

Selenium and History

Saturday, March 10, 2012 1:21 PM

What Is Selenium?

- Selenium is a portable software testing framework for web applications
- For Selenium 1, Selenium IDE provides a record/playback tool for authoring tests without learning a test scripting language. It is a Firefox plug-in.
- Selenium 1 provides a test domain-specific language (Selenese). The scripts consist commands in HTML table.
- Selenium2 is reworked to be based on WebDriver, which is now a W3C standard. One write tests in a number of popular programming languages, including Java, C#, Groovy, Perl, PHP, Python, and Ruby.
- The tests can be run against most modern web browsers.
- Selenium deploys on Windows, Linux, and Macintosh platforms.

History?

- Selenium was originally developed by Jason Huggins in 2004, who was later joined by other programmers and testers at ThoughtWorks.
- It is open-source software, released under the Apache 2.0 license, and can be downloaded and used without charge
- The latest side project is Selenium Grid, which provides a hub allowing the running of multiple Selenium tests concurrently on any number of local or remote systems, thus minimizing test execution time.

Selenium 1 Remote Control

Saturday, March 10, 2012 1:23 PM

What is Selenium Remote Control(RC)?

- **Selenium Remote Control (RC)** is a server, written in Java, that accepts commands for the browser via HTTP
- RC makes it possible to write automated tests for a web application in any programming language, which allows for better integration of Selenium in existing unit test frameworks
- To make writing tests easier, Selenium project currently provides client drivers for PHP, Python, Ruby, .NET, Perl and Java.

Selenium Architectures

Monday, July 15, 2013 8:42 AM

There are three architecture types in Selenium2:

1. Local
2. Remote
3. Grid

The browser machine (the "node") in all architectures (local, remote, and grid) of Selenium need to run an instance of Selenium "server", for example, *selenium-server-standalone-2.33.0.jar*.

Note: The word "server" should not be confused with the grid hub. It only means that it receives requests from test scripts. The Selenium "server" could also run on what we normally consider a "client" machine, that is, the browser machine.

Local Architecture

The browser is on the same machine as the test script. You do not need to run the Selenium server in this configuration.

The test script will drive the browser directly.

Note: Chromium/Chrome browser currently require you to also run a separate ChromeDriver.exe executable, but Firefox does not. However, there is a bug request to provide built-in support for WebDriver in Chromium.

Remote Architecture

The browser is on a different machine than the test script. You need to run a Selenium "server" on the browser machine.

Note: It is called a "server" even though it is running on what one usually considers a "client" (browser) machine.

See also: [Selenium2 remote WebDriver setup](#)

Example: [Selenium2 remote Chrome \(Java\) example](#)

Grid Architecture

The browser is on a remote machine. You need to run a Selenium "server" on the browser machine, as a "node". The test script sends requests to the hub, which forwards the requests to the browser nodes.

The test scripts connects to the remote Selenium server via WebDriver network protocol, and the hub forwards it to the browser node. The hub essentially acts as a "proxy"—it receives requests, farms it out to a node that satisfied the specified configuration request, receives response and relay back to the test script.

See also: [Selenium2 grid2 setup](#)

Example: [Selenium2 grid2 Firefox \(Java\) example](#)

Tip: The test script URL for remote and grid architectures have the same syntax. That is, it is transparent to the test script whether the remote Selenium server is a grid hub or a standalone server.

Selenium WebDriver introduction

Saturday, March 10, 2012 1:24 PM

tag specifications

What is Selenium 2 (WebDriver)?

- Selenium WebDriver is the successor to Selenium RC. It is based on the WebDriver wire protocol.
- WebDriver is now a W3C standard: <http://www.w3.org/TR/webdriver/>
- Selenium WebDriver accepts commands (sent via a Client API) and sends them to a browser on local or remote machine. This is implemented through a browser-specific driver, which sends commands to a browser, and retrieves results.
- Firefox has built-in WebDriver support. Chromium (and Chrome/CEF3) currently requires a separate ChromeDriver executable running on the browser machine. ChromeDriver can be downloaded here:

<https://code.google.com/p/chromedriver/downloads/list>

- **RemoteWebDriver Configuration:** When the browser is on a remote machine, that machine needs to run a Selenium Standalone Server. And the client uses Selenium RemoteWebDriver API to send commands to the standalone server. And the Standalone Server will drive the requested browser on its machine.
- **Grid Configuration:** In Grid configuration, one starts a Selenium server in a "hub" role, and a number of servers in "node" role, forming a server farm. The client sends commands to a hub, which automatically distribute testing to a remote node in a farm that matches the requested configuration. The WebDriver built-in supported configuration options include operating system type and version, and browser type and version. However, the built-in configurations do not support custom configurations. So we RemoteWebDriver may be more suitable if one requires custom configurations.

Selenium IDE

Saturday, March 10, 2012 1:25 PM

What is Selenium IDE?

(Selenium 1 only)

- **Selenium IDE** is a complete integrated development environment (IDE) for Selenium tests
- It's an easy-to-use Firefox plug-in
- It allows recording, editing, and debugging tests
- It was previously known as Selenium Recorder.
- Scripts may be automatically recorded and edited manually providing autocompletion support and the ability to move commands around quickly
- Scripts are recorded in *Selenese*, a special test scripting language for selenium. Selenese provides commands for performing actions in a browser (click a link, select an option), and for retrieving data from the resulting pages.

Key Features:

- Record and playback
- Intelligent field selection will use IDs, names, or XPath as needed
- Autocomplete for all common Selenium commands
- Walk through tests
- Debug and set breakpoints
- Save tests as Selenese, Ruby scripts, or other formats
- Support for Selenium user-extensions.js file
- Option to automatically assert the title of every page

Selenium Client API

- As an alternative to writing tests in Selenese, tests can also be written in various programming languages. These tests then communicate with Selenium by calling methods in the Selenium Client API. Selenium currently provides client APIs for Java, C#, Ruby and Python.
- With Selenium 2, a new Client API was introduced (with *WebDriver* as its central component). However, the old API (using class *Selenium*) is still supported

In Terms of Price

Saturday, March 10, 2012 1:26 PM

TestComplete Vs. Telerik


TestComplete Price

Automated Testing & Improved Software Quality on Any Budget

	Enterprise	Standard	Both versions include the following: <ul style="list-style-type: none">✓ Unlimited support via email✓ Free minor version and bug fix releases✓ Discounts on future major version upgrades
Node-Locked License	\$1999 Add to cart	\$999 Add to cart	
Floating User License	\$4499 Add to cart	\$2999 Add to cart	

[? Explain license differences](#)


Telerik Price



Purchase Test Studio

- ✓ Functional web and desktop testing
- ✓ Web performance testing
- ✓ Manual testing
- ✓ Visual Studio plug-in
- ✓ *Coming soon: Load testing*

\$2,499 [Add to Cart](#)



Test Studio Run-Time

Requires Test Studio license

- ✓ Test execution only
- ✓ Unlimited users
- ✓ Suitable for test servers

\$199 [Add to Cart](#)

When Preferred Telerik over TestComplete,



It adds an Extra 500\$ for each license

TestComplete Vs. Selenium

Selenium is open-source software, released under the Apache 2.0 license, and can be downloaded and used without charge

When preferred Selenium over TestComplete,

We Save an amount of \$1999 spent on each license for TestComplete



What language binding should be used for Selenium?

Saturday, March 10, 2012 1:28 PM

Note: Information provided below is just my opinion of what language to use for Selenium automation. One can have a complete different outlook of this topic so be sure that I am not asserting anything below.

Background information:

Currently Selenium bindings are available in Java, C#, Python, Ruby, Perl and JavaScript. Among these options we have API documentation for Java, C#, Ruby and Python.

One can question, why not use JavaScript for selenium ?

This is what I have found from code.google.com that pretty much explain why we should not use JavaScript.

Understanding the API

Unlike the other language bindings, which all provide blocking APIs, WebDriverJS is purely asynchronous. This document describes the libraries used to simplify working with WebDriver and JavaScript's asynchronous event loop.

The simplest asynchronous APIs use basic continuation passing to indicate when an operation has completed. Unfortunately, this can quickly lead to excessively verbose code that is difficult to understand. For instance, consider the canonical WebDriver example, a simple Google search:

```
driver.get("http://www.google.com");
driver.findElement(By.name("q")).sendKeys("webdriver");
driver.findElement(By.name("btnG")).click();
assertEquals("webdriver - Google Search", driver.getTitle());
```

Rewritten in JavaScript with continuation passing, this example quickly gets out of hand:

```
driver.get("http://www.google.com", function() {
  driver.findElement(By.name("q"), function(q) {
    q.sendKeys("webdriver", function() {
      driver.findElement(By.name("btnG"), function(btnG) {
        btnG.click(function() {
          driver.getTitle(function(title) {
            assertEquals("webdriver - Google Search", title);
          });
        });
      });
    });
  });
});
```

Instead of using continuation passing, WebDriverJS uses "promises" to track the state of each operation. Using promises, the Google search example can be rewritten as follows

```
driver.get("http://www.google.com");
driver.findElement(By.name("q")).sendKeys("webdriver");
driver.findElement(By.name("btnG")).click();
driver.getTitle().then(function(title) {
  assertEquals("webdriver - Google Search", title);
});
```

What language to use for Selenium? Selenium(Java) or Selenium(Scripting languages: Python, Perl, Ruby)

I looked at web to learn about what Selenium user base has to say about Selenium(Java) Vs Selenium(Scripting languages like Python or Ruby)

Stackoverflow discussion about this topic :

<http://stackoverflow.com/questions/7453550/why-selenium-ruby-for-testing>

<http://stackoverflow.com/questions/13960897/selenium-webdriver-with-java-vs-python>

<http://stackoverflow.com/questions/9379497/what-are-advantages-disadvantages-of-using-selenium-for-java-vs-net-application>

Snapshot of comments from these discussion,

2 Answers

active

oldest

votes

▲
1
▼

It's hard to give you a hard and fast answer. It seems you're looking for some sort of explanation indicating that the Ruby bindings are better than the Java ones. If this is what you're looking for it is not something we can provide. Neither language is 'better' than the other but in this case it is possibly relevant to compare the bindings. For Selenium, there is more documentation on the Java bindings and a larger **userbase** leading me to say that the Java bindings are actually better.

The reason for the choice of Ruby is more likely due to the **skillset of your team** or perhaps that using Ruby helped it fit better into your existing system.

share | edit

answered Sep 17 '11 at 8:48



Mike Kwan

11.7k ● 2 ● 11 ● 39

1

Ruby and python are both popular choices for testing because you can write very terse code in them with hardly any syntactic noise and you can use all the tools of a higher order language. Ruby is extremely strong in creating embedded DSLs which makes it an ideal candidate for testing purposes.

Java is more strict, requires more coding and the benefits in this case are negligible. You don't really need static type checking because you run your test all the time and all the code paths are constantly checked and you don't really benefit from the faster execution time of java because selenium is very slow and it's the real bottleneck.

So the quick answer is: probably because you can write the test code faster.

share | edit

answered Sep 17 '11 at 9:15



Karoly Horvath
32.8k • 1 • 18 • 55

Generally speaking, the Java selenium web driver is better documented. When I'm searching for help with a particular issue, I'm much more likely to find a Java discussion of my problem than a Python discussion.

Another thing to consider is, what language does the rest of your code base use? If you're running selenium tests against a Java application, then it makes sense to drive your tests with Java.

share | edit

answered Dec 19 '12 at 21:41



Aurand
1,461 • 1 • 10

You've got it spot on, there are ton load of documents for Java. All the new feature implementations are mostly explained with Java. Even stackoverflow has a pretty strong community for java + selenium.

share | edit

answered Dec 19 '12 at 21:37



amey
2,400 • 5 • 19

It really does not matter. Even the Documentation. Selenium lib is not big at all.

Moreover, if you are good in development, you'll wrap selenium in your own code, and will never use `driver.find(By.whatever(description))`. Also you'd use some standards and `By.whatever` will become `By.xpath` only.

Personally, I prefer python and the reason is that and my other tests for software use other python liibs -> this way I can unite my tests.

share | edit

answered Dec 21 '12 at 15:36



Alex Okrushko
1,046 • 5 • 17

I think the above comments cover pretty much the voice of selenium user base. For a better understanding of these opinions let us classify above information into supporting and not-supporting tags for Selenium(Java) and Selenium(Python, Ruby)

Selenium(Java)

#Supporting:

- More documentation on Java binding.
- Larger user base.
- Searching for help with a particular issue is more likely to be found on Java discussion.
- New feature implementations are mostly explained with Java

#Not Supporting:

- Java is strict
- Selenium itself is slow so we don't get benefited with faster execution time of Java

Selenium(Scripting language like Python, Ruby)

#Supporting:

- Can write terse code that need not be compiled like Java.
- Use all the tools of higher order language.
- Personal preference as other test software use Python.

#Not supporting

- Smaller user base. We can't expect much help from outside. Any problem we face should be solved ourselves.
- Skillset of team members in most of the cases been a strong reason for pick Python or Ruby for Selenium.

So what does all these opinions and facts mean to us ?

If we prefer to use Java,

- Skillset - no additional training is required, apart from little brush up of what already learned.
 - Existing team members of QA Automation have required skillset for Java.
 - Irrespective to stream of graduation from college(Computer Science Engineering - CSE , Electronics and Communication Engineering - ECE, Electrical and Electronics Engineer - EEE), Java is taught as a subject in course curriculum. This means,
 - For prospective/future members of QA Automation, we need not have to be keen on fresher profiles with Python or Ruby as a mandatory skillset. From my understanding, Ruby and Python as skill set is too difficult to find at a fresher level in Hyderabad, India market.
 - QA members from regression teams in Hyderabad are mostly from CSE, ECE and EEE streams. Going forward, it would be easy for these teams to participate in Selenium automation.
- As a newbie, when we are stuck with a problem, we can always get help form internet - larger user base for Java.
- Well document Java bindings for Selenium will come handy. We can always look back to them for help with the syntax or usage w hen ever needed.

If we prefer to use scripting language like Python, or Ruby,

- We use Jscript as language to program for TestComplete automation. So using a scripting language for Selenium will maintain s ingularity in type of language but not the language itself.
- We use python to program Tellus framework, so using python might help in uniting our tests.
- Though QA Automation team members have some experience in Python, In my opinion that is not adequate enough to use Selenium b indings for automation. Training in these languages would be a mandatory pre request for team members working on Selenium projects.
- As a newbie, when we are stuck with a problem, we may get little or no help from Selenium user base .
- Selenium documentation for Python and Ruby bindings are available. They may not be as comprehensive as documentation for Java bindings and may not cover everything like Java docs .
- Understanding and debugging automation code for application/automation bugs would be arduous for QA Automation and QA regres sion team members if we are working on a language that most of us are not familiar with. Like with TestComplete , QA regression t eam members will be engaged in looking for Selenium automation results and expected to replicate the bugs through running and debugging a utomation code, training them on Python or Ruby will be mandatory.

I just explicated my opinion about the topic at discussion. This could be different from others opinion. I could have missed or not covered certain perspective, so please feel free to add your opinion.

Thanks

Shiva Imminni

Ruby selenium-webdriver installation

Tuesday, February 11, 2014 10:49 AM

Usually, installation is like this:

```
gem install selenium-webdriver
```

But it has documentation installation error:

```
Parsing documentation for nokogiri-1.6.0-x86-mingw32  
unable to convert "\x90" from ASCII-8BIT to UTF-8 for lib/nokogiri/1.9/nokogiri.  
so, skipping  
unable to convert "\x90" from ASCII-8BIT to UTF-8 for lib/nokogiri/2.0/nokogiri.  
so, skipping
```

One could avoid those error by not installing documentation:

```
gem install selenium-webdriver --no-rdoc --no-ri
```

Selenium Ruby with existing browser

Monday, February 17, 2014 1:38 PM

Selenium Script in Ruby Driving existing (already-running) Chrome Browser

Need to separately run Chrome/CEFCClient and ChromeDriver as follows.

With Chrome:

```
C:\> "C:\Program Files (x86)\Google\Chrome\Application\chrome.exe" --remote-  
debugging-port=8088 --user-data-dir=C:\Test\temp
```

With CEFCClient:

```
C:\> C:\CEFCClient\cefclient.exe --remote-debugging-port=8088 --user-data-dir=c:\test  
\temp
```

Note:

- Start Chrome browser with specific remote debugging port.
- Chrome must specify its own user data dir
- Chrome DevTools can inspect the page at port 8088: <http://localhost:8088>
It will show Inspectable pages.

Also start ChromeDriver on specific port:

```
C:\> C:\ChromeDriver\chromedriver.exe --port=4444
```

Ruby WebDriver Example

```
require "selenium-webdriver"  
  
caps = Selenium::WebDriver::Remote::Capabilities.chrome  
caps['chromeOptions'] = {"debuggerAddress" => "localhost:8088"}  
driver = Selenium::WebDriver.for :remote, :url =>  
"http://localhost:4444", :desired_capabilities => caps  
  
driver.navigate.to "http://tellus/"  
  
element = driver.find_element(:name, 'q')  
element.send_keys "Cheese"  
element.submit  
  
puts driver.title  
  
driver.quit
```

Reference

<https://code.google.com/p/chromedriver/issues/detail?id=497>

Selenium Tutorial

Tuesday, July 11, 2017 3:22 PM

<http://www.software-testing-tutorials-automation.com/2013/07/what-is-selenium-webdriver.html>

Basic Selenium Commands

Tuesday, July 11, 2017 3:21 PM

<http://www.software-testing-tutorials-automation.com/2013/07/list-of-selenium-commands-with-examples.html>

Selenium IDE Online Test

Tuesday, July 11, 2017 3:26 PM

<http://www.software-testing-tutorials-automation.com/2013/08/selenium-ide-interview-questionsquiz.html>

SeleniumTraining

Thursday, December 19, 2013 3:21 PM

From Bhupender 1/8/2014

I have completed working on creating TWIKI page for selenium training. Please refer [this link](#) to view the TWIKI page.

Xpath Locators

Friday, October 18, 2013 12:32 AM



Locators_table_1_0_2

This document will be helpful while creating a XPATH for the element on the webpage. It comes very handy when you want to create a complex XPATHs. Please go through the document to understand it better.

Installation of Selenium Hub and Node in Factset

Friday, December 19, 2014 1:26 PM

Installation of Selenium Hub:

1. Create a folder "C:\selenium\hub" on the machine where you plan to install the hub.
2. Copy all the files from

H:\Department\QualityAssurance\QAAutomation\web\seleniumgrid\hub

over to the folder "C:\selenium\hub" on the target machine.

3. Create a folder "C:\selenium\logs" for the log files.
4. Download nssm from

<http://nssm.cc/download>

Then unzip and install nssm on the target machine. nssm is used to install the hub as a windows service.

5. Download and install java jdk (Java SE 7u71/72 or above) from

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

on the target machine. Java is used to run the hub.

