QA Division

Selenium IDE Installation Guidelines

Last Update	2009-01-30
Version	2.0

Detailed Document Change History

Version	Date	Author	Description
1.0	2008-10-24	Moises Siles	Initial Draft
2.0	2009-01-20	Johan Muir	Proofreading. Added General Script Example Section
2.0	2009-01-30	Adrián Rojas	Added StoreGlobal, GoToIf functions, FAQ and Annex section
2.0	2009-02-04	Adrián Rojas	Update FAQ section

Table of Contents

1 Introduction

1.1 Purpose

The purpose of this document is to describe, step by step, the installation and usage processes of the automation tool Selenium IDE.

1.2 Audience

This document is intended for VeraCarta testers and QA personnel that are either not familiar or new to Selenium.

1.3 Scope

This document focuses basically on the steps to be followed in order to install and run the application. It is not intended to be an exhaustive user's guide to Selenium IDE.

1.4 Related Documents

Ref.	Document

1.5 Summary

This document explains the general concepts and steps to be followed in order to install and run Selenium software. In addition, it will describe the capture process from Selenium IDE.

1.6 Terminology / Glossary

Term	Definition

2 Selenium IDE Installation Guidelines

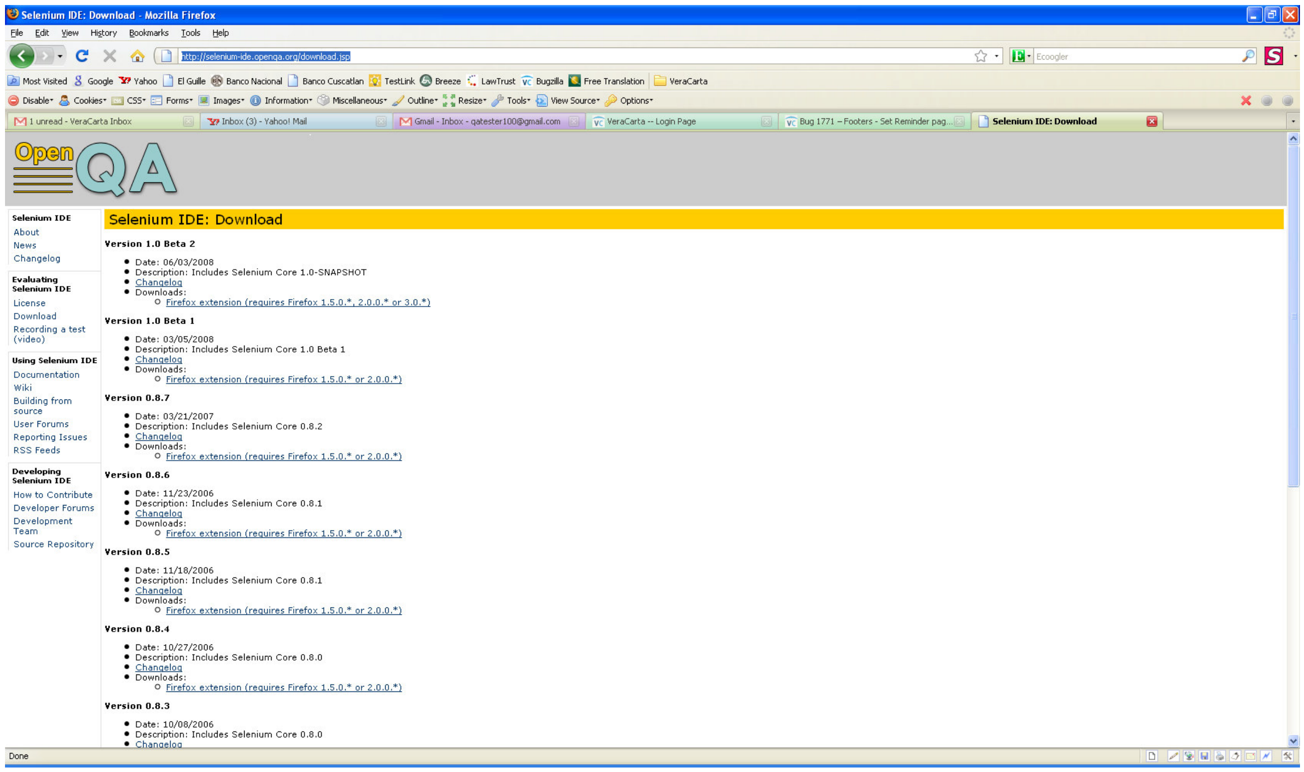
The following steps will describe the procedures to be used in order to install and run Selenium IDE.

Selenium IDE is an integrated development environment for Selenium tests. It is implemented as a Firefox extension, and allows you to record, edit, and debug tests. Selenium IDE includes the entire Selenium Core, allowing you to easily and quickly record and play back tests in the actual environment that they will run.

Selenium IDE is not only a recording tool: **it is a complete IDE**. You can choose to use its recording capability, or you may edit your scripts by hand. With autocomplete support and the ability to move commands around quickly, Selenium IDE is the ideal environment for creating Selenium tests no matter what style of tests you prefer.

2.1 Downloading the Firefox extension

Installation is very easy, simply go to the Firefox extensions page https://addons.mozilla.org/en-US/firefox/addon/2079 and add the extension to Firefox. Yes, the Selenium IDE only works with Firefox, although you can run the tests generated by Selenium IDE in other browsers using the TestRunner that comes with Selenium Core http://wiki.openqa.org/display/SEL/Getting+Started+with+Selenium+Core.


Selenium IDE: Download - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://selenium-ide.openqa.org/download.jsp
Most Visited Google Yahoo El Guille Banco Nacional Banco Cuscatlan TestLink Breeze LawTrust Bugzilla Free Translation VeraCarta
Disable Cookies CSS Forms Images Information Miscellaneous Outline Resize Tools View Source Options
1 unread - VeraCarta Inbox
Inbox (3) - Yahoo! Mail
Gmail - Inbox - qatester100@gmail.com
VeraCarta -- Login Page
Bug 1771 – Footers - Set Reminder pag...
Selenium IDE: Download
Open QA
Selenium IDE
About
News
Changelog
Evaluating Selenium IDE
License
Download
Recording a test (video)
Using Selenium IDE
Documentation
Wiki
Building from source
User Forums
Reporting Issues
RSS Feeds
Developing Selenium IDE
How to Contribute
Developer Forums
Development Team
Source Repository
Done

Selenium IDE: Download

Version 1.0 Beta 2

- Date: 06/03/2008
- Description: Includes Selenium Core 1.0-SNAPSHOT
- Changelog
- Downloads:
  - Firefox extension (requires Firefox 1.5.0.*, 2.0.0.* or 3.0.*)

Version 1.0 Beta 1

- Date: 03/05/2008
- Description: Includes Selenium Core 1.0 Beta 1
- Changelog
- Downloads:
  - Firefox extension (requires Firefox 1.5.0.* or 2.0.0.*)

Version 0.8.7

- Date: 03/21/2007
- Description: Includes Selenium Core 0.8.2
- Changelog
- Downloads:
  - Firefox extension (requires Firefox 1.5.0.* or 2.0.0.*)

Version 0.8.6

- Date: 11/23/2006
- Description: Includes Selenium Core 0.8.1
- Changelog
- Downloads:
  - Firefox extension (requires Firefox 1.5.0.* or 2.0.0.*)

Version 0.8.5

- Date: 11/18/2006
- Description: Includes Selenium Core 0.8.1
- Changelog
- Downloads:
  - Firefox extension (requires Firefox 1.5.0.* or 2.0.0.*)

Version 0.8.4

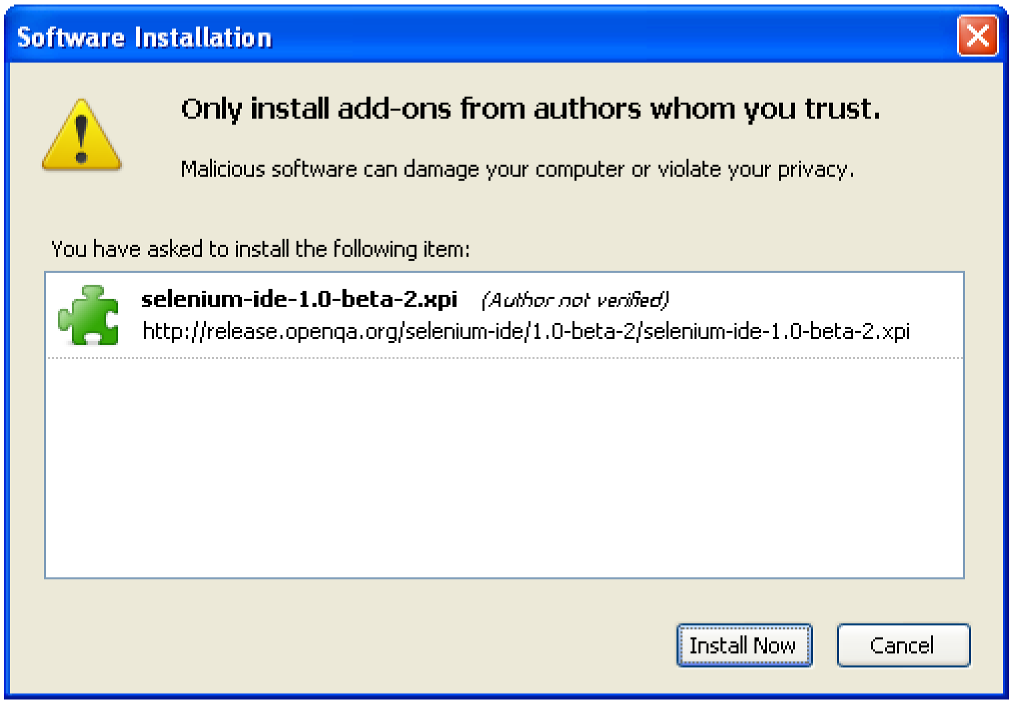
- Date: 10/27/2006
- Description: Includes Selenium Core 0.8.0
- Changelog
- Downloads:
  - Firefox extension (requires Firefox 1.5.0.* or 2.0.0.*)

Version 0.8.3

- Date: 10/08/2006
- Description: Includes Selenium Core 0.8.0
- Changelog

Picture #1

After we clicked on the version that you need, an installation window will be shown. See Picture #2.



