DZone Refcardz

CONTENTS INCLUDE:



# Getting Started with Selenium

By Frank Cohen

## ABOUT SELENIUM

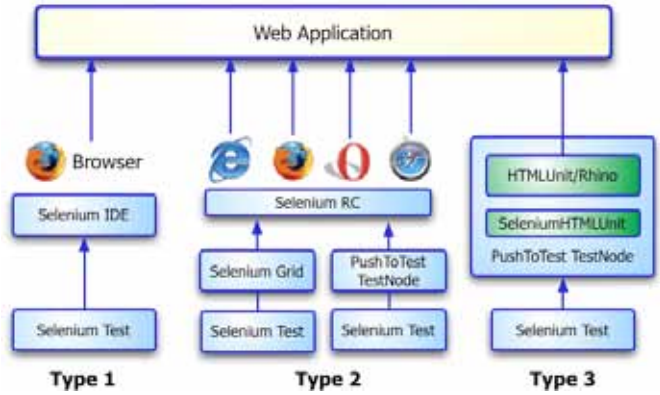
Selenium is a portable software testing framework for Web applications. Selenium works well for QA testers needing record/playback authoring of tests and for software developers needing to author tests in Java, Ruby, Python, PHP, and several other languages using the Selenium API. The Selenium architecture runs tests directly in most modern Web browsers, including MS IE, Firefox, Opera, Safari, and Chrome. Selenium deploys on Windows, Linux, and Macintosh platforms.

Selenium was developed by a team of programmers and testers at ThoughtWorks. Selenium is open source software, released under the Apache 2.0 license and can be downloaded and used without royalty to the originators.

## ARCHITECTURE IN A NUTSHELL

Selenium Browserbot is a JavaScript class that runs within a hidden frame within a browser window. The Browserbot runs your Web application within a sub-frame. The Browserbot receives commands to operate against your Web application, including commands to open a page, type characters into form fields, and click buttons.

Selenium architecture offers several ways to play a test.




Functional testing (Type 1) uses the Selenium IDE add-on to Firefox to record and playback Selenium tests in Firefox. Functional testing (Type 2) uses Selenium Grid to run tests in a farm of browsers and operating environments. For example, run install Selenium Grid on 3 operation environments (for example, Windows Vista, Windows XP, and Ubutu) and on each install 2 browser (for example, Microsoft Internet Explorer and Firefox) to smoke test, integration test, and functional test your application on 6 combinations of operating environment and browser. Many more combinations of operating environment and browser are possible. An option for functional testing (Type 2) is to use the PushToTest TestMaker/TestNode open source project. It uses Selenium RC to provide Selenium Grid-like capability with the added advantage of providing data-driven Selenium tests, results analysis charts and graphs, and better stability of the test operations.

The PushToTest open-source project provides Selenium data-driven testing, load testing, service monitoring, and reporting. TestMaker runs load and performance tests (Type 3) in a PushToTest TestNode using the PushToTest SeleniumHTMLUnit library and HTMLUnit Web browser (and Rhino JavaScript engine.)

**Hot Tip** HTMLUnit runs Selenium tests faster than a real browser and requires much less memory and CPU resources.

## INSTALLING SELENIUM

Selenium IDE installs as a Firefox add-on. Below are the steps to download and install Selenium IDE:

1. Download selenium-ide-1.0.2.xpi (or similar) from http://seleniumhq.org.
2. From Firefox open the .xpi file. Follow the Firefox instructions.
3. Note: Selenium Grid runs as an Ant task. You need JDK 1.6, Ant 1.7, and the Selenium Grid 1.0 binary distribution. Additional directions can be found at http://selenium-grid.seleniumhq.org/get_started.html
4. See http://www.pushtotest.com/products for TestMaker installation instructions.