6. Install the hub as a windows service like what Joseph said in his email:

- a. Right-click **Command Prompt** and select "**run as administrator**"
- b. Enter command:
nssm.exe install "Service Name"

Sample:

nssm.exe install "Selenium Hub"

- c. Under **Application** tab, enter your program path and arguments, for example:
Path: C:\Program Files (x86)\Java\jdk1.7.0_60\bin\java.exe
Arguments: -classpath C:\Selenium\hub\selenium-server-standalone-2.53.1.jar
org.openqa.grid.selenium.GridLauncher -role hub -hubConfig "C:\Selenium\hub\hubconfig.json"
- d. Click **Install Service**

7. Go to services window and start the service "Selenium Hub".
8. The following server url

<http://hub-machine-domainname:4444/wd/hub>

is used in the test script to create remote web driver to do the testing.

Installation of Selenium Node:

1. Create a folder "C:\selenium\node" on the machine where you plan to install the node

2. Copy all the files from

H:\Department\QualityAssurance\QAAutomation\web\seleniumgrid\node

over to the folder "C:\selenium\node" on the target machine.

3. Modify the file nodeConfig.json and point "hubHost" to the machine name of the target hub; Also point the "host" to the node machine.

4. Create a folder "C:\selenium\logs" for the log files of the node (debug info is also there).

5. Download nssm from

<http://nssm.cc/download>

Then unzip and install nssm on the target machine. nssm is used to install the hub as a windows service.

6. Download and install java jdk (Java SE 7u71/72 or above) from

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

on the target machine. Java is used to run the node.

7. Download and install firefox and chrome on the node machine.

8. Configure the firefox browser (firefox does not auto use system certs):

a. Copy all the certs files from

H:\Department\QualityAssurance\QAAutomation\web\certs
to a temp folder on the node machine.

b. Install all the three root certs:

Options -> Advanced -> Certificates -> View Certificate -> Import

Select .cer file one at a time

Click Open

Check Trust this CA to identify websites

Click OK

9. To configure IE, you need to following instructions here: <https://github.com/SeleniumHQ/selenium/wiki/InternetExplorerDriver>

10. To disable auto update for firefox, open firefox browser and do:

- Don't import from internet explorer
- Entering "about:config" in the browser url field
- Setting "app.update.enabled" to false
- Setting "app.update.auto" to false

11. To disable auto update for chrome, register a key by following <https://www.chromium.org/administrators/turning-off-auto-updates>

12. Install the node as a windows service like what Joseph said in his email:

a. Right-click **Command Prompt** and select "run as administrator"

b. Enter command:

nssm.exe install "Service Name"

Sample:

nssm.exe install "Selenium Node"

c. Under **Application** tab, enter your program path and arguments, for example:

Path: C:\Program Files (x86)\Java\jdk1.7.0_60\bin\java.exe

Arguments: -classpath C:\Selenium\node\selenium-server-standalone-2.53.1.jar; -

Djava.util.logging.config.file=C:\selenium\node\logging.properties org.openqa.grid.selenium.GridLauncher -

Dwebdriver.chrome.driver="C:\Selenium\node\chromedriver.exe" -Dwebdriver.ie.driver="C:\Selenium\node

\IEDriverServer.exe" -role node -nodeConfig "C:\Selenium\node\nodeConfig.json"

d. Click **Log On** tab

e. Click **This account** and enter Windows login with domain, for example, GREENWICH\svc-tellus

- f. Enter **Password** and **Confirm**
 - g. Click **Install Service**
13. Go to services window and start the service "Selenium Node".
 14. Check the registered browsers from the following hub url:
<http://hub-machine-domain-name:4444/grid/console>
 15. To test this grid, use the hub url: <http://hub-machine-domain-name:4444/wd/hub>

Instruction to Selenium Grid

Wednesday, December 17, 2014 2:43 PM

Link: <http://www.guru99.com/introduction-to-selenium-grid.html>

Introduction to Selenium Grid

What is Selenium Grid?

Selenium Grid is a part of the Selenium Suite that specializes on running multiple tests across different browsers, operating systems, and machines in parallel.

Selenium Grid has 2 versions - the older Grid 1 and the newer Grid 2. We will only focus on Grid 2 because Grid 1 is gradually being deprecated by the Selenium Team.



Selenium Grid uses a hub-node concept where you only run the test on a single machine called a **hub**, but the execution will be done by different machines called **nodes**.

When to Use Selenium Grid?

You should use Selenium Grid when you want to do either one or both of following :

- **Run your tests against different browsers, operating systems, and machines all at the same time.** This will ensure that the application you are testing is fully compatible with a wide range of browser-OS combinations.
- **Save time in execution of your test suites.** If you set up Selenium Grid to run, say, 4 tests at a time, then you would be able to finish the whole suite around 4 times faster.

Grid 1.0 Vs Grid 2.0

Following are the main differences between Selenium Grid 1 and 2.

Grid 1

Selenium Grid 1 has its own remote control that is different from the Selenium RC server. They are two different programs.

You need to install and configure Apache Ant first before you can use Grid 1.

Can only support Selenium RC commands/scripts.

You can only automate one browser per remote control.

Grid 2

Selenium Grid 2 is now bundled with the Selenium Server jar file

You do not need to install Apache Ant in Grid 2.

Can support both Selenium RC and WebDriver scripts.

One remote control can automate up to 5 browsers.

What is a Hub and Node?

The Hub

- The hub is the central point where you load your tests into.
- There should only be one hub in a grid.
- The hub is launched only on a single machine, say, a computer whose OS is Windows 7 and whose browser is IE.
- The machine containing the hub is where the tests will be run, but you will see the browser being automated on the node.

The Nodes

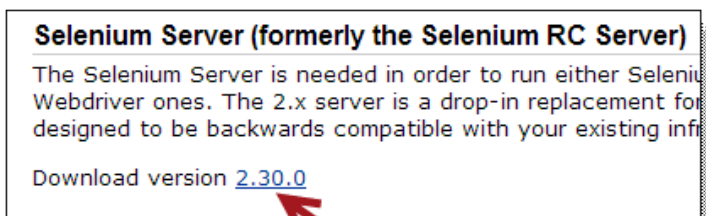
- Nodes are the Selenium instances that will execute the tests that you loaded on the hub.
- There can be one or more nodes in a grid.
- Nodes can be launched on multiple machines with different platforms and browsers.
- The machines running the nodes need not be the same platform as that of the hub.

How to Install and Use Grid 2.0?

In this section, you will use 2 machines. The first machine will be the system that will run the hub, while the other machine will run a node. For simplicity, let us call the machine where the hub runs as "Machine A" while the machine where the node runs will be "Machine B". It is also important to note their IP addresses. Let us say that Machine A has an IP address of 192.168.1.3 while Machine B has an IP of 192.168.1.4.

Step 1

Download the Selenium Server by [here](#).



click this to start
download

Step 2

You can place the Selenium Server .jar file anywhere in your HardDrive. But for the purpose of this tutorial, place it on the C drive of both Machine A and Machine B. After doing this, you are now done installing Selenium Grid. The following steps will launch the hub and the node.

Step 3

- We are now going to launch a hub. Go to Machine A. Using the command prompt, navigate to the root of Machine A's - C drive ,because that is the directory where we placed the Selenium Server.
- On the command prompt, type **java -jar selenium-server-standalone-2.30.0.jar -role hub**
- The hub should successfully be launched. Your command prompt should look similar to the image below

```
C:\Users\...>cd \
C:\>java -jar selenium-server-standalone-2.30.0.jar -role hub
Feb 23, 2013 4:02:11 AM org.openqa.grid.selenium.GridLauncher main
INFO: Launching a selenium grid server
2013-02-23 04:02:12.065:INFO:osjs.Server:jetty-7.x.y-SNAPSHOT
2013-02-23 04:02:12.113:INFO:osjs.ContextHandler:started o.s.j.s.ServletContext
Handler(/,null)
2013-02-23 04:02:12.126:INFO:osjs.AbstractConnector:Started SocketConnector@0.0.
0.0:4444
Feb 23, 2013 4:02:22 AM org.openqa.grid.internal.BaseRemoteProxy <init>
WARNING: Max instance not specified. Using default = 1 instance
```

Step 4

Another way to verify whether the hub is running is by using a browser. Selenium Grid, by default, uses Machine A's port 4444 for its web interface. Simply open up a browser and go to <http://localhost:4444/grid/console>

Grid Hub 2.30.0


[view config](#)

Also, you can check if Machine B can access the hub's web interface by launching a browser there and going to <http://iporhostnameofmachineA:4444/grid/console> where "iporhostnameofmachineA" should be the IP address or the hostname of the machine where the hub is running. Since Machine A's IP address is 192.168.1.3, then on the browser on Machine B you should type <http://192.168.1.3:4444/grid/console>

Step 5

- Now that the hub is already set up, we are going to launch a node. Go to Machine B and launch a command prompt there.
- Navigate to the root of Drive C and type the code below. We used the IP address 192.168.1.3 because that is where the hub is running. We also used port 5566 though you may choose any free port number you desire.

```
C:\> java -jar selenium-server-standalone-2.30.0.jar -role
webdriver -hub http://192.168.1.3:4444/grid/register -port 5566
```

 IP address of the machine
where the hub is running

- When you press Enter, your command prompt should be similar to the image below.

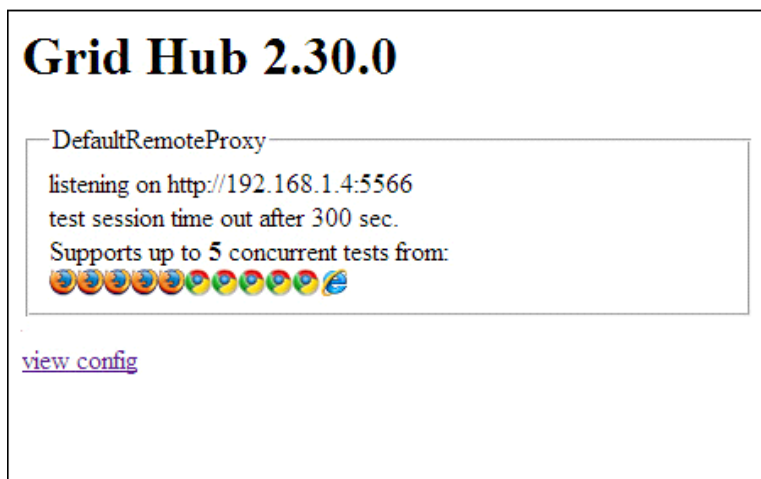
```

C:\>java -jar selenium-server-standalone-2.30.0.jar -role webdriver -hub http://
192.168.1.3:4444/grid/register -port 5566
Feb 23, 2013 4:34:39 PM org.openqa.grid.selenium.GridLauncher main
INFO: Launching a selenium grid node
16:34:40.733 INFO - Java: Oracle Corporation 23.7-b01
16:34:40.733 INFO - OS: Windows XP 5.1 x86
16:34:40.733 INFO - v2.30.0, with Core v2.30.0. Built from revision dc1ef9c
16:34:40.874 INFO - RemoteWebDriver instances should connect to: http://127.0.0.
1:5566/wd/hub
16:34:40.874 INFO - Version Jetty/5.1.x
16:34:40.874 INFO - Started HttpContext[/selenium-server/driver,/selenium-server
/driver]
16:34:40.889 INFO - Started HttpContext[/selenium-server,/selenium-server]
16:34:40.889 INFO - Started HttpContext[/,/]
16:34:40.889 INFO - Started org.openqa.jetty.servlet.ServletHandler@ed3512
16:34:40.889 INFO - Started HttpContext[/wd,/wd]
16:34:40.905 INFO - Started SocketListener on 0.0.0.0:5566
16:34:40.905 INFO - Started org.openqa.jetty.jetty.Server@1f77497
16:34:40.905 INFO - using the json request : {"class":"org.openqa.grid.common.Re
gistrationRequest","capabilities":{"platform":"XP","seleniumProtocol":"WebDrive
r","browserName":"firefox","maxInstances":5}, {"platform":"XP","seleniumProtocol"
:"WebDriver","browserName":"chrome","maxInstances":5}, {"platform":"WINDOWS","sel
eniumProtocol":"WebDriver","browserName":"internet explorer","maxInstances":1}},
"configuration":{"port":5566,"register":true,"host":"192.168.1.4","proxy":"org.o
penqa.grid.selenium.proxy.DefaultRemoteProxy","maxSession":5,"role":"webdriver",
"hubHost":"192.168.1.3","registerCycle":5000,"hub":"http://192.168.1.3:4444/grid
/register","hubPort":4444,"url":"http://192.168.1.4:5566","remoteHost":"http://1
92.168.1.4:5566"}}
16:34:40.905 INFO - starting auto register thread. Will try to register every 50
00 ms
16:34:40.905 INFO - Registering the node to hub :http://192.168.1.3:4444/grid/re
gister
16:34:46.171 INFO - Executing: org.openqa.selenium.remote.server.handler.Status@
1e5a771 at URL: /status
16:34:46.186 INFO - Done: /status

```

Step 6

Go to the Selenium Grid web interface and refresh the page. You should see something like this.



At this point, you have already configured a simple grid. You are now ready to run a test remotely on Machine B.

Designing Test Scripts That Can Run on the Grid

To design test scripts that will run on the grid, we need to use **DesiredCapabilities** and the **RemoteWebDriver** objects.

- **DesiredCapabilities** is used to set the type of **browser** and **OS** that we will automate
- **RemoteWebDriver** is used to set which node (or machine) that our test will run against.

To use the **DesiredCapabilities** object, you must first import this package

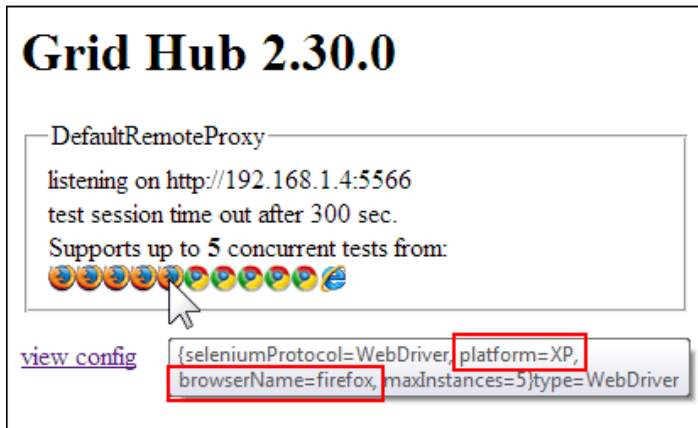
```
import org.openqa.selenium.remote.DesiredCapabilities;
```

To use the **RemoteWebDriver** object, you must import these packages.

```
import java.net.MalformedURLException;
import java.net.URL;
import org.openqa.selenium.remote.RemoteWebDriver;
```

Using the DesiredCapabilities Object

Go to the Grid's web interface and hover on an image of the browser that you want to automate. Take note of the **platform** and the **browserName** shown by the tooltip.



In this case, the platform is "XP" and the browserName is "firefox".

We will use the platform and the browserName in our WebDriver as shown below (of course you need to import the necessary packages first).

```
DesiredCapabilities capability = DesiredCapabilities.firefox();
capability.setBrowserName("firefox");
capability.setPlatform(Platform.XP);
```

Using the RemoteWebDriver Object

Import the necessary packages for RemoteWebDriver and then pass the DesiredCapabilities object that we created above as a parameter for the RemoteWebDriver object.

we used RemoteWebDriver and not FirefoxDriver

```
WebDriver driver;
driver = new RemoteWebDriver(
    new URL("http://192.168.1.4:5566/wd/hub"), capability);
```

IP address and port on Machine B

Running a Sample Test Case on the Grid

Below is a simple WebDriver TestNG code that you can create in Eclipse on Machine A. Once you run it, automation will be performed on Machine B.

```

import org.openqa.selenium.*;
import org.openqa.selenium.remote.DesiredCapabilities;
import java.net.MalformedURLException;
import java.net.URL;
import org.openqa.selenium.remote.RemoteWebDriver;
import org.testng.Assert;
import org.testng.annotations.*;

public class Grid_2 {
    WebDriver driver;
    String baseUrl, nodeURL;

    @BeforeTest
    public void setUp() throws MalformedURLException {
        baseUrl = "http://newtours.demoaut.com/";
        nodeURL = "http://192.168.1.4:5566/wd/hub";
        DesiredCapabilities capability = DesiredCapabilities.firefox();
        capability.setBrowserName("firefox");
        capability.setPlatform(Platform.XP);
        driver = new RemoteWebDriver(new URL(nodeURL), capability);
    }

    @AfterTest
    public void afterTest() {
        driver.quit();
    }

    @Test
    public void simpleTest() {
        driver.get(baseUrl);
        Assert.assertEquals("Welcome: Mercury Tours", driver.getTitle());
    }
}

```

The test should pass.

All suites

Default suite

Info

- C:\Users\...AppData\Local\Temp\testng-eclipse-1896183272\testng-customsuite.xml
- 1 test
- 0 groups
- Times
- Reporter output
- Ignored methods
- Chronological view

Results

- 1 method, 1 passed
- Passed methods (hide)
- ✓ simpleTest

Methods in chronological order

setUp	0 ms
simpleTest	9374 ms
afterTest	11570 ms

Summary

- Selenium Grid is used to run multiple tests simultaneously in different browsers and platforms.
- Grid uses the hub-node concept.
 - The hub is the central point wherein you load your tests.
 - Nodes are the Selenium instances that will execute the tests that you loaded on the hub.
- To install Selenium Grid, you only need to download the Selenium Server jar file - the same file used in running Selenium RC tests.
- There are 2 ways to verify if the hub is running: one was through the command prompt, and the other was through a browser
- To run test scripts on the Grid, you should use the DesiredCapabilities and the RemoteWebDriver objects.

- DesiredCapabilities is used to set the type of browser and OS that we will automate
- RemoteWebDriver is used to set which node (or machine) that our test will run against.

Pasted from <<http://www.guru99.com/introduction-to-selenium-grid.html>>

Selenium Grid

Saturday, March 10, 2012 1:25 PM

Selenium Grid?

- Selenium Grid is a server that allows tests to use web browser instances running on remote machines.
- With Selenium Grid, one server acts as the **hub**. Tests contact the hub to obtain access to browser instances. The hub has a list of servers that provide access to browser instances (**WebDriver nodes**), and lets tests use these instances.
- Selenium Grid allows to run tests in parallel on multiple machines, and to manage different browser versions and browser configurations centrally (instead of in each individual test).

Thief Selenium Grid Chrome test script example (Java)

Tuesday, December 17, 2013 10:25 AM

An example Java test script targeting Chrome browser in Thief team's Selenium Grid.