Only install add-ons from authors whom you trust.

Malicious software can damage your computer or violate your privacy.

You have asked to install the following item:

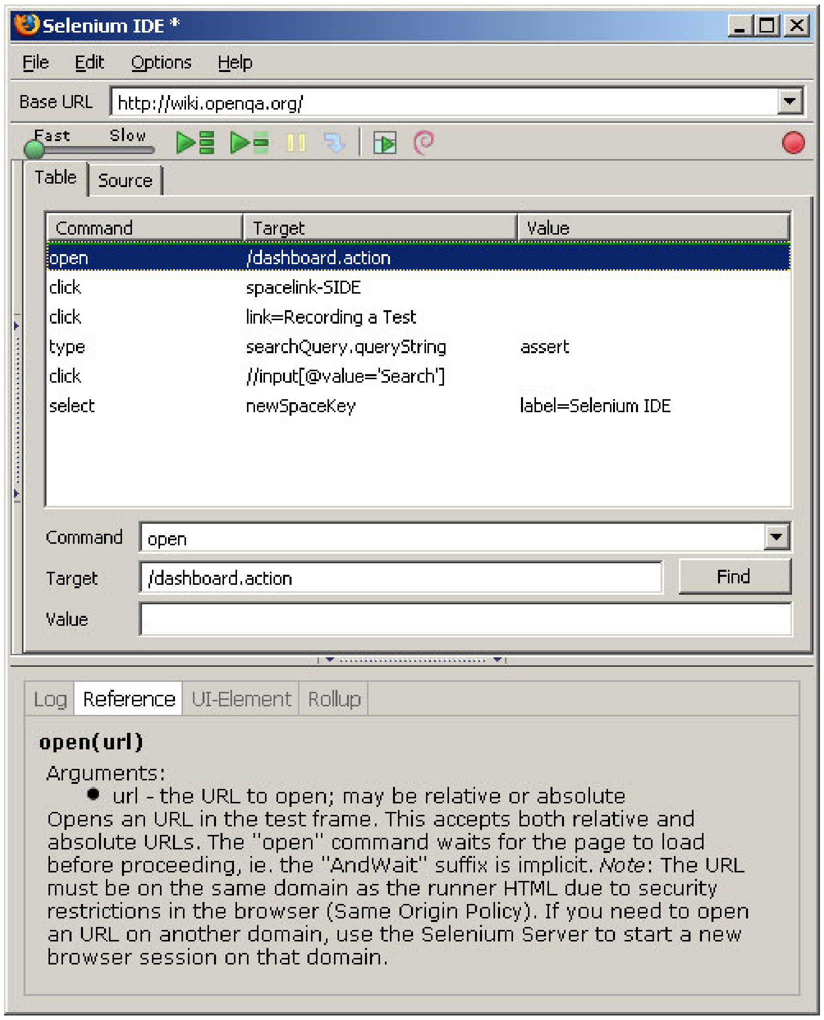
selenium-ide-1.0-beta-2.xpi (Author not verified)
http://release.openqa.org/selenium-ide/1.0-beta-2/selenium-ide-1.0-beta-2.xpi

Picture #2

Then Click on the Install Now button and wait until the installation finish.

2.2 Running the Application

After we install the application, open Firefox and go to Tools -> Selenium IDE and click on it, you will see the following window.



Command	Target	Value
open	/dashboard.action	
click	spacelink-SIDE	
click	link=Recording a Test	
type	searchQuery.queryString	assert
click	//input[@value='Search']	
select	newSpaceKey	label=Selenium IDE

Command: open
Target: /dashboard.action
Value:

open(url)
Arguments:
- url - the URL to open; may be relative or absolute

Opens an URL in the test frame. This accepts both relative and absolute URLs. The "open" command waits for the page to load before proceeding, ie. the "AndWait" suffix is implicit. *Note*: The URL must be on the same domain as the runner HTML due to security restrictions in the browser (Same Origin Policy). If you need to open an URL on another domain, use the Selenium Server to start a new browser session on that domain.

Picture #3

2.2.1 The "Command" column

The "Command" column represents the command that must be issued to duplicate your action. Following is a list of the most common commands used to build scripts.

- 'open' will open a URL and wait for it to load.
- 'click' will act just as if you had clicked on the link or button yourself.

- 'select' chooses a selection from a drop down list.
- ‘assertText’ Gets the text of an element. This works for any element that contains text.
- ‘clickAndWait’ Clicks on a link, button, checkbox or radio button. If the click action causes a new page to load (like a link usually does), call waitForPageToLoad.
- ‘keyPress’ Simulates a user pressing and releasing a key.
- ‘pause’ Wait for the specified amount of time (in milliseconds)
- ‘storeText’ Gets the text of an element. This works for any element that contains text.
- ‘type’ Sets the value of an input field, as though you typed it in.
- ‘verifyText’ Gets the text of an element. This works for any element that contains text.

2.2.2 The "Target" column

The "Target" column represents what you are trying to click on or interact with. If you look at the sample script above, you will notice a variety of different formats in the "Target" column. These are called "locators" in Selenium IDE, and can be represented a variety of ways:

- URLs - "/dashboard.action" - represents a url. The first part of the URL is the Base URL up at the top, and this URL is relative to that.
- HTML id - "spacelink-SIDE" - this is the HTML id of the link
- Text - "link=Recording a Test" - the text of the link
- HTML name = "searchQuery.queryString" - like the
- XPath - "//input[@value='Search']" - XPath is an XML-like language that can be used to specify the location of an element in a document
- Label - "label-Selenium IDE" - selects the item with the visible label of Selenium IDE.

There are other kinds of identifiers as well, such as HTML Dom, or a CSS-style reference, but are more complicated. You can read more about them here http://release.openqa.org/selenium-core/0.8.0/reference.html.

2.2.3 The "Value" column

The "Value" column represents any other data you might pass on to the command, also known as an argument or a parameter. The contents of the column varies from command to command, but is usually some kind of value - either the text you want to type, or the text you want to compare something to, or a second locator that is required to select an item.

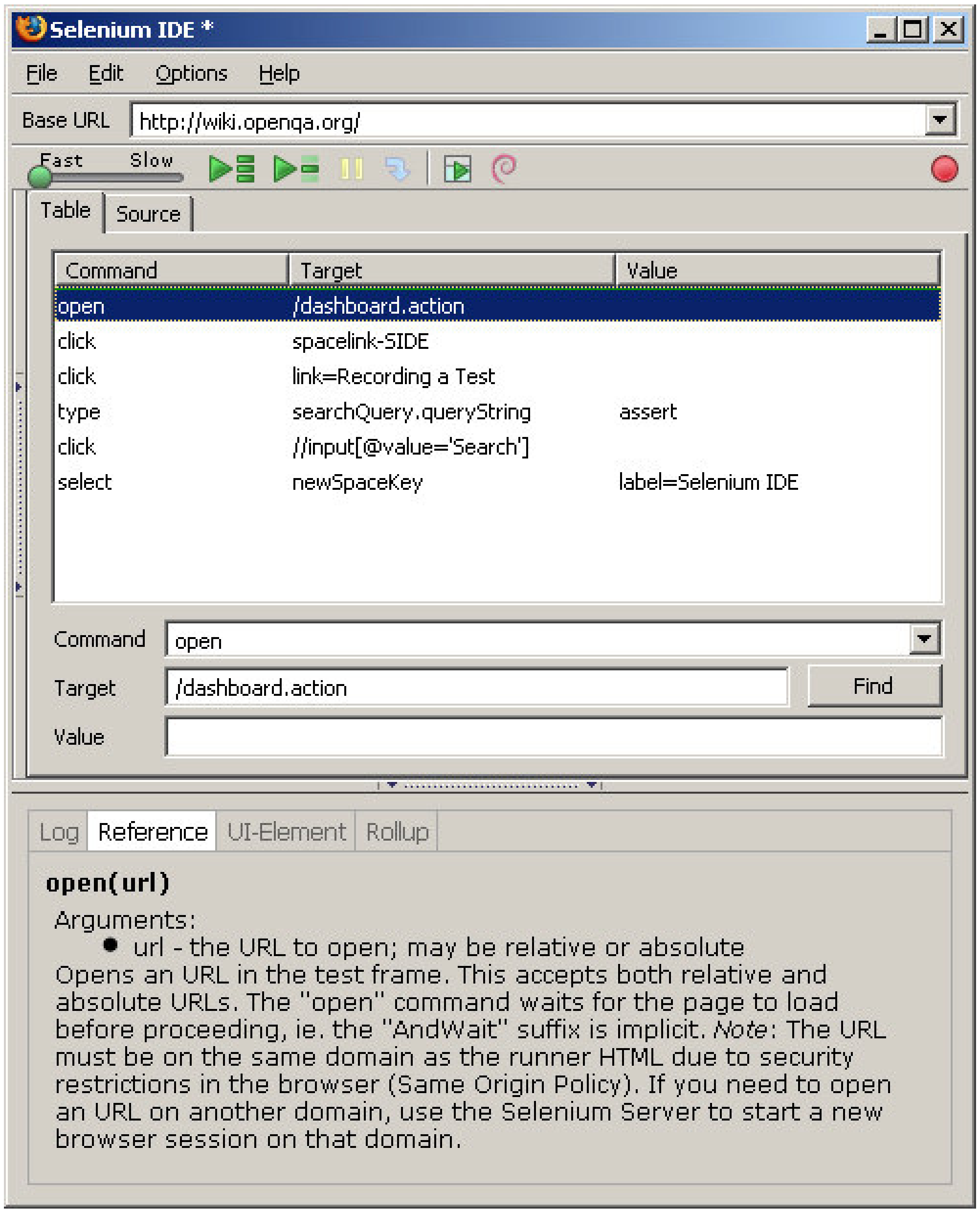
The best way to understand these values is to click on the command you are trying to use, and look at the bottom, on the Reference tab. It will give you information on how the command works, and what all the arguments are.