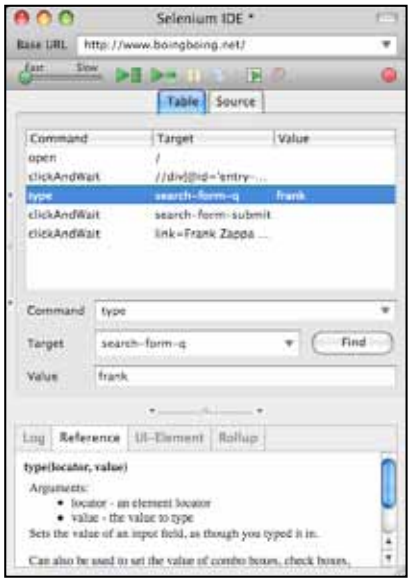
## RECORD/PLAYBACK USING SELENIUM IDE

Hot Tip

Selenium IDE is a Firefox add-on that records clicks, typing, and other actions to make a test, which you can play back in the Firefox browser. Open Selenium IDE from the Firefox Tools drop-down menu, Selenium IDE command.

Selenium IDE records interactions with the Web application, with one command per line. Clicking a recorded command highlights the command, displays a reference page, and displays the command in a command form editor. Click the command form entry down-triangle to see a list of all the Selenium commands.

Run the current test by clicking the Run Test Case icon in the icon bar. Right click a test command to choose the Set Breakpoint command. Selenium IDE runs the test to a breakpoint and then pauses. The icon bar Step icon continues executing the test one command at a time.




With Selenium IDE open, the menu bar context changes to provide access to Selenium commands: Open/Close Test Case and Test Suite. Test Suites contain one or more Test Cases.

Use the Options drop-down menu, Options command to set general preferences for Selenium IDE.

Selenium IDE provides an extensibility API set called User Extensions. You can implement custom functions and modify Selenium IDE behavior by writing JavaScript functions. We do not recommend writing User Extensions as the Selenium project makes no guarantees to be backwardly compatible from one version to the next.

Selenium Context Menu provides quick commands to insert new Selenium commands, evaluate XPath expressions within the live Web page, and to show all available Selenium commands. Right click on commands in Selenium IDE, and right-click on elements in the browser page to view the Selenium Context Menu commands.

## SELENESE TABLE FORMAT

Selenium IDE is meant to be a light-weight record/playback tool to facilitate getting started with Selenium. It is not designed to be a full test development environment. While Selenium records in an HTML table format (named Selenese) the table format only handles simple procedural test use cases. The Selenese table format does not provide operational test data support, conditionals, branching, and looping. For these you must Export Selenese files into Java, Ruby, or other supported languages.

## SELENIUM COMMAND REFERENCE

Selenium comes with commands to: control Selenium test operations, browser and cookie operations, pop-up, button, list, edit field, keyboard, mouse, and form operations. Selenium also provides access operations to examine the Web application (details are at http://release.seleniumhq.org/selenium-core/0.8.0/reference.html).

| Command | Value, Target, Wait Command |
|---|---|
| **Selenium Control** | |
| setTimeout | milliseconds |
| setMouseSpeed | number of pixels |
| | setMouseSpeedAndWait |
| setSpeed | milliseconds |
| | setSpeedAndWait |
| addLocationStrategy | strategyName |
| | addLocationStrategyAndWait |
| allowNativeXpath | boolean |
| | allowNativeXpathAndWait |
| ignoreAttributesWithoutValue | boolean |
| | ignoreAttributesWithoutValueAndWait |
| assignId | locator |
| | assignIdAndWait |
| captureEntirePageScreenShot | filename, kwargs |
| | captureEntirePageScreenShotAndWait |
| echo | message |
| pause | milliseconds |
| runScript | javascript |
| | runScriptAndWait |
| waitForCondition | javascript |
| waitForPageToLoad | milliseconds |
| waitForPopUp | windowID |
| fireEvent | locator |
| | fireEventAndWait |
| Browser Operations | |
| open | url |
| | openAndWait |
| openWindow | url |
| | openWindowAndWait |
| goBack | goBackAndWait |
| refresh | refreshAndWait |
| close | |
| deleteCookie | name |
| | deleteCookieAndWait |

