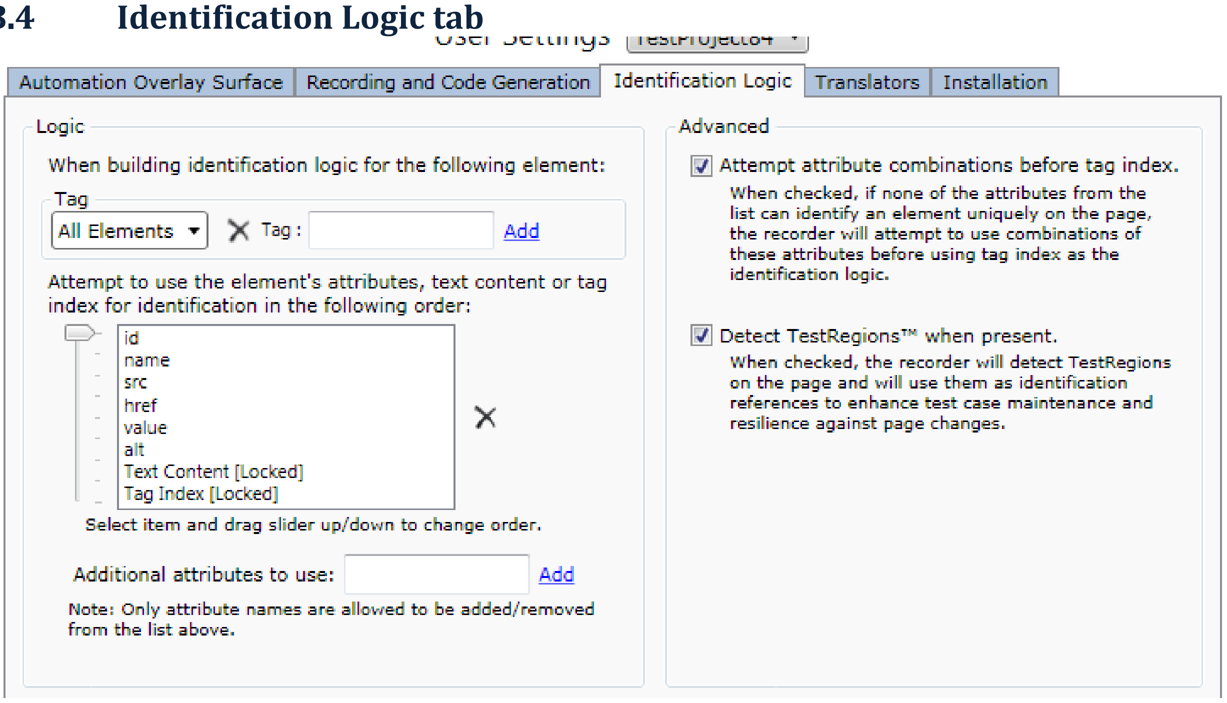
B.3.6 Use Absolute Drag Drop

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



Logic

When building identification logic for the following element:

Tag

All Elements ▾ ✕ Tag : Add

Attempt to use the element's attributes, text content or tag index for identification in the following order:

- id
- name
- src
- href
- value
- alt
- Text Content [Locked]
- Tag Index [Locked]

Select item and drag slider up/down to change order.

Additional attributes to use: Add

Note: Only attribute names are allowed to be added/removed from the list above.

Advanced

✓ Attempt attribute combinations before tag index.

When checked, if none of the attributes from the list can identify an element uniquely on the page, the recorder will attempt to use combinations of these attributes before using tag index as the identification logic.

✓ Detect TestRegions™ when present.

When checked, the recorder will detect TestRegions on the page and will use them as identification references to enhance test case maintenance and resilience against page changes.

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.

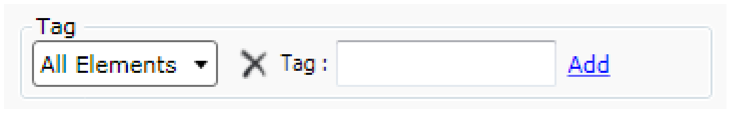
B.4.1 Tag

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.