2.3 Recording and Playing a Script

2.3.1 Recording

1. Browse to a page you'd like to test. Perhaps: http://wiki.openqa.org/
2. Open the IDE in Firefox (Tools -> Selenium IDE).
3. If it's not already activated, click on the red record button on the right hand side of the IDE.
4. Click through several links, and enter a search term.
5. Submit the search, and limit it by the Location drop down.

You will see the commands representing your actions getting added to the table as you go. The commands should look something like this:



Base URL: http://wiki.openqa.org/

Command	Target	Value
open	/dashboard.action	
click	spacelink-SIDE	
click	link=Recording a Test	
type	searchQuery.queryString	assert
click	//input[@value='Search']	
select	newSpaceKey	label=Selenium IDE

Command: open
Target: /dashboard.action
Value:

open(url)

Arguments:

- url - the URL to open; may be relative or absolute

Opens an URL in the test frame. This accepts both relative and absolute URLs. The "open" command waits for the page to load before proceeding, ie. the "AndWait" suffix is implicit. *Note*: The URL must be on the same domain as the runner HTML due to security restrictions in the browser (Same Origin Policy). If you need to open an URL on another domain, use the Selenium Server to start a new browser session on that domain.

Picture #4

To edit a command:

1. Click on the row with the command you want to edit in the table
2. Look below it where there are input fields that you can type into
3. By Command, click beside the "click" text, and start typing "AndWait"
4. You will notice a drop down, to help you select which command to call.
5. Choose or finish typing "clickAndWait"

2.3.2 Run the Script

1. Click on the red record button to STOP recording
2. Make sure you have your test browser window selected (so you don't navigate away from this tutorial!)
3. Click on the "Play current test case" button - the green arrow with one highlighted line by it.

2.3.3 Check your data

Here's where it gets a little more interesting, since there's a million functions to sort through and not all of them are just click and go. Let's add a couple of checks to the script, so we can see how it works:

1. Click on the second line in your script (in the table area)
2. In your browser window, highlight the title of the page, or some important text, such as "Welcome"
3. Right-click on your highlighted text, and you will see some commands have been added to the menu that pops up
4. Choose the verifyTextPresent option that shows your highlighted text. The command should be inserted just below the first command.
5. Now go down to your "clickAndWait" command. Choose the line in the table AFTER the clickAndWait item.
6. Right-click and select "Insert New Command"
7. In the Command Input, select "assertTextNotPresent" and in the target type "Mercury", or something else that shouldn't be on the page
8. Re-run your test, and you can see that the checks are working!

2.3.4 Verifying contents on inputs

Sometimes you will need to verify the contents on inputs, and this can be more complicated than just verifying text is present. **The most common way is the right-click menu**. There's a sub-menu called "Show All Commands" that lists more options. However, this menu doesn't show up for checkboxes or select boxes.

2.3.5 Verify vs. Assert

You will notice some of these commands start with "assert" and some start with "verify". "assert" will stop the entire execution of the script. This is useful for checking you are on the right page, or if some critical step has been reached. "verify" will log any error, but then keep on going with the test. This is useful for checking the values of inputs when there are a lot on the page.

Test Cases vs. Test Suites

Here is a description of the differences between a Test Case and a Test Suite.

- Test Case - this is one script, one set of instructions, saved in one file. If you look at the menu, you will see "Save Test Case" near the top.

- Test Suite - this is a collection of Test Cases, just a listing of locations of individual scripts, also in HTML format. If you look at the file menu, you will see near the bottom a set of commands for Test Suites. You can run an entire SET of test cases at once by using a Test Suite. If you work with Test Suites in Selenium IDE, it's important that you save BOTH the Test Case and the Test Suite after making changes.

2.3.6 How to create and run TestSuites

To automate your tests there are a few things you need to do. First of all you need to make sure all the tests you made in Selenium IDE are saved as .html files. You can do this by going to each of your test files and renaming them to have a .html file extension. Or you can save your test as an html file when you create it.

Now, in the directory of all your tests, you need to make an html file called TestSuite.html (it can be called whatever you want but here I'm going to use TestSuite.html). This file should contain a simple array of links to each of your test html's. An example is shown below.

TestSuite.html

```
        <table>
<tr>
<td>Test suite for the whole application</td>
</tr>
<tr>
<td><a target="testFrame" href="test_main_page.html" >Access main page</a></td>
</tr>
<tr>
<td><a target="testFrame" href="test_login.html" >Login to application</a></td>
</tr>
<tr>
<td><a target="testFrame" href="test_address_change.html" >Change address</a></td>
</tr>
<tr>
<td><a target="testFrame" href="test_logout.html" >Logout from application</a></td>
</tr>
</table>
```

Remember TestSuite.html should be kept in the same directory as all your tests. Obviously you will need to add new array rows, and change the html links as necessary, depending on your test html filenames and the number of tests you want to run.

Be sure to add the target="testFrame', or else Selenium won't be able to run your tests. This will force the tested pages to be opened in the bottom frame of the test window, so you can watch.

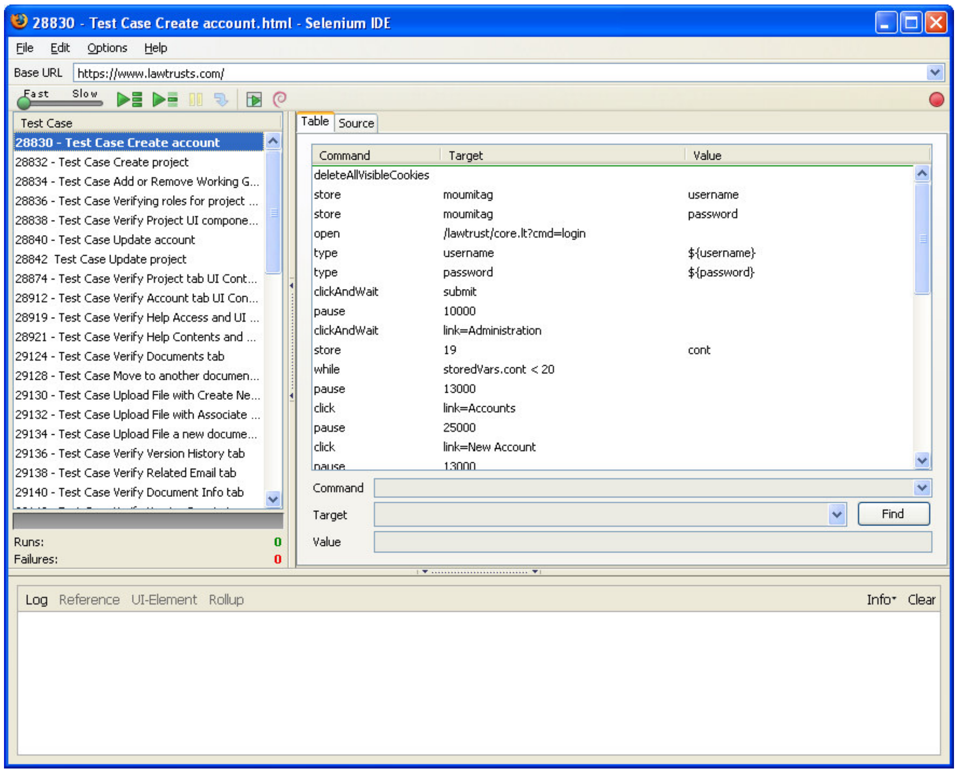
Now open Firefox, but not Selenium IDE. We are going to invoke Selenium Core, which runs as a sort of web application. In your browser, copy the following line and paste it into the address bar:

```
      chrome://selenium-
ide/content/selenium/TestRunner.html?baseURL=http://localhost&test=file:///dir/testsuite.
html&auto=true
```

Now you will see a new screen with 4 frames. The leftmost one should display the tables as described in your TestSuite.html file. The middle should show the table values of the current test, and the rightmost should show the commands available. Now you can click on the links of your tests in the leftmost window and run them using 'run selected' or you can run all the tests. Bear in mind that if a test fails, you will have to run the test on its own to see exactly where it has failed. For this reason it is important to make tests that in the test teardown phase, nullifies all the changes made in the setup phase. This will allow your tests to run properly while leaving the state of the program in the same way that it was found.

