Selenium at Auctionomics

Abstract

This paper describes implementing Selenium test automation at Auctionomics, Inc. This is a continuation of *Applied Selenium*, where I write about using Selenium at Playlist.com. Here I discuss only what is new or different about the Selenium implementation at Auctionomics.

Background

Auctionomics is a three person consulting company in Palo Alto, California. The company was founded on the auction theory ideas developed by Paul Milgrom. Auctionomics offers clients sophisticated auction consultation for high end auctions (millions through hundreds of millions of dollars). Auctionomics can also set up and run an auction using its proprietary MaaX software.

I was hired by the CTO, who is also the sole developer, to develop test automation for the MaaX system.

Implementation

MaaX is a software as a service system (SaaS) built on C#, .NET, IIS, and MS-SQL. In addition the system uses a linear program solver to find the best possible outcome for the auction. Selenium testing was done on the following systems:

Windows 7 running Firefox, Windows 7 running Internet Explorer 8
Windows XP running Firefox, Windows XP running Internet Explorer 7

Because of operating system and browser compatibility issues, two different versions of the Selenium RC libraries were used. See the following table below for details.

Table 1: Compatibility Table (these are the platforms automation is run on)

Config	Selenium RC Version[a]	Notes
WinXP/IE7	1.0.1	When using RC1.0.3 causes error: java lang runtime exception: sessionID nnnnnn doesn't exist; perhaps this session was already stopped?
WinXP/Firefox 3.5.9	1.0.3	When using RC1.0.1 causes error: java runtime exception: Firefox refused a shutdown while preparing a profile.
Win7/IE8	1.0.1	Need to turn protected mode OFF in IE8 and run batch file with the "iehta" argument
Win7/Firefox 3.6.3	1.0.3	Using rc1.0.3

a. Refers to the selenium-server.jar file in folder ..\selenium-server 1.0.n\

SeleniumRC was started by running the appropriate batch file for the target configuration. Each batch file would call the correct selenium version for that operating system and browser combination. For example, if running Selenium RC on Win7 with Firefox, a batch file containing the following was run, calling v.103 of Selenium:

```
java -jar selenium-server.jar -userExtensions C:\selenium103\selenium-server-1.0.3\user-exten-
sions.js -timeout 5000 -htmlSuite *firefox "http://dev.auctionomics.com" "C:\My Dropbox\Auction-
omics\sel_scripts\auc_testsuite.html" "C:\My Dropbox\Auctionomics\sel_scripts\results.html"
```

What to test?

The table below lists modules of the MaaX system that had been scripted. This was a small subset of the total Maax functionality.

Reset existing test auctions	Create forward auction	Create reverse auction
Clone auction	Solve simple auction	Solve auction with budgets
Solve auction with budgets2	Solve auction with minimum quantity	Solve auction with budget proof
Auction smoke test, menu check	Auction options check	Test supply/demand curve auction
Basic auction test		

Expanding Selenium

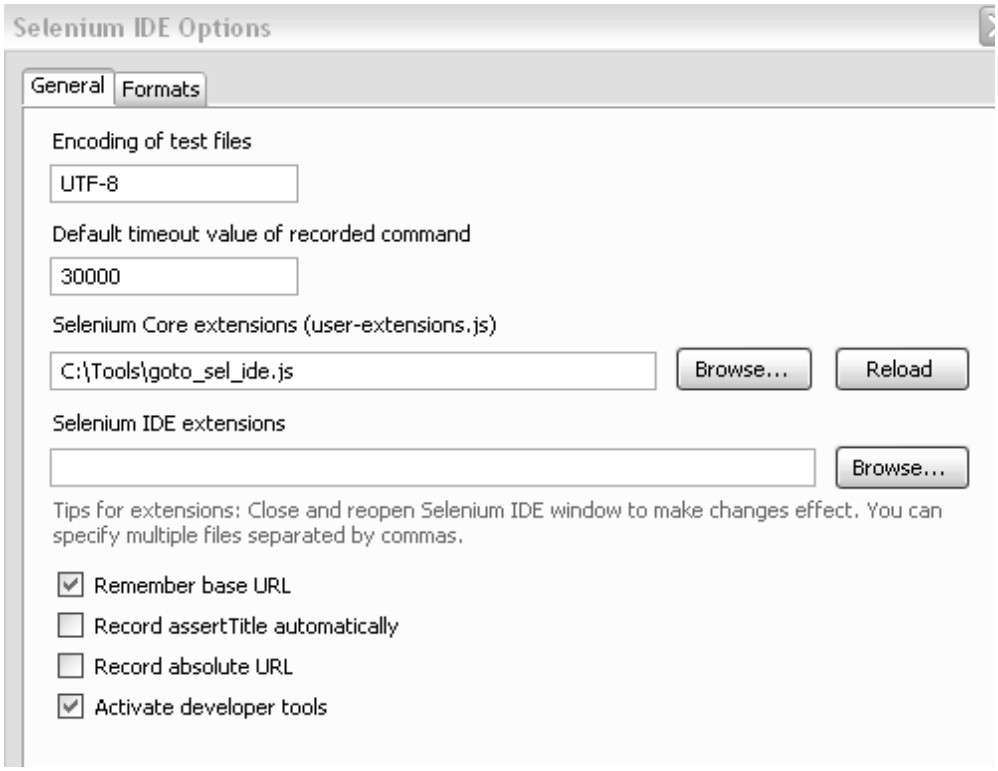
Building a loop in selenium

Before the main Selenium test suite ran, I wanted to make sure some pre-existing test auctions were set to the proper state. There were about ten auctions, and the best way to do this looked to me to loop through a list or array of auctions, examine the state, change if needed, then go on to the next auction. Enter the world of loops and flow control in Selenium.

