Web-testing automation

Selenium IDE

Maryna Didkovska

To Automate or Not to Automate? That is the Question!

Test automation supports:

- Frequent regression testing
- Rapid feedback to developers during the development process
- Virtually unlimited iterations of test case execution
- Customized reporting of application defects
- Support for Agile and eXtreme development methodologies
- Disciplined documentation of test cases
- Finding defects missed by manual testing

Selenium Components

Selenium is composed of three major tools. Each one has a specific role in aiding the development of web.

These tools are:

- Selenium-IDE;
- Selenium Remote Control (RC); Selenium WebDriver
- Selenium Grid;

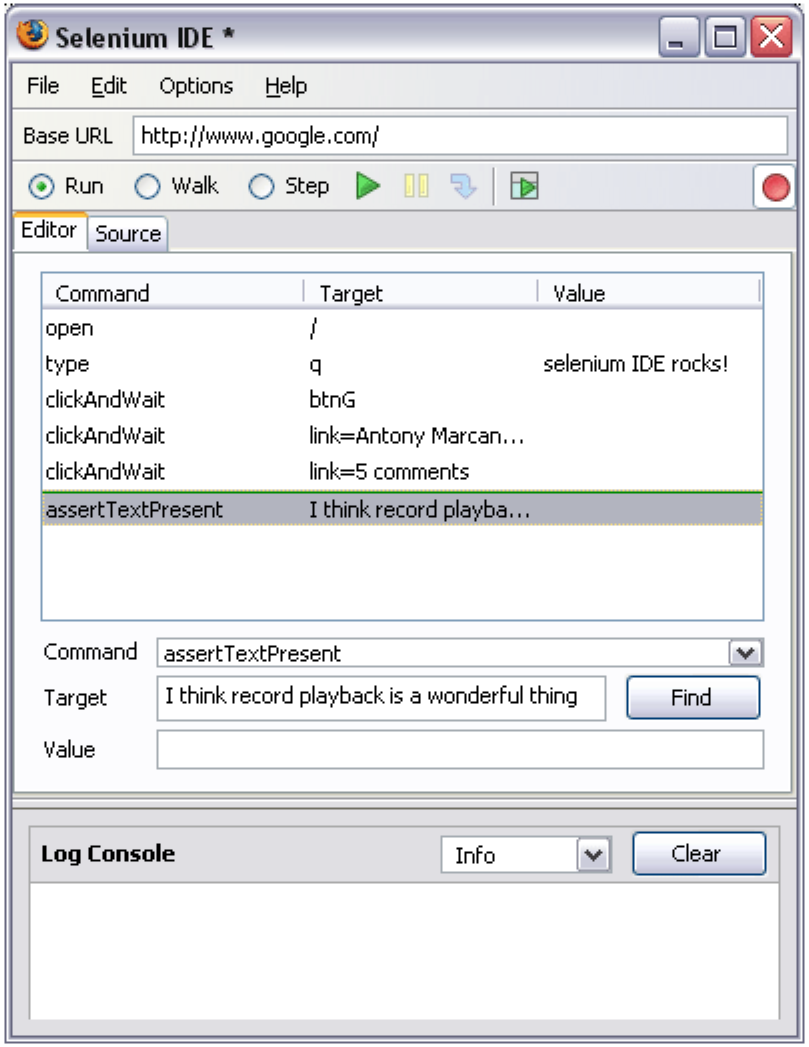
Selenium

Selenium is a robust set of tools that supports rapid development of test automation for web-based applications.

It is open source software, released under the Apache 2.0 license and can be downloaded and used without charge.

Selenium-IDE

Selenium-IDE (Integrated Development Environment) is a Firefox add-on that records clicks, typing, and other actions to make a test, which you can play back in the browser.