Other way to run the TestSuites:

1. Create the html file (Testsuite.html)
2. Open Selenium IDE
3. Go to File - > Open Testsuite… menu
4. Click on the Play entire test suite button
5. You will see the results below the TCs list



Base URL: https://www.lawtrusts.com/

Test Case
28830 - Test Case Create account
28832 - Test Case Create project
28834 - Test Case Add or Remove Working G...
28836 - Test Case Verifying roles for project ...
28838 - Test Case Verify Project UI compone...
28840 - Test Case Update account
28842 Test Case Update project
28874 - Test Case Verify Project tab UI Cont...
28912 - Test Case Verify Account tab UI Con...
28919 - Test Case Verify Help Access and UI ...
28921 - Test Case Verify Help Contents and ...
29124 - Test Case Verify Documents tab
29128 - Test Case Move to another documen...
29130 - Test Case Upload File with Create Ne...
29132 - Test Case Upload File with Associate ...
29134 - Test Case Upload File a new docume...
29136 - Test Case Verify Version History tab
29138 - Test Case Verify Related Email tab
29140 - Test Case Verify Document Info tab

Runs: 0
Failures: 0

Command	Target	Value
deleteAllVisibleCookies		
store	moumitag	username
store	moumitag	password
open	/lawtrust/core.lt?cmd=login	
type	username	${username}
type	password	${password}
clickAndWait	submit	
pause	10000	
clickAndWait	link=Administration	
store	19	cont
while	storedVars.cont < 20	
pause	13000	
click	link=Accounts	
pause	25000	
click	link=New Account	
pause	13000	

Command
Target
Value

Picture #5

2.4 How to run Test Suites on Internet Explorer

Following this how to you will be able to test remote websites from a windows computer using the Internet Explorer (IE) web browser and the test files that you create. This is very easy to set up and overcomes the problem of not being able to test remote sites due to the built in JavaScript security features of web browsers. It does not require you to change any settings of your browser. This how to only applies to testing with the Internet Explorer browser.

Steps

1. Download the latest release of selenium core. This doesn't need to be a special development version, just the normal zip file available from the selenium website. At time of writing this file was located here:

   URL: http://selenium-core.openqa.org/download.jsp

   Filename: selenium-core-0.8.3.zip

2. Unzip the file to a place on the hard disk of the same windows computer from which you wish to do testing from (this doesn't have to be the same computer that hosts the website you wish to test). For the purposes of this how to the zip file will be extracted to the C: drive resulting in the folder C:\selenium being created.

3. Go to "C:\selenium\tests" and make a copy of the file named TestSuite.html and name it MyTestSuite.html.

4. Open up the file MyTestSuite.html that you just created. This file contains a table of the tests that are part of this suite. Because you want to perform your own tests, edit this table to contain the names of the tests you would like to perform and the links to where those tests can be found. For this example we will remove all entries in the table (EXCEPT the first title row) and create the row:

   ```
   <tr><td><a href="./Test1.html">Test1</a></td></tr>
   ```

   So the table should now look like this:

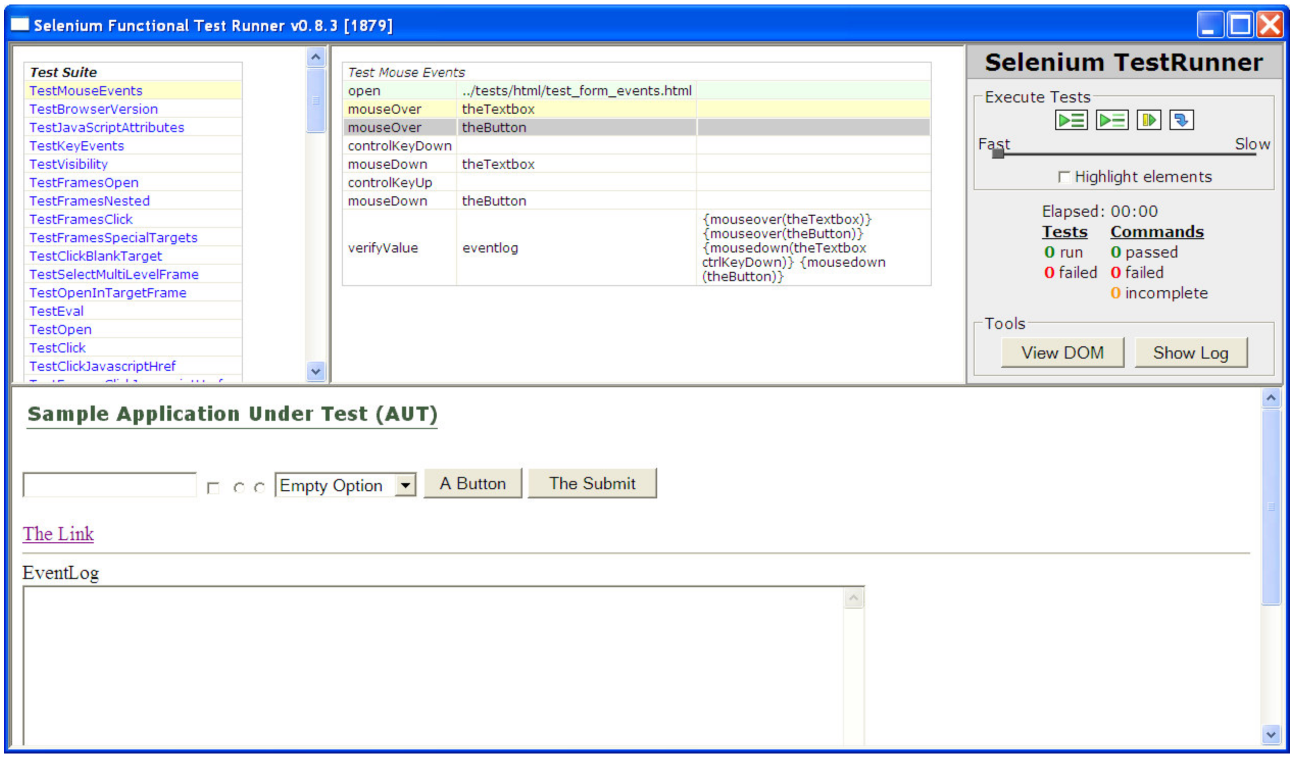
   ```
           ...
   <tbody>
           <tr><td><b>Test Suite</b></td></tr>
           <tr><td><a href="./Test1.html">Test1</a></td></tr>
           </tbody>
   ...
   ```

5. You are now ready to create your first test. The easiest way to start is to simply make a copy of the file named GoogleTestSearch.html and call it Test1.html. This file represents a

single test that goes to the google website and performs a search. Later you can come back and modify this test to go to the web site you wish to test and perform your own tests. Please refer to the Selenium homepage for a reference of the commands that you can use.

6. Open up Internet Explorer and type the following into the address bar: C:\selenium\core\TestRunner.hta

7. This should open up a window with for panels. In the first panels, update the path from: "../test/TestSuite.html" to "../test/MyTestSuite.html", and clicks on Go button. After that your test suite will be loaded on the left panel, your test in the middle and controls on the right. The lower frame holds what the browser is navigating as the tests are performed. Click "All" or "Selected" and the test will begin running. That's it! You've now performed your first test. Please note that this test may pass or fail, you should see rows in the test table being highlighted as the test is being performed and some or all rows being colored green for asses and some or all rows being colored pink for fails. If you do not see the test being performed and it seems to get "stuck" on a row then something has gone wrong with your set up.



Selenium Functional Test Runner v0.8.3 [1879]

Test Suite
TestMouseEvents
TestBrowserVersion
TestJavaScriptAttributes
TestKeyEvents
TestVisibility
TestFramesOpen
TestFramesNested
TestFramesClick
TestFramesSpecialTargets
TestClickBlankTarget
TestSelectMultiLevelFrame
TestOpenInTargetFrame
TestEval
TestOpen
TestClick
TestClickJavascriptHref

Test Mouse Events		
open	../tests/html/test_form_events.html	
mouseOver	theTextbox	
mouseOver	theButton	
controlKeyDown		
mouseDown	theTextbox	
controlKeyUp		
mouseDown	theButton	
verifyValue	eventlog	{mouseover(theTextbox)} {mouseover(theButton)} {mousedown(theTextbox ctrlKeyDown)} {mousedown (theButton)}

Selenium TestRunner

Execute Tests

Fast Slow

☐ Highlight elements

Elapsed: 00:00

Tests	Commands
0 run	0 passed
0 failed	0 failed
	0 incomplete

Tools

View DOM Show Log

Sample Application Under Test (AUT)

Empty Option A Button The Submit

The Link

EventLog

Picture #6

2.4.1 General Script Example

In the following section we'll go create a simple script that can be used as building block for more complex tests.

