# Draft: Applied Selenium

## Abstract

A case study on building and maintaining a Selenium RC test suite at Playlist.com.

Selenium test automation can be simple or complex. Start quickly by using the IDE which is simply a Firefox plugin that lets you create, edit, and debug scripts. Start recording, keeping the scripts simple and modular. Expand and deepen your coverage once you've got the hang of it. Fold scripts into a Selenium suite, which functions as a harness, and use with Selenium RC to test browser/os combinations of interest.

## Background

Playlist.com is an web based music discovery site. Users search for music, and create and save playlists. Users can create accounts, upload photos, friend each other, chat, and interact with as in any online social network. Playlist makes money through advertising and revenue share on ringtone purchases. I joined in August, 2008, Playlist had about fifteen developers, and all testing was done by development.

## The Problem

There were several problems I faced when I first started at Playlist:

1. Not enough qa personnel. Initially there was just me, supporting about fifteen developers.
2. There was poor release management, which resulted in the wrong code sometimes being pushed to production. Wrong code could be old code or code still under development.
3. Key features on the production web site often broke because of the release process (noted above) or developers did not test the integration of their changes with the rest of the code base. Examples of problems were:

   • New user registration was broken
   • Music search returned zero results when hundreds of results were expected
   • Users could not upload photos
   • Users could only see a subset of their playlists (often due to memcached problems)
   • Users could not embed the Playlist player at a social networking site or their blog
   • Ad placements or other revenue methods were broken

4. There was a backlog of testing (see #1 above).
5. There were new features being developed, and there was pressure to prioritize testing of these over the backlog of testing.

A common problem at start ups (and maybe bigger companies too): little budget for qa personnel, more testing than your qa department can handle, and often a weekly release, meaning shortened test cycles. Fortunately there tools that are force multipliers for understaffed, busy organizations.

## Choosing selenium

To extend my testing reach, I researched current, popular automation tools. Automation can be a lot activity without much progress, but in recent years automation tools have matured such that scripts can be created and maintained with much less time investment than even a few years ago.[1] I evaluated several automation tools, some open source, others proprietary: selenium, watir, twill, web replay, and squish. Somewhat on the merits of an O'Reilly article (http://oreilly.com/catalog/9780596527808/), I felt both selenium and twill looked to be the best choices for Playlist: both were proven automation tools, free, and there was ample support available.

Twill scripts would be used for a different, relatively simple testing (see my article on using twill at Playlist). Selenium would be our primary automation tool. Selenium offered a number of benefits:

1. A proven track record of successful web automation projects
2. Cross browser and operating system support via Selenium RC - this feature was a huge boast to our test effort.
3. Wide spread adoption by the development community (Google, IBM, Adobe)
4. Decent support, forums, and resources
5. Continued development of the tools by a dedicated, competent community
6. Very easy to learn and get started, yet extensible for more sophisticated testing via a variety of scripting languages
7. Free, and fun to work with

Selenium appealed to me for the benefit of not requiring a programming background to implement, yet at the same time being programmatically extendable, such as for conditions and loops, with common scripting languages such as javascript and python. That meant with my limited programming background, I could start creating scripts immediately, using the native HTML tables, or Selenese. Later, we could looking into extending the automation by integrating with a true scripting language.

## Details

### Team and environment

Initially I was the only one working with Selenium. I soon hired a senior qa engineer to help. All work was on Windows systems running either XP or Vista. We usually built and debugged on our primary computer, but then executed the tests on another system. We also ran the Selenium scripts on the Mac, with no modifications required of the script files, and only minor tweaks needed for harness files.
Tests were run against the operating system and browser combinations that owned significant market share. For example: IE7 on Windows XP, Firefox 3 on Windows and XP, and Safari on Mac OSX probably reached 90% of our audience.

1. I've run successful automation projects at Frame (using the Framemaker API/ SDK, and QA Partner) and Visioneer using Silk. By far the best discussion on automation is in *Lessons Learned in Software Testing* by Cem Kaner, James Bach, and Bret Pettichord (New York: John Wiley and Sons, 2002), see chapter 5.

We did not use any source code control, but kept that as an option if we grew.