1. Use "RemoteWebDriver", instead of plain "WebDriver" in your script
2. Use "/wd/hub" suffix to target a grid hub (central server),
3. The port number in URL is same as the "/grid/console" above

For example,

```
driver = new RemoteWebDriver(  
    new URL("http://swarmtsa01.prod.factset.com:4444/wd/hub"),  
    dc);
```

To see grid console:

<http://swarmtsa01.prod.factset.com:4444/grid/console>

```
/**  
 * Thief Selenium Grid Chrome example.  
 * JUnit unit test.  
 */  
package test;  
  
import java.net.URL;  
import java.util.List;  
import org.junit.After;  
import org.junit.AfterClass;  
import org.junit.Before;  
import org.junit.BeforeClass;  
import org.junit.Test;  
import org.openqa.selenium.By;  
import org.openqa.selenium.WebDriver;  
import org.openqa.selenium.WebElement;  
import org.openqa.selenium.chrome.ChromeOptions;  
import org.openqa.selenium.remote.DesiredCapabilities;  
import org.openqa.selenium.remote.RemoteWebDriver;  
  
public class TestGrid1 {  
    WebDriver driver;  
  
    public TestGrid1() {  
    }  
  
    @BeforeClass  
    public static void setUpClass() {  
    }  
  
    @AfterClass  
    public static void tearDownClass() {  
    }  
  
    @Before
```

```

public void setUp() {
    try {
        ChromeOptions options = new ChromeOptions();
        DesiredCapabilities dc = DesiredCapabilities.chrome();
        dc.setCapability(ChromeOptions.CAPABILITY, options);
        driver = new RemoteWebDriver(
            new URL("http://swarmtsa01.prod.factset.com:4444/wd/hub"),
            dc);
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}

@After
public void tearDown() {
    driver.quit();
}

@Test
public void test1() {
    try {
        // Go to the Google Suggest home page
        driver.get("http://www.google.com/webhp?complete=1&hl=en");

        // Enter the query string "Cheese"
        WebElement query = driver.findElement(By.name("q"));
        query.sendKeys("Cheese");

        // Sleep until the div we want is visible or 5 seconds is over
        long end = System.currentTimeMillis() + 5000;
        while (System.currentTimeMillis() < end) {
            WebElement resultsDiv = driver.findElement(By.className("gssb_e"));

            // If results have been returned, the results are displayed in a drop
            if (resultsDiv.isDisplayed()) {
                break;
            }
        }

        // And now list the suggestions
        List<WebElement> allSuggestions =
driver.findElements(By.xpath("//td[@class='gssb_a gbqfsf']"));

        for (WebElement suggestion : allSuggestions) {
            System.out.println(suggestion.getText());
        }

    } catch (Exception ex) {
        ex.printStackTrace();
    }
}
}

```

Expected console output:

```

cheesecake factory
cheesecake recipe
cheese

```


cheeseburger soup

MultipleBrowsers

Monday, October 20, 2014 3:30 PM

Parallel execution of WebDriver test using Selenium Grid2 on multiple machines.

Guide to Setup Selenium Grid2

<http://amitagarwal15.blogspot.com/2013/12/parallel-execution-of-webdriver-test.html>

Selenium-Grid test pass

Monday, October 20, 2014 3:36 PM

The first Selenium-Grid test passed this morning on Tellusdevb02 using Firefox, the other browsers need some work along with selenium-grid being a service.

Right now I'm running the below script from Joseph and the result "Google" looks good.

```
C:\Users\kmmorris>ruby GridTest01.rb  
Google
```

Script contents

```
# Example IE (Ruby)  
# It will output web page title, "Google"  
require "selenium-webdriver"  
caps = Selenium::WebDriver::Remote::Capabilities.ie  
driver = Selenium::WebDriver.for :remote, :url => "http://tellusdevb02.pc.factset.com:4444/wd/hub", :desired_capabilities => caps  
driver.navigate.to "http://google.com"  
puts driver.title  
element = driver.find_element(:name, 'q')  
element.send_keys "Hello WebDriver!"  
element.submit  
driver.quit
```

The Grid Console is here <http://tellusdevb02.pc.factset.com:4444/grid/console?config=true&configDebug=true>

Se Grid Console v.2.39.0

DefaultRemoteProxy (version : 2.43.1)
id : http://172.17.245.115:5556, OS : VISTA

Browsers **Configuration**

Remote Control (legacy)
v: [IE, FF, CH, SF, OP]
v: [IE, FF, CH, SF, OP]
v: [IE, FF, CH, SF, OP]
WebDriver
v: [IE, FF, CH, SF, OP]
v: [IE, FF, CH, SF, OP]
v: [IE, FF, CH, SF, OP]

Config for the hub :

```
host : null  
port : 4444  
cleanUpCycle : 5000  
timeout : 300000  
browserTimeout : 0  
newSessionWaitTimeout : -1  
grid1Mapping : {}  
throwOnCapabilityNotPresent : true  
capabilityMatcher : org.openqa.grid.internal.utils.DefaultCapabilityMatcher  
prioritizer : null  
servlets :
```

...

...

-Ken

NSSM for Grid

Friday, December 05, 2014 3:32 AM

From Joseph

FYI: Info for using NSSM to run a Windows Service as another user.

NSSM Installation:

1. Download from <http://nssm.cc/download>
2. Unzip to installation destination

Installing Windows Service to Run as Another User

It works with any Java or other program.

1. Right-click **Command Prompt** and select "run as administrator"
2. Enter command:
nssm.exe install "Service Name"
3. Under **Application** tab, enter your program path and arguments, for example:
Path: C:\Program Files (x86)\Java\jdk1.7.0_60\bin\java.exe
Arguments: -jar C:\Selenium\selenium-server-standalone-2.43.1.jar -role hub
4. Click **Log On** tab
5. Click **This account** and enter Windows login with domain, for example, GREENWICH\user
6. Enter **Password** and **Confirm**
7. Click **Install Service**

Tellus Selenium-Grid usage processes.

Thursday, December 18, 2014 4:17 AM

This is a work in progress by Ken, Last Updated 12/18/14

Writing you tests for Selenium Grid
Develop your script on you laptop first and test it.
Then have you script point at

Your test has to point at the HUB.

```
driver = Selenium::WebDriver.for :remote, :url =>  
"http://tellusdevb02.pc.factset.com:4444/wd/hub", :desired_capabilities => caps
```

For Development for all Testing point to Tellusgridb04.pc.factset.com:4444
It will point to Tellusdevbo2(NODE) for browsers.

For Development for IS Testing point to Tellusgridb04.pc.factset.com:4444
It will point to Tellusdevbo2(NODE) for brows

Grid Table

Thursday, December 18, 2014 12:01 PM

Work in progress Ken

<http://infonet/edit/QualityAssurance/SeleniumGrid?t=1418922055>

GridDocOverview

Wednesday, January 14, 2015 4:16 PM

Tellus Selenium Grid, Why?

Our Tellus software testing automation framework supported automated testing of the FactSet Workstation using a tool named Test Complete. The Tellus Framework launched Focal Point Environment(FPE) Virtual Machines to execute these automated tests once. As we moved forward and started to do more web based testing for Company Markets and Analytics we introduced Selenium2 and Protractor. A lot of the web applications only needed to be tested in the Browser only. This led to excessive use of the FPE VMs and a lot of time waiting.

To solve both of those problems stated above we decided to introduce Selenium-Grid as an alternative for Web Browser only testing. We decided to support Web only testing, Mobile testing, Internal Apps testing and allow for Software engineers to execute our automated tests in that environment.

Selenium Grid

Selenium Grid is a server that allows tests to use web browser instances running on remote machines. With Selenium Grid, one server acts as the **hub**. Tests contact the hub to obtain access to browser instances. The hub has a list of servers that provide access to browser instances (**WebDriver nodes**), and lets tests use these instances. Selenium Grid allows executing tests in parallel on multiple machines, and to manage different browser versions and browser configurations centrally (instead of in each individual test).

Tellus Selenium Grid, Where are we at today?

In our Tellus automation environment we support two environments. One is our Development environment and the other is our Production Testing environment. We decided to follow that plan with Selenium-Grid. Currently we have built a Development environment and are in the process of adding in our Tellus UI and Framework to support that. However, it is quite capable of taking tests driven from a computer (laptop) that points to our Selenium- Grid HUB.

As short while ago we announced that the Tellus Selenium-Grid(S-G)Development environment is available for testing Browser (FireFox and Chrome) Web Apps in our internal QA Automation Group. Note Internet Explorer (IE) currently doesn't like S-G yet.

What does the Tellus Selenium-Grid Development environment look like? It consists of 3 server class machines with multiple(6-8) CPUs and 6-8 Gigs of Memory. One of those machines (tellusgridb04) is the HUB and the other two are the NODEs. See [InstructionToSeleniumGrid](#)

The way it is setup now is that there is one HUB, it has one NODE that is registered for Web testing and we have another Mobile NODE that has registered with the same HUB. In addition we have a separate network (HUB and NODE) specifically for IS and Internally Application testing.

Tellus Selenium Grid Rules

First some simple rules, taken from Lilun's update in [RPD:15519848](#).

- (1) The test case should be short and independent**
- (2) The browser needs to be maximized for the clicks to work**
- (3) The script should be tested on the local hub/node first before being tested against the above hub**

And

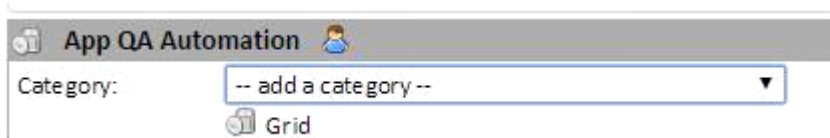
Your tests can not have any **Workstation** or external dependencies. Like **RunApp**, **AutoIt** or the following Tellus calls **OpenFDSW**, **OpenWorkstation**, **UtilsCommon.StartApp**, **UtilsCommon.TakeScreenshot** (it uses AutoIt for screen logging). There are probably more, work with your developer counterpart to make those test cases independent and that will execute in a Browser only environment.

If your tests leave Zombies (a Browser process is still running after you're done(test case is killed/dead) with the Browser) behind it will cause a slowness and eventually lock-up the NODE machine. We're working on a Zombie killer but not there yet as it can only run when the HUB has no activity with that NODE.

Flow- The computer that your test case executes on is the same machine that will have the results. It will point to the HUB <http://tellusgridb04.pc.factset.com:4444/wd/hub> and the HUB will manage the Browsers on the NODEs. Other than that you don't care about how the other stuff works. Each of the two nodes have 60 Browsers available (30 FireFox and 30 Chrome Browsers). Sorry but no, you cannot logon to the HUB or NODE machines.

[Instructions Using Tellus Devel Selenium Grid](#)

We all have a very large learning curve on S-G and we don't expect to "get it" at first. If you are confused and in danger please file an RPD to App QA Automation(Grid), however read the OneNotes(above and below) that should answer a lot of your questions.



Installing the Hub and Node. [Installation of Selenium Hub and Node](#)

Tellus Selenium Grid, Where are we at tomorrow?

Well after we add S-G to our Tellus UI and Framework the next thing is to add usage metrics. After that it build a production testing S-G environment. Then followed by Continuous Integration and Continuous Delivery automated testing environments..

SeleniumGrid Workflow

Thursday, January 22, 2015 11:19 AM

For Production

- Think of it
- First there has to be a Tellus Development Selenium-Grid environment setup.

Tellus Selenium Grid automated tests

Monday, February 02, 2015 12:32 PM

[RPD:15519990](#)

Determine which Selenium test cases use RunApp or other workstations dependencies

/smqa/Tellus/Selenium/Utils/

UtilsAll.rb

UtilsDriver.rb

UtilsInstall.rb

UtilsSTU.rb

/smqa/Tellus/Selenium/Snapshot/

Snapshot2 - AutomationURL.rb

/smqa/Tellus/Selenium/ProtractorWithRuby/

AutoTestList.rb

URL.rb

/smqa/Tellus/Selenium/BookBuilder/

BBCreatBook_DynamicReports.rb

Create BookURL.rb

CreateApplyEditDeleteTemplates_Dynamic_Placeholders.rb

CreateApplyEditDeleteTemplates_Static_Placeholders.rb

TemplatesURL.rb

View LinkURL.rb

/smqa/Tellus/Selenium/STU_Automation/

STU_AutomationURL.rb

/smqa/Tellus/Selenium/Filings2.0/

BlacklineExhibits.rb

BlacklineURL.rb

Filings CategoriesURL.rb

Key Exhibits CategoriesURL.rb

SEC Red Flags CategoriesURL.rb

ToolbarURL.rb

smqa/Tellus/Selenium/Callstreet/

Callstreet - Automation Part1URL.rb

DaysViewAndColumns.rb

WeekView.rb

/smqa/Tellus/Selenium/All Estimates - Automation/

All Estimates - Part1 - AutomationURL.rb

All Estimates - Part2 - AutomationURL.rb

All Estimates - Part3 - AutomationURL.rb

/smqa/Tellus/Selenium/Company_Security Analysis/

Supply Chain - AutomationURL.rb

Ken, still waiting on an easy to understand list of what can run on Grid vs. not for all of your Selenium tests across all apps.

A table like this would be a big help:

APP	TEST CASE NAME	TEST CASE ID (link)	Selenium Grid Native Status	Grid Notes
PA3	Blah	123	Yes	
PA3	Bleh	234	No	Needs IL4 to be HTML, link to tellus script with the dependent call: ...
etc				

-Jay

Look in this folder.



Have 1 test case from Mobile (iPad - Login and General Testing - IOS 7.0) on my laptop, pending code review.

Thanks

-Krishna

In the above I forgot to mention that these test cases need morph functionality to be added from engineers so it doesn't run on selenium grid now .

Thanks,

Surendra

I have 3 test cases in perforce of FactSet Notify SWE_Messages test plan ([RPD:13859914](#)) which can be run on selenium grid , currently those are running on a Dedicated laptop

Thanks,

Surendra

I have 5 test cases of **Callstreet Automation Part1** test plan on Perforce as well as on my laptop.

I also tested these on S-G setup on my laptop (on Chrome and IE browsers), and they are working fine.

Thanks,

Arvind

Zero from Analytics ☺

-Bhupender

I have two test cases (from List Builder) in Perforce and six test cases(from Orders 2.0) on my laptop that need to run in the selenium grid. But they need morph tool to run in the grid. Since there is an issue with the morph, these test cases were unable to run in the grid.

Thanks,

Gireesh

PA3 test scripting introduction

Monday, July 14, 2014 1:06 PM

Description

For PA3 testing automation scripting, we use JavaScript with WebStorm IDE, Node.js, Karma, and Protractor.

And we use "Page Objects" for re-usable abstraction of the Web page and its elements:

<https://code.google.com/p/selenium/wiki/PageObjects>

Development Environment Setup

[Installing Node.js, Protractor, and Karma](#)

[WebStorm: Node.js Setup](#)

[WebStorm: Karma Setup](#)

[WebStorm: Protractor Setup](#)

[Introduction to Selenium Promises](#)

[Selenium Page Object Pattern](#)

[Getting Started with jasmine-reporters with Protractor](#)

[Protractor default timeout 11 seconds](#)

Selenium JavaScript Documentation

<https://code.google.com/p/selenium/wiki/WebDriverJs>

Installing Node.js, Protractor, and Karma

Thursday, February 13, 2014 1:26 PM

tag installation

Prerequisite:

EDIT: It turns out that protractor's selenium-webdriver is written in JavaScript, so it doesn't need Java.

Installing node.js

1. Download the installer from <http://nodejs.org/download/>
2. Click **Windows Installer** (will download 32-bit MSI file)
3. Run the downloaded MSI installer

Installing Jasmine (for node.js)

```
npm install -g jasmine-node
```

Note: Jasmine must be installed globally with the -g option.

Reference: <https://github.com/mhevery/jasmine-node>

Installing Protractor

```
npm install -g protractor
```

That will install protractor command under this path:

C:\Users\username\AppData\Roaming\npm

or on some machines, under C:\Program Files\...

Put that path in your **PATH** environment variable.

Reference: <https://github.com/angular/protractor>

Installing Selenium Server from Protractor

Note: The following Standalone Server from Protractor is not needed if you are going to run against an existing Selenium Server or Selenium Grid.

The following will install selenium standalone server and chromedriver:

```
webdriver-manager update
```

To start Selenium server:

```
webdriver-manager start
```

It will start standalone Selenium Server on port 4444, by default.

Protractor Test Setup

Note: Protractor requires that the web page under test uses AngularJS:

1. <html> tag has "ng=app" attribute
2. Need to add <script> tag for angular.js

<Web server root>\hello.html:

```
<!doctype html>
<html ng-app>
  <head>
    <title>hello</title>
  </head>
  <body>
    <h1>Hello, Protractor!</h1>
  </body>
  <script
    src="https://ajax.googleapis.com/ajax/libs/angularjs/1.2.7/angular.min.js">
  </script>
</html>
```

Note: If the hello.html is not in the root of the web server, then you need to adjust the path to match in the index.spec.js file below.

Myproject\test\protractor.conf.js:

```
// This is a minimal example
exports.config = {

  specs: [
    './*.spec.js'
  ],

  baseUrl: 'http://127.0.0.1:80'
};
```

Myproject\test\index.spec.js:

```
// Gets title and compare with 'hello'
// Test Suite
describe("protractor hello test suite", function() {
  var ptor = protractor.getInstance();

  describe("index", function() {
    it("should display the correct title", function () {
      ptor.get("/hello.html");
      expect(ptor.getTitle()).toBe('hello');
    });
  });
});
```

Running Protractor Test

```
Prompt> cd Myproject
Prompt> protractor test\protractor.conf.js
```

It should start Chrome browser and console messages:

```
Prompt> protractor test\protractor.conf.js  
Starting selenium standalone server...  
Selenium standalone server started at http://164.55.146.120:57496/wd/hub  
.
```

```
Finished in 0.595 seconds  
1 test, 1 assertion, 0 failures
```

```
Shutting down selenium standalone server.
```

It shows that it passed (0 failures).

Reference

<https://egghead.io/lessons/angularjs-getting-started-with-protractor>
(Video)

WebStorm: Node.js Setup

Friday, January 10, 2014 10:52 AM

Node.js Setup

Setting up Node.js

1. Select **File > Settings**
2. Create a project:
 - a. Select **File > New Project**
 - b. **Project Type**: Empty project
3. Under Project Settings, Select **JavaScript > Libraries**
4. Check **Enabled** for "**Node.js v0.10.26 Core Modules**" with Type: Global, *or* click **Add** to add your node.js:
 - a. **Name**: Node.js v0.10.26 Core Modules
 - b. **Framework** type: Node.js Core Modules
 - c. **Version**: v0.10.26
 - d. **Visibility**: Global
 - e. **Documentation** URLs: <http://nodejs.org/api>
 - f. Click **OK**
5. Check **HTML5/EcmaScript 5**
6. Click **OK**

Setting up Node.js Source

1. Select **File > Settings**
2. Select **JavaScript > Node.js** and **NPM**
3. Click **Directory for Node.js source location**
4. C:\src\node-v0.10.24\
Note: It expect *.js to be in lib subdirectory under that directory.
5. Leave Usage Scope as "Associate with project"
6. Click **Configure**
7. Click **OK**

WebStorm: Karma Setup

Thursday, January 09, 2014 1:08 PM

Karma Setup

WebStore supports Karma by default.