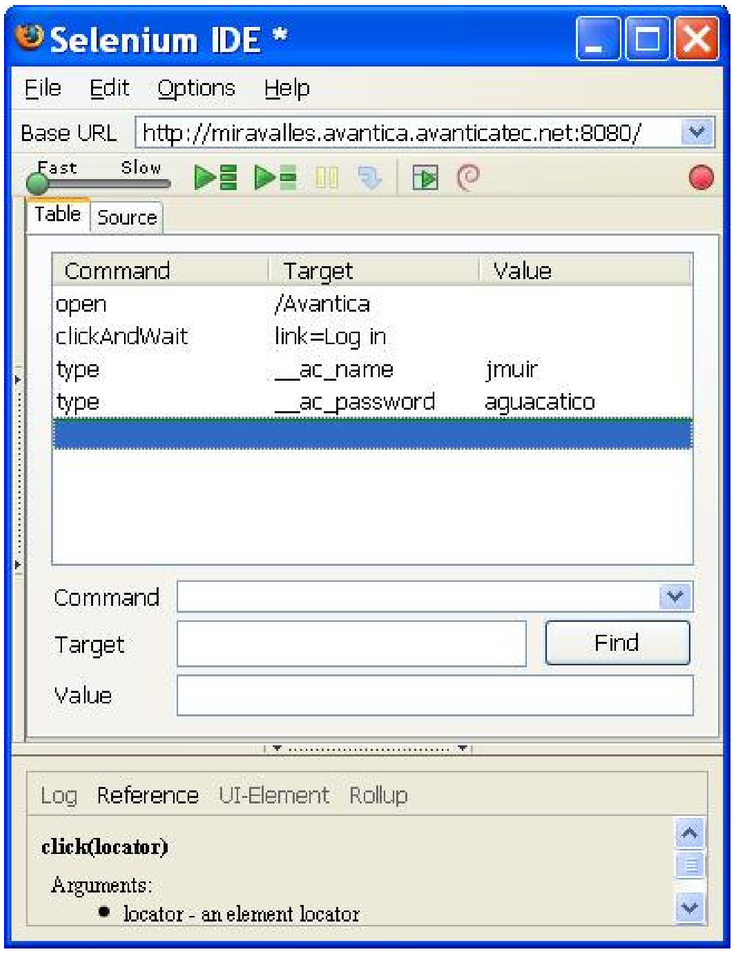
2.4.1.1 Verify existence of text or images

This test will check for certain text or image tags. The first we need to do is open a FireFox browser window. Start the Selenium IDE by going to Tools>Selenium IDE; make sure the application is set on recording mode, this can be verified by checking that the red button has been turned on.

Next, go to the browser and point it to the following url: http://miravalles.avantica.avanticatec.net:8080/Avantica and navigate to the specified address. If you go to the IDE you'll notice that it appears as if nothing has been recorded; don't be alarmed, this is a known issue and the way to get around this bug is by simply continuing to record the script.

Now, we'll login to the intranet. Click on the login link located on the right side of the page. The reason why we are clicking on this link and not adding the username and password directly on the main page is because we need to save the navigation to the intranet prior to performing any other critical step.

Now go to the IDE and add a new line by clicking on the line right below the last command. Add 'type' on the command field, and for the target we'll add the name of the username text field. Selenium has an easy way to find the names of fields or ids on html. The way to achieve this is by simply right clicking on the username field and on the last menu section you'll see Selenium's options for that field; you'll notice that the field's name is <__ac_name>. Enter the name on the target section on the IDE and your username on the value field. Once completed, repeat the same steps but this time adding the password toy our account. Your script should look something like this:



Base URL: http://miravalles.avantica.avanticatec.net:8080/

Command	Target	Value
open	/Avantica	
clickAndWait	link=Log in	
type	__ac_name	jmuir
type	__ac_password	aguacatico

Command
Target
Value

click(locator)

Arguments:

- locator - an element locator

Picture #7

Once you have added the fields, go back to the browser and click on the login button. This would conclude the steps needed to automate the login action on the intranet. Now, we'll verify that we are on the correct page by adding verifications.

There are two different ways in which you can verify text with Selenium, you can either check that the text is present on the page or that the html code referring to that text is on the source of the page. Knowing which to use will depend on if you can assure that the text you're looking for will only appear given certain actions performed on the site or if what you're looking for is that it is displayed on a certain place within the page.

To verify text all we need to do is select <verifyText> from the Command dropdown menu. On the Target section add the XPATH code for the element you want to check for and on the Value field enter the text you want to check. There's an easier way to do this by selecting the text from the page and right click on it. You'll notice that there are two options to choose from on the Selenium menu section; one contains the readable text and the other one contains the source code text. Select the latter one.

You can also do the same for images and tables, by simply right clicking on them you'll be able to select the verification you want to perform on the selected option.

If for example you need to check that some text matches a regular expression, you can use the following syntax within the value field of the <verifyText> command: "regexp:text<regex>text…".

3 Added Functions in Selenium IDE and Core

The users that have the Selenium applications can add new functions to be used in the scripts that have been created by the community of Selenium.

3.1 StoreGlobal Function

The StoreGlobal feature will store variables globally, making these variables available in the following test cases of the test Suite.

The main use of this feature is to make permanent values throughout the execution of the test suite, to facilitate the use of variables without declaring them in each test case.

3.1.1 Installation of the StoreGlobal

3.1.1.1 Installation of the StoreGlobal in Selenium Core

To add the extension to support the management of StoreGlobal variable in Selenium Core, follow these steps:

Requirements:

- ✓ Having a local or remote Selenium Core

Steps:

1. Go to the directory of Selenium Core
2. Located in the following path: Selenium-Core-VersionNumber\core\scripts
3. Locate and open the file user- extension.js
4. Copy and paste the script located in Annex I
5. Save the changes

3.1.1.2 Installation of the StoreGlobal in Selenium IDE

To add the extension to support the management of StoreGlobal variable in Selenium IDE follow these steps:

Requirements:

- ✓ Having installed the plug in Selenium IDE on Firefox

Steps:

1. If you have a local copy of Selenium Core
    - Locate the file user-extentions.js
2. If you don't have installed Selenium Core, then choose a directory location and easy to create the user-extentions.js
3. In the user-extentions.js copy and paste the script located in Annex I
4. Save the changes
5. Open Selenium IDE
6. Go to Menu: Options/Options
7. On the General tab, go to the section: Selenium Core extensions
8. Using the Browse button to locate the file user-extensions.js
9. Close and open de Selenium IDE

3.1.2 Example of the StoreGlobal

On this section of the document we find a simple example of the use of the StoreGlobal function.

TestA.html

Command	Target	Value
storeGlobal	sllona	companyCode
storeGlobal	admin	userLogin
storeGlobal	password	userPassword

Test B.html

Command	Target	Value
Open	https://server.expensable.com	
Type	companyCode	${companyCode}
Type	userLogin	${userLogin}
Type	userPassword	${userPassword}
Click		

TestSuite.html

```
<html>
<head>
<meta content="text/html; charset=ISO-8859-1"
http-equiv="content-type">
<title>Test Suite - Global actions tests</title>

</head>

<body>

<table     cellpadding="1"
          cellspacing="1"
          border="1">
       <tbody>
           <tr><td><b>Test Suite</b></td></tr>
           <tr><td><a href="./Test A.html">TestGlobal (set vars)</a></td></tr>
           <tr><td><a href="./Test B.html">TestGlobal (use vars)</a></td></tr>
       </tbody>
   </table>

</body>
</html>
```

3.2 Gotoif Function

The Gotoif feature controls conditional flows.

3.2.1 Installation of the Gotoif

3.2.1.1 Installation of the Gotoif in Selenium Core

To add the extension to support the management of Gotoif variable in Selenium Core, follow these steps:

Requirements:

- ✓ Having a local or remote Selenium Core

Steps:

1. Go to the directory of Selenium Core
2. Located in the following path: Selenium-Core-VersionNumber\core\scripts
3. Locate and open the file user- extension.js
4. Copy and paste the script located in Annex II
5. Save the changes

3.2.1.2 Installation of the Gotoif in Selenium IDE

To add the extension to support the management of Gotoif variable in Selenium IDE follow these steps:

Requirements:

- ✓ Having installed the plugin Selenium IDE on Firefox

Steps:

1. If you have a local copy of Selenium Core
   - Locate the file user-extentions.js inside the directory ..\core\scripts\
2. If you don't have installed Selenium Core, then choose a directory location and easy to create the user-extentions.js
3. In the user-extentions.js copy and paste the script located in Annex III
4. Save the changes
5. Open Selenium IDE

6. Go to Menu: Options/Options
7. On the General tab, go to the section: Selenium Core extensions
8. Using the Browse button to locate the file user-extensions.js
9. Close and open de Selenium IDE

3.2.2 Example of the Gotoif

On this section we describe a simple example on the use of the Gotoif function.

TestA.html

Command	Target	Value
Store	3	X
store	2	Y
gotoIf	${x} > ${y}	Imprime_1
Echo	Y es mayor que X	
gotoLabel	Imprime_fin	
Label	Imprime_1	
Echo	X es mayor que Y	
Label	Imprime_fin	
Echo	Fin del Flujo	

Result when X is greater than Y

- [info] Executing: lstore l 3 l x l
- [info] Executing: lstore l 2 l y l
- [info] Executing: lgotoIf l ${x} > ${y} l imprime_1 l
- [info] Executing: lecho l X es mayor que Y l l
- [info] echo: X es mayor que Y
- [info] Executing: llabel l imprime_fin l l
- [info] Executing: lecho l Fin del flujo l l
- [info] echo: Fin del flujo

Result when Y is greater than X

- [info] Executing: lstore l 3 l x l
- [info] Executing: lstore l 5 l y l
- [info] Executing: lgotoIf l ${x} > ${y} l imprime_1 l
- [info] Executing: lecho l Y es mayor que X l l
- [info] echo: Y es mayor que X
- [info] Executing: lgotolabel l imprime_fin l l

- [info] Executing: |echo | Fin del flujo | |
- [info] echo: Fin del flujo

4 FAQ

On this section of the document we can find a frequent asked questions and answers for some problems that appear to automate with Selenium.