The native language of Selenium, known as selenese, does not lend itself easily to even slightly complex constructs like loops. However, by calling the proper user extensions at run time, simple conditional statements can be executed.

To create a loop I had to:

1. Call the user extension file goto_sel_ide.js in the Selenium IDE. This user extension was written by Darren Derrider, and can be found at http://51elliot.blogspot.com/2008/02/selenium-ide-goto.html. Below is a screen shot from the IDE:



2. Write the code for the loop. The html code is as follows:

```
<!--put auctions to check into an array, will loop through these-->
<tr>
<td>getEval</td>
<td>testAuctions = new Array
(&quot;basic_auction&quot;,&quot;budget01&quot;,&quot;budget02&quot;,&quot;minQty01&quot;,&quot;
testSupply-
Curve&quot;,&quot;budget_proof&quot;,&quot;buyer_minimum&quot;,&quot;Reverse_Auction&quot;);</
td>
<td></td>
</tr>

<tr>
<td>getEval</td>
<td>index=0</td>
<td></td>
</tr>

<tr>
<td>while</td>
<td>index &lt; testAuctions.length</td>
<td></td>
</tr>

<tr>
<td>storeEval</td>
<td>javascript:  testAuctions[index];</td>
<td>aucname</td>
</tr>

<!--select first auction in array-->
<tr>
<td>select</td>
<td>ctl00_ctl00_mainPage_auctionChoose1_wholeControl_ddl1</td>
<td>label=${aucname}</td>
</tr>

<tr>
<td>clickAndWait</td>
<td>ctl00_ctl00_mainPage_auctionChoose1_wholeControl_b3</td>
<td></td>
</tr>

<!--make sure we are at the right auction-->
<tr>
<td>verifyTextPresent</td>
<td>${aucname}</td>
<td></td>
</tr>

<!--check the state of the auction, solved or unsolved?-->
<tr>
<td>storeTextPresent</td>
<td>Solve Auction</td>
<td>auctionstate</td>
</tr>

<!--if unsolved, go to next auction -->
<tr>
<td>gotoIf</td>
<td>storedVars['auctionstate']==true;</td>
<td>keepgoing</td>
</tr>
```

```
<!--If solved, set to unsolved, then go to next auction-->
<tr>
<td>clickAndWait</td>
<td>ctl00$ctl00$mainPage$auctionChoose1$wholeControl$unSolveItBtn</td>
<td></td>
</tr>

<tr>
<td>label</td>
<td>keepgoing</td>
<td>keepgoing</td>
</tr>

<tr>
<td>getEval</td>
<td>index++</td>
<td></td>
</tr>

<tr>
<td>endWhile</td>
<td></td>
<td></td>
</tr>
```

3. Once debugged and working, add the pre-processing script to the test suite. Then, when running Selenium RC, call a a different user extension so the loop would work with RC. This extension is called goto_sel08.js and can be found here: http://wiki.openqa.org/display/SEL/flowControl. The user extension invoked when running the script in the Selenium IDE is not compatible with under RC.

Selector and combo box Madness

The Maax system used several types of combo boxes to select various entities used in an auction: bidders, items, even the auctions themselves. Some types of the combo boxes were easily accessed, while others were opaque objects to Selenium and difficult to act upon.

Below are a couple of instances where I was able to get the selectors to work (I realize without seeing the Maax system itself, it’s hard to put this into context).

Example of choosing from a combo box of roles: Observer, Buyer, Seller, Swapper, etc. In this case the Selenium **type** command is used to select the role of Buyer.

<tr>
<td>type</td>
<td>ctl00_ctl00_mainPage_rightColumn_wholeControl_grid_cell3_3_roleX_I</td>
<td>Buyer</td>
</tr>

Maax had a number of domains used for development, testing, and client demonstrations. The one for QA was called Test Organization. Here is the Selenium code to select Test Organization at log in time:

<tr>
<td>type</td>
<td>ctl00_ctl00_mainPage_rightColumn_login1_notLoggedIn_orgCombo_I</td>
<td>Test Organization</td>
</tr>

In this case the **select** command with **label=** was the better choice for picking from a combo box listing auctions:

```
<tr>
<td>select</td>
<td>ctl00_ctl00_mainPage_auctionChoose1_wholeControl_ddl1</td>
<td>label=basic_auction</td>
</tr>
```

Finally, for some scripts that interacted heavily with the combo boxes, I needed to slow down the speed of that particular test script. For this I used the **SetSpeed** command:

```
<tr>
<td>setSpeed</td>
<td>1000</td>
<td></td>
</tr>
<tr>
```

Summary

At Auctionomics I was able to leverage Selenium work done at my previous company, Playlist.com. In addition, I built a slight more complex test routine in which I used a loop flow control as a part of the test suite. Also, various problem with selectors in different types of combo boxes kept me busy trying different Selenium commands until I found something that worked.

The most important aspect of my Selenium implementation at Auctionomics was keeping organized. In particular:

1. For the loop function, I had to use one flow control user extension file when running tests in the Selenium IDE, but a different user extension when running SeleniumRC.
2. For the various configurations under test (operating system and browser), I had to use two sets of the Selenium libraries: 1.0.1 (for Internet Explorer) and 1.0.3 (for Firefox)

One last note is to mention a great tool that helped me keep my Selenium source files organized: Dropbox. I installed Dropbox on all computers running Selenium. I set up the same file and directory structure for the Dropbox folders on all the machines. Changes made to a Selenium file, batch file, or whatever, were almost immediately and automatically updated on the other machines. This simple, no budget solution was an easy to keep all systems up to date with the latest files.