## Which Selenium?

See the links in the Reference section below for detailed discussions of each Selenium. In our case we used:

**Core**: the original Selenium implementation and now also a part of the Selenium RC package. This version is installed to a directory on your web server and runs the scripts on the server. We first attempted to use Selenium Core but quickly abandoned it. We learned Selenium RC was more useful for our needs; in addition, the directory where the Selenium Core files resided was frequently deleted by other engineering groups fiddling with the test server

**Selenium IDE**: the Firefox Plugin which creates native HTML (or selenese) scripts. We used this initially to create tests and debug tests. Often we would start scripting a page using the IDE, but then finish by writing in commands directly in the file using a text editor (we liked Notepad ++).

**Selenium RC**: a locally installed HTTP proxy server that bundles Selenium Core. This server launches and closes browser instances, and interprets and executes commands from test scripts in Selenese or a supported scripting language. Even better, RC reports back results from test scripts.

At Playlist we settled in with the IDE and RC. Eventually, Selenium **Grid** might have been of interest to accelerate the a test run, but we had not yet reached that stage. See the reference section for information on Grid.

## What to test?

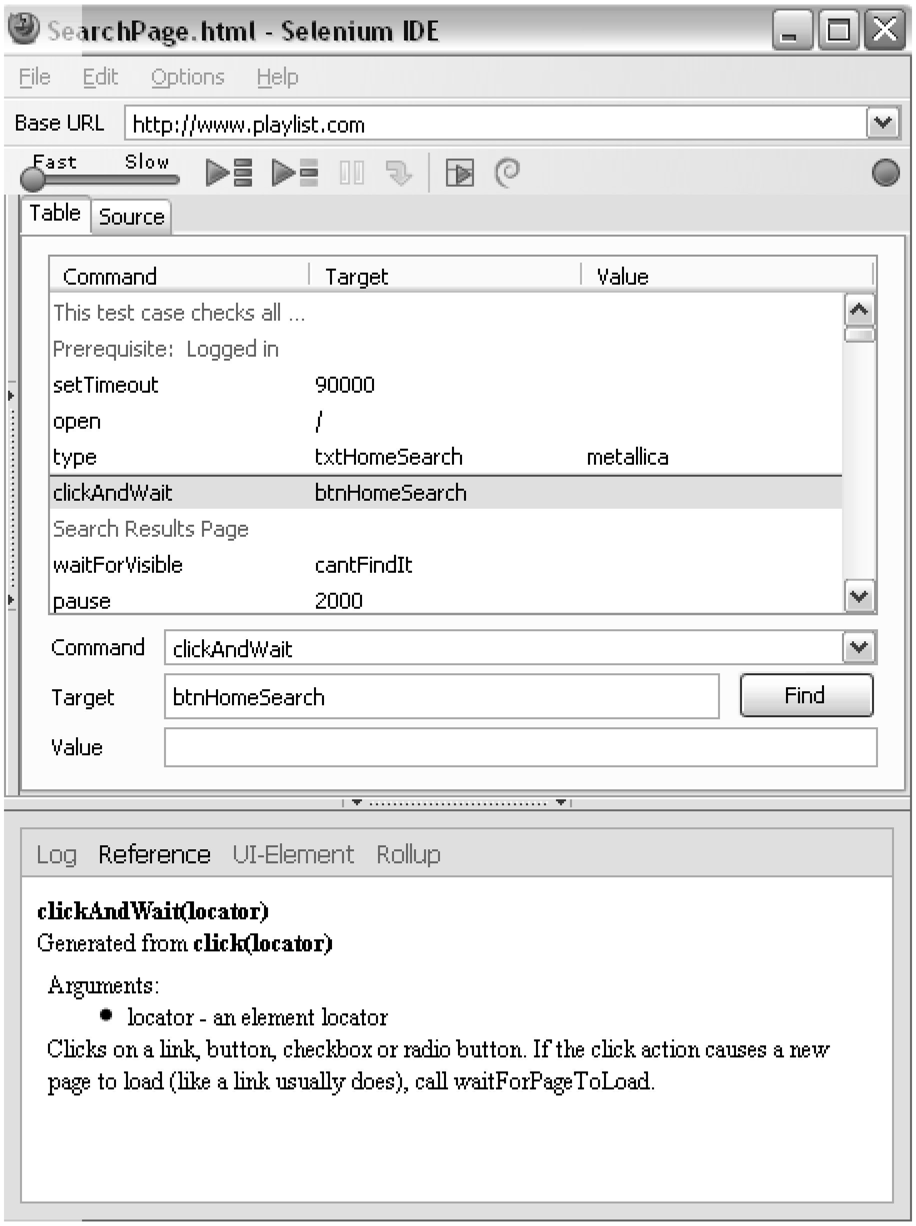
It all starts with figuring out what you want to test, and then after that, what you want to automate. Do that first. At Playlist, our list was driven by features (pages) that had the highest usage or visitation, and also by features that drove revenue. When done with our initial implementation, we broke out the tests into thirty different test scripts, each testing a different module. These were:

| | | |
|---|---|---|
| Home page | Manage songs tab | Add mp3 link |
| New account creation | Ringtones | Search Page |
| Search and add music | View songs tab | Comment post |
| Account settings | Friend connection | Feature a friend |
| About me dashboard | Invite friends | Create custom url |
| My Blog | Change featured playlist | Forums |
| Create new playlist | Get player embed code | Private messages |
| Edit playlist | Dashboard | Group Playlists |
| Logout | Forgot password | Purchase music |

Get help from dev as needed to determine what should be automated. See also the books and websites in the reference section for guidance in what to test.

## A typical script

We used the Selenium IDE and along with editing in Notepad++ to create a script.Below is a screen shot followed by an excerpt of the same scripts that works on the music search page (admittedly, this is not very interesting, but for brevity I did not want to include the whole file, which runs to over six hundred lines):


SearchPage.html - Selenium IDE
File Edit Options Help
Base URL
Fast Slow
Table Source
Find
Log Reference UI-Element Rollup

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

| Command | Target | Value |
|---|---|---|
| This test case checks all ... | | |
| Prerequisite: Logged in | | |
| setTimeout | 90000 | |
| open | / | |
| type | txtHomeSearch | metallica |
| clickAndWait | btnHomeSearch | |
| Search Results Page | | |
| waitForVisible | cantFindIt | |
| pause | 2000 | |

Command: clickAndWait
Target: btnHomeSearch
Value:

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

Arguments:
- locator - an element locator

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.

