Selenium Remote Control: Tutorial

This Selenium Remote Contrtol tutorial will teach you how to start the Selenium Server from the command line, and how to use the Server in interactive mode. It assumes that you are familiar with running programs from the command line. At the end, we'll demonstrate how to write a simple test that does the same thing as what we did in interactive mode.

The Selenium Server is written in Java, and requires the Java Runtime Environment (JRE) in order to start. You may already have it installed. Try running this from the command line:

```
java -version
```

You should see a brief message telling you what version of Java is installed. If you see an error message instead, you may need to install the JRE, or you may need to add it to your PATH environment variable.

For this example, you'll also need to make sure that you have a supported browser installed.

- **Windows**: If you're on Windows XP or Windows 2003, you can just use Internet Explorer for this example, or install Mozilla Firefox or Opera. If you're using Windows 2000, you'll need to install reg.exe in order to use Internet Explorer, but Firefox should work regardless. We recommend (but do not require) that you add your browser executable to your PATH environment variable. (If you do not explicitly add your browser's installation directory to the PATH, then you must install your browser in its standard location; Firefox's standard location is "c:\Program Files\Mozilla Firefox\firefox.exe"; Internet Explorer's standard location is "c:\Program Files\Internet Explorer\iexplore.exe".)
- **Unix/Linux**: For this tutorial, install Firefox and add the Firefox directory to your PATH environment variable. Note that on Unix/Linux we'll be trying to invoke "firefox-bin" directly, so make sure *that* executable is on the path; also don't forget to add the Firefox libraries to your LD_LIBRARY_PATH. If needed, we can invoke Firefox using a shell script (e.g. "firefox" or "run-mozilla.sh"), but in that case, we may not be able to stop Firefox until the server is shut down.
- **Mac OS X**: On Mac OS X, it should be enough to install Firefox.app in your /Applications directory. Note in order to control the browser accurately, we need to invoke the embedded Firefox executable (firefox-bin) directly in /Applications/Firefox.app/Contents/MacOS; if your browser isn't installed there, then you'll want to add the correct embedded location to your PATH environment variable as well as your DYLD_LIBRARY_PATH environment variable.

Interactive Mode

Selenium Server "interactive mode" is a way of rapidly prototyping tests that requires no coding whatsoever, so it's a good way to introduce new users to Selenium Remote Control. In interactive mode, you type your commands one by one into the Selenium Server command window; this allows you to immediately see the results of running your command in a working browser, on-the-spot. With that said, normally you'll be coding these tests in your favorite programming language, so the whole thing is completely automated.

Once you've got Java installed and ready to go, you can start the Selenium Server from the command line like this:

```
java -jar selenium-server.jar -interactive
```

That will start the Selenium Server and allow you to type commands in the command window. After a number of log messages, you should see the following message:

```
Entering interactive mode... type Selenium commands here (e.g:
cmd=open&1=http://www.yahoo.com)
```

Let's begin by using Selenium Server to open up a browser. If you're on Windows and want to run your Selenium commands using Internet Explorer, try typing this:

```
cmd=getNewBrowserSession&1=*iexplore&2=http://www.google.com
```

If you want to use Firefox, try this:

```
cmd=getNewBrowserSession&1=*firefox&2=http://www.google.com
```

You're running the "getNewBrowserSession" command, using the browser of your choice (*iexplore for Internet Explorer, *firefox for Firefox, or *opera for Opera), starting at www.google.com. Typing commands in this window automatically fires off HTTP web requests to the Selenium Server, requesting work to be done. (In non-interactive mode, you can use any automated tool you like to send these HTTP requests to the server, instead of typing them by hand.) When you press Enter, you'll see a message describing the request you just made:

```
---> Requesting
http://localhost:4444/selenium-server/driver?cmd=getNewBrowserSession&1=*firefox&2=http://www.goog
```

If all goes well, you should see a new browser window starting with the browser of your choice. Back in the Selenium Server command window, you should see the following message:

```
OK,1143670264928
```

(If this doesn't happen, you may need to take a look at our fine Troubleshooting guide.)

The first part of this message "OK" says that your request for work was successful. The second part of this message, the number, is a Session ID. The Session ID will be different every time you run the "getNewBrowserSession" command from the command line.

Let's try some more commands; let's do a Google search! We'll start by opening up the Google website. Type this in the Selenium Server command window, replacing the Session ID below with the number you got when you ran "getNewBrowserSession":

```
cmd=open&1=http://www.google.com/webhp&sessionId=1143670264928
```

When this command finishes, your browser window should reveal google.com in the lower frame. (The */webhp* sends us to a page Google maintains which is not subject to redirects to country-specific Google front ends, e.g., *www.google.fr*.) Don't forget that you need to replace the Session ID above with your own personal Session ID, the one you got when you ran the "getNewBrowserSession" command. (If you use the wrong Session ID, you won't see an error message; the Selenium Server will just sit there, waiting for some browser to come in and do its work.)