| | |
|---|---|
| deleteAllVisibleCookies | deleteAllVisibleCookiesAndWait |
| setBrowserLogLevel | logLevel |
| | setBrowserLogLevelAndWait |
| Cookie Operations | |
| createCookie | nameValuePair |
| | createCookieAndWait |
| deleteCookie | name |
| | deleteCookieAndWait |
| deleteAllVisibleCookies | deleteAllVisibleCookiesAndWait |
| Popup Box Operations | |
| answerOnNextPrompt | answer |
| | answerOnNextPromptAndWait |
| chooseCancelOnNextConfirmation | chooseCancelOnNextConfirmationAndWait |
| chooseOkOnNextConfirmation | chooseOkOnNextConfirmationAndWait |
| Checkbox & Radio Buttons | |
| check | locator |
| | checkAndWait |
| uncheck | locator |
| | uncheckAndWait |
| Lists & Dropdowns | |
| addSelection | locator |
| | addSelectionAndWait |
| removeSelection | removeSelectionAndWait |
| removeAllSelections | removeAllSelectionsAndWait |
| Edit Fields | |
| type | locator |
| | typeAndWait |
| typeKeys | locator |
| | typeKeysAndWait |
| setCursorPosition | locator |
| | setCursorPositionAndWait |
| Keyboard Operations | |
| keyDown | locator |
| | keyDownAndWait |
| keyPress | locator |
| | keyPressAndWait |
| keyUp | locator |
| | keyUpAndWait |
| altKeyDown | altKeyDownAndWait |
| altKeyUp | altKeyUpAndWait |
| controlKeyDown | controlKeyDownAndWait |
| controlKeyUp | controlKeyUpAndWait |
| metaKeyDown | metaKeyDownAndWait |
| metaKeyUp | metaKeyUpAndWait |
| shiftKeyDown | shiftKeyDownAndWait |
| shiftKeyUp | shiftKeyUpAndWait |
| Mouse Operations | |
| click | locator |
| | clickAndWait |
| clickAt | locator |
| | clickAtAndWait |
| doubleClick | locator |
| | doubleClickAndWait |
| doubleClickAt | locator |
| | doubleClickAtAndWait |
| contextMenu | locator |
| | contextMenuAndWait |
| contextMenuAt | locator |
| | contextMenuAtAndWait |
| mouseDown | locator |
| | mouseDownAndWait |
| mouseDownA | locator |
| | mouseDownAtAndWait |
| mouseMove | locator |
| | mouseMoveAndWait |
| mouseMoveAt | locator |
| | mouseMoveAtAndWait |
| mouseOut | locator |
| | mouseOutAndWait |
| mouseOver | locator |
| | mouseOverAndWait |
| mouseUp | locator |
| | mouseUpAndWait |
| mouseUpAt | locator |
| | mouseUpAtAndWait |
| dragAndDrop | locator |
| | dragAndDropAndWait |
| dragAndDropToObject | sourceLocator |
| | dragAndDropToObjectAndWait |
| Form Operations | |
| submit | formLocator |
| | submitAndWait |
| Windows/Element Selection | |
| select | locator |
| | selectAndWait |
| selectFrame | locator |
| selectWindow | windowID |
| focus | locator |
| | focusAndWait |
| highlight | locator |
| | highlightAndWait |
| windowFocus | windowFocusAndWait |
| windowMaximize | windowMaximizeAndWait |