1. **Q:** How should I record the scripts and run the test cases?

   A: To record the scripts and run the test cases it is best to have the browser window maximized and all participants must have the same monitor resolution on their computers, if not, you can write scripts in different ways and have problems when running the test cases

2. **Q:** What speed should be used to execute the test in Selenium IDE or Selenium Core?

   A: If the application is a bit slow in loading pages is recommendable to run the tests in a medium Speed

3. **Q:** How to put the test cases in the testsuite?

   A: The test cases should be placed in a logical flow, so you can reuse the scripts and not collide with each other, for example:

   TestSuite.html

   ```
   <tr><td><a href="Create a submitter user.html"> Create a submitter user</a></td></tr>
   <tr><td><a href=”Verify submitters options.html"> Verify submitters options</a></td></tr>
   <tr><td><a href="Delete submitter user.html">Delete submitter user</a></td></tr>
   ```

4. **Q:** What can be used to find the elements of a page?

   A: You can use the DOM for BadBoy, following these steps:

   a) Open the BadBoy tool
   b) Record the page that you want to find the elements
   c) Press Ctrl+D
   d) Must appear the DOM of Badboy
   e) You can see the elements contained in the page

5. **Q:** What use when an element is not located correctly when recording the script?

   A: As the first choice you can use the Xpath of Selenium IDE, if the item is still not recognized, you can refer to it through javascript, following these steps to use the Xpath in Selenium IDE:

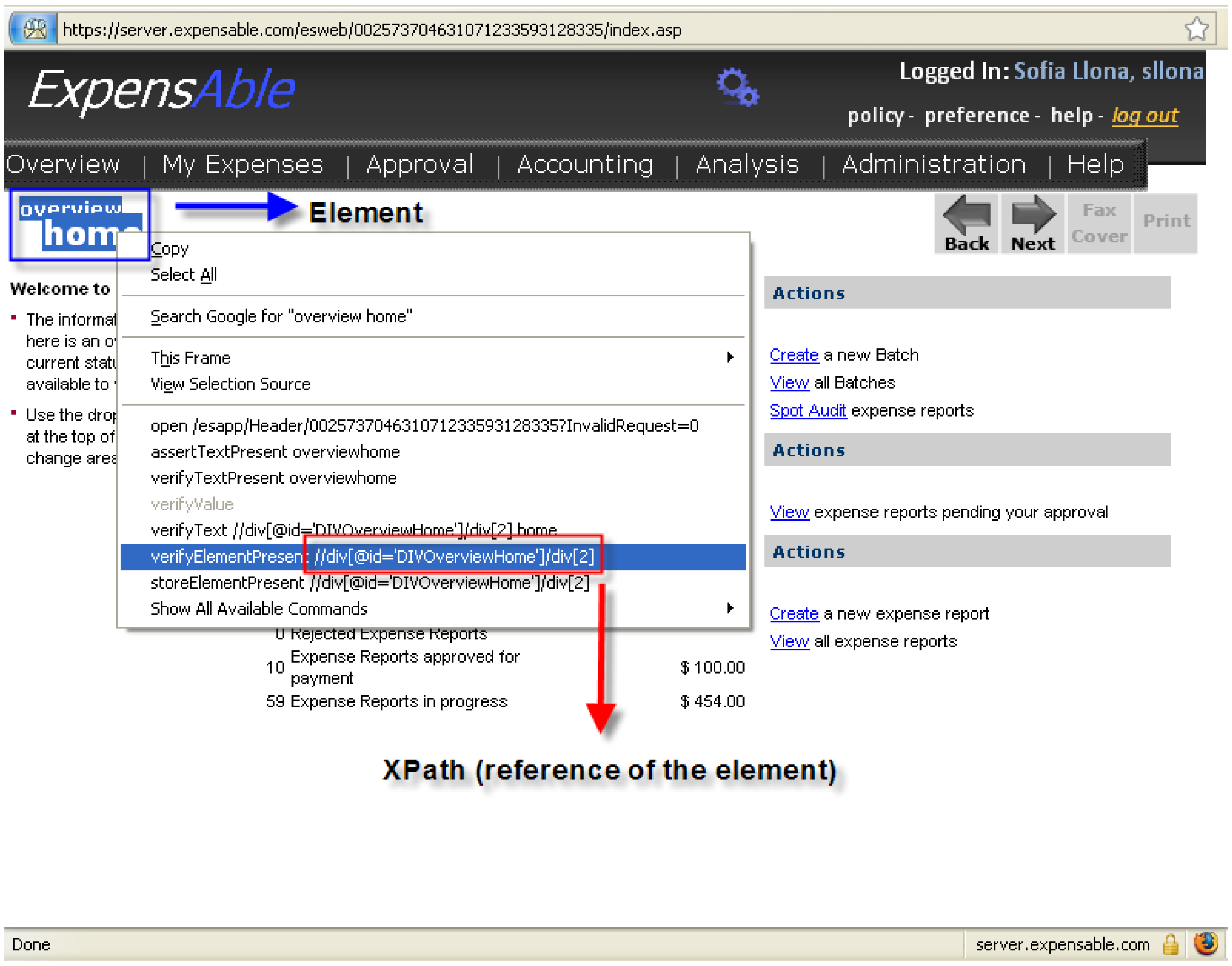
a) Open the Selenium IDE

b) Record the page that you want to find the elements

c) Select the element that you find

d) Press right click

e) Look the options the Selenium IDE presents


https://server.expensable.com/esweb/002573704631071233593128335/index.asp
ExpensAble
Logged In: Sofia Llona, sllona
policy - preference - help - log out
Overview | My Expenses | Approval | Accounting | Analysis | Administration | Help
overview home
Element
Back
Next
Fax Cover
Print
Welcome to
Actions
Create a new Batch
View all Batches
Spot Audit expense reports
View expense reports pending your approval
Create a new expense report
View all expense reports
Copy
Select All
Search Google for "overview home"
This Frame
View Selection Source
Show All Available Commands
XPath (reference of the element)
Done
server.expensable.com

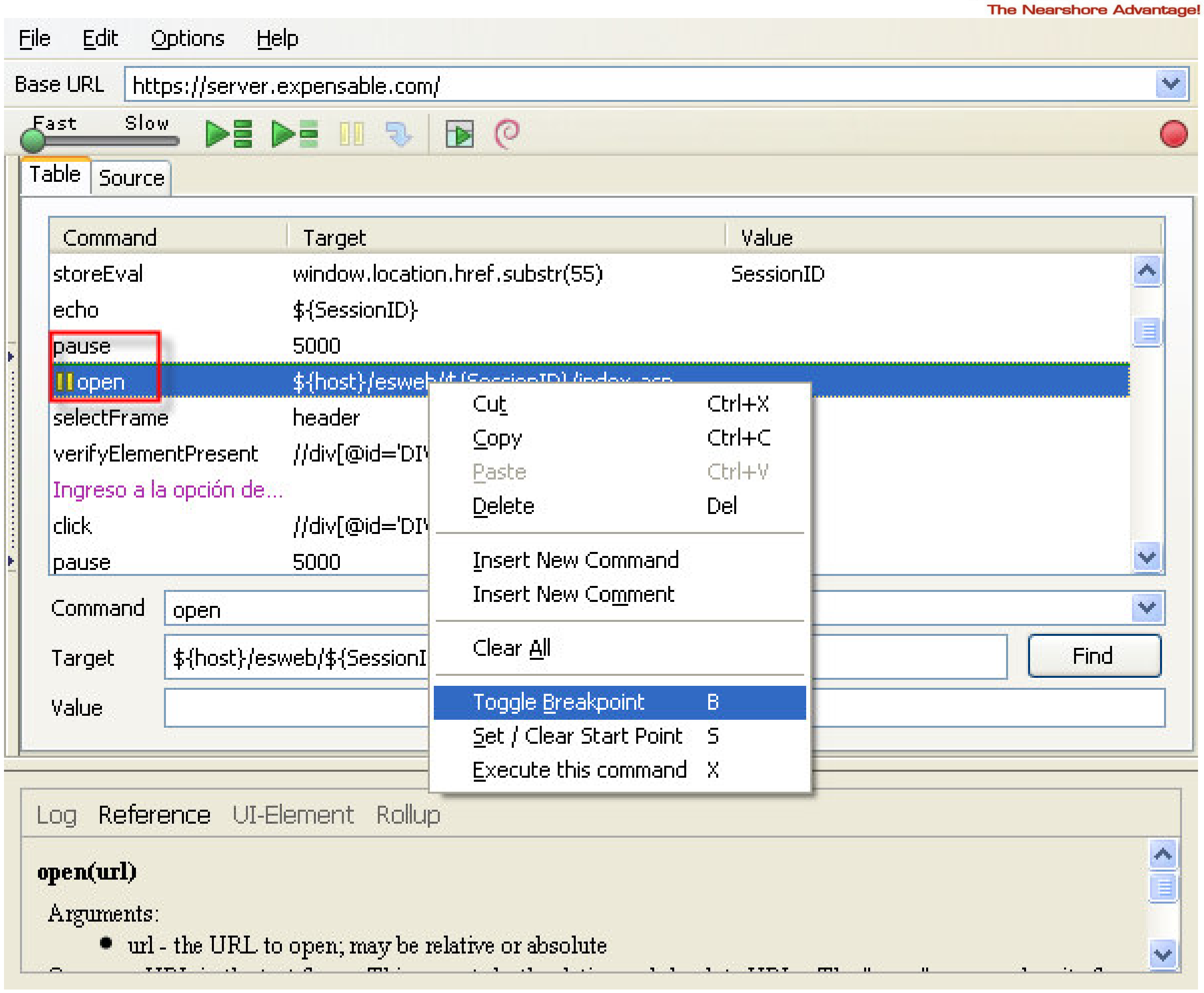
open /esapp/Header/002573704631071233593128335?InvalidRequest=0
assertTextPresent overviewhome
verifyTextPresent overviewhome
verifyValue
verifyText //div[@id='DIVOverviewHome']/div[2] home
verifyElementPresent //div[@id='DIVOverviewHome']/div[2]
storeElementPresent //div[@id='DIVOverviewHome']/div[2]