Now that Google is open, let's try typing something in the search box. Type this in the Selenium Server command window:

```
cmd=type&1=q&2=hello world&sessionId=1143670264928
```

Again, don't forget to replace the Session ID with your own Session ID. If all goes to plan, you should see "hello world" in the search box for your Google search.

Now, let's do a search!

```
cmd=click&1=btnG&sessionId=1143670264928
```

You should now see the results of your Google Search in your browser window. You can run dozens of Selenium commands to automate all manner of browsing tasks. For more information on particular commands, you can go look at the Selenium Core section at the Open QA website, or check out the reference materials available for any of our Client Drivers (Java, .NET, Perl, Python or Ruby).

The next thing we might want to do is read some information back out of the page... for example, let's retrieve the HTML title of the current page, like this:

```
cmd=getTitle&sessionId=1143670264928
```

The browser will return the title of the HTML page we've loaded, like this:

```
OK,hello world - Google Search
```

That's enough Interactive Mode for now; let's move on to writing some code! But before we go, let's stop the browser we started. Type this command in the Selenium Server command window:

```
cmd=testComplete&sessionId=1143670264928
```

When this command finishes, your browser window should automatically close. To quit the Selenium Server, type "quit" or just press Ctrl-C.

Programming a Selenium RC Test

Now that you know how to run commands in interactive mode, let's code this test up! Here's examples of writing the test we just wrote in four different programming languages:

(+) Java (JUnit)

(+) C# (NUnit)

(+) Perl (Test::More)

(+) Python (unittest)

(+) Ruby (Test::Unit)

The Same Origin Policy

As you were running your tests, you may have noticed that your browser started at the following URL:

```
http://www.google.com/selenium-server/SeleneseRunner.html?sessionId=1143670728535
```

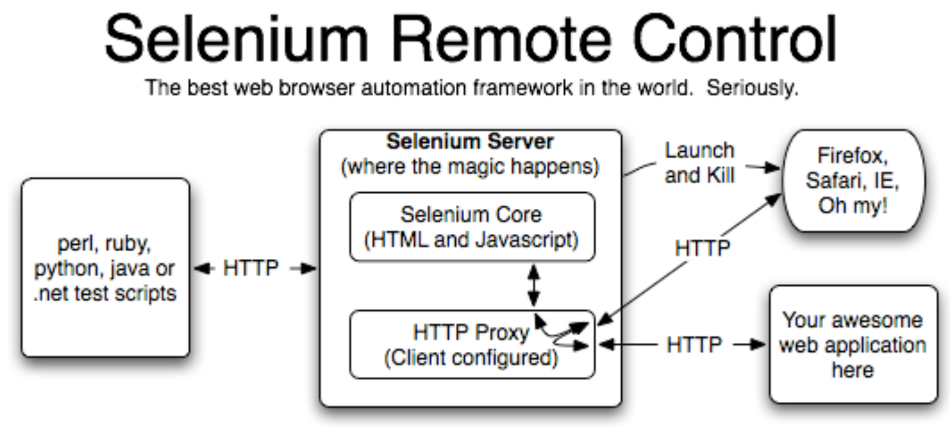
That's a rather unusual URL, because, of course, there is no such file available on www.google.com. If you open up your browser manually and browse to that URL, you'll get a 404 error. What's going on?

The Selenium Server is attempting to circumvent a very difficult problem in JavaScript automated testing: normally, JavaScript you write yourself can't be run on google.com, due to a policy known as the same origin policy. (That write-up is from the Mozilla website, but all modern JavaScript browsers enforce this policy.) The same origin policy makes a lot of sense. Let's say you've got your browser window pointing at, let's say, your bank's website, but you also have another webpage open pointing to someone's blog. JavaScript is allowed to *read* values from web pages, as well as *change* data appearing on webpages you've loaded. A malicious blogger could read your bank data, or worse, rewrite your bank page to make it think it was saying something else. He could use this to trick you into giving him sensitive information. The Same Origin Policy states that JavaScript is only allowed to read/modify HTML from the *same origin* as its source.

That creates a big problem for Selenium automated tests. If you wrote a .js file designed to test google.com, the same origin policy denies you the right to just run that .js file on google.com; instead, you'd have to somehow install that .js file *on google.com* in order to write automated tests against it. In the case of google.com we don't have the right to do this; but even if we did, it would be a hassle to do so.

That's where the Selenium Server comes in. The Selenium Server is acting as a *client-configured proxy* for the browser that you automatically started with "getNewBrowserSession". Specifically, it configures your browser to use the Selenium Server as a proxy in its browser preferences.

A proxy normally *fetches* HTML pages on your behalf; if a page can't be found, it honestly reports that the page wasn't there. But the Selenium Server is a very different kind of proxy; when the browser requests a page through the proxy that contains "/selenium-server/" in its URL, the Selenium Server doesn't simply fetch the page from the remote server, but instead automatically returns its own page instead. That makes the browser think that the remote server itself actually contained the page in question, which allows us to "inject" arbitrary JavaScript into google.com without modifying google.com in any way at all.


