Selenium

Selenium is a portable software testing framework for web applications. Selenium provides a record/playback tool for authoring tests without learning a test scripting language. Selenium provides a test domain specific language (DSL) to write tests in a number of popular programming languages, including Java, Ruby, Groovy, Python, PHP, and Perl. Test playback is in most modern web browsers. Selenium deploys to Windows, Linux, and Macintosh platforms.

There are differing approaches to using Selenium in the Project WOnder community and code base. Tests can be written as tables in HTML or in a more concise format, a text file with the extension ".sel" where the parts of a command are separated by a '|' character. Tests can be running using a Firefox extension called "Selenium IDE" or that can be run from the command-line using the "Selenium-RC" classes. Tests can be run from within a Project WOnder application using the ERSelenium direct action. Tests can also be run with an ant task or using the Hudson/Jenkins continuous integration system.

Project Wonder makes it easy to integrate Selenium testing into your own WebObjects application using the ERSelenium framework from Project Wonder. The advantage of this is that test tools do not have to be installed on non-development machines to run the tests. Also the tests are part of the project resources and thus ERSelenium usage for Selenium testing is most effective for WebObjects development teams who are using source code management tools such as subversion. Finally ERSelenium provides meta tags that allow you to extend your Selenium tests. The operation of ERSelenium test script meta tags is similar, for example, to the use of server side includes in html files in apache.

Resources

- http://www.openqa.org/selenium/
- Neal Ford's NJFS presentation slides on Selenium
- http://www.openqa.org/selenium-core/usage.html
- http://www.openqa.org/selenium-rc/tutorial.html
- a short video on recording tests by using the Selenium IDE

Creating Tests

Selenium uses a basic "action-wait-assert" pattern in testing the UI of a web application. Commands do one or more of these things. In general, "action" commands start with "click", "select" or "type", "wait" commands are "pause" or "waitFor", and the "assert" commands start with "assert" or "verify". Generally, every "assert" command has a "verify" counterpart, such as "assertText" and "verifyText". The difference is that if something is asserted and found not to be true, then the test will abort as the test fails. If something is verified and found not to be true, the the test will fail and the tests will continue to run. So, in order to write robust tests, one would want to verify if one can, but abort if a particular test failure will prevent the running of other tests. Note that a test failure may prevent the **correct** running of a set of other tests, but you may still want to let those tests run. More errors may lead to more information. There are also "store" commands that can be used to store and examine state in a running test.

A command has three parts: a name, a locator, and an optional parameter. Figuring out the names of desired commands is usually not difficult, though plugins can be added to Selenium that complicate the question. Writing locators is often the hardest part of test writing. There are different approaches to this that are described in the literature one will find. In general, locators will either find some text, and one needs to provide the text to find, or they will use xpath to find an element or reference the dom tree directly. Other strategies are documented as well.

One can use tools, such as the Selenium IDE, to record tests. This will result in some strange looking locators, but they will work. Or one can write web applications in such a way that important things are easy to find. The liberal use of "id" parameters on HTML tags, for example, will ensure this. It is arguably easier to record tests, and creating tests by recording them means that one does not have to change one's application, but the resulting tests will turn out to be somewhat fragile. Any change in the page source, regardless of whether or not it changes the meaning of a test, will probably necessitate a re-generation of the test. However, if one is using identifiers on the tags in the application, one can document the intent of the test. The page may be changed, but if the logical meaning of the elements does not change, then one may presumably not need to change the tests. Finding the right labels to use on the correct tags to mark up the logical structure of the page is, though, a skill that must be learned and some people will be better at it than others.

For example, here is an example test in "Selenese":

```
|open|<a web page URL>||
|assertText|Hello world||
|click|link="Do you hear me?"||
|pause|1200||
|verifyText|Yes, I hear you.||
```

Also see: How to Develop Selenium Tests

Running Selenium Tests

If you are using ERSelenium, then you can start tests in your application by going to the Direct Action URL: http:///cgi-bin/WebObjects/.woa/wa /SeleniumStartTesting This will bring up a page that has a test navigator/launcher in a frame in the top of the window. This page shows page content in the frame below that. In the right hand box of the top frame are buttons to start all tests, or to start a particular test after it has been selected in the left-hand box, which shows a list of all available tests.

When using Firefox, tests can be launched from within the Selenium IDE window, if that add-on is installed. See http://addons.mozilla.org/en-US/firefox /addon/2079for more information.

Tests can be launched from the command-line. There is, right now, only limited support for this option. For example, tests written in "selenese" (in files using the .sel exptension) cannot be loaded in this manner. The tests need to be written in html. It is possible that the build system could translate .sel files to .html files so that this would work. Note that the selenium-server.jar file is not, at this time, distributed wit Project Wonder. Construct the TestSuite.xml file as demonstrated in http://seleniumhq.org/docs/02_selenium_basics.html#test-suites. Launching is as shown here:

```
java -jar /wo/lib/selenium-remote-control-1.0.1/selenium-server-1.0.1/selenium-server.jar \
 -htmlSuite '*firefox' \
    http://localhost:55555/cgi-bin/WebObjects/AjaxExample.woa/ \
    /wo/pwo/Wonder/Examples/Ajax/AjaxExample/Tests/AjaxTestSuite.xml \
    /wo/pwo/Wonder/Examples/Ajax/AjaxExample/Tests/Ajax_Results_SomeOther.html
```

Useful Links:

- Project WONDER-Frameworks-ERSelenium
- How to Develop Selenium Tests
- Testing-Load Testing WO Apps with JMeter
- How to test a Diva app with Selenium
- Specifying Robust Selenium Element Locators
- Selenium IDE Flow Control - Goto and While Loops
- Selenese Selenium Commands
- Selenium Reference