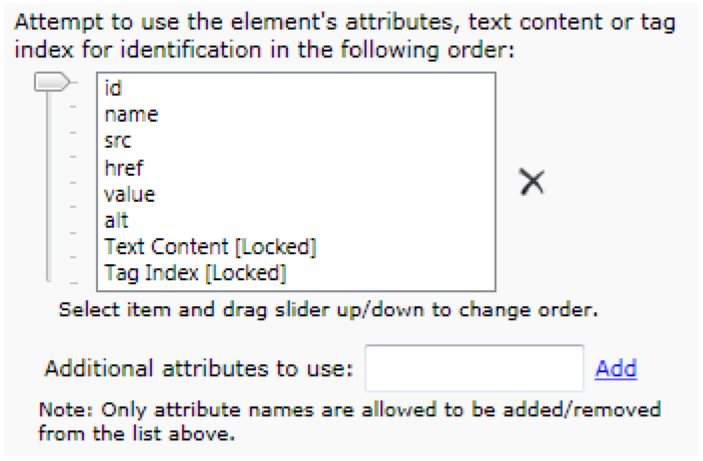
- Repeat the above steps for the <img> tag, entering “img” instead of “a”. You can continue doing this for as many different tags you want to create a custom attribute priority scheme.



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


Attempt to use the element's attributes, text content or tag index for identification in the following order:

id
name
src
href
value
alt
Text Content [Locked]
Tag Index [Locked]

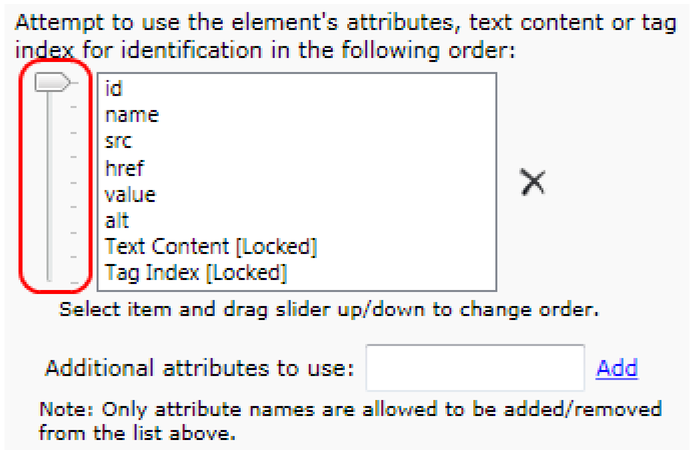
Select item and drag slider up/down to change order.

Additional attributes to use: Add

Note: Only attribute names are allowed to be added/removed from the list above.

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


Attempt to use the element's attributes, text content or tag index for identification in the following order:

id
name
src
href
value
alt
Text Content [Locked]
Tag Index [Locked]

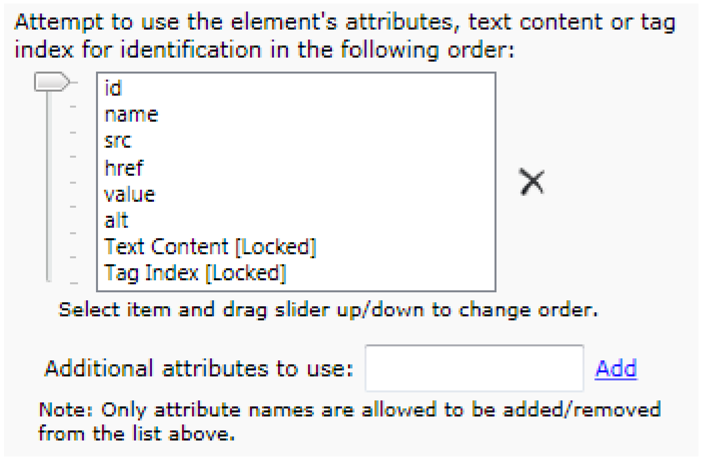
Select item and drag slider up/down to change order.

Additional attributes to use: Add

Note: Only attribute names are allowed to be added/removed from the list above.

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:



Attempt to use the element's attributes, text content or tag index for identification in the following order:

id
name
src
href
value
alt
Text Content [Locked]
Tag Index [Locked]

Select item and drag slider up/down to change order.