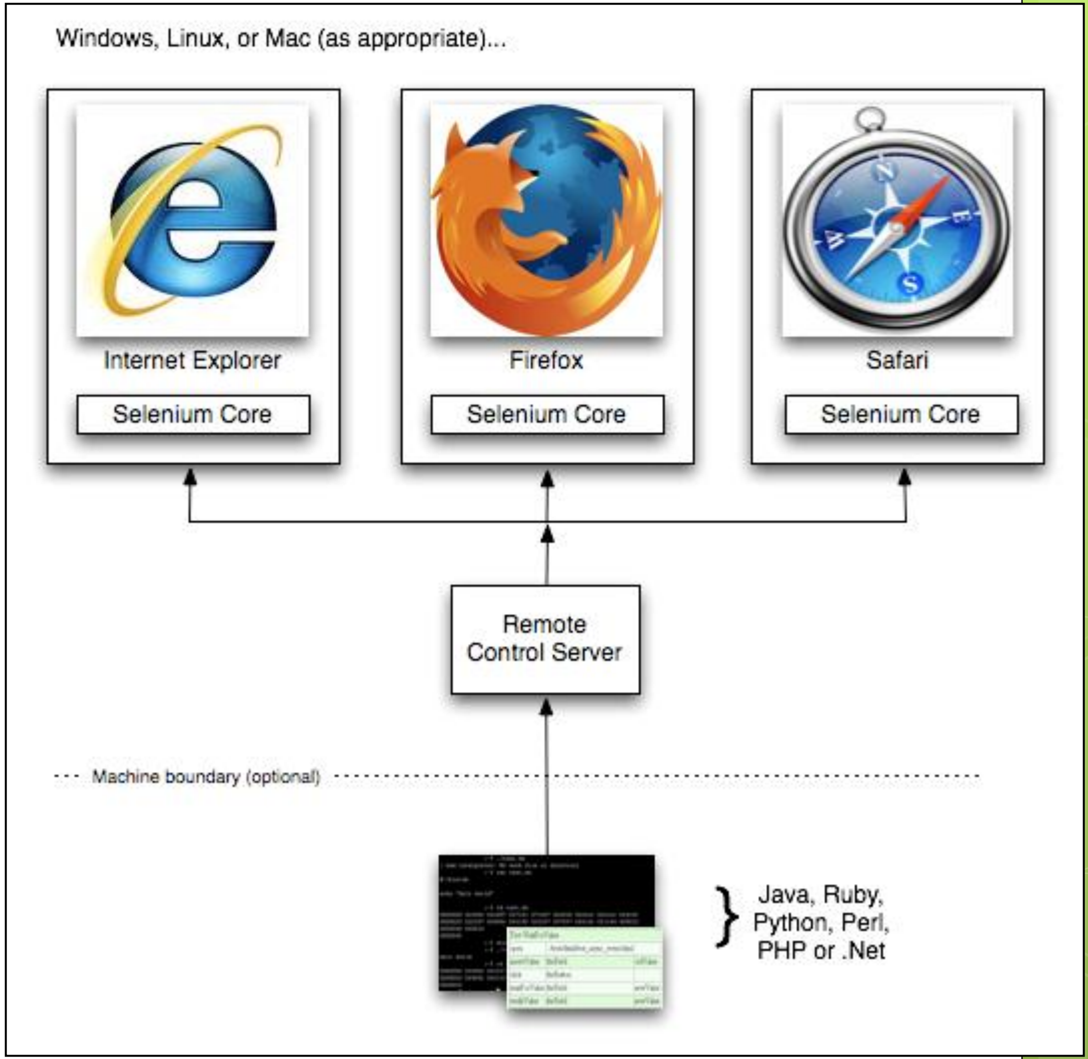
Base URL: http://www.google.com/

Command	Target	Value
open	/	
type	q	selenium IDE rocks!
clickAndWait	btnG	
clickAndWait	link=Antony Marcan...	
clickAndWait	link=5 comments	
assertTextPresent	I think record playba...	

Command: assertTextPresent
Target: I think record playback is a wonderful thing
Value:

Selenium-RC

Selenium Remote Control (RC) is a test tool that allows to write automated web application UI tests in any programming language against any HTTP website using any mainstream JavaScript-enabled browser.


Windows, Linux, or Mac (as appropriate)...
Internet Explorer
Selenium Core
Firefox
Selenium Core
Safari
Selenium Core
Remote Control Server
Machine boundary (optional)
Java, Ruby, Python, Perl, PHP or .Net

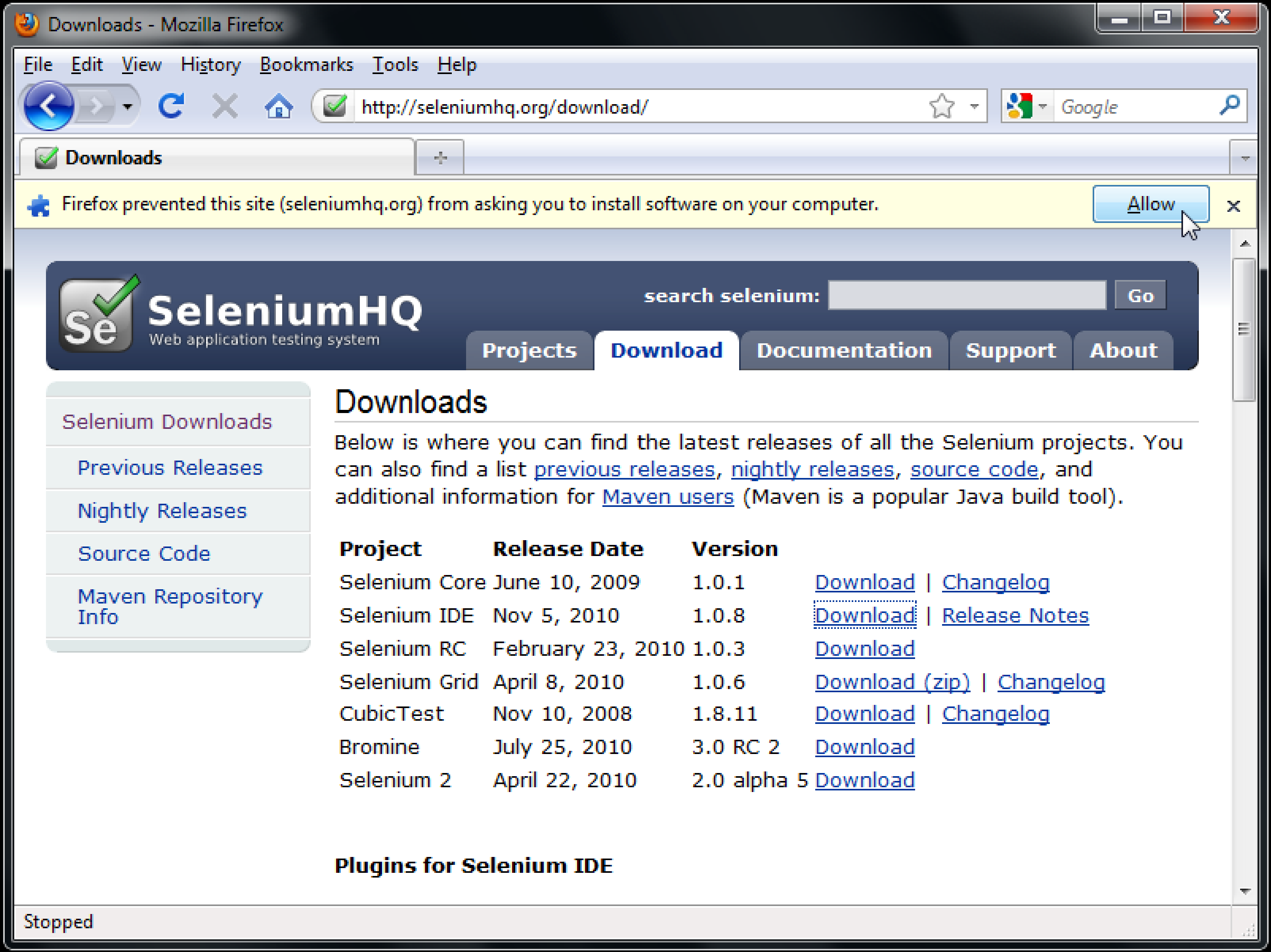
Selenium Grid

Selenium Grid extends Selenium RC to distribute your tests across multiple servers, saving you time by running tests in parallel, cutting down on the time it takes to test multiple browsers or operating systems.