Now the first forty lines or so of the script as seen in Notepad++:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/
DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head profile="http://selenium-ide.openqa.org/profiles/test-case">
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<link rel="selenium.base" href="" />
<title>SearchPage</title>
</head>
<body>
<table cellpadding="1" cellspacing="1" border="1">
<thead>
<tr><td rowspan="1" colspan="3">SearchPage</td></tr>
</thead><tbody>
<!--This test case checks all elements on the Search Results page.  Verifies 10 search
results appear, verifies all elements in 1st search result, sends song to friend, goes
to visit site link, goes to get ringtone link, goes to search page for artists featured
in playlist, checks pagination (next, previous, etc.), and Searches for EMI blocked
artist and verifies alert message for EMI blocked artists.-->
<!--Prerequisite:  Logged in-->
<tr>
             <td>setTimeout</td>
             <td>90000</td>
             <td></td>
</tr>
<tr>
             <td>open</td>
             <td>/</td>
             <td></td>
</tr>
<tr>
             <td>type</td>
             <td>txtHomeSearch</td>
             <td>metallica</td>
</tr>
<tr>
             <td>clickAndWait</td>
             <td>btnHomeSearch</td>
             <td></td>
</tr>
<!--Search Results Page-->
```

Again, not super interesting to look at.

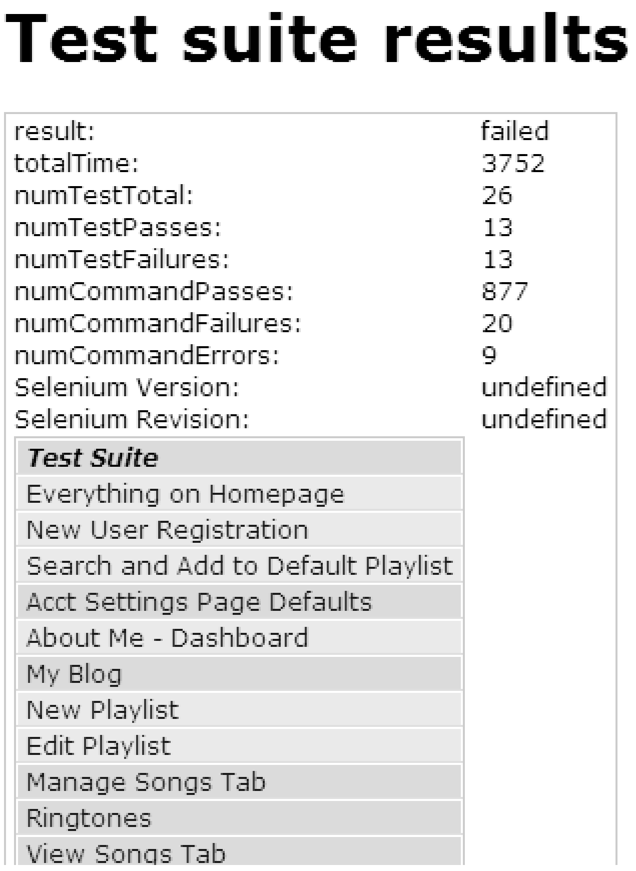
## Many scripts, one suite

Soon qa had written about thirty scripts. There were some interactions between the scripts, but at least twenty could be run independently. Next we created a test **suite** file to contain all the test scripts. This was the file that would be passed into Selenium RC. The test suite file is show here as it appears in Notepad++:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/
DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
  <meta content="text/html; charset=UTF-8" http-equiv="content-type" />
  <title>Test Suite</title>
</head>
<body>
<table id="suiteTable" cellpadding="1" cellspacing="1" border="1" class="sele-
nium"><tbody>
<tr><td><b>Test Suite</b></td></tr>
<tr><td><a href="Homepage_Guest.html">Everything on Homepage</a></td></tr>
<tr><td><a href="Signup.html">New User Registration</a></td></tr>
<tr><td><a href="Search_AddSongs.html">Search and Add to Default Playlist</a></td></tr>
<tr><td><a href="Settings_AcctSettingsTabDefaults.html">Acct Settings Page Defaults</
a></td></tr>
<tr><td><a href="Dashboard_AboutMe-Settings_ProfileInfo_MusicTastes.html">About Me -
Dashboard</a></td></tr>
<!--<tr><td><a href="Photos.html">My Photos</a></td></tr>-->
<tr><td><a href="Blogs.html">My Blog</a></td></tr>
<tr><td><a href="Playlist_New.html">New Playlist</a></td></tr>
<tr><td><a href="Playlist_Edit.html">Edit Playlist</a></td></tr>
<tr><td><a href="Playlist_ManageSongs.html">Manage Songs Tab</a></td></tr>
<tr><td><a href="Ringtones.html">Ringtones</a></td></tr>
<tr><td><a href="Playlist_ViewSongs.html">View Songs Tab</a></td></tr>
<tr><td><a href="Friend_Connections.html">Friend Connection</a></td></tr>
<tr><td><a href="Friends_Invite.html">Invite Friends</a></td></tr>
<tr><td><a href="Settings_FeaturedPlaylist_Change.html">Change Featured Playlist</a></
td></tr>
<tr><td><a href="Get_PlayerCode.html">Get Player Code</a></td></tr>
<tr><td><a href="Dashboard.html">Dashboard</a></td></tr>
<tr><td><a href="Add_MP3link.html">Add MP3 Link</a></td></tr>
<tr><td><a href="SearchPage.html">Search Page</a></td></tr>
<tr><td><a href="CommentUser.html">Comment Poster</a></td></tr>
<tr><td><a href="Friend_Featured.html">Feature a Friend</a></td></tr>
<tr><td><a href="Settings_CustomURL.html">Create Custom URL</a></td></tr>
<tr><td><a href="Forums.html">Forums</a></td></tr>
<tr><td><a href="PrivateMessages.html">Private Messages</a></td></tr>
<tr><td><a href="Playlist_Group.html">Group Playlist</a></td></tr>
<tr><td><a href="Logout.html">Logout</a></td></tr>
<tr><td><a href="Forgot_Password.html">Forgot Password</a></td></tr>
</tbody></table>
</body>
</html>
```