WebStorm seems to need Karma installed globally:

```
$ npm install -g karma
```

If you installed it locally (without -g), the karma test runs may fail.

Create project

1. Select File > New Project
2. Enter Project name: TryKarma
3. Select Project type: Empty Project
4. Click OK

Create directories

1. Right-click project
2. Select New > Directory
3. Enter test for directory name
4. Click OK

Create a 'basic' folder under test folder.

Create Karma config file

```
C:\Dev\TryKarma>karma init karma.conf.js
```

Which testing framework do you want to use ?

Press tab to list possible options. Enter to move to the next question.

```
> jasmine
```

Do you want to use Require.js ?

This will add Require.js plugin.

Press tab to list possible options. Enter to move to the next question.

```
> no
```

Do you want to capture a browser automatically ?

Press tab to list possible options. Enter empty string to move to the next question.

```
> Chrome
```

```
>
```

What is the location of your source and test files ?

You can use glob patterns, eg. "js/*.js" or "test/**/*.Spec.js".

Enter empty string to move to the next question.

```
> test/**/*.Spec.js
```

```
WARN [init]: There is no file matching this pattern.
```

```
>
```

Should any of the files included by the previous patterns be excluded ?
You can use glob patterns, eg. "**/*.swp".
Enter empty string to move to the next question.
>

Do you want Karma to watch all the files and run the tests on change ?
Press tab to list possible options.
> no

Config file generated at "C:\Dev\TryKarma\karma.conf.js".

Setup Autocomplete for Jasmine

1. Select **File > Settings** (Preferences on Mac)
2. Select **JavaScript > Libraries**
3. Click **Add...**
4. Click **+** button
5. Select **Attach Directories...**
6. Select **karma-jasmin/lib**,
under %appdata%/roaming/npm/node_modules/ if installed globally
Note: `npm install karma` creates karma-jasmin as well karma folder.
7. Click **OK**
Note: It will show jasmin.js, index.js, and adaptor.js files added to the list box.
8. Enter Name: **karma-jasmin**
9. Leave "**Framework type**" to the default "**Custom**" for this example.
10. Click **OK** again
11. Check to enable the newly added karma-jasmine, if not already
12. (Optional) To limit karma to test directory:
 - a. Click **Manage Scopes...**
 - b. Select the test directory name under the project
 - c. Click **Apply**
 - d. Click **OK**
13. Click **OK**

Now the code will have autocomplete.

Project/test/basic/mySpec.js:

```
describe("my suite", function() {  
  it("should be true", function() {  
    expect(true).toBe(true);  
  });  
});
```

Running Test

1. Right-click karma.conf.js
2. Select Run karma.conf.js
3. Or click the green arrow "Rerun karma.conf.js" (if you have run it before)

WebStorm: Protractor Setup

Friday, January 10, 2014 10:58 AM

WebStorm Protractor Configuration (for each test script)

1. Get the node path. (If on UNIX, type: `which node` command in your terminal; on Windows 7, it would be `C:\Program Files\nodejs\node.exe`)
2. Click on **Run > Edit Configurations**
3. Click on the + ("plus") button to add a new Node configuration.
4. Select **Node.js**
5. Under **Configuration** tab, enter the following:
 - **Path to Node ("Node interpreter" on Version 7):** `C:\Program Files\nodejs\node.exe`
(This is path from step 1.)
 - **Node parameters:** (blank)
 - **Working directory:** `C:\Users\username\AppData\Roaming\npm\node_modules\protractor`
 - **IMPORTANT:** Working Directory must be Protractor installation directory in order for debugging to work.
 - **Javascript file:** `lib\cli.js`
 - **Application parameters:** `/full/path/to/protractor.conf.js`
6. Click **OK**

Example Files

1. Create a project first: Select **File > New Project...** (project type "Empty Project")
2. Create the following files.

Project/test/protractor.conf.js

```
// Protractor test configuration file
exports.config = {
  seleniumAddress: 'http://localhost:4444/wd/hub',
  specs: [
    './e2e/**/*.spec.js'
  ],

  // Capabilities to be passed to the webdriver instance.
  capabilities: {
    browserName: 'chrome'
    //chromeOptions: {binary: 'C:\\Applications\\CEFClient\\cefclient.exe'}
  },

  baseUrl: 'http://localhost:80'
};
```

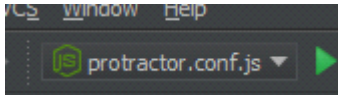
Project/test/e2e/index/index.spec.js

```
// This test gets page title
describe("test suite", function() {
  // Increase jasmine default timeout (default 5 seconds),
  // else may timeout with CEF client
  var timeout = 10000; // 10 seconds
  jasmine.getEnv().defaultTimeoutInterval = timeout;
  var ptor = protractor.getInstance();
```

```
describe("index test case", function() {
  it("should display the correct title", function () {
    ptor.get("/test.html");
    expect(ptor.getTitle()).toEqual('Test');
  });
});
});
```

Running Protractor Test

1. You need to run Selenium Standalone Server first, on Command Prompt:
`webdriver-manager start`
2. Put some breakpoints in your test script (*index.spec.js*)
3. On toolbar, select the new configuration:



4. Click **Debug** (Shift+F9)



CAUTION: Do not run test by right-clicking *protractor.conf.js* in Project Explorer and select Run; it will create a new Run Configuration again, and it might not run correctly.

5. It should now stop at the breakpoints

Reference

<https://github.com/angular/protractor/commit/8154adf5ca01ab43e47d5fa02e4123d1dcbd0aea>

Introduction to Selenium Promises

Friday, April 25, 2014 4:48 PM

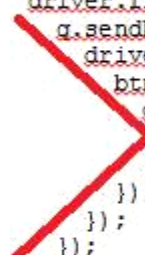
Introduction

Selenium *WebDriverJS* API is a JavaScript language binding for Selenium WebDriver. (It is not to be confused with a separate project called *Camme/WebDriverJS*.) Selenium WebDriverJS is entirely asynchronous. The asynchronous functional programming is a different paradigm that may require a different mode of thinking. Instead of returning the result from the function directly, the result is not available until sometime later in a callback function. That creates a program flow of control that looks very different from a synchronous program, and the program could look very complex very quickly.

For example, in Java,

```
driver.get("http://some.example.com");
driver.findElement(By.name("q")).sendKeys("webdriver");
driver.findElement(By.name("btnG")).click();
assertEquals("Example Test", driver.getTitle());
```

Rewritten in JavaScript, it looks much more complex. The asynchronous results are handled in callback functions. The increasing indentation with increasing number of callbacks looks like a “pyramid”, and it could quickly get out of hand:



```
driver.get("http://some.example.com", function() {
  driver.findElement(By.name("q"), function(q) {
    q.sendKeys("webdriver", function() {
      driver.findElement(By.name("btnG"), function(btnG) {
        btnG.click(function() {
          driver.getTitle(function(title) {
            assertEquals("Example Test", title);
          });
        });
      });
    });
  });
});
```

That coding pattern using callbacks is called *continuation passing*.

Selenium WebDriverJS provides *promises* that may simplify the syntax. It tracks the state of operations internally, so that the example above becomes simpler again:

```
driver.get("http://some.example.com");
driver.findElement(By.name("q")).sendKeys("webdriver");
driver.findElement(By.name("btnG")).click();
driver.getTitle().then(function(title) {
  assertEquals("Example Test", title);
});
```

Under the hood, each asynchronous call puts a task on the queue, and each is executed in order, when the previous one completes.

The Selenium WebDriverJS Promise API is based on that of Dojo Toolkit.

The Concept of Promise

Let's image this scenario: You go to a watch repair shop to repair a watch, but it will take some time to repair it, and you cannot wait for it. So the repairman gives you a promise to deliver the watch to you later. You leave your phone number with him. When the watch has been repaired, the repairman calls you and delivers the watch to you.

There are two parts of a Promise API, the *defer* object, which is the producer, and the *promise* object, which is the consumed. Like the repairman giving a promise to deliver, the deferred object returns a promise.

Example of a promise returned by WebDriverJS:

```
var promise = driver.getTitle();
```

Like leaving a callback phone number, you register a callback function (`function(title){...}`) by calling the `then` method of the promise object. For example:

```
driver.getTitle().then(function(title) {  
    assertEquals("Example Title", title);  
});
```

Like the repairman calling you and delivering the goods to you when the watch is ready, the *defer* object calls your callback function and pass it the result. For example:

```
defer.fulfill(result);
```

The *defer* object could *fulfill* the promise (in the “success” case), or *reject* it (in the “failure” or “error” case). The *fulfill* method is also called *resolve* in some implementations.

Simplified Promise Object

The following is a simplified implementation of what a promise object might look like. This is to illustrate the workings of a Promise, and it is not a real implementation of a library.

First, we need to create a promise object with an array of callbacks:

```
var promise = {  
    callbacks: []  
}
```

Now add callbacks with the method called `then`:

```
var promise = {  
    callbacks: [],  
    then: function (callback) {  
        callbacks.push(callback);  
    }  
}
```

You can use call the `then` function to register your callback function.

Now we rename the “fulfill” callbacks to `fulfillCallbacks`, and implement the error callbacks, `rejectCallbacks`:

```
var promise = {  
    callbacks: [],
```

```

    fulfillCallbacks: [],
    rejectCallbacks: [],
    then: function (fulfillCallback, rejectCallback) {
        fulfillCallbacks.push(fulfillCallback);
        if (rejectCallback) {
            rejectCallbacks.push(rejectCallback);
        }
    }
}

```

Note: The erased code is shown crossed out.

Simplified Deferred Object

A defer object could contain a promise object:

```

var defer = {
    promise: promise
};

```

It needs a “fulfill” function (also called “*resolve*”):

```

var defer = {
    promise: promise,
    resolve: function (data) {
        this.promise.fulfillCallbacks.forEach(function(callback) {
            window.setTimeout(function () {
                callback(data)
            }, 0);
        });
    },
};

```

That resolves a deferred action (the success case). It will call your callback function, and pass the result to it as an argument.

The defer object also needs a reject function for the error case:

```

var defer = {
    promise: promise,
    resolve: function (data) {
        this.promise.fulfillCallbacks.forEach(function(callback) {
            window.setTimeout(function () {
                callback(data)
            }, 0);
        });
    },
    reject: function (error) {
        this.promise.rejectCallbacks.forEach(function(callback) {
            window.setTimeout(function () {
                callback(error)
            }, 0);
        });
    }
};

```

The reject method notifies that the result failed.

Creating Promise and Deferred Types

Now, let's convert both objects to types, first the promise:

```
var Promise = function () {
  this.fulfillCallbacks = [];
  this.rejectCallbacks = [];
};

Promise.prototype = {

  fulfillCallbacks: null,

  rejectCallbacks: null,

  then: function (fulfillCallback, rejectCallback) {

    fulfillCallbacks.push(fulfillCallback);

    if (rejectCallback) {

      rejectCallbacks.push(rejectCallback);

    }

  }

};
```

And the following is the Deferred type:

```
var Deferred = function () {
  this.promise = new Promise();
};

Deferred.prototype = {

  promise: null,

  resolve: function (data) {

    this.promise.fulfillCallbacks.forEach(function(callback) {

      window.setTimeout(function () {

        callback(data)

      }, 0);

    });

  },

  reject: function (error) {

    this.promise.rejectCallbacks.forEach(function(callback) {

      window.setTimeout(function () {

        callback(error)

      }, 0);

    });

  }

};
```



```
}  
};
```

Usage:

And here is an example usage:

```
function doAsync() {  
    var defer = new Deferred();  
    // Do some async task...  
    somethingAsync(function (request) {  
        if (request.status === 200) { // status success  
            // Resolve the deferred job (success case)  
            defer.resolve(request.responseText);  
        } else {  
            // Fail the job  
            defer.reject(new Error("Status " + request.status));  
        }  
    });  
    // This is async, so return a promise  
    return defer.promise;  
}  
  
// doAsync() returns a promise, and we  
// use then() to register callback with that promise  
var promise = doAsync();  
promise.then(function (text) {  
    // "Fulfill" callback  
    alert(text);  
}, function (error) {  
    // "Reject" callback  
    alert(error.message);  
});
```

It calls the `doAsync` function, which returns a promise. Then it calls the `then` function of the promise to add fulfill and reject callback functions. The defer object later calls the “fulfill” or “reject” method, which will call your “fulfill” callback or “reject” callback function, respectively.

Using the Selenium WebDriverJS Promises

First, require the library:

```
var webdriver = require('selenium-webdriver');
```

To create a deferred object:

```
var defer = webdriver.promise.defer();
```

The returned object is Deferred type.

To get a promise:

```
var promise = defer.promise;
```

To fulfill the promise:

```
defer.fulfill(result);
```

The `result` is the result of some asynchronous operation. Calling `fulfill` will trigger your callback function that is passed to the `then` function.

To reject the promise:

```
defer.reject(msg);
```

Usually one would pass a reason for the rejection as the argument.

Promise Chaining

One could chain the promises, for example:

```
getUserById(id)
  .then(getFilesByUser)
  .then(deleteFile)
  .then(promptResult);
```

The implementation chaining gets more complex, but the main change is that the `then` function returns a promise. Here is the promise with chaining:

```
var Promise = function () {
  this.fulfillCallbacks = [];
  this.rejectCallbacks = [];
};
```

```
Promise.prototype = {
  fulfillCallbacks: null,
  rejectCallbacks: null,
  status: 'pending',
  error: null,
```

```
  then: function (fulfillCallback, rejectCallback) {
```

```
    var defer = new Deferred();
```

```
    // Add callbacks to the arrays with the defer binded to these callbacks
```

```
    this.fulfillCallbacks.push({
```

```
      func: fulfillCallback,
```

```
      defer: defer
```

```
    });
```

```
    if (rejectCallback) {
```

```
      this.rejectCallbacks.push({
```

```
        func: rejectCallback,
```

```
        defer: defer
```

```
      });
```

```
    }
```

```
    // Check if the promise is not pending. If not call the callback
```

```

    if (this.status === 'resolved') {
        this.resolveCallback({
            func: fulfillCallback,
            defer: defer
        }, this.data)
    } else if (this.status === 'rejected') {
        this.executeCallback({
            func: rejectCallback,
            defer: defer
        }, this.error)
    }

    return defer.promise;
},

executeCallback: function (callbackData, result) {
    window.setTimeout(function () {
        var res = callbackData.func(result);

        if (res instanceof Promise) {
            callbackData.defer.bind(res);
        } else {
            callbackData.defer.resolve(res);
        }
    }, 0);
}
};

```

And the defer:

```

var Deferred = function () {
    this.promise = new Promise();
};
Deferred.prototype = {
    promise: null,
    resolve: function (data) {
        var promise = this.promise;
        promise.data = data;
        promise.status = 'resolved';
        promise.fulfillCallbacks.forEach(function (callbackData) {
            promise.executeCallback(callbackData, data);
        });
    }
};

```

```

    });
  },
  reject: function (error) {
    var promise = this.promise;
    promise.error = error;
    promise.status = 'rejected';
    promise.rejectCallbacks.forEach(function(callbackData) {
      promise.executeCallback(callbackData, error);
    });
  },
  // Make this promise behave like another promise:
  // When the other promise is resolved/rejected this is also resolved/rejected
  // with the same data
  bind: function (promise) {
    var that = this;
    promise.then(function (res) {
      that.resolve(res);
    }, function (err) {
      that.reject(err);
    })
  }
};

```

WebDriverJS Promise Chaining

WebDriverJS Promise chaining has the same syntax as the simplified implementation above. However, unlike that implementation, if you return a promise from the then function callback in WebDriverJS, the next chained `then` method callback will receive the result of the returned promise.

Example:

```

var input = driver.findElement(webdriver.By.name('person_name'));
input.sendKeys('joe');
driver.findElement(webdriver.By.name('person_email')).sendKeys('joe@example.com');
// This will run through 'then's one-by-one,
// till the "oops_bad_name" fails
driver.findElement(webdriver.By.name('person_name')).getText()
.then(function(text) { // first then
  // Check name result
  if (text !== "joe") {
    console.log("=> ERROR: unexpected name: ", text);
    return; // will not continue with next operation
  } else {
    console.log("INFO: name success");
  }
  // First operation is successful, queue up next operation

```

```

    return driver.findElement(webdriver.By.name('person_email')).getText();
}).then(function(text) { // second then
    // Check email result
    if (text !== "joe@example.com") {
        console.log("==> ERROR: unexpected email: ", text);
        return;
    } else {
        console.log("INFO: email success");
    }
    // Do next operation:
    return driver.findElement(webdriver.By.name('person.oops_bad_name')).getText();
    // will intentionally fail
}).then(null, function(e) { // third then
    // Catch failure for all above
    console.log("caught ", e);
    // => caught { [Error: No driver.findElement found using locator:
webdriver.By.name("person.oops_bad_name")] webdriver_promise_error_: true }
});

// We intentionally fail the second operation below.
driver.findElement(webdriver.By.name('person_name')).getText().then(function(text) {
    if (text !== "joe") {
        console.log("==> ERROR: unexpected name: ", text);
        return; // will not continue with next operation
    }
    return driver.findElement(webdriver.By.name('person_email')).getText(); // next
}).then(function(text) {
    if (text !== "oops.typo@example.com") {
        console.log("==> ERROR: unexpected email: ", text); // will fail
        // Cause test failure (shows as 1 failure):
        throw new Error("User-thrown ERROR: unexpected email");
    }
    return driver.findElement(webdriver.By.name('person.oops_bad_name')).getText();
});

```

Output:

```

INFO: name success
INFO: email success
caught { [Error: No element found using locator: by.binding("person.oops_bad_name")]
webdriver_promise_error_: true }
==> ERROR: unexpected email:  joe@example.com

```

What Just Happened

In the callback of the first `then`, we return the promise for finding “person_email” element, then we handle that element in the second `then`. In the second `then`, we return a promise with a bad element name, causing the test to fail (throw exception). But as the third `then` defines a `failureCallback` (second parameter),

```

}).then(null, function(e) { // third then
...

```

Then it will catch the failure.

Conditional Branching or Processing Result

That promise chaining behavior could be useful if you do further processing of asynchronous result in the `then` callback, such as conditional branching based on the result.

Tip: Conditional branching may increase complexity, so for simple tests, that is usually not necessary. Using `expect` (Jasmine unit test) or `assertEquals` function to validate the results may be more desirable in many cases, in terms of simplicity:

```
expect(browser.getTitle()).toBe("Portfolio Analysis");
```

Example

```
element(by.xpath(xpath)).isPresent()  
.then(function(isPresent){  
    var msg;  
    if(isPresent) {  
        return element(by.id('a'));  
    } else {  
        return element(by.id('b'));  
    }  
}).then(function(result_a_or_b) {  
    // handle 'a' or 'b' above  
});
```

Handling Multiple Promises

Example: Fulfilling Promise

```
var webdriver = require('selenium-webdriver');  
...  
var deferred = webdriver.promise.defer();  
var promise = deferred.promise;  
var deferred2 = webdriver.promise.defer();  
var promise2 = deferred2.promise;  
  
webdriver.promise.all([promise, promise2])  
    .then(function(resultArray) {  
        console.log("success: " + resultArray);  
    }, function(resultArray) {  
        console.log("fail: " + resultArray);  
    })  
  
// To resolve promise:  
deferred.fulfill("abc");  
deferred2.fulfill("xyz");
```

Output:

```
success: abc,xyz
```

The abc,xyz in output is string representation or resultArray, which is a simple array.