With Selenium-Grid, multiple instances of Selenium-RC are running on various operating system and browser configurations.

Installation

- http://seleniumhq.org/download/



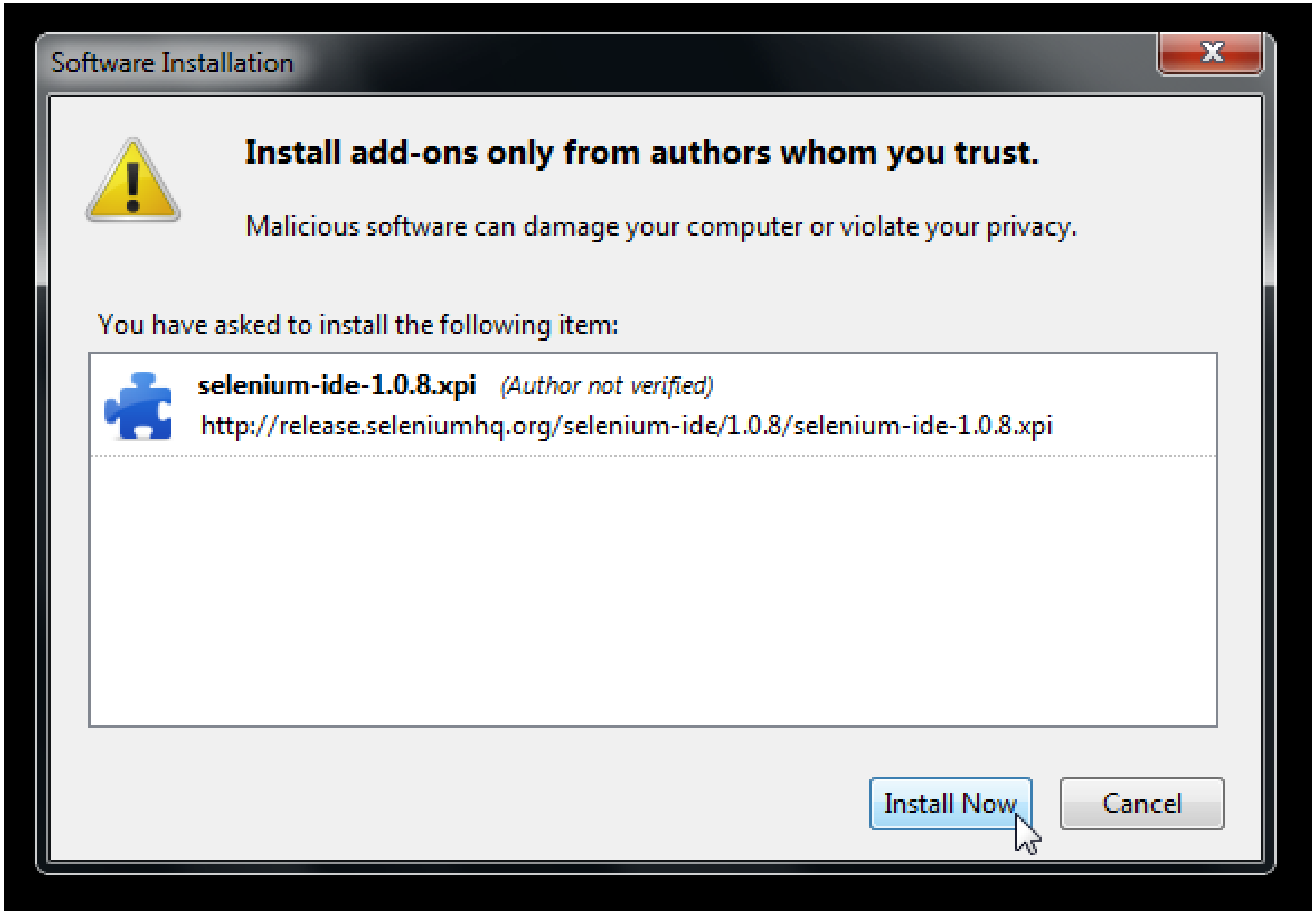
Firefox prevented this site (seleniumhq.org) from asking you to install software on your computer.

Downloads

Below is where you can find the latest releases of all the Selenium projects. You can also find a list previous releases, nightly releases, source code, and additional information for Maven users (Maven is a popular Java build tool).

Project	Release Date	Version	
Selenium Core	June 10, 2009	1.0.1	Download | Changelog
Selenium IDE	Nov 5, 2010	1.0.8	Download | Release Notes
Selenium RC	February 23, 2010	1.0.3	Download
Selenium Grid	April 8, 2010	1.0.6	Download (zip) | Changelog
CubicTest	Nov 10, 2008	1.8.11	Download | Changelog
Bromine	July 25, 2010	3.0 RC 2	Download
Selenium 2	April 22, 2010	2.0 alpha 5	Download