## Results

When a test suite run is complete, an html results file is created. This file is written locally but also mailed around to the qa team (see below in the information about Blat). Below are screen shots first of the summary of a test run, then a detail of the test report.

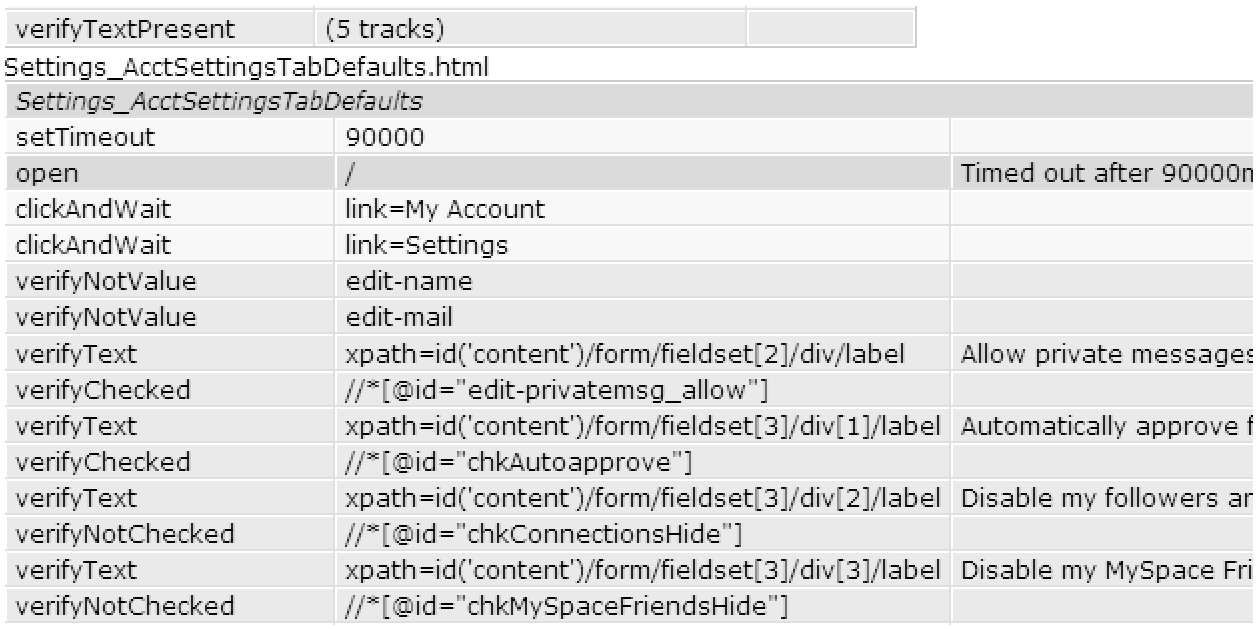


# Test suite results

| | |
|---|---|
| result: | failed |
| totalTime: | 3752 |
| numTestTotal: | 26 |
| numTestPasses: | 13 |
| numTestFailures: | 13 |
| numCommandPasses: | 877 |
| numCommandFailures: | 20 |
| numCommandErrors: | 9 |
| Selenium Version: | undefined |
| Selenium Revision: | undefined |

| *Test Suite* |
|---|
| Everything on Homepage |
| New User Registration |
| Search and Add to Default Playlist |
| Acct Settings Page Defaults |
| About Me - Dashboard |
| My Blog |
| New Playlist |
| Edit Playlist |
| Manage Songs Tab |
| Ringtones |
| View Songs Tab |