## SELENESE TABLE FORMAT

Selenium commands identify elements within a Web page using:

| | |
|---|---|
| identifier=id | Select the element with the specified @id attribute. If no match is found, select the first element whose @name attribute is id. |
| name=name | Select the first element with the specified @name attribute. The name may optionally be followed by one or more element-filters, separated from the name by whitespace. If the filterType is not specified, value is assumed. For example, name=style value=carol |
| dom=javascriptExpression | Find an element using JavaScript traversal of the HTML |

| | |
|---|---|
| dom=javascriptExpression (continued) | Document Object Model. DOM locators must begin with "document." For example:<br>`dom=document.forms['form1'].myList`<br>`dom=document.images[1]` |
| xpath=xpathExpression | Locate an element using an XPath expression. Here are a few examples:<br>`xpath=//img[@alt='The image alt text']`<br>`xpath=//table[@id='table1']//tr[4]/td[2]`<br>`/html/body/table/tr/td/a`<br>`//div[@id='manage_messages_iterator']`<br>`//tr[@class='SelectedRow']/td[2]`<br>`//td[child::text()='myemail@me.com']`<br>`//td[contains(child::text(),'@')]` |
| link=textPattern | Select the link (anchor) element which contains text matching the specified pattern. |
| css=cssSelectorSyntax | Select the element using css selectors. For example:<br>`css=a[href="#id1"]`<br>`css=span#firstChild + span`<br>Selenium 1.0 css selector locator supports all css1, css2 and css3 selectors except namespace in css3, some pseudo classes(:nth-of-type, :nth-last-of-type, :first-of-type, :last-of-type, :only-of-type, :visited, :hover, :active, :focus, :indeterminate) and pseudo elements(::first-line, ::first-letter, ::selection, ::before, ::after). Without an explicit locator prefix, Selenium uses the following default strategies:<br>dom, for locators starting with "document."<br>xpath, for locators starting with "//"<br>identifier, otherwise |

Your choice of element locator type has an impact on the test playback performance. The following table compares performance of Selenium element locators using Firefox 3 and Internet Explorer 7.

| Locator used | Type | Firefox 3 | Internet Explorer 7 |
|---|---|---|---|
| q | Locator | 47 ms | 798 ms |
| //input[@name='q'] | XPath | 32 ms | 563 ms |
| //html[1]/body[1]//form[1]//input[2] | XPath | 47 ms | 859 ms |
| //input[2] | XPath | 31 ms | 564 ms |
| document.forms[0].elements[1] | DOM Index | 31 ms | 125 ms |

Additional details on Selenium performance can be found at: http://www.pushtotest.com/docs/thecohenblog/symposium

## SCRIPT-DRIVEN TESTING

Selenium implements a domain specific language (DSL) for testing. Some applications do not lend themselves to record/playback: 1) The test flow changes depending on the results of a step in the test, 2) The input data changes depending on the state of the application, and 3) The test requires asynchronously operating test flows. For these conditions, consider using the Selenium DSL in a script driven test. Selenium provides support for Java, Python, Ruby, Groovy, PHP, and C#.

Selenium IDE helps get a script-driven test started by exporting to a unit test format. For example, consider the following test in the Selenese table format:

| franktest | | |
|---|---|---|
| open | / | |
| type | q | sock puppet |
| clickAndWait | sa | |
| click | //div[@id='res']/div[1]/ol/li[1]/div/h2/a/em | |
| clickAndWait | //div[@id='res']/div[1]/ol/li[1]/div/h2/a/em | |

Use the Selenium IDE File menu, Export, Python Selenium RC command to export the test to a jUnit-style TestCase written in Python. The following shows the Java source code:

```
package com.example.tests;

from selenium import selenium
import unittest, time, re

class franktest(unittest.TestCase):
    def setUp(self):
        self.verificationErrors = []
        self.selenium = selenium("localhost", 4444, "*chrome", \
           "http://change-this-to-the-site-you-are-testing/")
        self.selenium.start()
    def test_franktest(self):
        sel = self.selenium
        sel.open("/")
        sel.type("q", "sock puppet")
        sel.click("sa")
        sel.wait_for_page_to_load("30000")
        sel.click("//div[@id='res']/div[1]/ol/li[1]/div/h2/a/em")
        sel.click("//div[@id='res']/div[1]/ol/li[1]/div/h2/a/em")
        sel.wait_for_page_to_load("30000")

    def tearDown(self):
        self.selenium.stop()
        self.assertEqual([], self.verificationErrors)

if __name__ == "__main__":
    unittest.main()
```

An exported test like the one above has access to all of Python's functions, including conditionals, looping and branching, reusable object libraries, inheritance, collections, and dynamically typed data formats.