Plugins for Selenium IDE



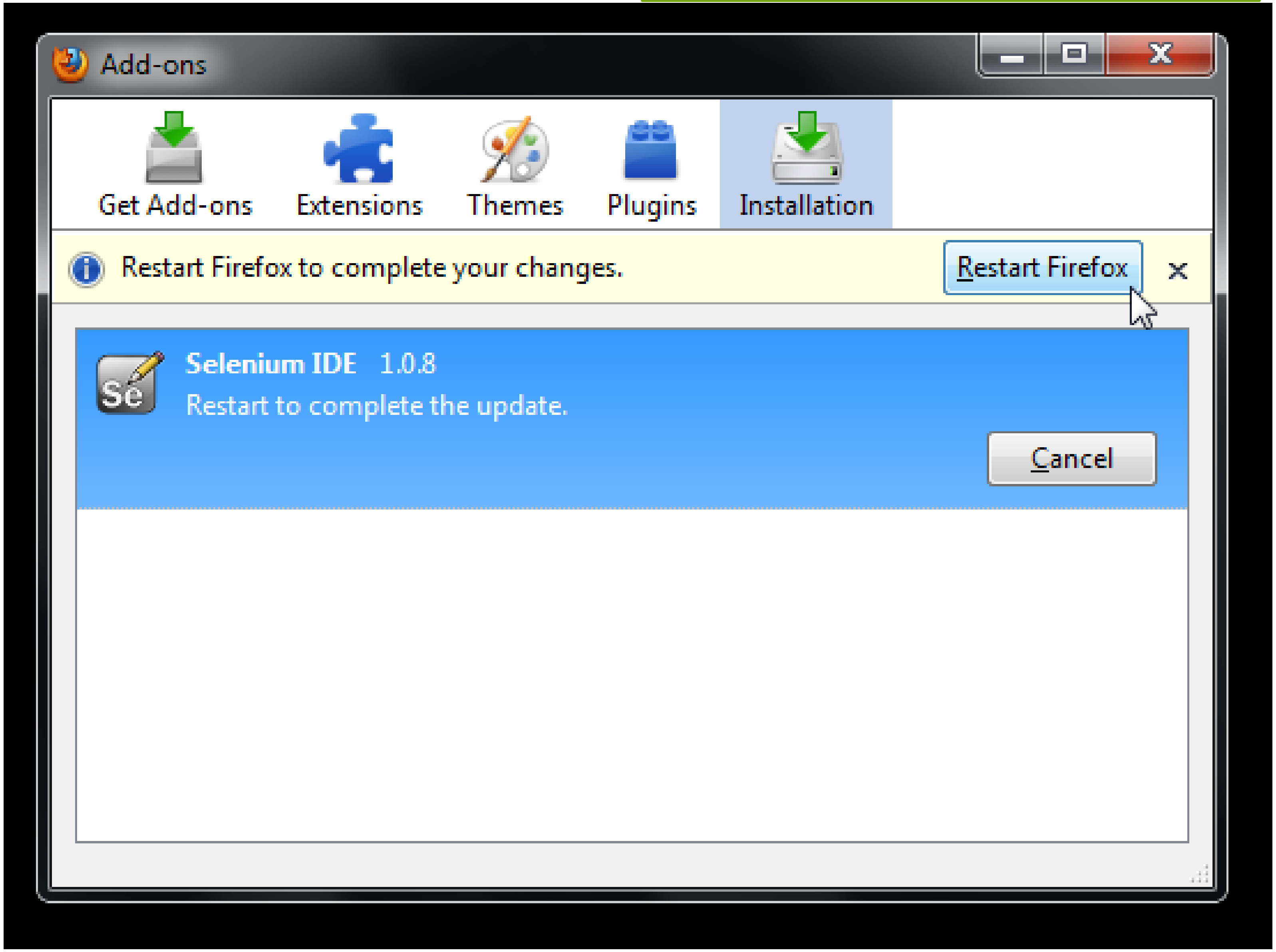
Software Installation

Install add-ons only from authors whom you trust.

Malicious software can damage your computer or violate your privacy.

You have asked to install the following item:

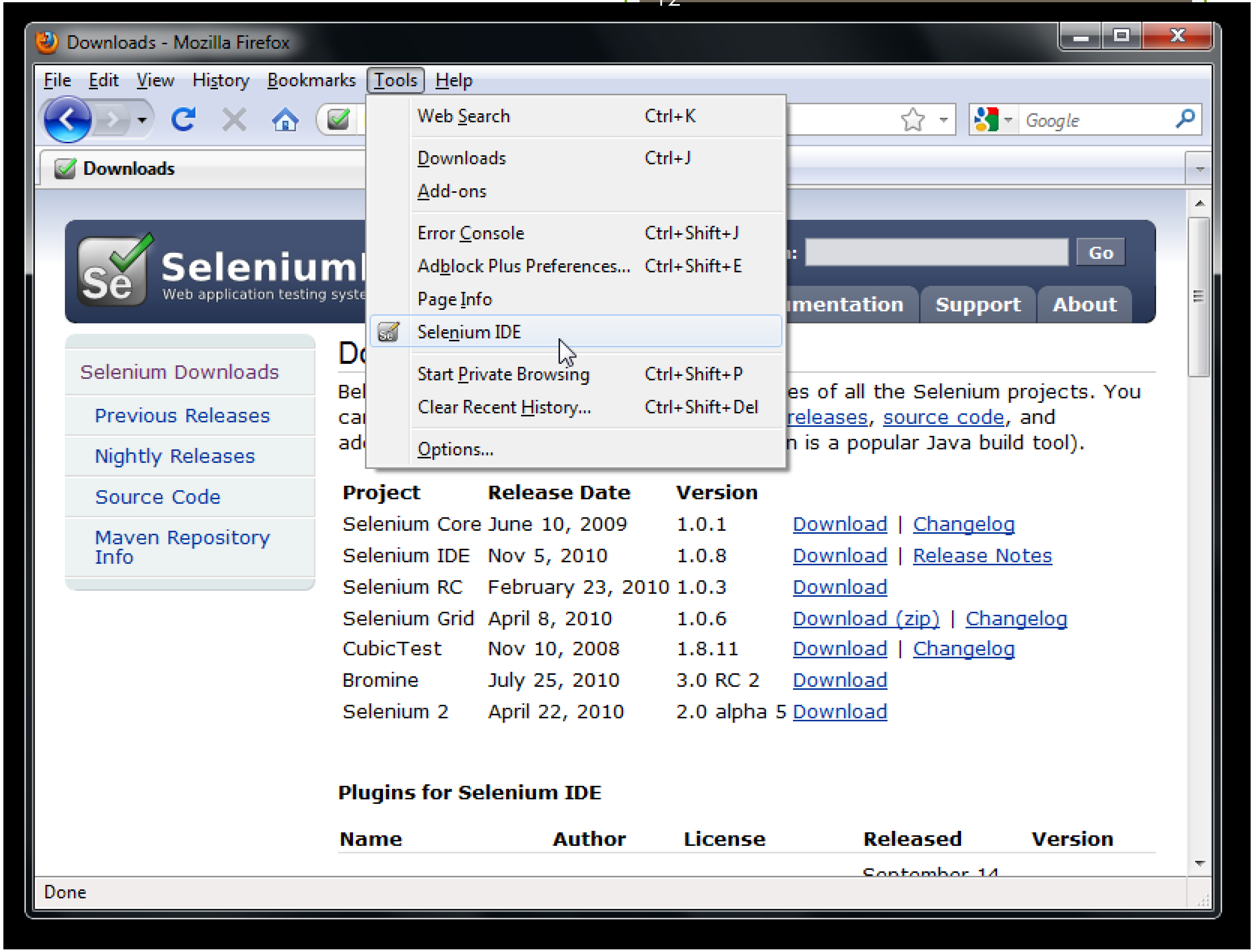
selenium-ide-1.0.8.xpi *(Author not verified)*
http://release.seleniumhq.org/selenium-ide/1.0.8/selenium-ide-1.0.8.xpi