0 Rejected Expense Reports
10 Expense Reports approved for payment $ 100.00
59 Expense Reports in progress $ 454.00

6. **Q:** When is not recommendable to use breakpoints in the script?

A: It is not recommendable to use breakpoints in comments or after a pause, because Selenium ignores the breakpoint when you run the script, for example:

ScriptA.html



Command	Target	Value
storeEval	window.location.href.substr(55)	SessionID
echo	${SessionID}	
pause	5000	
open	${host}/esweb/${SessionID}/index.asp	
selectFrame	header	
verifyElementPresent	//div[@id='DI	
Ingreso a la opción de...		
click	//div[@id='DI	
pause	5000	

Command: open
Target: ${host}/esweb/${SessionI
Value:

open(url)

Arguments:

- url - the URL to open; may be relative or absolute

7. **Q:** How to use JavaScript with dynamic data?

A: The use of JavaScript in Selenium is the same like in other normal JavaScripts application, for example, to extract the session number for an application you can use the next syntax:

ScriptB.html

```
<tr>
        <td>storeEval</td>
        <td>window.location.href.substr(55)</td> /*(store variable with JavaScript)*/
        <td>SessionID</td> /*(name of the variable)*/
</tr>
<tr>
        <td>open</td>
        <td>/esweb/${SessionID}/index.asp</td> /*(use of the variable)*/
        <td></td>
```

```
</tr>
```

8. **Q:** Why alerts and confirmations cannot be handled correctly?

   A: Selenium does NOT support JavaScript alerts and confirmations that are generated in a page's onload() event handler. In this case a visible dialog WILL be generated and Selenium will hang until someone manually clicks OK or Cancel

9. **Q:** How to select the window where you are?

   A: Use the SelectWindow function with the target null, for example:

```
<tr>
        <td>selectWindow</td>
        <td></td>
        <td></td>
</tr>
```

10. **Q:** How to select a frame that has or is part of a hierarchy of Frames?

    A: You must first select the parent frame and then the child in descending order until you reach the desire frame, for example:

    Parent Frame: body

    Child Frame: status

```
<tr>
        <td>selectFrame</td>
        <td>body</td>
        <td></td>
</tr>
<tr>
        <td>selectFrame</td>
        <td>status</td>
        <td></td>
</tr>
```

11. **Q:** Which is the correct syntax to select a Frame?

    A: The correct syntax to select a frame is:

    Correct

```
<tr>
        <td>selectFrame</td>
        <td>body</td>
        <td></td>
</tr>
```

    Incorrect

```
<tr>
        <td>selectFrame</td>
        <td>name=body</td>
        <td></td>
</tr>
```

12. **Q:** Function ClickAndWait vs. Click?

    A: It is more recommendable to use the Click function than the ClickAndWait function.

    If you are recording the functionality of a button and you are using the ClickAndWait function, would be better to use the Click function and then put a pause of X time, for example:

    Before

```
<tr>
        <td> clickAndWait </td>
        <td>button</td>
        <td></td>
</tr>
```

    After change

```
<tr>
        <td>click</td>
        <td>button</td>
        <td></td>
</tr>
```

```
<tr>
        <td>pause</td>
        <td>10000</td>
        <td></td>
</tr>
```

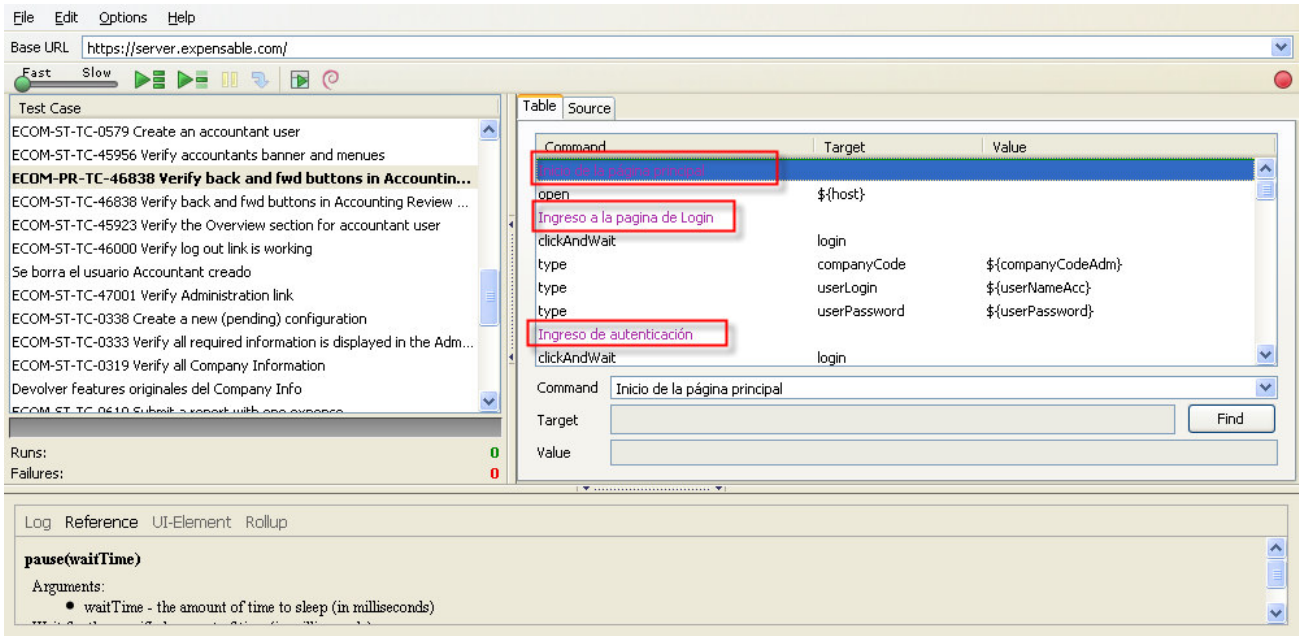
13. **Q:** Function Assert vs. Verify?

    A: "Assert" will stop the entire execution of the script. This is useful for checking if you are on the right page, or if some critical step has been reached. "Verify" will log any error, but then keep on going with the test

14. **Q:** Should comments be used in the scripts?

    A: We recommend using comments within the scripts, so an external user to the project can be guided into the script and understanding the flow of the script, you can use the following steps to use the comments in Selenium IDE:

    a) Open the Selenium IDE
    b) Record the page that you want
    c) Press right click
    d) Select the option "Insert New Comment"
    e) Enter the comment in the Command field



Base URL https://server.expensable.com/

Test Case
ECOM-ST-TC-0579 Create an accountant user
ECOM-ST-TC-45956 Verify accountants banner and menues
ECOM-PR-TC-46838 Verify back and fwd buttons in Accountin...
ECOM-ST-TC-46838 Verify back and fwd buttons in Accounting Review ...
ECOM-ST-TC-45923 Verify the Overview section for accountant user
ECOM-ST-TC-46000 Verify log out link is working
Se borra el usuario Accountant creado
ECOM-ST-TC-47001 Verify Administration link
ECOM-ST-TC-0338 Create a new (pending) configuration
ECOM-ST-TC-0333 Verify all required information is displayed in the Adm...
ECOM-ST-TC-0319 Verify all Company Information
Devolver features originales del Company Info

Runs: 0
Failures: 0

Command	Target	Value
Inicio de la página principal		
open	${host}	
Ingreso a la pagina de Login		
clickAndWait	login	
type	companyCode	${companyCodeAdm}
type	userLogin	${userNameAcc}
type	userPassword	${userPassword}
Ingreso de autenticación		
clickAndWait	login	

Command: Inicio de la página principal
Target:
Value:

pause(waitTime)
Arguments:
- waitTime - the amount of time to sleep (in milliseconds)