Example: Rejecting Promise

Similarly, if you call `deferred.reject()`, then it will call the failure callback with the first reject reason.

```
var webdriver = require('selenium-webdriver');  
...  
var deferred = webdriver.promise.defer();  
var promise = deferred.promise;  
var deferred2 = webdriver.promise.defer();  
var promise2 = deferred2.promise;  
  
webdriver.promise.all([promise, promise2])  
    .then(function(resultArray) {  
        console.log("success: " + resultArray);  
    }, function(resultArray) {  
        console.log("fail: " + resultArray);  
    })
```

```

        }, function(resultArray) {
            console.log("fail: " + resultArray);
        })

// To reject promise:
deferred.fail("no bood");

deferred2.fulfill("xyz");

```

Output:

fail: no good

Control Flow

Logging Issue

The following code looks like it will output the first log message, then call the asynchronous method, then when that's done.

```

Logger.log("starting test");
Browser.getTitle().then(function(title) {
    console.log("title");
})
Logger.log("test ended");

```

However, there is one problem. The second log method call could execute before the asynchronous method completes.

The ControlFlow object lets use put tasks in the flow of WebDriverJS execution. You can get that object from,

```
webdriver.promise.controlFlow()
```

If one wraps the Logger.log method in ControlFlow.execute method, then it will execute in the flow of WebDriverJS. Then the second log method call will be executed after the asynchronous method call (getTitle()) completes. For example:

```

Logger = {
    log: function(s) {
        webdriver.promise.controlFlow().execute(function() {
            logger._log(s);
        });
    },
    _log: function(s) {
        ...
    }
    ...
};

```

Tip: In some cases, using expect (Jasmine unit test) or assertXX (such as assertEquals) functions might be sufficient greatly simplifies the test script without logging.

Some Promise Library Implementations

Q.js

<https://github.com/kriskowal/q>

When.js

<https://github.com/cujojs/when>

rsvp.js

https://github.com/tildeio/rsvp.js?utm_source=javascriptweekly

node-promise

<https://github.com/kriszyp/node-promise>

jQuery.Promise

<http://jquery.com/>

Standard Proposals

CommonJS Promises Proposal

<http://wiki.commonjs.org/wiki/Promises>

Promise/A+ Proposal

<http://promisesaplus.com/>

Reference

<http://stackoverflow.com/questions/17718673/how-is-a-promise-defer-library-implemented>

<https://github.com/kriskowal/q>

<https://code.google.com/p/selenium/wiki/WebDriverJs>

<https://github.com/angular/protractor/blob/master/docs/api.md>

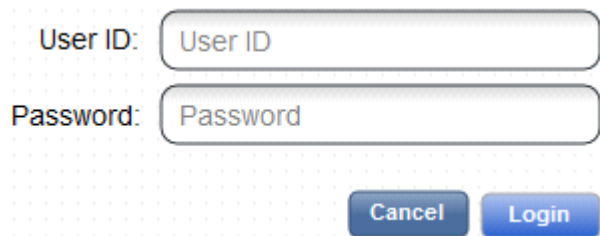
Selenium Page Object Pattern

Tuesday, May 27, 2014 10:02 AM

Basic Concept

Page Object is a layer of abstraction over the HTML and its automation code (a "wrapper"). The main goal is code reuse and abstraction. One would write a JavaScript "class" (type) for each HTML page or panel under test. The page object "class" resides in its own .js file. The test script "imports" the page object definition using the `require()` function.

Suppose we are testing a login dialog:



The image shows a login dialog box. It has a light gray background with a thin border. Inside, there are two input fields. The first is labeled "User ID:" and contains the text "User ID". The second is labeled "Password:" and contains the text "Password". Below the input fields, there are two buttons: "Cancel" and "Login". The "Cancel" button is on the left and the "Login" button is on the right. Both buttons are blue with white text.

Then we can write a page object in, say login.js, consisting of constructor and methods:

```
// Constructor
var Login = function() {
}

// Method
Login.prototype.login(userId, password) {
    element(by.id('#user-id')).sendKeys(userId);
    // ...
}

// Method
Login.prototype.logout = function() {
}

module.exports = Login;
```

Another style is:

```
// constructor
module.exports = function() {
    return {
        // method
        login = function(userId, password) {
            element(by.id('#user-id')).sendKeys(userId);
            // ...
        },
        // method
        logout = function() {
            // ...
        }
    };
};
```

```

    }
  };
};

```

In `protractor.spec.js` (test script), one `requires` the page object file:

```
var Login = require('./login.js');
```

Then construct the page object:

```
var login = new Login();
```

and login with different user IDs and passwords:

```
login.login('a', 'x');
```

So the test script looks like this:

```

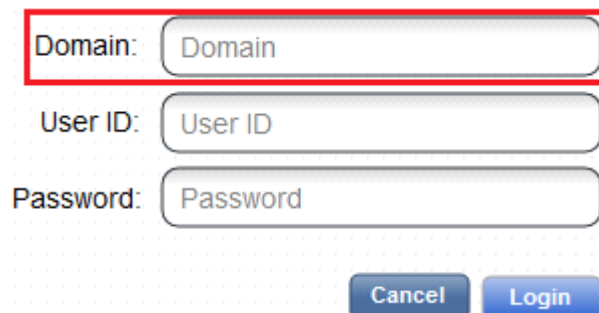
var Login = require('./login.js');
var login = new Login();
describe('Portfolio', function() {
  describe('login', function() {
    it('should login with minimum length', function() {
      login.login('a', 'x');
    });
    it('should login with special characters',
function() {
      login.login('@...', '!...');
    });
    afterEach(function() {
      login.logout();
    }
  });
});
});

```

The `login()` function is reused with different login arguments (just like a "utility" function).

Test Script Immune to Page Changes

If the page under test changes, then one only needs to change the page object (`login.js`), and the page object remains intact (`protractor.spec.js`). For example, suppose that the page now has a new Domain field:



Domain:

User ID:

Password:

Then the page object `login()` method will account for the new field, for example:

```
// ...
login = function(userId, password, domain) {
    // ...
    if (typeof domain != 'undefined') {
        // handle optional domain
    }
}
```

The benefit is that your existing test scripts will be immune to the change--they still work!

A Page Object for Every Page?

Do you need to write a page object for every page or panel? Some people have the opinion that "if you have WebDriver APIs in your test methods, you are doing it wrong". But others say that you only need page objects for reusable tests. Page object does have some overhead, but the overhead is small. Please exercise your best judgment and be consistent.

Reference

1. <https://code.google.com/p/selenium/wiki/PageObjects>
2. http://docs.seleniumhq.org/docs/06_test_design_considerations.jsp#page-object-design-pattern
3. <http://martinfowler.com/bliki/PageObject.html>
4. http://chon.techliminal.com/page_object/#/intro

Getting Started with jasmine-reporters with Protractor

Thursday, May 22, 2014 10:08 AM

Installing locally (in your NODE_PATH):

```
npm install jasmine-reporters
```

In protractor.conf.js:

```
// 1.0
exports.config = {
  ...
  onPrepare: function() {
    require('jasmine-reporters');
    jasmine.getEnv().addReporter(new
jasmine.JUnitXmlReporter('C:\\options'));
  }
};
```

jasmine-reporters 2.0:

Note: jasmine-reporters is broken in 2.0, such that it has no log output. Revert back to 1.0.

```
exports.config = {
  ...
  onPrepare: function() {
    var jasmineReporters = require('jasmine-reporters');
    var options = {
      savePath: 'C:\\test',
      consolidate: true,
      useDotNotation: true
    };
    jasmine.getEnv().addReporter(new
jasmineReporters.JUnitXmlReporter(options));
  }
};
```

Note: The jasmine global variable is not visible in outside onPreparse(), so you need to require('jasmine-reporters') inside onPrepare() (that module will check for existence of jasmine global variable). There doesn't seem to be an option to turn off stack trace in file. And the terminal output has no stack trace.

JUnitXmlReporter Constructor Description

JUnitXmlReporter (options)
(as of version 2.0.0.0)

Generates JUnit XML for the given spec run.

Properties of options Parameters

{string} savePath
where to save the files

{boolean} consolidateAll
 whether to save all test results in a single file (default: true)
 NOTE: if true, {filePrefix} is treated as the full filename (excluding extension)

{boolean} consolidate
 whether to save nested describes within the same file as their parent; default: true

{boolean} useDotNotation
 whether to separate suite names with dots rather than spaces (ie "Class.init" not "Class init"); default: true

{string} filePrefix
 is the string value that is prepended to the xml output file; default: 'TEST-'

Output

The output file will be in <savePath>/TEST-<test_suite_title>.xml
 where <test_suite_title> is top-level test suite title ('describe' block) converted to one word, no spaces.

Example XML Output

```
<?xml version="1.0" encoding="UTF-8" ?>
<testsuites>
<testsuite name="Angular test suite" errors="0" tests="0" failures="0"
time="0" timestamp="2014-05-21T10:07:36">
</testsuite>
<testsuite name="Angular test suite.Test case" errors="0" tests="1"
failures="1" time="4.525" timestamp="2014-05-21T10:07:32">
  <testcase classname="Angular test suite.Test case" name="expects to
equal with logging" time="4.524"><failure type="expect" message="Expected
&apos;Test&apos; to equal &apos;Wrong expected
title&apos;.">Error: Failed expectation
  at null.&lt;anonymous&gt; (C:\Users\username\Documents\Source
\TestProtractor\ex.spec.js:96:37)
  at flow.execute.then.e.stack (C:\Users\username\AppData\Roaming\npm
\node_modules\protractor\jasminewd\index.js:43:14)
  at webdriver.promise.ControlFlow.runInNewFrame_ (C:\Users\username
\AppData\Roaming\npm\node_modules\protractor\node_modules\selenium-
webdriver\lib\webdriver\promise.js:1598:20)
  at webdriver.promise.ControlFlow.runEventLoop_ (C:\Users\username
\AppData\Roaming\npm\node_modules\protractor\node_modules\selenium-
webdriver\lib\webdriver\promise.js:1463:8)
  at wrapper [as _onTimeout] (timers.js:252:14)
  at Timer.listOnTimeout [as ontimeout] (timers.js:110:15)</failure>
</testcase>
</testsuite>
</testsuites>
```

Example Terminal Output

Using ChromeDriver directly...
 F

Failures:

- 1) Angular test suite Test case expects to equal with logging
 Message:
 Expected 'Test' to equal 'Wrong expected title'.

Finished in 1.425 seconds
1 test, 2 assertions, 1 failure

Process finished with exit code 1

Protractor default timeout

Friday, February 21, 2014 1:12 PM

By default, Protractor sets the timeout for actions to 11 seconds. You can change this in your config with the `allScriptsTimeout` options.

```
// protractor.conf.js:
exports.config = {
  // Override the timeout for webdriver to 20 seconds.
  allScriptsTimeout: 20000,
}
```

The timeout for loading Web page is set in second parameter of `ptor.get()`:

```
ptor.get(url, timeout);
```

where `ptor` is the protractor instance.

Reference: <<https://github.com/angular/protractor/blob/master/docs/faq.md#angular-cant-be-found-on-my-page>>

Page Objects Documentation

Monday, July 14, 2014 1:20 PM

<https://code.google.com/p/selenium/wiki/PageObjects>

Selenium and Python

Thursday, March 15, 2012 8:35 AM

Selenium 2.20.0

Python bindings for Selenium

<http://pypi.python.org/pypi/selenium>

We want to use and control Selenium must like we do the Test Complete. Test Complete executes open source Jscirpt and is driven by Open Source STAF.

What we want to avoid is using Selenese. Our first choice would be Jscript followed by Python.

QA Mac Mini login information

Friday, October 31, 2014 3:06 PM

z

First:

Download and install (put the .exe in your PATH) TightVNC or any other VNC client.

If your Windows is 64-bit: <\\fds1.pc.factset.com\apps\VNC\T>

ightVNC\2.7.10\x64\tnviewer.exe

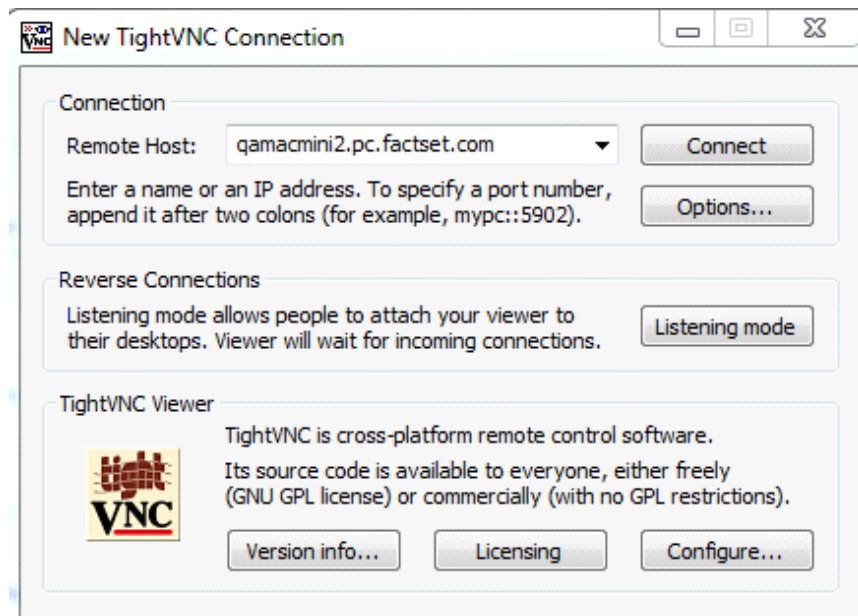
If your Windows 32-bit: <\\fds1\apps\vnc\TightVNC\2.7.10\x86\tnviewer.exe>

Two QA Mac Mini Machines:

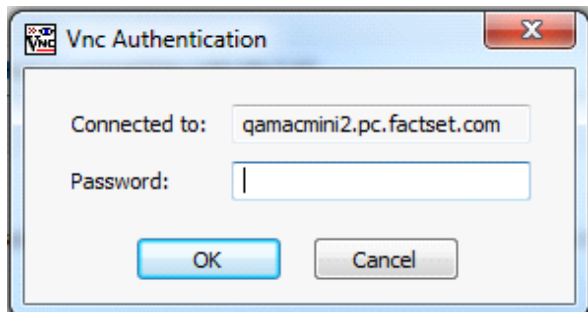
- qamacmini.pc.factset.com
- qamacmini2.pc.factset.com

Login example:

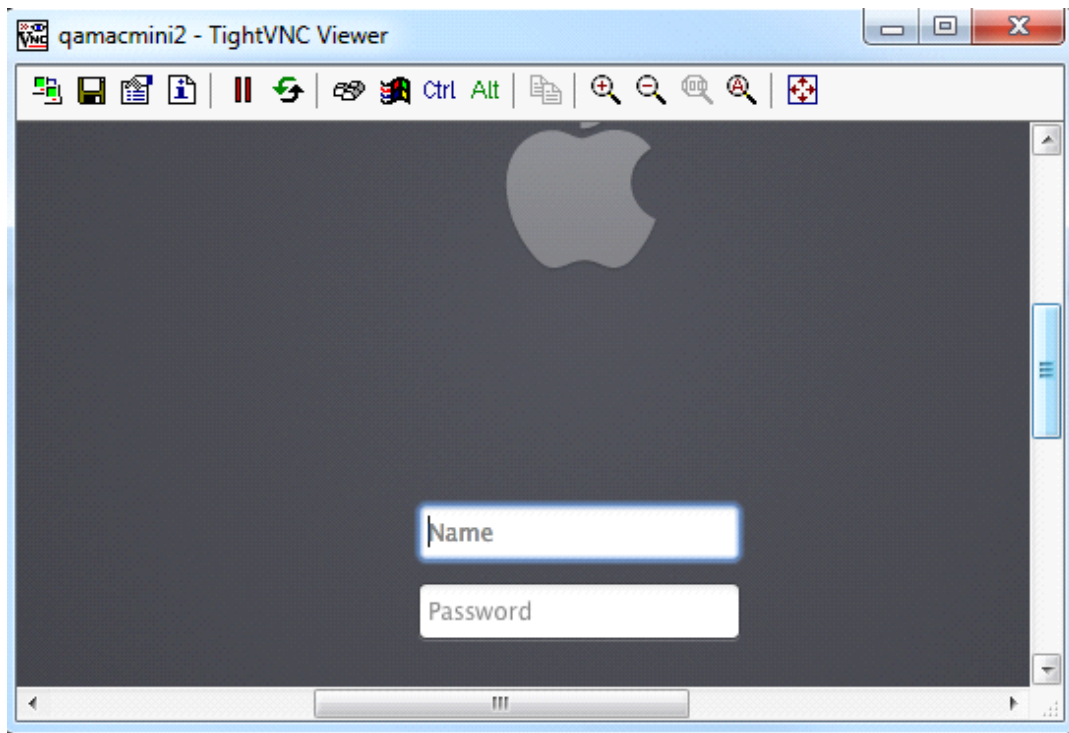
1. Start the VNC client **tnviewer.exe**
2. Enter **Remote Host:** **qamacmini.pc.factset.com** or **qamacmini2.pc.factset.com**
3. Click **Connect**



3. Password: **qa** and hit OK button



1. Enter your Windows username (without the domain) and the password
4. Press **Enter** key or click the → (arrow) next to the password box
5. And you will login Mac Mini



Appium Development Environment Setup

Thursday, December 11, 2014 11:39 AM

Overview

Multiple users can run Appium GUI at the same time. However, each user must own his/her own copy of Appium.app (GUI), tmp directory, and port.

Client script should use that port, if not using Selenium Grid.

Prerequisite

1. Appium requires Xcode
2. Xcode Installation:
 - a. Got to <https://developer.apple.com/xcode/downloads/>
 - b. You will need to login (free sign-up)
 - c. Currently, Appium (v1.2.x, 1.3.x) has compatibility issues with Xcode 6.1. Currently, it works better with Xcode version 5.1.1. Please make sure that other users can use that Xcode version (as Xcode configuration appears to be for all users.)
 - d. After Xcode installation, select the active version:
`sudo xcode-select -s /Applications/Xcode5.app/Contents/Developer`
3. Install Xcode command line tools:
 - a. `sudo xcode-select --install`
 - b. If there's error with accessing the server, download the .pkg manually from developer.apple.com, and double-click it to install

Installing ruby and packages

<http://appium.io/slate/en/tutorials/ios.html?ruby#install-ruby>

Note: These instructions install user's home directory, so each user will need to go through these steps.