Restart Firefox to complete your changes.

Selenium IDE 1.0.8

Restart to complete the update.



Do

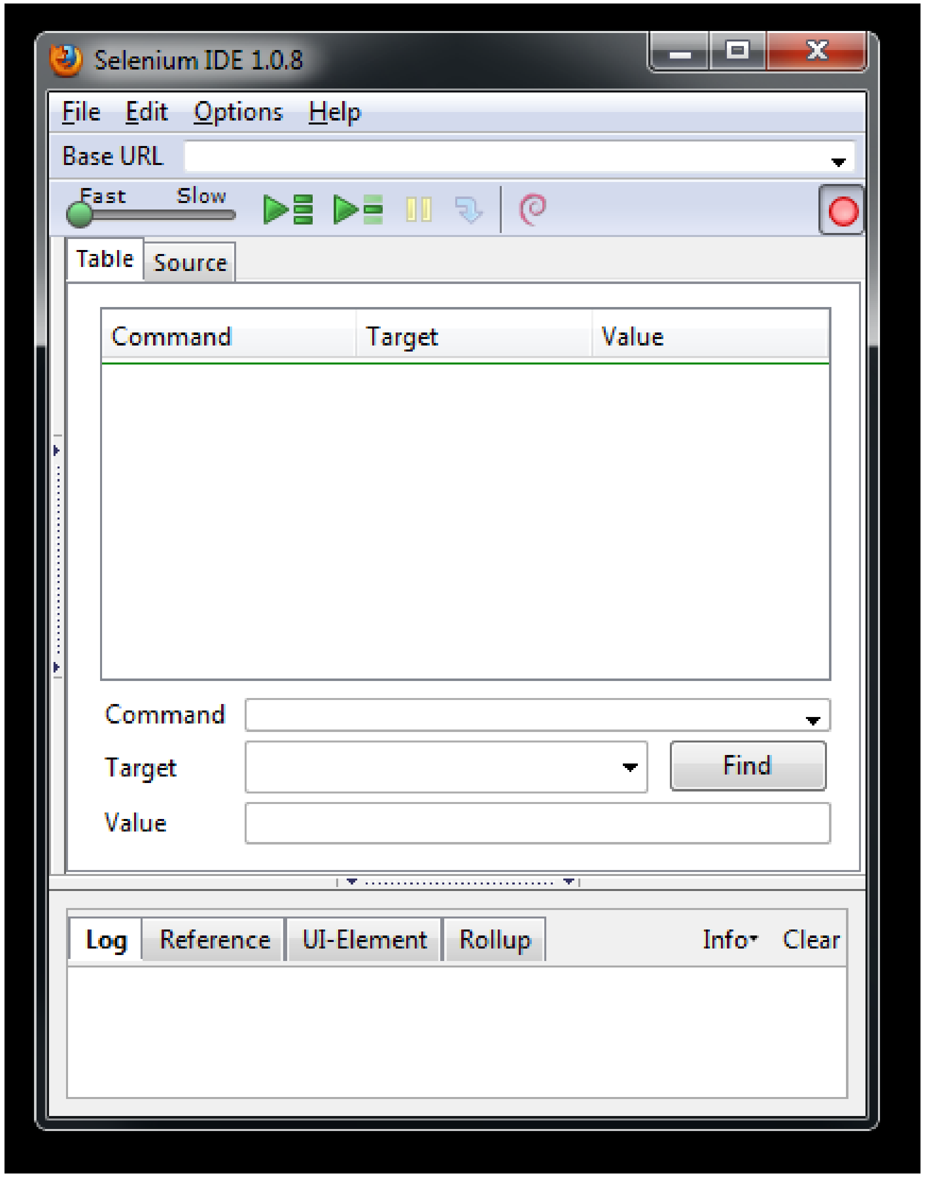
Bel es of all the Selenium projects. You car releases, source code, and ad n is a popular Java build tool).