Additional attributes to use: Add

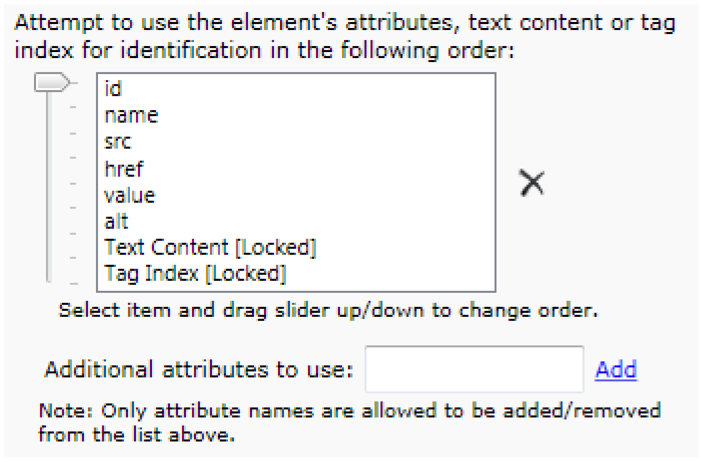
Note: Only attribute names are allowed to be added/removed from the list above.

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


Attempt to use the element's attributes, text content or tag index for identification in the following order:

id
name
src
href
value
alt
Text Content [Locked]
Tag Index [Locked]

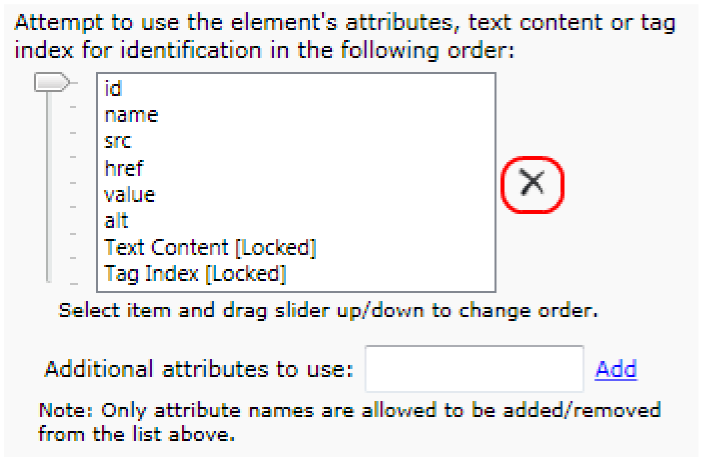
Select item and drag slider up/down to change order.

Additional attributes to use: Add

Note: Only attribute names are allowed to be added/removed from the list above.

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


Attempt to use the element's attributes, text content or tag index for identification in the following order:

id
name
src
href
value
alt
Text Content [Locked]
Tag Index [Locked]

Select item and drag slider up/down to change order.

Additional attributes to use: Add

Note: Only attribute names are allowed to be added/removed from the list above.

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

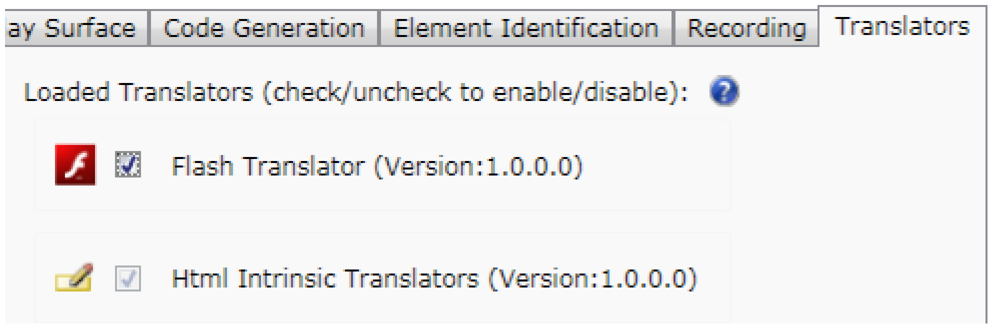
This option will craft the FindParam using a test region (if present) as part of the FindParam generation. More information about using test regions in web page's is available online at http://www.artoftest.com/Resources/WebAii/Documentation/topicsindex.aspx?topic=introtestregions.

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



Loaded Translators (check/uncheck to enable/disable):

✓ Flash Translator (Version:1.0.0.0)

✓ Html Intrinsic Translators (Version:1.0.0.0)

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.