Selenium provides a Selenium RC client package for Java, Python, C#, Ruby, Groovy, PHP, and Perl. The client object identifies the Selenium RC service in its constructor:

```
        self.selenium = selenium("localhost", 4444, "*iexplore", \
           "http://change-this-to-the-site-you-are-testing/")
        self.selenium.start()
```

The above code identifies the Selenium RC service running on the localhost machine at port 4444. This client will run the test in Microsoft Internet Explorer. The third parameter identifies the base URL from which the recorded test will operate.


Using the selenium.start() command initializes and starts the Selenium RC service. The Selenium RC client module (import selenium in Python) provides methods to operate the Selenium DSL commands (click, type, etc.) in the Browserbot running in the browser. For example, selenium.click("open") tells the Browserbot to a click command to the element with an id tag equal to "open". The browser responds to the click command and communicates with the Web application.

At the end of the test the selenium.stop() command ends the Selenium RC service.

## SELENIUM AND AJAX

Ajax uses asynchronous JavaScript functions to manipulate the browser's DOM representation of the Web page. Many Selenium commands are not compatible with Ajax. For example, ClickAndWait will time-out waiting for the browser

to load the Web page because Ajax functions that manipulate the current Web page in response to a click event do not reload the page. We recommend using Selenium commands that poll the DOM until the Ajax methods complete their tasks. For example, waitUntilElementPresent polls the DOM until the JavaScript function adds the desired element to the page before continuing with the rest of the Selenium script.

Consider the following checklist when using Selenium with Ajax applications:

| | |
|---|---|
| ✓ | Your Selenium tests may require a large number of extra commands to ensure the test stays in synchronization with the Ajax application. Consider an Ajax application that requires a log-in, then displays a selection list of items, then presents an order form. Ajax enabled applications often deliver multiple steps of function on a single page and show-and-hide elements as you work with the application. Some even disable form submit buttons and other user interface elements until you enter enough valid information. For an application like this you will need a combination of Selenium commands. Consider the following Selenium test:<br><br>waitForElementPresent pauses the test until the Ajax application adds the requisite element to the page. waitForCondition pauses the test until the JavaScript function evaluates to true. |
| ✓ | Some Ajax applications use lazy-loading techniques to improve user interaction with the application. A stock market application provides a list of 10 stock quotes asynchronously after the user clicks the submit button. The list may take 10 to 50 seconds to completly update on the screen. Using waitForXPathCount pauses the test until the page contains the number of nodes that match the specified XPath expression. |
| ✓ | Many Ajax applications use dynamic element id tags. The Ajax application that named the Log-out button app_6 may later rename the button to app_182. We recommend using DOM element locator techniques, or XPath techniques if needed, to dynamically find elements on a positional or other attribute means. |

| Command | Target | Value |
|---|---|---|
| open | http://localhost:8080/calendar/ | |
| waitForElementPres... | app_6 | |
| click | app_6 | |
| waitForElementPres... | //input[@value='Cancel'] | |
| waitForElementPres... | //input[@id='event'] | |
| type | event | FlyToMiami |
| type | event_date | 6/12/2008 |
| select | event_hour | label=8 |
| waitForCondition | selenium.browserbot.getCurrentWindow().document.getElementById... | 10000 |
| click | //input[@value='Save Event'] | |

## WORKING WITH TINYMCE AND AJAX OBJECTS

Ajax is about moving functions off the server and into the browser. Selenium architecture supports innovative new browser-based functions because Selenium's Browserbot is a JavaScript class itself. The Browserbot even lets Selenium tests operate JavaScript functions as part of the test. For example, TinyMCE (http://tinymce.moxiecode.com) is a graphical text editor component for embedding in Web pages. TinyMCE supports styled text and what-you-see-is-what-you-get editing. Testing a TinyMCE can be challenging. Selenium offers click and type functions that interact with TinyMCE but no direct commands for TinyMCE's more advanced functions. For example, imagine testing TinyMCE's ability to stylize text. The test needs to insert test, move the insertion point, select a sentence, bold the text, and drag the sentence to another paragraph. This is beyond Selenium's DSL. Instead, the Selenium test may include JavaScript commands that interact with TinyMCE's published API (http://tinymce.moxiecode.com/documentation.php).