Project	Release Date	Version	
Selenium Core	June 10, 2009	1.0.1	Download | Changelog
Selenium IDE	Nov 5, 2010	1.0.8	Download | Release Notes
Selenium RC	February 23, 2010	1.0.3	Download
Selenium Grid	April 8, 2010	1.0.6	Download (zip) | Changelog
CubicTest	Nov 10, 2008	1.8.11	Download | Changelog
Bromine	July 25, 2010	3.0 RC 2	Download
Selenium 2	April 22, 2010	2.0 alpha 5	Download

Plugins for Selenium IDE

Name	Author	License	Released	Version

Open the IDE


Selenium IDE 1.0.8
File Edit Options Help
Base URL
Fast Slow
Table Source
Command Target Value
Command
Target
Find
Value
Log Reference UI-Element Rollup Info Clear

Building Test Cases

During recording, Selenium-IDE will automatically insert commands into your test case based on your actions. Typically, this will include:

- clicking a link - *click* or *clickAndWait* commands
- entering values - *type* command
- selecting options from a drop-down listbox - *select* command
- clicking checkboxes or radio buttons - *click* command

Task 1

Record script with following steps:

- Open http://www.google.com.ua/
- Search for web application testing
- Goto Selenium site

Playback it

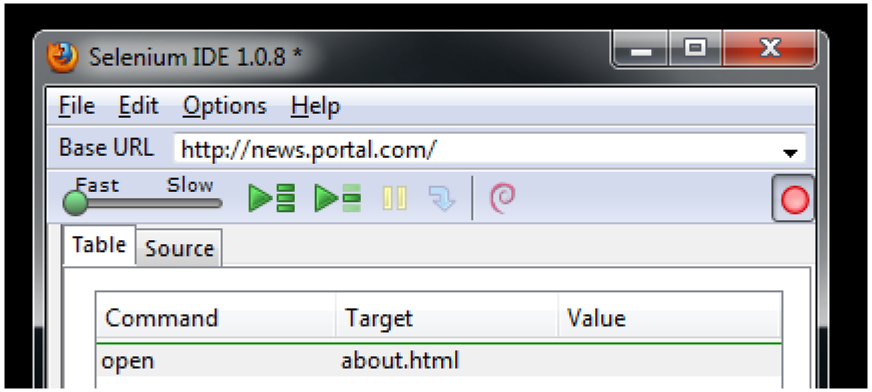
Base url : http://www.google.com.ua/

Open	/	
type	id=lst-ib	web application testing
clickAndWait	name=btnK	
clickAndWait	link=Selenium web application testing system	

Running Test Cases

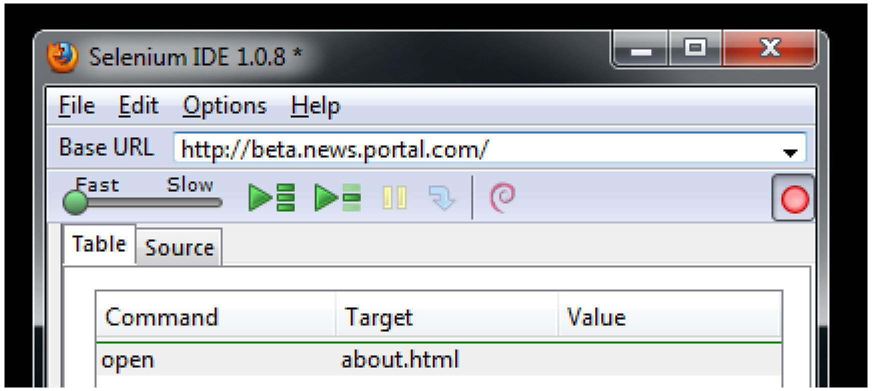
- Run a Test Case
- Run a Test Suite
- Stop and Start
- Stop in the Middle
- Start from the Middle
  - To set a startpoint, select a command, right-click, and from the context menu select Set/Clear Start Point.
- Run Any Single Command
  - Double-click any single command to run it by itself.

Using Base URL to Run Test Cases in Different Domains



Base URL http://news.portal.com/

Command	Target	Value
open	about.html	



Base URL http://beta.news.portal.com/

Command	Target	Value
open	about.html	

Selenium Commands – Selenese

Selenium provides a rich set of commands for fully testing your web-application in virtually any way you can imagine.

A *command* is what tells Selenium what to do.

The command set is often called *selenese*.

These commands essentially create a testing language.

Selenium Basics

Selenium commands come in three "flavors":

- Actions are commands that generally manipulate the state of the application. They do things like "click this link" and "select that option". If an Action fails, or has an error, the execution of the current test is stopped.
- Accessors examine the state of the application and store the results in variables, e.g. "storeTitle".
- Assertions are like Accessors, but they verify that the state of the application conforms to what is expected. Examples include "make sure the page title is X" and "verify that this checkbox is checked".

Assertions

All Selenium Assertions can be used in 3 modes:

- “assert”
- “verify”
- “waitFor”

When an “assert” fails, the test is aborted.

When a “verify” fails, the test will continue execution, logging the failure.

“waitFor” commands wait for some condition to become true. They will succeed immediately if the condition is already true. However, they will fail and halt the test if the condition does not become true within the current timeout setting.

Script Syntax

Selenium commands are simple, they consist of the command and two parameters. For example:

Command	Target	Value
verifyText	//div//a[2]	Login

The parameters are not always required; it depends on the command.

Command	Target	Value
goBackAndWait		
verifyTextPresent		Welcome to My Home Page
type	id=phone	(555) 666-7066
type	id=phone	${myVariableAddress}

Parameters

Parameters vary, however they are typically:

- a *locator* for identifying a UI element within a page.
- a *text pattern* for verifying or asserting expected page content.
- a *text pattern* or a selenium variable for entering text in an input field or for selecting an option from an option list.

Locators

For many Selenium commands, a target is required. This target identifies an element in the content of the web application, and consists of the location strategy followed by the location in the format

locatorType=location.