Selenium Remote Control
The best web browser automation framework in the world. Seriously.
Selenium Server
(where the magic happens)
Selenium Core
(HTML and Javascript)
HTTP Proxy
(Client configured)
perl, ruby, python, java or .net test scripts
HTTP
Launch and Kill
Firefox, Safari, IE, Oh my!
HTTP
HTTP
Your awesome web application here

However, the solution isn't perfect. Try running your tests like this:

```
cmd=getNewBrowserSession&1=*iexplore&2=http://www.yahoo.com
cmd=open&1=http://www.google.com&sessionId=1143670264928
cmd=type&1=q&2=hello world&sessionId=1143670264928
```

In this case, the "start URL" was yahoo.com, so the browser opened to

```
http://www.yahoo.com/selenium-server/SeleneseRunner.html?sessionId=1143670728535
```

and then we tried to open up google.com (which did work...) and type "Hello World" into the search box. But yahoo.com doesn't have the right to modify google.com, so that operation fails.

All this means is that you have to choose your browser's start URL wisely: if you open up the browser to yahoo.com, you can't use it to test google.com, and vice versa.

(A note for the future: we hope to have a slightly cooler solution for you in an upcoming release, allowing you to change a running browser session from one domain to another. The two domains still won't be able to talk to each other, but at least then you'll be able to switch back and forth easily. Most people won't need this functionality; if you're only testing one website at a time, you should be OK.)

Automatically Launching Other Browsers

Selenium Server can automatically launch/kill other browsers that we don't yet explicitly support. (Most of the browsers on the "Should Work" supported list need to be launched in this way.) When running your "getNewBrowserSession" command, use the "*custom" browser launcher instead of "*firefox" or "*iexplore", specifying the absolute path to your browser.

```
cmd=getNewBrowserSession&1=*custom c:\Program
Files\Netscape\Netscape\Netscp.exe&2=http://www.google.com
```

Note that when we launch the browser in this way, you'll have to *manually* configure your browser to use the Selenium

Server as a proxy. (Normally this just means opening your browser preferences and specifying "localhost:4444" as an HTTP proxy, but instructions for this can differ radically from browser to browser, so consult your browser's documentation for details.)

If you attempt to launch a *custom browser but don't configure the proxy correctly, you won't be able to get through even a simple Google test, because you'll get a 404 (File not found) error trying to access http://www.google.com/selenium-server/. Remember, that directory doesn't really exist on Google.com; it only appears to exist when the proxy is properly configured.

Also beware that Mozilla browsers can be a little fidgety about how they start and stop. Avoid launching your browser using a shell script; always prefer to use the binary executable directly. You may need to set the MOZ_NO_REMOTE environment variable to make Mozilla browsers behave a little more predictably.

If you want to use one of the explicitly supported browsers but have installed it in an unusual location, you can tell us that by using the special browser string followed by the absolute path to the executable.

```
cmd=getNewBrowserSession&1=*firefox c:\firefox\firefox.exe&2=http://www.google.com
```

That will automatically configure Firefox's proxy settings, but does not require you to add Firefox directly to the path. (You can use this to test multiple different versions of Firefox installed in separate directories.)

You can also use "*custom" to automatically launch a supported browser *without* configuring its proxy. For example, you could launch Firefox like this:

```
cmd=getNewBrowserSession&1=*custom c:\Program Files\Mozilla
Firefox\firefox.exe&2=http://www.google.com
```

Again, note that you'll need to manually configure your proxy prior to launching Firefox in this way; otherwise you'll get a 404 error while following the steps in the tutorial.

Running without a Session ID - the "null" session

If you get annoyed with typing out your Session ID over-and-over, or if you want to try manually testing with another browser that we don't yet officially support, you can always just launch the browser yourself and browse to "/selenium-server/SeleneseRunner.html" manually. (Don't forget to set your proxy settings, as discussed in the previous section.)

If you just point straight at "/selenium-server/SeleneseRunner.html" you can send commands to that browser without specifying a Session ID. When no Session ID is specified, the Selenium Server uses the "null" session to communicate with any browser that comes to speak with it.

You can even make up your own Session IDs, just by starting your browser while pointing to "/selenium-server/SeleneseRunner.html?sessionId=foo". That will use the Session ID "foo" instead of some crazy long number. (Session IDs don't have to be numbers, though they usually are when the Selenium Server generates one.) Hence, if you use a Session ID that isn't on record with the Selenium Server, no error message will be thrown; the Selenium Server will just wait patiently for some browser to come along and request work from that Session ID.

With that, you're ready to try out one of our Client Drivers. We have Client Drivers available in a variety of languages, allowing you to automatically submit HTTP requests to the Selenium Server and automate your browsing tasks. Click on one of them on the left to get more information about any particular Client Driver, or get information on how to write your own from scratch.

Good luck!