Here is an example of using the TinyMCE API from a Selenium test context:

```
this.browserbot.getCurrentWindow().tinyMCE.execCommand
('mceInsertContent',false,'<b>Hello world!!</b>');
```

Run the above JavaScript function from within a Selenium test using the AssertEval command.

```
AssertEval javascript:this.browserbot.getCurrentWindow().tinyMCE.
execCommand
  ('mceInsertContent',false,'<b>Hello world!!</b>');
```

## DATA PRODUCTION

Selenium offers no operational test data production capability itself. For example, a Selenium test of a sign-in page usually needs sign-in name and sign-in password operational test data to operate. Two options are available: 1) Use the data access features in Java, Ruby, or one of other supported languages, 2) Use PushToTest TestMaker's Selenium Script Runner to inject data from comma separated value (CSV) files, relational databases, objects, and Web services.
See http://tinyurl.com/btxvn4 for details.

Create a Comma-Separated-Value file. Use your favorite text editor or spreadsheet program. Name the file data.csv. The contents must be in the following form.

| | A | B | C | D |
|---|---|---|---|---|
| 1 | EventName | EventDate | ConfirmName | |
| 2 | BrushTeeth | 8/21/08 | BrushTeeth | |
| 3 | CleanRoom | 8/21/08 | CleanRoom | |
| 4 | Trash | 8/21/08 | Trash | |
| 5 | FeedDog | 8/21/08 | FeedDog | |
| 6 | KissHusband | 8/21/08 | KissHusband | |
| 7 | | | | |

The first row of the data file contains column names. These will be used to map values into the Selenium test. Change the Selenium test to refer to mapping name. PushToTest maps the data from the named column in the CSV data file to the Selenium test data using the first row definitions.

Connect the Data Production Library (DPL) to the Selenium test in a TestMaker TestScenario. Begin by definition a HashDPL. This DPL reads from CSV data files and provides the data to the test.

```
<DataSources>
   <dpl name="mydpl" type="HashDPL">
      <argument name="file" dpl="rsc" value="getDataByIndex" index="0"/>
   </dpl>
</DataSources>
```

Next, tell the TestScenario to send the data.csv and Selenium test files to the TestNodes that will operate the test.

```
  <resources>
    <data path="data.csv"/>
    <selenese path="CalendarTest.selenium"/>
  </resources>
```

Then tell the Selenium ScriptRunner to use the DPL provided data when running the Selenium test.

```
<run name="CalendarTest" testclass="CalendarTest.selenium"
        method="runSeleneseFile" langtype="selenium">
        <argument dpl="mydpl" name="DPL_Properties"
         value="getNextData"/>
</run>
```

The getNextData operation gets the next row of data from the CSV file. The Selenium ScriptRunner injexts the data into the Selenium test.

## BROWSER SANDBOX, REDIRECT, AND PROXY ISSUES

Selenium RC launches the browser with itself as the proxy server to inject the Javascript of the Browserbot and your test. This architecture makes it possible to run the same test on multiple browsers. However, some browsers will warn the user of possible security threats when the proxy starts and when the test requests functions or pages outside of the originating domain. The browser takes control and stops the Browserbot operations to display the warning message. When this happens, the test stops until a user dismisses the warning. There are no reliable cross-browser workarounds.

Some Web applications redirect from http to https URLs. The browser will often issue a warning that stops the Selenium test.

Selnium does not support a test moving across domains. For example, a test that started with a baseurl of www.mydomain.com may not open a page on www.secondomain.com.