Locator Types:

id	(id=loginForm);
name	(name=username);
xpath	(xpath=/html/body/form[1]);
link	(link=Cancel);
dom	(dom=document.forms[0]);
css	(css=input.required[type="text"]);

Commonly Used Selenium Commands

open	opens a page using a URL
click/clickAndWait	performs a click operation, and optionally waits for a new page to load
verifyTitle/assertTitle	verifies an expected page title
verifyTextPresent	verifies expected text is somewhere on the page
verifyElementPresent	verifies an expected UI element, as defined by its HTML tag, is present on the page
verifyText	verifies expected text and it's corresponding HTML tag are present on the page
waitForPageToLoad	pauses execution until an expected new page loads. Called automatically when clickAndWait is used
waitForElementPresent	pauses execution until an expected UI element, as defined by its HTML tag, is present on the page

Matching Text Patterns - Globbing Patterns

Globbing is fairly limited. Only two special characters are supported in the Selenium implementation:

- ***** which translates to “match anything,” i.e., nothing, a single character, or many characters.
- **[ ]** (*character class*) which translates to “match any single character found inside the square brackets.” A dash (hyphen) can be used as a shorthand to specify a range of characters (which are contiguous in the ASCII character set).

examples:

- [aeiou] matches any lowercase vowel
- [0-9] matches any digit
- [a-zA-Z0-9] matches any alphanumeric character

- example of two commands that use globbing patterns. The actual link text on the page being tested was "Film/Television Department"; by using a pattern rather than the exact text, the **click** command will work even if the link text is changed to "Film & Television Department" or "Film and Television Department". The glob pattern's asterisk will match "anything or nothing" between the word "Film" and the word "Television".

Command	Target	Value
click	link=glob:Film*Television Department	
verifyTitle	glob:*Film*Television*	

- The actual title of the page reached by clicking on the link was "De Anza Film And Television Department - Menu". By using a pattern rather than the exact text, the verifyTitle will pass as long as the two words "Film" and "Television" appear (in that order) anywhere in the page's title.

Regular Expression Patterns

PATTERN	MATCH
.	any single character
[ ]	character class: any single character that appears inside the brackets
*	quantifier: 0 or more of the preceding character (or group)
+	quantifier: 1 or more of the preceding character (or group)
?	quantifier: 0 or 1 of the preceding character (or group)
{1,5}	quantifier: 1 through 5 of the preceding character (or group)
|	alternation: the character/group on the left or the character/group on the right
( )	grouping: often used with alternation and/or quantifier

Command	Target	Value
click	link=regexp:Film.*Television Department	
verifyTitle	regexp:.*Film.*Television.*	

Command	Target	Value
open	http://weather.yahoo.com/forecast/USAK0012.html	
verifyTextPresent	regexp:Sunrise: *[0-9]{1,2}:[0-9]{2} [ap]m	

Sunrise: *	The string Sunrise: followed by 0 or more spaces
[0-9]{1,2}	1 or 2 digits (for the hour of the day)
:	The character : (no special characters involved)
[0-9]{2}	2 digits (for the minutes) followed by a space
[ap]m	"a" or "p" followed by "m" (am or pm)

Exact Patterns

select	**//select**	**exact:Real ***

select	**//select**	**regexp:Real \***

It uses no special characters at all. So, if you needed to look for an actual asterisk character (which is special for both globbing and regular expression patterns), the **exact** pattern would be one way to do that.

Store Commands and Selenium Variables

Command	Target	Value
store	paul@mysite.org	userName

Command	Target	Value
verifyText	//div/p	${userName}

Command	Target	Value
type	id=login	${userName}

A little bit more about store

- **storeElementPresent**

This corresponds to verifyElementPresent. It simply stores a boolean value–"true" or "false"–depending on whether the UI element is found.

- **storeText**

StoreText corresponds to verifyText. It uses a locater to identify specific page text. The text, if found, is stored in the variable. StoreText can be used to extract text from the page being tested.

echo - The Selenese Print Command

- Selenese has a simple command that allows you to print text to your test's output. This is useful for providing informational progress notes in your test which display on the console as your test is running.

Command	Target	Value
echo	Testing page footer now.	
echo	Username is ${userName}	

What to test?

The simplest type of test, a *content test*, is a simple test for the existence of a static, non-changing, UI element. For instance

- Does each page have its expected page title? This can be used to verify your test found an expected page after following a link.
- Does the application's home page contain an image expected to be at the top of the page?
- Does each page of the website contain a footer area with links to the company contact page, privacy policy, and trademarks information?
- Does each page begin with heading text using the <h1> tag? And, does each page have the correct text within that header?

Verify

Command	Target	Value
verifyTextPresent	Marketing Analysis	

Command	Target	Value
verifyElementPresent	//div/p	
verifyElementPresent	id=Login	
verifyElementPresent	link=Go to Marketing Research	

Command	Target	Value
verifyText	//table/tr/td/div/p	This is my text and it occurs right after the div inside the table.

Alt : // ~ css=

Task 2

- Open http://seleniumhq.org
- Goto link http://seleniumhq.org/download/
- Assure that Title of the new page is Downloads, when not – stop testing
- Verify that text Download is present on the page and is used with h2 tag
- Assure that in table language Ruby is present on the page, when not – stop testing
- Verify that version 2.25.0 is written in the table

Tips and Tricks

- Do I only check that the text exists on the page? (*verify/assertTextPresent*)
- Do I only check that the HTML element exists on the page? That is, the text, image, or other content is not to be checked, only the HTML tag is what is relevant. (*verify/assertElementPresent*)
- Must I test both, the element and it's text content? (*verify/assertText*)

Task 3

- Open http://www.google.com.ua/
- Search for web application testing
- Goto Selenium site
- Save PageTitle of Selenium page into variable – (store)
- Print out the value of this variable (echo)

Task 4

- Go to http://demo.opensourcecms.com/wordpress/wp-login.php
- Enter “admin” in the “Username” field
- Enter “demo123" in the “Password” field
- Click on the “Log In” button
- Verify that the text “Howdy, admin” is present
- Click on the “Posts” link
- Click on the “Add New” button
- Type “Selenium Demo Post” in the title field
- Click on the “Publish” button
- Verify that the text “Post published” is present

Location Strategy

- the element ID
- the element name attribute
- an Xpath statement
- document object model (DOM)

Generally, using an ID locator is more efficient as it makes your test code more readable. Using the name attribute also has similar advantages. These also give the best performance. Xpath statements have been known to be slow in Internet Explorer due to limations of IE's Xpath processor.

Sometimes you must use an Xpath or DOM locator , i.e. if the page source does not have an ID or name attribute.

There is an advantage to using Xpath or DOM, you can locate an object with respect to another object on the page.