- Install the latest stable release of Ruby.
`\curl -sSL https://get.rvm.io | bash -s stable`
`rvm get stable`
`rvm pkg install openssl`

If prompted for directory to install Homebrew, enter directory under home directory: `/Users/username/local`

```
export HOMEBREW_HOME=$HOME/local
export PATH=$PATH:$HOMEBREW_HOME/bin
And add those lines to ~/.bash_profile
```

- Clear previous installations:
`rvm uninstall all --force`
- Install Ruby:
`rvm install 2.1.5 --with-openssl-dir=$HOME/.rvm/usr --verify-downloads 1`
`rvm use 2.1.5`
- Check that it's installed properly by printing the ruby version.
`ruby --version`
- Update RubyGems and Bundler.
`gem update --system`
`gem install --no-rdoc --no-ri bundler`
`gem update`
`gem cleanup`
- Check that RubyGems is `>= 2.1.5`
`gem --version`
- Install appium_console gem.
`gem uninstall -aIx appium_lib`
`gem uninstall -aIx appium_console`
`gem install --no-rdoc --no-ri appium_console`
(That will include appium_lib as dependency.)

- Install [flaky](#) gem.
`gem uninstall -aIx flaky`
`gem install --no-rdoc --no-ri flaky`

Installing Node.js and packages

- If you use the big green install button on [nodejs.org](#) or all npm commands will require sudo.
 - If you don't have root privilege, then install [nodejs](#) using nvm, <https://github.com/creationix/nvm>:
 - Installing nvm:
`curl https://raw.githubusercontent.com/creationix/nvm/v0.20.0/install.sh | bash`
`source ~/.bashrc`
 - To list available node.js versions on remote repo:
`nvm list-remote`
 - To install node.js:
`nvm install 0.10`
 - In `~/.bash_profile`, add line:
`export PATH=$PATH:$HOME/.nvm/v0.10.33/bin`
(adjust for actual version)
 And source it:
`source ~/.bash_profile`
 - Node should be v0.10.26 or better.
`node --version`
`npm --version`
 - Install grunt.
`npm install -g grunt grunt-cli`
 - Run the version command from the appium folder below. If you're not in that folder, the grunt version will not display.
`grunt --version`
- Output:
`grunt-cli v0.1.13`
`grunt v0.4.2`

Appium GUI App Installation

1. Download from <http://appium.io>
 Note: Appium version 1.3.3 might cause simulator to lose Mobile Safari browser, so you should use version 1.3.4, instead.
2. Double click the downloaded .dmg
3. Drag the mounted Appium.app to a directory under your home folder in Finder, for example, `~/Apps/`.

Authorize access to iOS Simulator

When you start Appium.app for the first time, it will prompt to authorize that. You will need an admin account.

Using Appium GUI with Alternative Code (Optional)

1. Download Appium source:
`git clone https://github.com/appium/appium.git appium`
2. Check out version 1.3.4
`cd appium`
`git checkout tags/v1.3.4`
3. Build Appium with sample apps and Safari Launcher:
`./reset.sh --ios --launcher`

Note: `--launcher` builds SafariLauncher for simulator. An optional `--dev` builds sample apps (including `WebViewApp.app`)

Configuring Appium GUI to use alternative code:

1. Start Appium.app (GUI)
2. Click **Developer Settings** ("Wrench" icon)
3. Check **"Usage External Appium Package"** and enter its directory path, for example, `/Users/username/src/appium`

Starting Appium Server on Command Line (no GUI)

Note: This cannot be used with Appium.app GUI at the same time.

Starting Appium as standalone Selenium server:

```
node lib/server/main.js --log-level=warn
```

Note: Appium logging level is "debug" by default.

Starting Appium with "no-reset" option:

```
node lib/server/main.js --no-reset
```

Note: "Reset" sets simulator back to "factory state".

Reference

<http://appium.io/slate/en/tutorial/ios.html?ruby#install-overview>

<http://appium.io/slate/en/v1.2.0/?ruby#server-flagssm>

Appium iOS Ruby Example

Thursday, May 14, 2015 11:48 AM

```
require 'selenium-webdriver'
require 'appium_lib'

server_url = 'http://tellusdevb03.pc.factset.com:4444/wd/hub'
appPath = '/BuildArchives/2015
_V1/base_iphone_clientspec/2.14.32.1/release/bin/Release_Sim/FactSet.app'
capabilities = {
  :platform => 'MAC',
  :browserName => 'FactSet',
  :platformName => 'ios',
  :platformVersion => '8.2',
  :deviceName => 'iPhone 6',
  :app => appPath
}
driver = Appium::Driver.new(caps: capabilities, :appium_lib => {:server_url
=> server_url} ).start_driver
Appium.promote_appium_methods Object
driver.manage.timeouts.implicit_wait = 60

driver.find_element(:name, "OK").click # "Send Notifications" prompt
driver.find_element(:name, 'email or factset.net ID').send_keys
'user@factset.com'

sleep 5 # see better for debugging

driver_quit
```

Appium Selenium Grid node installation

Tuesday, November 4, 2014 8:51 AM

Note:

Appium v1.4.13 is compatible with node.js v0.12.7, iOS 9 and Xcode 7.

Appium Installation

1. Go to <http://appium.io>
2. Download the Appium .dmg (v1.3.4 or above)
3. Double click the downloaded .dmg
4. Drag the mounted Appium.app to a directory under your home folder in Finder, for example, ~/Server/.

Adding node.js to your PATH:

5. Add lines in ~/.bash_profile:
PATH="\$PATH:/Users/appium/Server/Appium.app/Contents/Resources/node/bin"
export PATH
6. Make it take effect right away:
source ~/.bash_profile

Authorizing access to iOS Simulator

1. Right-click Appium.app
2. Click Open
3. When you start Appium.app for the first time, it will prompt to authorize that.
4. Click Yes to authorize
5. You will need to enter an admin account username and password.

or on command line:

/Users/appium/Server/Appium.app/Contents/Resources/node_modules/.bin/authorize_ios

nodeconfig.js

This example configuration includes Mobile Safari, FactSet app on simulator or real device.

```
{
  "capabilities": [
    {
      "platform": "MAC",
      "platformName": "iOS",
      "browserName": "safari",
      "version": "8.0",
      "maxInstances": 1
    },
    {
      "platform": "MAC",
      "platformName": "iOS",
      "browserName": "FactSet",
      "maxInstances": 1
    }
  ],
  "configuration": {
    "hubHost": "tellusdevb03.pc.factset.com",
    "hubPort": 4444,
    "host": "selgridct01.pc.factset.com",
    "port": 4723,
    "url": "http://selgridct01.pc.factset.com:4723/wd/hub",
    "proxy": "org.openqa.grid.selenium.proxy.DefaultRemoteProxy",
    "maxSession": 1,
    "register": true,
    "registerCycle": 5000,
  }
}
```



```

    "cleanupCycle": 2000
  }
}

```

Note: It uses "FactSet" as (custom) browser name, so that the grid will find it.

Tip: Easier management for native app:

- In nodeconfig.json, omit "version"; it will accept any app version
- In client script, specify "app" path in capabilities

That way, you don't need to reconfigure server for every new app version.

Running Appium Server

```

NODE="/Users/appium/Server/Appium.app/Contents/Resources/node/bin/node"
APPIUM="/Users/appium/Server/Appium.app/Contents/Resources/node_modules/appium/lib/server/main.js"
"$NODE" "$APPIUM" --nodeconfig "$HOME/Server/nodeconfig.json"

```

You can check it in the grid console:

<http://tellusdevb03.pc.factset.com:4444/grid/console>

Appium as Launchd Service

This configuration automatically starts Appium as a launchd service when this user logs in, and stops when the user logs out:

1. Create com.factset.appium.plist file (example below)
2. Put the .plist in ~/Library/LaunchAgents/

Example ~/Library/LaunchAgents/com.factset.appium.plist

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
  <dict>
    <key>RunAtLoad</key>
    <true/>
    <key>KeepAlive</key>
    <dict>
      <key>SuccessfulExit</key>
      <false/>
    </dict>
    <key>Label</key>
    <string>com.factset.appium</string>
    <key>ProgramArguments</key>
    <array>
      <string>/Users/appium/Server/Appium.app/Contents/Resources/node/bin/node</string>
      <string>/Users/appium/Server/Appium.app/Contents/Resources/node_modules/appium/lib/server/main.js</string>
    </array>
    <key>Debug</key>
    <true/>
    <key>WorkingDirectory</key>
    <string>/Users/appium</string>
    <key>StandardOutPath</key>
    <string>/Users/appium/Library/Logs/appium.log</string>
    <key>StandardErrorPath</key>
    <string>/Users/appium/Library/Logs/appium_error.log</string>
  </dict>
</plist>

```

Registering Service

```
launchctl load ~/Library/LaunchAgents/com.factset.appium.plist
```

It will take effect right away.

Automating User Login

In order to keep the user logged in, one way is to use AutoIt script driving VNC viewer (for example, TightVNC) to programmatically log in the user on the Macintosh, and exit the VNC viewer without logging out.

Example AutoIt script

```
# Login Mac user via TightVNC Viewer and exit without log out
#region --- Au3Recorder generated code Start (v3.3.9.5 KeyboardLayout=00000409) ---

#region --- Internal functions Au3Recorder Start ---
Func _Au3RecordSetup()
    Opt('WinWaitDelay',100)
    Opt('WinDetectHiddenText',1)
    Opt('MouseCoordMode',0)
    Local $aResult = DllCall('User32.dll', 'int', 'GetKeyboardLayoutNameW', 'wstr', '')
    If $aResult[1] <> '00000409' Then
        MsgBox(64, 'Warning', 'Recording has been done under a different Keyboard layout' & @CRLF &
            '(00000409->' & $aResult[1] & ')')
    EndIf
EndFunc

Func _WinWaitActivate($title,$text,$timeout=0)
    WinWait($title,$text,$timeout)
    If Not WinActive($title,$text) Then WinActivate($title,$text)
    WinWaitActive($title,$text,$timeout)
EndFunc

_AU3RecordSetup()
#endregion --- Internal functions Au3Recorder End ---

Run('C:\Program Files\TightVNC\tnvviewer.exe')
_WinWaitActivate("New TightVNC Connection","Enter a name or an I")
MouseClick("left",347,58,1)
_WinWaitActivate("Vnc Authentication","Connected to:")
Send("pass{SHIFTDOWN}w{SHIFTUP}ord{ENTER}")
_WinWaitActivate("jhwangmac - TightVNC Viewer","jhwangmac - TightVNC")
; Clear Username field
Send("{CTRLDOWN}a{CTRLUP}{SHIFTDOWN}{END}{SHIFTUP}{BACKSPACE}")
Send("myname{TAB}")
Send("pass{SHIFTDOWN}w{SHIFTUP}ord{SHIFTDOWN}1{SHIFTUP}{ENTER}")
;Sleep(5000) ; debug
; Click X button to exit without logout
MouseClick("left",1428,9,1)
#endregion --- Au3Recorder generated code End ---
```

Appium iOS Basic Usage

Friday, February 06, 2015 10:35 AM

Appium GUI App Installation

1. Download from <http://appium.io>
Note: Appium version 1.3.3 might cause simulator to lose Mobile Safari browser, so you should use version 1.3.4, instead.
2. Double click the downloaded .dmg
3. Drag the mounted Appium.app to a directory under your home folder in Finder, for example, ~/Apps/.

Authorize access to iOS Simulator

When you start Appium.app for the first time, it will prompt to authorize that. You will need an admin account.

iOS Settings in Appium GUI

1. Click **iOS Settings**
2. Enter **App Path**: */path/to/FactSet.app*
3. Enter **BundleID** (this should be same as App Path)
4. Check **Force Device** and enter, for example, iPhone 6
IMPORTANT: Don't select any entry with a UUID in it. Manually type: iPhone 6
5. Enter **Platform Version**, for example, 8.1

Basic Operations

- Click **Launch** to start Appium server
- Click **Inspector** to start the Inspector

Using Appium GUI with FactSet app in iOS Simulator

Thursday, November 20, 2014 7:18 AM

iOS Settings

1. Start Appium (/Applications/Appium.app)
2. Click **iOS Settings**
3. Check **App Path** and enter /path/to/FactSet.app
4. Check **BundleID** and enter /path/to/FactSet.app
Note: Normally BundleID should be in the form, domain.company.ProductID, but for simulator, it currently works with a path.
5. Check **Force Device** and select a device
Note: This is required. And the device must be existing simulator.
6. Select a **Platform Version**
7. (Optional) Check **No Reset**, if need to re-use existing app in simulator. (Usually you don't need this.)

Using Appium Inspector

1. Click **Launch**
2. Wait for Status 200 ("OK") in the console message
3. Click **Inspector**

Introduction to iOS Test Scripting with Appium

Thursday, January 29, 2015 8:03 AM

How Appium Works with iOS automation

Appium is an implementation of Selenium server, it starts up an iOS simulator and drives an iOS app or Mobile Safari browser. The iOS app can be native or hybrid (with WebView embedded browser.)

Appium can run as a standalone Selenium server (non-grid), or as a node in a Selenium Grid.

When Appium is running as a standalone Selenium server, the client library would create a "Remote WebDriver" driver object using a local URL (in the form, <http://hostname:port/wd/hub>, for example, <http://0.0.0.0:4444/wd/hub>). The client library will connect to the Appium server and send WebDriver network protocol messages. Appium server will start the iOS simulator, install the app onto the iOS simulator (if with default "Reset" option, which resets the simulator to "factory condition"). It will launch the app, and drive it.

When Appium is running as a node in Selenium Grid, then one would use the hub hostname in the URL mentioned above, for example, <http://hub.example.com:4444/wd/hub>. The hub will relay the requests to the node, which is the Appium server.

There are Appium client libraries for various programming languages, including Ruby, Python, and Java. For ruby, you would use the `appium_lib` gem.

Contexts

In order to support both native and hybrid apps, Appium has the concept of *contexts*, which does not exist in the regular Selenium. The default `NATIVE_APP` context works with native objects.

When a WebView (an embedded Web browser) component is visible in the app, the dropdown also includes a `WEBVIEW` context, for example, `WEBVIEW_1`. You can use the Appium library `set_context` method (`set_context`) to switch context. After switching to a `WEBVIEW` context, you can use `find_element` as if it were targeting a Web page. It is usually less effective to access HTML element in `NATIVE_APP` context, so it is recommended to automate WebView in `WEBVIEW` context. If, after accessing WebView, you need to access native objects, then you can switch back to `NATIVE_APP` context.

Difference from Regular Selenium

In iOS, there is no `id` property in native objects, but there is `name` property, which is expected to be unique. So for iOS, one could find element by name: `find_element(:name, "someName")`

but not by `:id` for native objects. In `WebView` context, your `XPATH` could specify the `ID` attribute of the HTML element.

Testing Scripting

Appium Server

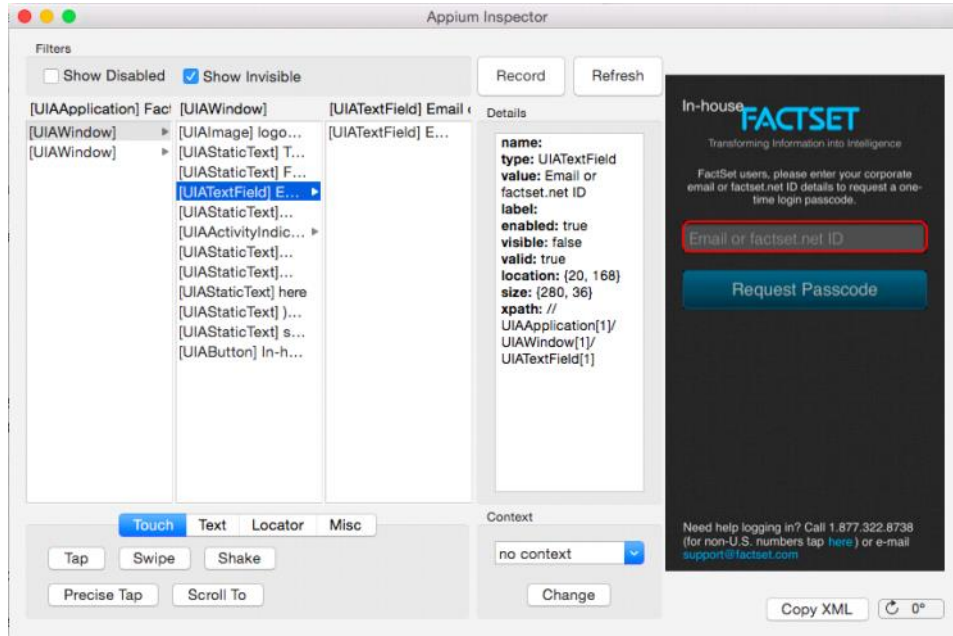
For developing tests, start the Appium GUI (Appium.app downloaded from <http://appium.io>). Please see the installation instructions in OneNote for the configurations: [Appium Mac OS/X Development Environment Setup](#)

To start the Appium server, click Launch. Wait for the status code 200 (Success) on the console, which indicates that the server is ready.

During development, it is recommended to run Appium as a standalone server, so that you can see the iOS simulator and the app on your local machine. (With Selenium grid, you lose the ability of control which machine will actually run the iOS simulator.)

Appium Inspector

Click Inspector button to start the Inspector. It is a tool that shows the object tree, as shown below.



You can select an object, and select an action to perform on it.

Inspecting WebView

By default, it is in NATIVE_APP context, which shows up in the Context dropdown list in the Inspector. When there is a WebView object on the screen, a WEBVIEW context will be available. You could select it and click Change to switch context. However, the XPATH in WEBVIEW context generated by Appium is not particularly useful. Instead, use Safari's Developer menu to inspect the HTML elements.

To Enable Developer Menu:

1. Select **Safari > Preferences**
2. Click **Advanced**
3. Check **Show Develop menu in menu bar**

To use WebKit Inspector:

1. Select **Developer > iOS Simulator > page name**
2. Click **Inspect** icon

Appium Recorder

In the Appium Inspector, you can click Record to start recording. It will generate a Selenium test script.

```
Ruby  Add Boilerplate  XPath Only  Replay  Undo  Redo  Clear
6   'platformVersion': '7.1',
7   'deviceName': 'ipad',
8 }
9
10 server_url = "http://0.0.0.0:4723/wd/hub"
11
12 Appium::Driver.new(caps: capabilities).start_driver
13 Appium.promote_appium_methods Object
14
15 find_element(:xpath, "//UIAApplication[1]/UIAWindow[1]/UIATextField[1]/UIATextField[1]").send_keys
16   "jhwang@factset.com"
17 driver_quit
```

Scripting

We will use Ruby for the the following examples. Other languages would have equivalent APIs.

First import the Appium client library:

```
require 'appium_lib'
```

If you are also using Watir, then import that too:

```
require 'watir-webdriver'
```

Specify the "Remote WebDriver" URL in the form <http://servername:port/wd/hub>, where *servername* is the Appium server or the hub:

```
server_url = 'http://hub.example.com:4444/wd/hub'
```

For local standalone server, the `server_url` would be <http://0.0.0.0:4444/wd/hub> for default port. For a grid environment, it would target the hub instead: <http://hub.example.com:4444/wd/hub>

You need to specify the capabilities for the driver. The keys, `platformName`, `platformVersion`, `deviceName`, and `app` are specific to Appium. For testing environment that will run the same test against different iOS device types and versions, the capabilities would need to be dynamic.