## SELENIUM RC BROWSER PROFILES

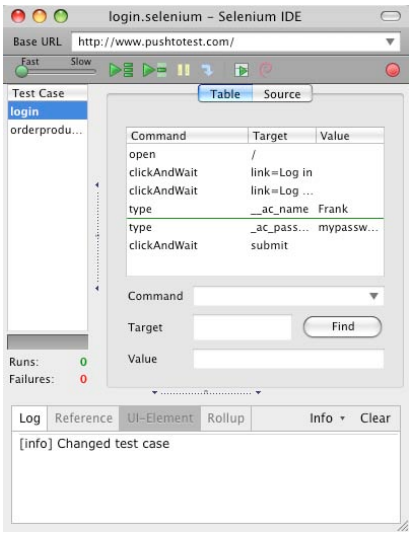
Selenium Remote Control (RC) enables test operation on multiple real browsers. A browser profile attribute may be any of the following installed browsers: chrome, konqueror, piiexplore, iehta, mock, opera, pifirefox, safari, iexplore and custom. Append the path to the real browser after browser profile if your system path does not state the path to the browser. For example:

*firefox /Applications/Firefox.app/Contents/MacOS/firefox

## COMPONENT APPROACH EXAMPLE

Many organizations pursue a "Test and Trash" methodology to achieve agile software development lifecycles. For example, an organization in pursuit of agile techniques may change up to 30% of an application with an application lifecycle of 8 weeks. Without giving the change much thought, up to 30% of their recorded tests break!

We recommend a component approach to building tests. Test components perform specific test operations. We write or record tests as individuals components of test function. For example, a component operates the sign-in function of a private Web application. When the sign-in portion of the application changes, we only need to change the sign-in test and the rest of test continues to perform normally.




Selenium supports the component approach in three ways: Selenium IDE supports Test Suites and Test Cases, exporting Selenium tests to dynamic languages (Java, Ruby, Perl, etc.) creates reusable software classes, and 3) PushToTest TestMaker supports multiple use cases with parameterized test use cases.

In Selenium IDE, the File menu enables tests to be saved as test cases or test suites. Record a test, use File -> Save Test Case. Create a second Test Case by choosing File -> New Test Case. Record the second test use case. Save the TestSuite for these two test use cases by choosing File -> Save TestSuite. Click the "Run entire test suite" icon from the Selenium IDE tool bar.

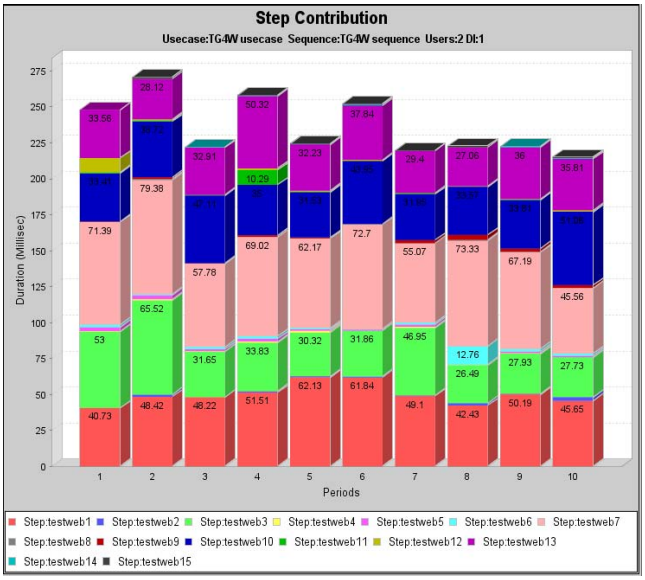
TestMaker defines test use cases using a simple XML notation:

```
<usecases>
  <usecase name="MailerCheck_usecase">
      <test>

        <run name="LogIn" testclass="Login.selenium"
         instance="myinst"

          method="runSeleneseFile" langtype="selenium">

        </run>


        <run name="OrderProduct" testclass="OrderProduct.
         selenium" instance="myinst"

          method="runSeleneseFile" langtype="selenium">

        </run>


      </test>
  </usecase>
</usecases>
```

## REPORTING OPTIONS

Selenium offers no results reporting capability of its own. Two options are available: 1) Write your tests as a set of JUnit tests and use JUnit Report (http://ant.apache.org/manual/OptionalTasks/junitreport.html) to plot success/failure charts, 2) Use PushToTest TestMaker Results Analysis Engine to produce more than 300 charts from the transaction and step time tracking of Selenium tests.