Detailed results:



| | | |
|---|---|---|
| verifyTextPresent | (5 tracks) | |

Settings_AcctSettingsTabDefaults.html

| *Settings_AcctSettingsTabDefaults* | | |
|---|---|---|
| setTimeout | 90000 | |
| open | / | Timed out after 90000n |
| clickAndWait | link=My Account | |
| clickAndWait | link=Settings | |
| verifyNotValue | edit-name | |
| verifyNotValue | edit-mail | |
| verifyText | xpath=id('content')/form/fieldset[2]/div/label | Allow private messages |
| verifyChecked | //*[@id="edit-privatemsg_allow"] | |
| verifyText | xpath=id('content')/form/fieldset[3]/div[1]/label | Automatically approve f |
| verifyChecked | //*[@id="chkAutoapprove"] | |
| verifyText | xpath=id('content')/form/fieldset[3]/div[2]/label | Disable my followers ar |
| verifyNotChecked | //*[@id="chkConnectionsHide"] | |
| verifyText | xpath=id('content')/form/fieldset[3]/div[3]/label | Disable my MySpace Fri |
| verifyNotChecked | //*[@id="chkMySpaceFriendsHide"] | |

At present the thirty script files we wrote make about 1,000 checks and take about twenty-five minutes to run. As previously mentioned, we'd like to shorten the run time to about ten minutes, and there are a few ways to do this, one using Selenium Grid.

## Other notes

One set of tests, uploading photos to your Playlist account, required that some images be available at a pre-determined location on your local machine, e.g. C:\PlaylistPhotos\. I believe this was one of the few local dependencies, and otherwise the scripts were very portable across computers and operating systems.

## File locations, Selenium RC, batch files, Blat

Here's how directories were set up on a typical PC running Vista:

- Test Case and Test Suite - C:\Users\blake\Selenium
- Selenium RC installation - C:\SeleniumRC_Trunk
- Batch Files - C:\Users\blake\Selenium\Batch
- Blat Email Files - C:\Users\blake\Selenium\Blat_emails

And the same on a Mac:

- Test Case and Test Suite - MacintoshHD/Users/blake/Selenium_TestCases
- Selenium RC installation - MacintoshHD/Users/blake/SeleniumRC_Trunk
- Batch Files - MacintoshHD/Users/blake/SeleniumRC_Trunk/Batch

The Selenium RC web site has all the information you need about setting up Selenium RC. Once we had everything ready to go, a batch file would fire off the Selenium RC commands:

```
java -jar selenium-server.jar -timeout 5000 -htmlSuite *iehta http://stag-
ing.playlist.com""c:\absolute\path\to\my\TestSuite_Playlist.html" "c:\abso-
lute\path\to\my\results.html"
```

The above command does the following:

1. Launches Selenium RC (`java -jar selenium-server.jar`)
2. Sets an absolute timeout to stop the suite if it does not finish earlier (`-timeout 5000`)
3. Passes in a suite to Selenium (`-htmlSuite`)
4. Tells RC what browser to launch (`*iehta`)
5. Tells RC what environment to test against (`http://staging.playlist.com"`)
6. Tells RC the suite file name and path (`"c:\absolute\path\to\my\TestSuite_Playlist.html"`)
7. Tells RC where to log the test results (`"c:\absolute\path\to\my\results.html"`)

On Windows we used **batch** files to make things easier. Here is an example of a batch file we ran if we wanted to run the test suite on our production environment using Firefox3:

```
cd\
cd \SeleniumRC_Trunk\selenium-remote-control-1.0-SNAPSHOT\selenium-server-1.0-SNAPSHOT

Blat - -body "<eom>" -to qateam@playlist.com -i automation@playlist.com -sf
C:\Users\blake\Selenium\Blat_emails\Started_WWW_FF3.txt
```

```
java -jar selenium-server.jar -timeout 5000 -htmlSuite *firefox "http://www.play-
list.com" "C:\Users\blake\Selenium\TESTSUITE_Playlist.html" "C:\Users\blake\Sele-
nium\results.html" > sel_stdout.txt

Blat C:\Users\blake\Selenium\Blat_emails\body.txt -to qateam@playlist.com -i automa-
tion@playlist.com -sf C:\Users\blake\Selenium\Blat_emails\Finished_WWW_FF3.txt -ps
sel_stdout.txt -attach C:\Users\blake\Selenium\results.html
```

**Blat** is a Windows command line tool that emails the contents of a file via SMTP. We configured Blat to send emails as notification when a suite run was started, send another email when a suite finished, and include the results of the test run. On the Mac this was easier as we just used shell scripts and could call SMTP directly.

