Introducing CFSelenium - A Native ColdFusion Client Library for Selenium-RC

Posted At : February 22, 2011 9:34 AM | Posted By : Bob Silverberg
Related Categories: CFSelenium, Selenium

We were very lucky to have Adam Goucher, a member of the core Selenium team and the maintainer of Selenium-IDE, as a speaker at the Toronto ColdFusion User Group this month. Adam delivered an awesome presentation about automated testing in general and Selenium in particular.

During the session I asked a bunch of questions about the architecture of the Selenium components, in particular Selenium-RC and WebDriver, which is now part of Selenium 2.0. Through our conversation I came to understand that it would be fairly simple to create a native ColdFusion client library for Selenium-RC, so I decided to give it a try. Thankfully it was as simple as I expected and I decided to complete the library so that the full API of Selenium-RC is supported.

Thus CFSelenium was born and was given a home at GitHub. If you have no idea what I'm talking about, fear not, I will provide some more background information on Selenium and Selenium-RC in the rest of this post.

What is Selenium?

At its most basic Selenium is a tool that lets you automate a web browser, which is mostly used for automated testing. With it you can instruct a browser to do pretty much anything that a user of your web application might do, and you can ask the browser questions about the resulting Document Object Model (DOM), the answers to which can be fed into asserts in tests. Selenium works by injecting JavaScript into the browser which is then used to control the browser. Selenium is free and open source.

What is Selenium-IDE?

Selenium-IDE is a Firefox plugin which can be used to record and playback scripts which automate the browser. It is an excellent entry point into using Selenium as it's extremely easy to set up and use, and you can create complete tests in it that include verifications and asserts. As you can walk through your web app recording actions as you proceed it is a nice way of getting familiar with Selenium commands, and I've used it in very simple tests and one-off tests quite a bit, but when you start getting into serious functional tests you generally want to use an actual programming language to define and drive your tests, not to mention allowing tests to be run on browsers other than Firefox. That's where Selenium-RC comes in.

What is Selenium-RC?

Selenium-RC, which stands for *Selenium Remote Control*, consists of two parts:

1. The Selenium Server, a Java application that launches and kills browsers, and controls the browsers by interpreting Selenese commands that have been passed to it.
2. Client libraries, which provide the interface between each programming language and the Selenium RC Server.

There are some major advantages to using Selenium-RC for your functional tests rather than Selenium-IDE, including:

- Your tests can now be executed against practically any browser that supports Javascript. Specifically Selenium-RC supports Firefox, Internet Explorer, Safari, Opera and Google Chrome, but you can control other browsers as well using the **custom* browser command.
- You can write your tests using a *real* programming language which means you can include loops and conditional logic, as well as programmatically generating test data and/or pulling data from a file or database.

Prior to CFSelenium, client libraries were available for Java, Python, Ruby, C#, Perl, PHP, and perhaps others that I'm missing. Although it's possible to control the RC server via ColdFusion code using the Java client library, I thought it would be interesting to see if I could create a library that used only ColdFusion, as this seemed to offer a couple of advantages:

1. It's easier to get started with as you don't need to find and load the Java driver.
2. As the cfc implements the complete API, you can get code assist when using the component from within ColdFusion Builder.

Using the Selenium-RC server and the new ColdFusion client library you can now write tests using your favourite ColdFusion testing framework (e.g., MXUnit).

Requirements

Because of the way the cfc was written, you must have ColdFusion 9.0 or above to use it. There is no functionality required that is not present in earlier versions of ColdFusion, but I chose to write the component as a pure script component, which is why the requirement of CF9 exists. It would not be terribly difficult to change the format of the component to work under earlier versions of CF if someone was so inclined to do so.

In order to actually run tests you need to have a copy of the Selenium RC Server jar, but a copy is included with the distribution, in a folder called *Selenium-RC*.

How to Use It

Step 1 - Start the RC Server

Simply start the server as you would any other Java program. I'm on OS X, so I just open up a terminal window, navigate to the folder that contains the jar, and execute the following command:

```
java -jar selenium-server-standalone-2.0b2.jar
```

Step 2 - Create an Instance of the ColdFusion Client Library

The remaining steps are all code that is likely to exist inside a test method, although steps 1 and 2 might be in *setup()* and step 5 might be in *teardown()*. You simply instantiate an instance of the selenium.cfc component that comes with the distribution. The only required argument to the constructor it the url from which you wish to start your test. Here's an example:

```
selenium = new selenium("http://github.com/");
```

You can optionally pass in arguments for the *host* and *port* on which the RC Server is running (which default to *localhost* and *4444*, respectively) as well as the browser that you wish to launch (which defaults to Firefox). For example:

```
selenium = new selenium("http://github.com/","localhost",4444,"*googlechrome");
```

Step 3 - Launch the Browser

To launch the browser you call the *start()* method on the selenium object:

```
selenium.start();
```

Step 4 - Control the Browser and Do Asserts

You now work with the selenium object to control the browser and interrogate the resulting DOM to feed data to your asserts. For example:

```
selenium.open("/bobsilverberg/CFSelenium");

assertEquals("bobsilverberg/CFSelenium - GitHub", selenium.getTitle());

selenium.click("link=readme.md");

selenium.waitForPageToLoad("30000");

assertEquals("readme.md at master from bobsilverberg's CFSelenium - GitHub", selenium.getTitle());

selenium.click("raw-url");

selenium.waitForPageToLoad("30000");

assertEquals("", selenium.getTitle());

assertTrue(selenium.isTextPresent("[CFSelenium]"));
```

The above test would:

1. Navigate to the page *http://github.com/bobsilverberg/CFSelenium*.
2. Verify that we're on the correct page by asserting that the page title is *bobsilverberg/CFSelenium - GitHub*.
3. Click on the anchor tag with the link text of *readme.md*, and wait for the resulting page to load.
4. Verify that we're on the correct page by asserting that the page title is *readme.md at master from bobsilverberg's CFSelenium - GitHub*.
5. Click on the anchor tag with the id of *raw-url*, and wait for the resulting page to load.
6. Verify that we're on the correct page by asserting that the page title is an empty string.
7. Verify that we're on the correct page by asserting that the text *[CFSelenium]* is found on the page.

Step 5 - Kill the Browser

To kill the browser you call the *stop()* method on the selenium object:

```
selenium.stop();
```

You should always kill the browser at the end of a test, pass or fail, so this is a good candidate for *teardown()*.

What's Next?

Testing

I admit to having done only rudimentary testing on the client library thus far, and I've already identified and fixed one bug. I have had a few folks volunteer to help me test the library, for which I am grateful, but I'd love to have more help. If you're interested in giving it a try I'm happy to help you get set up and attempt to answer any questions you may have. Either send me an email or add a comment to this post to let me know if you'd like to help.

A Formatter

I'd like to add a formatter to Selenium-IDE which would allow you to export a script built in the IDE into an MXUnit test that uses the ColdFusion client library. Adam has been giving me some advice on this and I hope to have something in the next week or two. In the meantime, if you export from Selenium-IDE into Java format you can pretty much take the bulk of the generated code and drop it into a test method in an MXUnit test case as is.

Debugging

I'm thinking of adding a debug mode which would automatically capture information to make debugging your tests easier. When turned on this would slow the tests down tremedously, but it might be a nice to have feature for those times that your tests aren't working and you're not sure why.

In Conclusion

I'd like to thank Adam again for his help and if anyone has any questions about, or suggestions for, the client library please let me know.