For example, TestMaker tracks Selenium command duration in a test suite or test case. Consider the following chart. This shows the "Step" time it takes to process each Selenium command in a test use case over 10 equal periods of time that the test took to operate.



## SELENIUM BIOSPHERE

Test Maker allows repurposing Selenium tests as load test service monitors. http://www.pushtotest.com

BrowserMob facilitates low-cost Selenium load testing. http://browsermob.com/load-testing

SauceLabs provides a farm of Selenium RC servers for testing. http://saucelabs.com/

ThoughtWorks Twist can be used for test authoring and management. http://studios.thoughtworks.com/twist-agile-test-automation

Running a Selenium test as a functional test in TestMaker. TestMaker displays the success/failure of each command in the test and the duration in milliseconds of each step.

## THE FUTURE, SELENIUM 2.0 (AKA WEBDRIVER)

The Selenium Project started the WebDriver project, to be delivered as Selenium 2.0. WebDriver is a new architecture that plays Selenium tests by driving the browser through its native interface. This solves the test playback stability issue in Selenium 1.0 but requires the Selenium project to maintain individual API drivers for all the supported browsers. While there is no release date for Selenium 2.0, the WebDriver code is already functional and available for download at http://code.google.com/p/webdriver.

## AVAILABLE TRAINING

SkillsMatter.com, Think88com, PushToTest.com, RTTSWeb.com, and Scott Bellware (http:blog.scottbellware.com) offer training courses fro Selenium. PushToTest offers free Open Source Test Workshops (http://workshop.pushtotest.com) as a meet-up for Selenium and other Open Source Test tool users.

## ABOUT THE NAME SELENIUM

Selenium lore has it that the originators chose the name of Selenium after learning that Selenium is the antidote to Mercury poisoning. There appears to be no love between the Selenium team and HP Mercury, but perhaps a bit of envy

## ABOUT THE AUTHOR

**Frank Cohen** is, Founder of PushToTest, Author of FastSOA. Through his speaking, writing and consulting, Frank Cohen, is the expert that information technology professionals and enterprises go to when they need to understand and solve problems in complex interoperating information systems, especially Service Oriented Architecture (SOA), Ajax, and Web services. PushToTest is the open-source test automation solutions business, and maintainer of the popular TestMaker open-source project

**Website:** www.pushtotest.com

**The Cohen Blog:** www.pushtotest.com/docs/thecohenblog

## RECOMMENDED BOOK


As a Java developer, you want a guide that shows you how to add Ajax functionality to your web applications with a minimum of effort. Well look no further than Pro Ajax and Java Frameworks. In this book, recognized Java experts and authors of the best-selling Apress title, Foundations of Ajax, will show you how.

**BUY NOW**
books.dzone.com/books/pro-ajax-java



DZone

DZone communities deliver over 6 million pages each month to more than 3.3 million software developers, architects and decision makers. DZone offers something for everyone, including news, tutorials, cheatsheets, blogs, feature articles, source code and more. **"DZone is a developer's dream,"** says PC Magazine.

DZone, Inc.
1251 NW Maynard
Cary, NC 27513

888.678.0399
919.678.0300

**Refcardz Feedback Welcome**
refcardz@dzone.com

**Sponsorship Opportunities**
sales@dzone.com

ISBN-13: 978-1-934238-62-2
ISBN-10: 1-934238-62-7
50795
$7.95

Version 1.0