```
capabilities = {
  :platform => 'MAC',
  :platformName => 'ios',
  :platformVersion => '8.1',
  :deviceName => 'iPhone 6',
  :browserName => 'FactSet',
  :app => '/Users/Apps/2.11.12.0/FactSet.app'
}
```

Now start the driver:

```
driver = Appium::Driver.new(caps: capabilities, :appium_lib => {:server_url =>
server_url} ).start_driver
Appium::promote_appium_methods Object
driver.manage.timeouts.implicit_wait = 60
```

The `Appium::promote_appium_methods` line will promote the driver variable and Appium functions such as `find_element` to global scope. They would be visible to the main Ruby script, even if it is called in a Ruby module.

That is, you can access driver global variable in your test script like this:

```
driver.find_element(:xpath, "//UIAApplication[1]/UIAWindow[2]/UIATextField[1]").click
```

The driver variable is a regular Selenium WebDriver object, so you can use regular Selenium methods with it, for example `driver.find_element()`.

Some functions are also available in global scope directly:

```
find_element(:xpath, "//UIAApplication[1]/UIAWindow[2]/UIATextField[1]").click
```

WEBVIEW Context

When you navigate to a window with WebView object, then a WEBVIEW context will be available. To switch to a WEBVIEW context, for example,

```
driver.appium_device.set_context(driver.appium_device.available_contexts.last)
```

The last element in the `available_contexts` array is usually the WEBVIEW name (string), for example, "WEBVIEW_1", but you could verify that it begins with the "WEBVIEW" substring.

Tip: If you turn on "No Reset" option in Appium, then you need to switch back to "NATIVE_APP" context before closing driver, else it might show an empty Web page next time.

In WEBVIEW context, you can use regular Selenium element selectors, for example, by CSS selector, or XPATH with HTML elements:

```
# Find element by ID
driver.find_element(:css, "#mwTabEU").click
# Find HTML element by XPATH
head_line = driver.find_element(:xpath, "//div[text()='Latest on Bonds']/following-sibling::div").text
```

Using Watir with Appium

(Only if you use Watir library)

First import Watir library near the top of the script:

```
require 'watir-webdriver'
```

As the driver object returned by Appium is a regular Selenium WebDriver object, you can create Watir browser object from it:

```
browser = Watir::Browser.new(driver)
```

Then you could use Watir methods, for example `browser.select_list(...)`.

Using Appium GUI with Selenium Grid

To attach Appium (GUI) to a Selenium hub, click General Settings, check Selenium Grid Configuration File and enter the path to the `nodeconfig.json` (node configuration) file.

Example `nodeconfig.json`:

```
{
  "capabilities": [
    {
      "platform": "MAC",
      "platformName": "ios",
      "browserName": "safari",
      "version": "6.0",
      "maxInstances": 1
    },
    {
      "platform": "MAC",
      "platformName": "ios",
      "platformVersion": "7.1",
      "browserName": "WebViewApp",
      "version": "1.0",
      "maxInstances": 1
    },
    {
      "platform": "MAC",
      "platformName": "ios",
      "platformVersion": "7.1",
```



```

        "browserName": "FactSet",
        "version": "2.9",
        "maxInstances": 1
    },
    ],
    "configuration": {
        "proxy": "org.openqa.grid.selenium.proxy.DefaultRemoteProxy",
        "host": "jhwangmac.pc.factset.com",
        "port": 4723,
        "url": "http://jhwangmac.pc.factset.com:4723/wd/hub",
        "maxSession": 1,
        "register": true,
        "registerCycle": 5000,
        "hubPort": 4444,
        "hubHost": "tellusgridb04.pc.factset.com",
        "timeout": 120000,
        "browserTimeout": 120000,
        "cleanupCycle": 2000
    }
}

```

Please see Appium Selenium Grid setup instructions for more details.

Closing the Driver

You must quit the driver using,
`driver_quit`

Note: `driver close()` only closes the browser and might not clean up the session sufficiently.

Troubleshooting

Underneath, Appium uses the Instruments UI automation framework of Mac OS X to perform the actual automation. You could use Instruments directly to inspect the elements for troubleshooting purposes. However, its automation scripting language and API are not directly applicable to Appium.

Call Street iOS/Web through grid sample

Monday, November 03, 2014 1:29 PM

CallStreetGrid Demo

Sample Call Street automation with Selenium Grid
Source Location:

To run a sample test as a single thread:

H:\Department\QualityAssurance\QAAutomation\iOS\Samples\CallStreetGrid.zip

Prerequisites

```
gem install appium_lib
gem install watir-webdriver
```

CallStreetGrid usage - single thread (from *CallStreetGrid.zip*)

Safari on simulator:
`ruby callstreet.rb safari`

Safari on real device:
`ruby callstreet.rb safari`

Note: Currently this will redirect to FactSet app, so it is not workable.

WebViewApp on simulator:
`ruby callstreet.rb webviewapp simulator`

WebViewApp on real device:
`ruby callstreet.rb webviewapp device`

Firefox (Web):
`ruby callstreet.rb firefox`

Chrome (Web):
`ruby callstreet.rb chrome`

Internet Explorer (Web):
`ruby callstreet.rb ie`
or
`ruby callstreet.rb "internet explorer"`

Example capabilities configuration config.json (called in Config.rb):

```
{
  "firefox": {
    "capabilities": {
      "platform": "WINDOWS",
      "browserName": "firefox"
    }
  }
}
```

```

    }
  },
  "chrome": {
    "capabilities": {
      "platform": "WINDOWS",
      "browserName": "chrome"
    }
  },
  "internet explorer": {
    "capabilities": {
      "platform": "WINDOWS",
      "browserName": "internet explorer"
    }
  },
  "safari": {
    "capabilities": {
      "platform": "MAC",
      "platformName": "ios",
      "platformVersion": "7.1",
      "deviceName": "iPhone 5s",
      "orientation": "LANDSCAPE",
      "browserName": "safari"
    }
  },
  "serverURL": "http://tellusgridb04.pc.factset.com:4444/wd/hub"
}

```

Example to load the configuration file config.json in Config.rb:

```

require 'json'

module Configuration
  def Configuration.GetConfiguration()
    dir = File.expand_path(File.dirname(__FILE__))
    fileContent = File.read(dir + '/config.json')
    config = JSON.parse(fileContent)
    return config
  end
end

```

Example to get configs in callstreet.rb:

```

configs = Configuration.GetConfiguration()

```

Example to get config for a specific browser and the hub server URL in callstreet.rb:

```

config = configs[browserName]
serverURL = configs['serverURL']

```

Getting Selenium Driver in Utils/UtilsCommon.rb:

```

def getDriver(config, serverURL)
  capabilities = config['capabilities']

```

```

browserName = capabilities['browserName']
if capabilities['platformName'] == "ios" then
  require 'appium_lib'
  driver = Appium::Driver.new(caps: capabilities).start_driver
  Appium.promote_appium_methods Object
  driver.manage.timeouts.implicit_wait = 60
  # switch to "WEBVIEW" context
  driver.appium_device.set_context
driver.appium_device.available_contexts.last
else
  if browserName == "internet explorer" or browserName == "ie"
then
    caps = Selenium::WebDriver::Remote::Capabilities.ie
  elsif browserName == 'firefox' then
    caps = Selenium::WebDriver::Remote::Capabilities.firefox
  elsif browserName == 'chrome' then
    caps = Selenium::WebDriver::Remote::Capabilities.chrome
  end
  driver = Selenium::WebDriver.for :remote,
    :url => serverURL,
    :desired_capabilities

=> caps
end
  driver.manage.timeouts.implicit_wait = 120
  #driver.manage.timeouts.page_load_timeout = 120
  driver.manage.timeouts.script_timeout = 120
  driver.manage.window.maximize
  return driver
end

```

Then create Watir browser object like this (test/DaysViewAndSortColumns.rb):

```

log = Log.new
utilsCommon = UtilsCommon.new(log)
...
config = ...
serverURL = "http://tellusdevb01.pc.factset.com:4444/wd/hub"
driver = utilsCommon.getDriver(config, serverURL)
oPage = Watir::Browser.new driver

```

Check the test results in ./Results folder

Call Street concurrent iOS grid sample

Monday, November 03, 2014 1:29 PM

CallStreetGridConcurrent Demo

This is multi-thread version of CallStreetGrid sample. It uses multi-threading in Ruby to start multiple browser session in the same script, without need for external batch file.

Sample Source Code Location:

H:\Department\QualityAssurance\QAAutomation\iOS\Samples\CallStreetGridConcurrent.zip

Prerequisites

```
gem install appium_lib
gem install watir-webdriver
gem install json
```

CallStreetGridConcurrent usage

```
ruby callstreet.rb BROWSERNAME[:INSTANCES] [...]
Example: ruby callstreet.rb ie firefox:2 chrome:3
BROWSERNAME: firefox, chrome, ie or "internet explorer", or safari (for iOS)
```

Example capabilities

```
firefox:
{
  "platform": "WINDOWS",
  "browserName": "firefox"
}

chrome:
{
  "platform": "WINDOWS",
  "browserName": "chrome"
}

internet explorer:
{
  "platform": "WINDOWS",
  "browserName": "internet explorer"
}

safari:
{
  "platform": "MAC",
  "platformName": "ios",
  "platformVersion": "7.1",
  "deviceName": "iPhone 5s",
  "orientation": "LANDSCAPE",
  "browserName": "safari"
}
```

Getting Selenium Driver

```
def getDriver(config, serverURL)
  capabilities = config['capabilities']
  browserName = capabilities['browserName']
  if capabilities['platformName'] == "ios" then
    require 'appium_lib'
    driver = Appium::Driver.new(caps: capabilities).start_driver
    Appium.promote_appium_methods Object
    driver.manage.timeouts.implicit_wait = 60
    # switch to "WEBVIEW" context
    driver.appium_device.set_context driver.appium_device.available_contexts.last
  else
    if browserName == "internet explorer" or browserName == "ie" then
      caps = Selenium::WebDriver::Remote::Capabilities.ie
    elsif browserName == 'firefox' then
      caps = Selenium::WebDriver::Remote::Capabilities.firefox
    elsif browserName == 'chrome' then
      caps = Selenium::WebDriver::Remote::Capabilities.chrome
    end
    driver = Selenium::WebDriver.for :remote,
      :url => serverURL,
      :desired_capabilities => caps
  end
  return driver
end
```

Then create Watir browser object like this:

```
log = Log.new
utilsCommon = UtilsCommon.new(log)
...
driver = utilsCommon.getDriver(config, serverURL)
oPage = Watir::Browser.new driver
```

FactSet iOS app through Selenium Grid sample

Tuesday, November 18, 2014 9:47 AM

Sample code for automating FactSet iOS app through Selenium Grid Demo

Sample automation script for FactSet iOS app through Selenium Grid. It gets the temporary passcode from your emails, and logs in to FactSet app. Then it taps some native elements. It uses tellusdevb01 as Selenium Hub.

Demo Video: <http://tellusdevb02.pc.factset.com/files/FactSetiOSGridLogin.avi>

It shows Login to FactSet iOS app and tapping native elements through Selenium Grid.

Source Location:

H:\Department\QualityAssurance\QAAutomation\iOS\Samples\FactSetiOSGrid.zip

Setup

```
gem install selenium-webdriver
gem install appium_lib
gem install kconv
gem install viewpoint
```

Note: Appium 1.3 seems to have the issue of losing Mobile Safari in iOS Simulator. You could use Appium 1.2.4 as workaround.

1. Download Appium source:
`git clone https://github.com/appium/appium.git appium`
2. Check out version 1.2.4:
`cd appium`
`git checkout tags/v1.2.4`
3. Build Appium with sample apps and Safari Launcher:
`./reset.sh --ios --launcher --dev`

Starting Appium Grid Node on command line

```
cd appiumzzz
node lib/server/main.js --nodeconfig /path/to/nodeconfig.json
```

nodeconfig.json

```
{
  "capabilities": [
    {
      "platform": "MAC",
      "platformName": "ios",
      "browserName": "safari",
      "version": "6.0",
      "maxInstances": 1
    },
    {
      "platform": "MAC",
      "platformName": "ios",
      "platformVersion": "7.1",
      "browserName": "FactSet",

```

```

        "version": "2.9",
        "maxInstances": 1
    }
  ],
  "configuration": {
    "proxy": "org.openqa.grid.selenium.proxy.DefaultRemoteProxy",
    "host": "jhwangmac.pc.factset.com",
    "port": 4723,
    "url": "http://jhwangmac.pc.factset.com:4723/wd/hub",
    "maxSession": 1,
    "register": true,
    "registerCycle": 5000,
    "hubPort": 4444,
    "hubHost": "tellusdevb01.pc.factset.com",
    "timeout": 120000,
    "browserTimeout": 120000,
    "cleanupCycle": 2000
  }
}

```

You need to change the host and url under configuration to your Selenium node.

Configuration

You need to enter your email address, email username and password in *conf\email_config.yaml*.

You may also need to change the :app path in factset_ios_grid.rb. The FactSet app for iOS simulator is here:

H:\Department\QualityAssurance\QAAutomation\iOS\Apps\FactSet.app.zip

Example email_config.yaml

```

---
email: jhwang@factset.com
username: jhwang
password: somepassword

```

Usage

```

ruby factset_ios_grid.rb

```


Appium iOS WebView example

Monday, November 24, 2014 12:08 PM

Inspecting WebView DOM elements for Development

1. In iOS Simulator, navigate FactSet.app to a windows with WebView object (for example, "Dow Jones NewsPlus in menu -> News > Market News -> North American)
2. In Safari, select menu Developer -> iOS Simulator -> page (for example, "www.djnewsplus.com")
3. Click Inspect icon

Example Code

```
require 'appium_lib'
require 'watir-webdriver'
...

# `driver' is Appium WebDriver from Appium::Driver.new()
# In default NATIVE_APP context
# Menu
driver.find_element(:xpath,
  "//UIAApplication[1]/UIAWindow[1]/UINavigationBar[1]/UIButton[1]").click
wait = Selenium::WebDriver::Wait.new(:timeout=>10)
wait.until{driver.find_element(:xpath,
  "//UIAApplication[1]/UIAWindow[1]/UITableView[1]")}}

# News (use precise tap to workaround 'cannot tap menu item' issue)
Appium::TouchAction.new.press(:x => 30, :y => 219).wait(0.5).release.perform

# Market News
driver.find_element(:name, "MARKET NEWS").click

# North American
driver.find_element(:xpath,
  "//UIAApplication[1]/UIAWindow[1]/UITableView[1]/UITableViewCell[2]/UIAStaticText[1]").
click

# Switch to WEBVIEW
pp driver.appium_device.available_contexts
driver.appium_device.set_context driver.appium_device.available_contexts.last
puts driver.title

# In WEBVIEW context
browser = Watir::Browser.new driver

# Europe market chart
browser.find_element(:css, "#mwTabEU").click
sleep(5) # see better for demo

# Latest on Bonds
puts browser.find_element(:xpath, "//div[text()='Latest on Bonds']/following-
sibling::div").text
browser.find_element(:xpath, "//div[text()='Latest on Bonds']/following-
sibling::div//a").click
sleep(5) # see better for demo

# If no reset, should switch back to native context, else next time might get blank
page
driver.appium_device.set_context "NATIVE_APP"
```

Appium 1.4.13 compatible with iOS 9 and Xcode 7

Tuesday, October 13, 2015 10:11 AM

Appium 1.4.13 compatible with iOS 9 and Xcode 7

Documentation for Infrastructure Chart Tests CLI :

Wednesday, August 05, 2015 11:41 AM

FDSChart Tests:

Prerequisites:

- First we need to download CLI folder located at H:/Users/Norwalk/ahines/FDSChartTestBuilder
 - Download "TestAutomation" folder from H:/Department/QualityAssurance/AtlasQA/FDSChart (Route is set as ".C:\users\qaadmin\Desktop\Chart\TestAutomation\Tests")
 - To run locally we also need to download fdschart.fvm and fdschartunicode.fvm files from both base and test builds.
1. Place all these files at any accessible location.

Command to run tests:

Open Command Prompt at CLI folder and run the following command

Using GUIDs:

You can pre-download the files for future tests by just running the following command first in which we don't need to specify any tests to run:

```
C:\CLI> FDSChartTestRunner.exe -v --base "{1419226f-b146-410a-840c-706353f5b617}" --test "{df0ddbe8-1c43-4b93-b2b7-c06b94d01af1}"  
//optional command you can directly run below command to execute
```

And then subsequent calls will use downloaded files instead of re-downloading:

```
C:\CLI> FDSChartTestRunner.exe -v --base "{1419226f-b146-410a-840c-706353f5b617}" --test "{df0ddbe8-1c43-4b93-b2b7-c06b94d01af1}" "FDSChart Tests\Areas\*.charttest"
```

Using FTP Path:

We just need to give FTP path(the path where we save .fvm files) in place of Build IDs.

FDSChartJS Tests:

Prerequisites:

- Install git bash from <https://git-scm.com/downloads>
- Install NodeJS from <https://nodejs.org/>
- Install bower using "npm install -g bower"
- Install grunt using "npm install -g grunt-cli"
- Download/Clone "fdschartjs-testrunner" repository from <https://gitlab.factset.com/dataviz/fdschartjs-testrunner>
- Download/clone "ChartingInfrastructureAutomation" to fdschartjs-testrunner repository from <https://gitlab.factset.com/dataviz/ChartingInfrastructureAutomation>.
 - We need to download test-runner specific dependencies(npm, bower etc.) to this directory, for that we have to run following commands:

npm install
bower install
 - Use following command to run fdschartjs tests:

sh run.sh -----> output/TEST-test.json will be generated

Or

grunt --baseline= http://fdschartjs.factset.com/versions/4.4/lib/FDSChartJS.js --	output/<number>-test.json will be generated
sample= http://fdschartjs.factset.com/versions/edge/lib/FDSChartJS.js --	
id=<number> ----->	
- Download/Clone "fdschartjs-testresults" repository from <https://gitlab.factset.com/dataviz/fdschartjs-testresults>
 - Copy the content in TEST-test.json file from testrunner directory and paste in "latest.json" (create this file in resultreader directory).
 - We need to download result-reader specific dependencies(npm, bower etc.) to this directory, for that we have to run following commands:

npm install
grunt
It will give us a local host url through which we can see results.

Installation of Apache Server running PHP

Monday, December 22, 2014 4:12 PM

Steps:

H:\Department\QualityAssurance\QAAutomation\web\apache

Step 2: Download and install Visual C++ Redistributable "VSU_4\vc redistrib_x86.exe" for Visual Studio 2012 Update 4 from the following URL:

<http://www.microsoft.com/en-us/download/details.aspx?id=30679>

This is needed to start the apache server as a windows service.

Instructions to run tests in Selenium Grid on Devel

Wednesday, December 17, 2014 2:19 PM

What remote hub server URL should you use in your script to run test against the hub on Devel?

<http://tellusgridb04.pc.factset.com:4444/wd/hub>

What needs to be considered for your testing to be successful?

- (1) The test case should be short and independent
- (2) The browser needs to be maximized for the clicks to work
- (3) The script should be tested on the local hub/node first before being tested against the above hub

What modules you need to include in your ruby script to run tests?

For web:

```
require 'selenium-webdriver'
```

For IOS:

```
require 'appium_lib'
```

Configure the firefox browser (firefox does not auto use system certs):

1. Copy all the certs files from
H:\Department\QualityAssurance\QAAutomation\web\certs
to a temp folder on the node machine.
2. Install all the three root certs:
Options -> Advanced -> Certificates -> View Certificate -> Import
Select .cer file one at a time
Click Open
Check Trust this CA to identify websites
Click OK

Where can you run your tests against the selenium hub on Devel?

You can run your tests on your pc and verify the test results in your log files. Just use the remote server url mentioned above in your script.

How to install a simple hub/node on your local pc for your initial test?

(1) Install a hub:

- (a) Copy all the files from the network folder

H:\Department\QualityAssurance\QAAutomation\web\seleniumgrid

to your local folder like "c:\selenium" on your pc

(b) Download and install java jdk on your pc if you don't have

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

(c) Run the following command in DOS window to start your hub:

```
java -jar C:\selenium\selenium-server-standalone-2.43.1.jar -port 4444 -role hub
```

(2) Install a node for your hub:

(a) Install browsers firefox and chrome if you don't have on your pc.

(b) Register browsers with the hub and start the node in DOS window on your pc on the command line:

```
java -jar C:\selenium\selenium-server-standalone-2.43.1.jar -Dwebdriver.ie.driver=C:\selenium\IEDriverServer.exe -Dwebdriver.chrome.driver=C:\selenium\chromedriver.exe -browser browserName=firefox,maxInstances=5 -browser browserName=ie,platform=WINDOWS -browser browserName=chrome,platform=WINDOWS,maxInstances=5 -role node -hub http://localhost:4444/grid/register -port 5555 -maxSession 5
```

(c) Use the following url to create your remote web driver in your test script:

<http://localhost:4444/wd/hub>

(d) Check the available browsers on the hub using the following url:

<http://localhost:4444/grid/console>

(e) Check the following URL for more information about installation of selenium grid on your pc:

<https://github.com/SeleniumHQ/selenium-docs/blob/master/grid.rst>
<http://elementalselenium.com/tips/52-grid>

Sample script to test Selenium Grid: CallStreetGrid (based on Joseph's initial note)

Sample Call Street automation with Selenium Grid
Source Location:

[H:\Department\QualityAssurance\QAAutomation\web\Samples\CallStreetGridV2.zip](#)

(modified based on Joseph's original versions:

H:\Department\QualityAssurance\QAAutomation\iOS\Samples\CallStreetGrid.zip and

H:\Department\QualityAssurance\QAAutomation\iOS\Samples\CallStreetGridConcurrent.zip)

Prerequisites

```
gem install appium_lib  
gem install watir-webdriver
```

CallStreetGrid usage (from *CallStreetGridV2.zip*)

Firefox (Web):

```
ruby callstreet.rb firefox
```

Chrome (Web):

```
ruby callstreet.rb chrome
```

Internet Explorer (Web):

```
ruby callstreet.rb ie
```

or

```
ruby callstreet.rb "internet explorer"
```

Check the test results in `./Results` folder.

Example capabilities configuration `config.json` (called in `Config.rb`):

```
{
  "firefox": {
    "capabilities": {
      "platform": "WINDOWS",
      "browserName": "firefox"
    }
  },
  "chrome": {
    "capabilities": {
      "platform": "WINDOWS",
      "browserName": "chrome"
    }
  },
  "internet explorer": {
    "capabilities": {
      "platform": "WINDOWS",
      "browserName": "internet explorer"
    }
  },
  "safari": {
    "capabilities": {
      "platform": "MAC",
      "platformName": "ios",
      "platformVersion": "7.1",
      "deviceName": "iPhone 5s",
      "orientation": "LANDSCAPE",
      "browserName": "safari"
    }
  },
  "serverURL": "http://tellusgridb04.pc.factset.com:4444/wd/hub",
  "pacingInterval": 2
}
```

Example to load the configuration file `config.json` in `Config.rb`:

```
require 'json'
```

```

module Configuration
  def Configuration.GetConfiguration()
    dir = File.expand_path(File.dirname(__FILE__))
    fileContent = File.read(dir + '/config.json')
    config = JSON.parse(fileContent)
    return config
  end
end

```

Example to get configs in callstreet.rb:

```
configs = Configuration.GetConfiguration()
```

Example to get config for a specific browser and the hub server URL in callstreet.rb:

```

config = configs[browserName]
serverURL = configs['serverURL']

```

Getting Selenium Driver in Utils/UtilsCommon.rb:

```

def getDriver(config, serverURL)
  capabilities = config['capabilities']
  browserName = capabilities['browserName']
  if capabilities['platformName'] == "ios" then
    require 'appium_lib'
    driver = Appium::Driver.new(caps: capabilities).start_driver
    Appium.promote_appium_methods Object
    #driver.manage.timeouts.implicit_wait = 60
    # switch to "WEBVIEW" context
    driver.appium_device.set_context driver.appium_device.available_contexts.last
  else
    if browserName == "internet explorer" or browserName == "ie" then
      caps = Selenium::WebDriver::Remote::Capabilities.ie
    elsif browserName == 'firefox' then
      caps = Selenium::WebDriver::Remote::Capabilities.firefox
    elsif browserName == 'chrome' then
      caps = Selenium::WebDriver::Remote::Capabilities.chrome
    end
    driver = Selenium::WebDriver.for :remote,
      :url => serverURL,
      :desired_capabilities => caps
  end
  driver.manage.timeouts.implicit_wait = 120
  #driver.manage.timeouts.page_load_timeout = 120
  driver.manage.timeouts.script_timeout = 120
  driver.manage.window.maximize
  return driver
end

```

Then create Watir browser object like this (test/DaysViewAndSortColumns.rb):

```
log = Log.new
```



```
utilsCommon = UtilsCommon.new(log)
...
config = ...
serverURL = "http://tellusgridb04.pc.factset.com:4444/wd/hub"
driver = utilsCommon.getDriver(config, serverURL)
oPage = Watir::Browser.new driver
```

Scirocco Selenium IDE ("Scirocco Recorder") Version 1.1 Release Notes

Scirocco Selenium IDE 1.1 is now available for general release. The major new features are exports to JavaScript (Protractor/Jasmine) and Ruby (Watir and WebDriver).

New Features

1. [RPD:12770738](#): Scirocco Selenium IDE: Adding Export to JavaScript (Protractor)
2. [RPD:12991367](#): In Scirocco Selenium IDE, generate protractor.conf.js template
3. [RPD:12991290](#): For Scirocco Selenium IDE, create node.js module for JavaScript helper library
4. [RPD:13222600](#): In Scirocco Selenium IDE, adding export to Ruby WebDriver
5. [RPD:12770860](#): Scirocco Selenium IDE: Adding Export to Ruby (Watir)

Bug Fixes

1. [RPD:12765277](#): In Scirocco Recorder open source edition, Recording records nothing
2. [RPD:12765550](#): Scirocco Recorder Export Download error: WebKitBlobBuilder is not defined
3. [RPD:12768965](#): In Scirocco Recorder, Export gets "select not defined in lineOfTestCase" error
4. [RPD:12970395](#): In Scirocco Selenium IDE, Export gets error with addSelection and removeSelection
5. [RPD:12781611](#): In Scirocco Selenium IDE, need ability to delete Command entry
6. [RPD:12886192](#): Scirocco Selenium IDE generates incorrect URL
7. [RPD:12973757](#): In Scirocco Selenium IDE, multi-select recording cannot tell if value or text

Known Issues

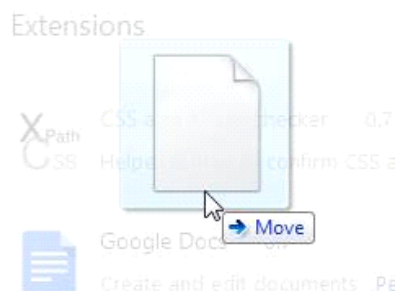
1. [RPD:12770331](#): Scirocco Selenium IDE fails to record popups, including datepicker
2. [RPD:12782324](#): Scirocco Catch-22 issue: Recorded script clicks element with CSS class that does not exist until clicked
3. [RPD:13258388](#): In Scirocco Selenium IDE, command target value is cut off in UI
4. [RPD:13258507](#): In Scirocco Selenium IDE export, long lines are cut off in UI
5. [RPD:13291045](#): Scirocco Selenium IDE integration with test management system
6. [RPD:13321414](#): In Scirocco Selenium IDE, recorded script sometimes cannot find element
7. [RPD:13321727](#): For Scirocco Selenium IDE, use text in xpath selector for TD element

Download

Installation package is available here: <http://tellus/scirocco/SeleniumIDE.zip>

Installation

1. Unzip SeleniumIDE.zip; it will extract SeleniumIDE.crx
2. Open Google Chrome browser
3. Select **Settings** icon on toolbar
4. Click **Extensions**
5. Drag and drop the .crx file to the Extensions page



History

FactSet's Scirocco Selenium IDE is a fork of the open source Scirocco Recorder Version 1.0.0 at <https://github.com/sonixlabs/selenium-ide-for-chrome>.

ReleaseNotesVersion1..1

Tuesday, June 24, 2014 9:27 AM

Scirocco Selenium IDE ("Scirocco Recorder") Version 1.1 Release Notes

Scirocco Selenium IDE 1.1 is now available for general release. The major new features are exports to JavaScript (Protractor/Jasmine) and Ruby (Watir and WebDriver).

New Features

1. [RPD:12770738](#): Scirocco Selenium IDE: Adding Export to JavaScript (Protractor)
2. [RPD:12991367](#): In Scirocco Selenium IDE, generate protractor.conf.js template
3. [RPD:12991290](#): For Scirocco Selenium IDE, create node.js module for JavaScript helper library
4. [RPD:13222600](#): In Scirocco Selenium IDE, adding export to Ruby WebDriver
5. [RPD:12770860](#): Scirocco Selenium IDE: Adding Export to Ruby (Watir)

Bug Fixes

1. [RPD:12765277](#): In Scirocco Recorder open source edition, Recording records nothing
2. [RPD:12765550](#): Scirocco Recorder Export Download error: WebKitBlobBuilder is not defined
3. [RPD:12768965](#): In Scirocco Recorder, Export gets "select not defined in lineOfTestCase" error
4. [RPD:12970395](#): In Scirocco Selenium IDE, Export gets error with addSelection and removeSelection
5. [RPD:12781611](#): In Scirocco Selenium IDE, need ability to delete Command entry
6. [RPD:12886192](#): Scirocco Selenium IDE generates incorrect URL
7. [RPD:12973757](#): In Scirocco Selenium IDE, multi-select recording cannot tell if value or text

Known Issues

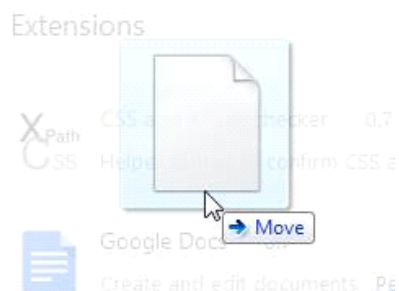
1. [RPD:12770331](#): Scirocco Selenium IDE fails to record popups, including datepicker
2. [RPD:12782324](#): Scirocco Catch-22 issue: Recorded script clicks element with CSS class that does not exist until clicked
3. [RPD:13258388](#): In Scirocco Selenium IDE, command target value is cut off in UI
4. [RPD:13258507](#): In Scirocco Selenium IDE export, long lines are cut off in UI
5. [RPD:13291045](#): Scirocco Selenium IDE integration with test management system
6. [RPD:13321414](#): In Scirocco Selenium IDE, recorded script sometimes cannot find element
7. [RPD:13321727](#): For Scirocco Selenium IDE, use text in xpath selector for TD element

Download

Installation package is available here: <http://tellus/scirocco/SeleniumIDE.zip>

Installation

1. Unzip SeleniumIDE.zip; it will extract SeleniumIDE.crx
2. Open Google Chrome browser
3. Select **Settings** icon on toolbar
4. Click **Extensions**
5. Drag and drop the .crx file to the Extensions page



History

FactSet's Scirocco Selenium IDE is a fork of the open source Scirocco Recorder Version 1.0.0 at <https://github.com/sonixlabs/selenium-ide-for-chrome>.

Selenium recorders (IDEs)

Thursday, October 23, 2014 3:58 AM

Scircoco

[ReleaseNotesVersion1..1](#)

<https://code.google.com/p/scirocco/wiki/QuickStartScirocco>

Howdy,

You have been un-randomly selected to use Selenium-Builder and Scirocco to do recordings and code generation (we don't care about Playbacks). The idea is to do a recording, generate code and see if it makes sense for a scripter and analyst to mold that process into something usable.

If you generate Protractor code (JavaScript) please do NOT show it to the HYD Analytics scripters as they are far too busy.

Selenium Builder is a Firefox add-on (yes it needs a port to Chrome). IS internal apps does support Firefox, so they could use it as-is.

Install Instructions provided by Joseph.

Selenium-Builder Installation

Installation

1. Open URL in Firefox: <http://sebuilder.github.io/se-builder/>
2. Click **Install**
3. When prompted for permission, click **Allow**
4. Click **Install Now**
5. It will prompt to **Restart Firefox**

Usage


1. On menu, select **Tools > Web Developer > Launch Selenium Builder** or press **Ctrl+Alt+B**
2. Enter the URL to be tested under "**Start recording at**"
3. Click **Selenium 2**
4. It will start recording

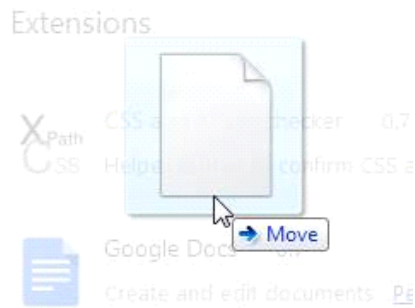
Scirocco Installation

Download

Installation package is available here: <http://tellus/scirocco/SeleniumIDE.zip>

Installation

1. Unzip SeleniumIDE.zip; it will extract SeleniumIDE.crx
2. Open Google Chrome browser
3. Select **Settings** icon on toolbar

4. Click **Extensions**
5. Drag and drop the .crx file to the Extensions page



History

FactSet's Scirocco Selenium IDE is a fork of the open source Scirocco Recorder Version 1.0.0 at <https://github.com/sonixlabs/selenium-ide-for-chrome>.

Some Scirocco and Selenium Builder comments also provided by Joseph

- Scirocco originally does not generate Protractor code, but Joseph added that feature.
- Selenium Builder (SB) generates Protractor out of the box
- Protractor and SB do not generate code for AngularJS (Angular is not relevant, as it is not a test automation library, but a Web app development library; Protractor and SB potentially generate Protractor code instead.)
- Scirocco originally supports only Java, but Joseph added Ruby and JavaScript.
- Selenium Builder supports export to:
 - JSON
 - Java
 - Java/JUnit
 - Java/TestNG
 - Ruby (uses 'selenium-webdriver' library instead of the 'watir' library that Bhupender uses)
 - Python
 - Python/unittest
 - PHPUnit – Selenium 2
 - PHP – WebDriver
 - Node.JS – WD
 - Node.JS – Mocha
 - Node.JS – Protractor
 - C#
 - C#/NUnit
 - English

-Ken

Understanding Selenium Builder locators

Friday, January 30, 2015 7:55 AM

This is a description of Selenium Builder element locator algorithm in plain English.

As of Selenium Builder 2.3 (Jan. 2015)

In short, the `id` attribute has the best result, as its uniqueness is enforced by the browser. The `name` attribute and link text also have good results if they are actually unique. The following element attributes would help: `id`, `name`, `class`, `type`, `alt`, `title`, and `value`.

It will fall back to XPATH absolute path, or by CSS classes *alone*. Using absolute path is fragile, as it could break if element moved. And using CSS classes alone often fails to locate the element, as CSS is often dynamically changed.

Notes:

- Selenium Builder gives you multiple choices of locators, and the *preferred method* is the default one
- The example generated code format below is in Ruby

Algorithm:

1. If the element has **id** attribute, it will use `id` as the preferred method
Generate code format:
`find_element(:id, "userID")`
2. Else if it has **name** attribute, then `name` will be the preferred method
Generated code format:
`find_element(:name, "userName")`
3. Else if the element is an **A** (address link) tag, the link text will be the preferred method
Generated code format:
`find_element(:link_text, "European Market")`
4. Generate **CSS** locator expression using any of these attributes: `id`, `name`, `class`, `type`, `alt`, `title`, `value`; multiple elements of same type will use CSS `:nth-of-type()` index
Generated code format:
`find_element(:css, "input[type=\"text\"]")`
 - a. If the CSS locator does not locate the element, then add its parent's CSS locator, and so on, up to the top html tag.
Generated code format:
`find_element(:css, "div > input[type=\"text\"]")`
 - b. And adds this CSS selector as a user choice
 - c. If no preferred method yet, use CSS as preferred method
5. Generate **html-based XPATH**, based on HTML tag name and attributes such as `id` attribute, `class` attribute, `for` attribute for label tag only, `body` tag, and `html` tag, as absolute path.
 - a. Add this XPATH as a user choice
 - b. If no preferred method yet, use XPATH as preferred method**Generated code format:**
`find_element(:xpath, "//html/body/div[@class='x']/div[@class='y']")`
6. Generate **full XPATH** with tag names and indices, as absolute path
 - a. Add this XPATH as a user choice

- b. If no preferred method yet, use XPATH as preferred method

Generated code format:

```
find_element(:xpath, "//html/body/div[1]/input[2]")
```

- 7. Generate CSS selector with **class** attribute and tag name (not path)
 - a. If it does not already have CSS selector, then use this CSS selector
 - b. If it does not already have preferred method, then use CSS as preferred method

Generated code format:

```
find_element(:css, "div[class=\"x.y\"]")
```

Selenium

Friday, June 09, 2017 1:13 PM

Selenium Training Twiki

<http://infonet.factset.com/view/QualityAssurance/SeleniumTraining>

Selenium Recordings <H:\Department\QualityAssurance\QAAutomation\Selenium>