5 Annex

5.1 Annex I - Script to add the StoreGlobal function

```
globalStoredVars = new Object();
/*
 * Globally store the value of a form input in a variable
 */
Selenium.prototype.doStoreValueGlobal = function(target, varName) {
    if (!varName) {
        // Backward compatibility mode: read the ENTIRE text of the page
        // and stores it in a variable with the name of the target
        value = this.page().bodyText();
        globalStoredVars[target] = value;
        return;
    }
    var element = this.page().findElement(target);
    globalStoredVars[varName] = getInputValue(element);
};

/*
 * Globally store the text of an element in a variable
 */
Selenium.prototype.doStoreTextGlobal = function(target, varName) {
    var element = this.page().findElement(target);
    globalStoredVars[varName] = getText(element);
};

/*
 * Globally store the value of an element attribute in a variable
 */
Selenium.prototype.doStoreAttributeGlobal = function(target, varName) {
    globalStoredVars[varName] = this.page().findAttribute(target);
};

/*
 * Globally store the result of a literal value
 */
Selenium.prototype.doStoreGlobal = function(value, varName) {
    globalStoredVars[varName] = value;
};
```

```
/*
 * Search through str and replace all variable references ${varName} with their
 * value in storedVars (or globalStoredVars).
 */
Selenium.prototype.replaceVariables = function(str) {
    var stringResult = str;

    // Find all of the matching variable references
    var match = stringResult.match(/\$\{\w+\}/g);
    if (!match) {
        return stringResult;
    }

    // For each match, lookup the variable value, and replace if found
    for (var i = 0; match && i < match.length; i++) {
        var variable = match[i]; // The replacement variable, with ${}
        var name = variable.substring(2, variable.length - 1); // The replacement variable
without ${}
        var replacement = storedVars[name];
        if (replacement != undefined) {
            stringResult = stringResult.replace(variable, replacement);
        }
        var replacement = globalStoredVars[name];
        if (replacement != undefined) {
            stringResult = stringResult.replace(variable, replacement);
        }
    }
    return stringResult;
};
```

5.2 Annex II - Script to add the Gotoif function in Selenium Core

```
/*
* Adding Control Flow script
*/

/*
 (C) Copyright MetaCommunications, Inc. 2006.
     http://www.meta-comm.com
     http://engineering.meta-comm.com

Distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF
ANY KIND.
*/
```

```
        function map_list( list, for_func, if_func )
          {
          var mapped_list = [];
          for ( var i = 0; i < list.length; ++i )
            {
            var x = list[i];
            if( null == if_func || if_func( i, x ) )
              mapped_list.push( for_func( i, x ) );
            }
          return mapped_list;
          }


        // Modified to initialize GoTo labels/cycles list
        HtmlRunnerTestLoop.prototype.old_initialize =
  HtmlRunnerTestLoop.prototype.initialize

        HtmlRunnerTestLoop.prototype.initialize = function(htmlTestCase, metrics,
  seleniumCommandFactory)
          {
          this.gotoLabels  = {};
          this.whileLabels = { ends: {}, whiles: {} };

          this.old_initialize(htmlTestCase, metrics, seleniumCommandFactory);

          this.initialiseLabels();
          }

        HtmlRunnerTestLoop.prototype.initialiseLabels = function()
          {
          var command_rows = map_list( this.htmlTestCase.getCommandRows()
                              , function(i, x) {
                                  return x.getCommand()
                                  }
                              );

          var cycles = [];
          for( var i = 0; i < command_rows.length; ++i )
            {
            switch( command_rows[i].command.toLowerCase() )
              {
              case "label":
                this.gotoLabels[ command_rows[i].target ] = i;
                break;
              case "while":
              case "endwhile":
                cycles.push( [command_rows[i].command.toLowerCase(), i] )
                break;
              }
            }
          var i = 0;
          while( cycles.length )
            {
            if( i >= cycles.length )
              throw new Error( "non-matching while/endWhile found" );

            switch( cycles[i][0] )
              {
              case "while":
                if(   ( i+1 < cycles.length )
                  && ( "endwhile" == cycles[i+1][0] )
                  )
                  {
                  // pair found
                  this.whileLabels.ends[ cycles[i+1][1] ] = cycles[i][1]
                  this.whileLabels.whiles[ cycles[i][1] ] = cycles[i+1][1]

                  cycles.splice( i, 2 );
                  i = 0;
                  }
                else
                  ++i;
                break;
              case "endwhile":
                ++i;
                break;
              }
            }

          }

        HtmlRunnerTestLoop.prototype.continueFromRow = function( row_num )
          {
          if(   row_num == undefined
            || row_num == null
            || row_num < 0
            )
            throw new Error( "Invalid row_num specified." );

          this.htmlTestCase.nextCommandRowIndex = row_num;
          }


        // do nothing. simple label
        Selenium.prototype.doLabel      = function(){};

        Selenium.prototype.doGotolabel  = function( label ) {

          if( undefined == htmlTestRunner.currentTest.gotoLabels[label] )
```

```
                throw new Error( "Specified label '" + label + "' is not found." );

            htmlTestRunner.currentTest.continueFromRow(
    htmlTestRunner.currentTest.gotoLabels[ label ] );
            };

        Selenium.prototype.doGoto = Selenium.prototype.doGotolabel;


        Selenium.prototype.doGotoIf = function( condition, label ) {
            if( eval(condition) )
                this.doGotolabel( label );
            }



        Selenium.prototype.doWhile = function( condition ) {
            if( !eval(condition) )
                {
                var last_row =
    htmlTestRunner.currentTest.htmlTestCase.nextCommandRowIndex - 1
                var end_while_row = htmlTestRunner.currentTest.whileLabels.whiles[ last_row ]
                if( undefined == end_while_row )
                    throw new Error( "Corresponding 'endWhile' is not found." );

                htmlTestRunner.currentTest.continueFromRow( end_while_row + 1 );
                }
            }


        Selenium.prototype.doEndWhile = function() {
            var last_row =
    htmlTestRunner.currentTest.htmlTestCase.nextCommandRowIndex - 1
            var while_row = htmlTestRunner.currentTest.whileLabels.ends[ last_row ]
            if( undefined == while_row )
                throw new Error( "Corresponding 'While' is not found." );

            htmlTestRunner.currentTest.continueFromRow( while_row );
            }
```

5.3 Annex III - Script to add the Gotoif function in Selenium IDE

```
        /*
        * Adding Flow Control script
        */
```

```
var gotoLabels= {};
var whileLabels = {};

// overload the oritinal Selenium reset function
Selenium.prototype.reset = function() {
  // reset the labels
  this.initialiseLabels();
  // proceed with original reset code
  this.defaultTimeout = Selenium.DEFAULT_TIMEOUT;
  this.browserbot.selectWindow("null");
  this.browserbot.resetPopups();
}

Selenium.prototype.initialiseLabels = function()
{
   gotoLabels  = {};
   whileLabels = { ends: {}, whiles: {} };
   var command_rows = [];
   var numCommands = testCase.commands.length;
   for (var i = 0; i < numCommands; ++i) {
     var x = testCase.commands[i];
     command_rows.push(x);
   }
   var cycles = [];
   for( var i = 0; i < command_rows.length; i++ ) {
      if (command_rows[i].type == 'command')
      switch( command_rows[i].command.toLowerCase() ) {
         case "label":
            gotoLabels[ command_rows[i].target ] = i;
            break;
         case "while":
         case "endwhile":
            cycles.push( [command_rows[i].command.toLowerCase(), i] )
            break;
      }
   }
   var i = 0;
   while( cycles.length ) {
      if( i >= cycles.length ) {
         throw new Error( "non-matching while/endWhile found" );
      }
      switch( cycles[i][0] ) {
         case "while":
            if(    ( i+1 < cycles.length )  && ( "endwhile" == cycles[i+1][0] ) ) {
               // pair found
               whileLabels.ends[ cycles[i+1][1] ] = cycles[i][1];
               whileLabels.whiles[ cycles[i][1] ] = cycles[i+1][1];
               cycles.splice( i, 2 );
               i = 0;
            } else ++i;
```

```
                    break;
                case "endwhile":
                    ++i;
                    break;
            }
        }
    }

    Selenium.prototype.continueFromRow = function( row_num )
    {
        if(row_num == undefined || row_num == null || row_num < 0) {
            throw new Error( "Invalid row_num specified." );
        }
        testCase.debugContext.debugIndex = row_num;
    }

    // do nothing. simple label
    Selenium.prototype.doLabel      = function(){};

    Selenium.prototype.doGotolabel  = function( label )
    {
        if( undefined == gotoLabels[label] ) {
            throw new Error( "Specified label '" + label + "' is not found." );
        }
        this.continueFromRow( gotoLabels[ label ] );
    };

    Selenium.prototype.doGoto = Selenium.prototype.doGotolabel;

    Selenium.prototype.doGotoIf = function( condition, label )
    {
        if( eval(condition) ) this.doGotolabel( label );
    }

    Selenium.prototype.doWhile = function( condition )
    {
        if( !eval(condition) ) {
            var last_row = testCase.debugContext.debugIndex;
            var end_while_row = whileLabels.whiles[ last_row ];
            if( undefined == end_while_row ) throw new Error( "Corresponding 'endWhile' is
not found." );
            this.continueFromRow( end_while_row );
        }
    }

    Selenium.prototype.doEndWhile = function()
    {
        var last_row = testCase.debugContext.debugIndex;
        var while_row = whileLabels.ends[ last_row ] - 1;
        if( undefined == while_row ) throw new Error( "Corresponding 'While' is not found."
    );
                this.continueFromRow( while_row );
            }
```