## Next steps

Below are some improvements I'd like to make to our Selenium suite:

1. Review the method of identifying objects. There are several ways Selenium can locate an object in the page: id, name, css, or Xpath. We may have made our tests too brittle by using Xpath as the primary means of finding an object. Our scripts would sometimes report errors when only minor, known, and acceptable changes had been made to a page.
2. Are we using the Selenium in an optimal manner? There are any number of places automation can play a role in the development cycle:
   • smoke or acceptance test
   • unit tests (there are tools supporting integrating Selenium with Junit and TestNg)
   • load tests
   • performance benchmarks
   • configuration testing
   • reliability or endurance testing

Depending on how implemented, Selenium can play a role in all of these.

3. Some what related to #2 above, review and expand when in the development cycle we run the Selenium automation. Right now the automation is run on staging and production, that is, post integration. The earlier we can run Selenium, or even a subset of our suite, the sooner we can find bugs. Ideally these scripts could be run by developer as a part of their unit testing, and also after a nightly build process, once we have that in place.
4. As already mentioned, look into extending with scripting language. I'd especially like the ability to loop through a list of inputs from a file, for example to created a 200 track playlist. I did this with Web Replay, and would like to extend Selenium as such.
5. Somewhat related to #3, get input and review from development on our script creation, and suggestions for improving the how we script and test a page or module. To borrow from what one developer said, "Incorporate any refactoring suggestions to make more object oriented as needed."
6. Be able to test the flash widget. Adobe and the Selenium have developed some tools to support this. For the most part I treated the widget as an opaque object, and mostly tested around it. See the Reference section below for links about Selenium and Flash.

## ROI

Was it worth the effort? How did Selenium help address the problems I outlined at the beginning of this paper? I give our first score a B-, here’s why:

1. We stopped shipping the big, stupid bugs such as new accounts cannot be created. Mostly.
2. For features that rarely changed, we could script those pages, then not worry about them. This freed up people to work on other things.
3. The scripts often caught changes that developers had forgotten to tell us about.

Yet we’ve just scratched the surface of what we can do. So in all, considering we have at most a couple of months invested, it’s a great start.

## Last thoughts

You must continually review your automation plan to make sure it is working *efficiently*. Keep your automation effort in perspective. Automation is a part of a larger qa effort, which serves development, your customers, and therefore your business. Make sure your automation contributes to your process, and is not simply a project of it’s own. I mentioned earlier about automation being a force multiplier, but it can also be a force divider, robbing your other projects of attention while you keep busy with what’s turned into a science project. No science projects!

Good luck!

## Reference

Most topics you can search the internet on, but the links below should save you some time.

### Selenium

Start here for Selenium: http://seleniumhq.org/

The **Google group**: http://groups.google.com/group/selenium-users

Selenium and **Flash/Flex**: http://www.adobe.com/devnet/flash/articles/flash_selenium_print.html

**IBM** has a lot of good online information. Here’s an article on using Selenium and TestNG: http://www.ibm.com/developerworks/java/library/j-cq04037/.

Search on Selenium at **YouTube** and **SlideShare**. There are lots of good references here, also.

A company making Selenium tools, including a Grid like tool, and where the first author of Selenium works: http://saucelabs.com//

**Ajax** and Selenium (I have not researched everything at InfoQ): http://www.infoq.com/articles/testing-ajax-selenium

## Blat

http://www.blat.net/

## O’Reilly article on twill and selenium

http://oreilly.com/catalog/9780596527808/

## Books

There area lot of great test books out there, but I’d start with *Lessons Learned in Software Testing*: http://www.amazon.com/Lessons-Learned-Software-Testing-Kaner/dp/0471081124/ref=sr_1_1?ie=UTF8&s=books&qid=1263407512&sr=8-1