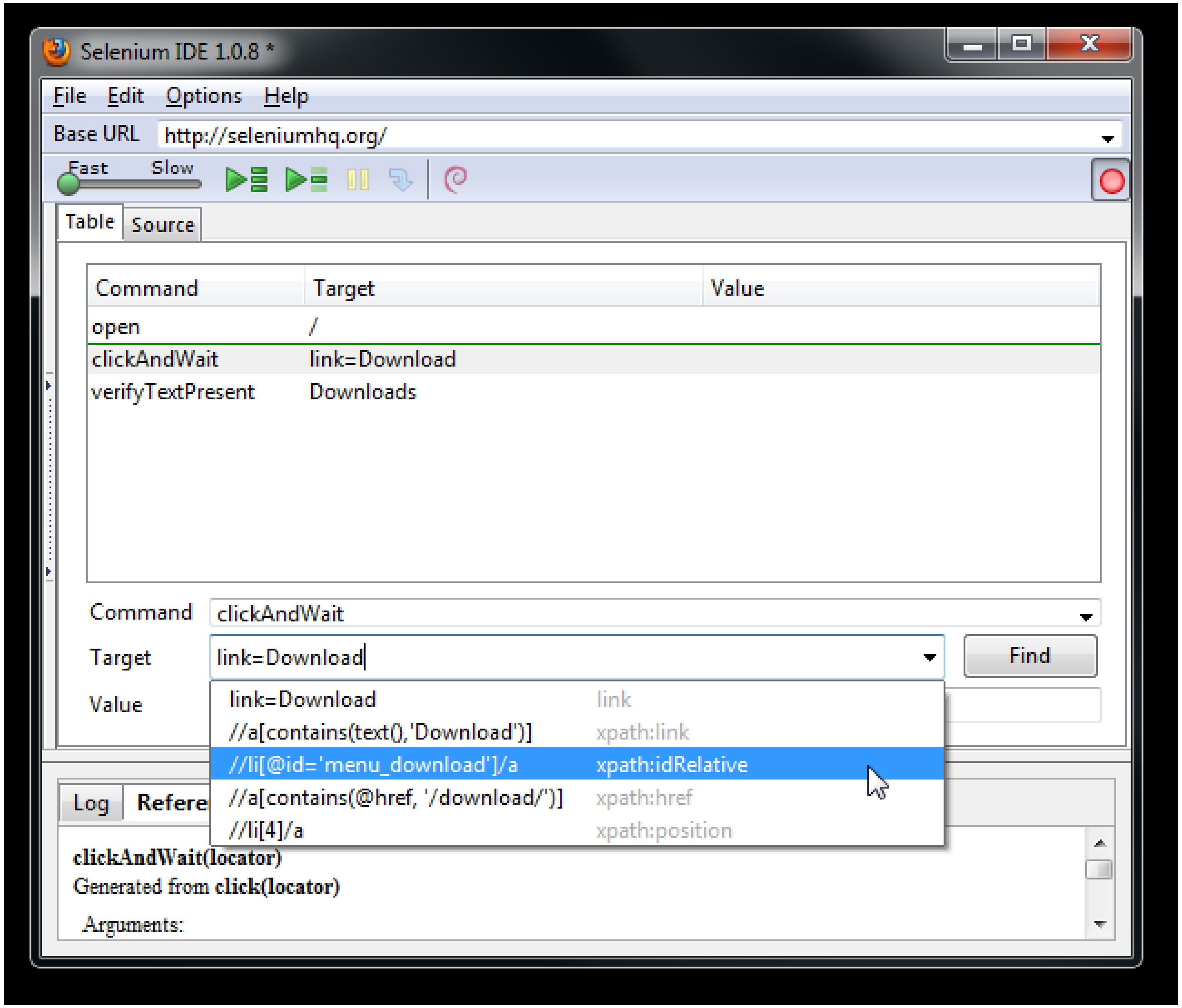
For example, if there is a link that must occur within the second paragraph within a <div> section, you can use Xpath or DOM to specify this. With ID and name locators, you can only specify that they occur on the page–somewhere on the page. If you must test that an image displaying the company logo appears at the top of the page within a header section Xpath may be the better locator.

Find Button

The Find button is used to see which UI element on the currently displayed webpage (in the browser) is used in the currently selected Selenium command. This is useful when building a locator for a command's first parameter (see the section on *locators* in the Selenium Commands chapter). It can be used with any command that identifies a UI element on a webpage, i.e. *click*, *clickAndWait*, *type*, and certain *assert* and *verify* commands, among others.

From Table view, select any command that has a locator parameter. Click the Find button. Now look on the webpage: There should be a bright green rectangle enclosing the element specified by the locator parameter.

Locator Assistance



Base URL http://seleniumhq.org/

Command	Target	Value
open	/	
clickAndWait	link=Download	
verifyTextPresent	Downloads	

Command: clickAndWait
Target: link=Download
Value:

```
link=Download                        link
//a[contains(text(),'Download')]     xpath:link
//li[@id='menu_download']/a          xpath:idRelative
//a[contains(@href, '/download/')]   xpath:href
//li[4]/a                            xpath:position
```

clickAndWait(locator)
Generated from **click(locator)**

Arguments:

Selenium and Javascript

- JavaScript can also be used to help generate values for parameters, even when the parameter is not specified to be of type **script**.
- Special syntax is required–the JavaScript snippet must be enclosed inside curly braces and preceded by the label javascript, as in javascript {*yourCodeHere*}.

Command	Target	Value
store	league of nations	searchString
type	q	javascript{storedVars['searchString'].toUpperCase()}

Task 5

- Goto http://www.theautomatedtester.co.uk/tutorials/selenium/Selenium_JavaScript.htm
- Enter numbers into each of the text boxes and then click on one of the buttons
- Compare it against what we think the answer is going to be. For this we need to use the verifyEval or assertEval commands. Your next step in the script should look a lot like the following: verifyEval | javascript{1+1} | ${total}

TBC…

See you next week!