



# SELENIUM WEBDRIVER

Amit Kumar



“[Selenium automates browsers.](#) That's it! What you do with that power is entirely up to you.”

# Agenda

- ☐ Introduction to Selenium WebDriver
- ☐ Java Refresher
- ☐ Introduction to Maven
- ☐ Selenium IDE
- ☐ Selenium WebDriver
- ☐ Selenium Grid
- ☐ JUnit vs. TestNg
- ☐ Frameworks

# History

According to [SeleniumHQ](#), the history of Selenium started in 2004 at Thoughtworks with Jason Huggins building the core mode as JavaScript Test Runner. He built the core mode because he did not want to manually step through the same test after every change. Therefore, Jason developed a JavaScript library that permitted him to run test repeatedly against multiple browsers such as Firefox, Google, and Internet Explorer. That library eventually became Selenium Core, which underlies all the functionality of **Selenium Remote Control (RC)** and **Selenium IDE**. Selenium RC was ground-breaking because no other product allowed you to control a browser from a language of your choice.

While Selenium was a tremendous tool, it wasn't without its drawbacks. Because of its Javascript based automation engine and the security limitations browsers apply to Javascript, different things became impossible to do. To make things worse, webapps became more and more powerful over time, using all sorts of special features new browsers provide and making these restrictions more and more painful.

In 2006 an engineer at Google named Simon Stewart started working on a project he called **WebDriver**.

Google had long been a heavy user of Selenium, but testers had to work around the limitations of the product. Simon wanted a testing tool that spoke directly to the browser using the 'native' method for the browser and operating system, thus avoiding the restrictions of a sandboxed Javascript environment. The WebDriver project began with the aim to solve the Selenium' pain-points.

Then came 2008 which led to the merger of Selenium and WebDriver into Selenium 2 a.k.a Selenium WebDriver. Selenium had massive community and commercial support, but WebDriver was clearly the tool of the future. The joining of the two tools provided a common set of features for all users and brought some of the brightest minds in test automation under one roof. Why did the projects merge? In Simon's words, partly because WebDriver addresses some shortcomings in selenium (by being able to bypass the JS sandbox, for example. And we've got a gorgeous API), partly because selenium addresses some shortcomings in WebDriver (such as supporting a broader range of browsers) and partly because the main selenium contributors and I felt that it was the best way to offer users the best possible framework. Selenium 2 got released in 2011.

# What is Selenium?

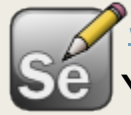
Selenium is a suite of tools to automate web browsers across many platforms.

Selenium...

- runs in many browsers and operating systems
- can be controlled by many programming languages and testing frameworks.



# Components of Selenium



[Selenium IDE](#) is a Firefox add-on that makes it easy to record and playback tests in Firefox 2+. You can even use it to generate code to run the tests with Selenium Remote Control.



[Selenium Remote Control](#) is a client/server system that allows you to control web browsers locally or on other computers, using almost any programming language and testing framework.



[Selenium WebDriver](#) can drive a browser natively either locally or on remote machines.



[Selenium Grid](#) takes Selenium Remote Control to another level by running tests on many servers at the same time, cutting down on the time it takes to test multiple browsers or operating systems.

# Before we begin...

## A Java Refresher:

### Object Oriented Programming

- ✓ Object (Data & Logic)
- ✓ Class (Variables & Methods)
- ✓ Encapsulation (access modifiers)
- ✓ Inheritance (single, multi-level, hierarchical, hybrid)
- ✓ Polymorphism (overloading & overriding)
- ✓ Abstraction (abstract class & interface)

### Learning the language:

- ✓ Packages
- ✓ Data Types
- ✓ Non-access modifiers
- ✓ Operators
- ✓ Control Structures
- ✓ I/O
- ✓ Collections (List – ordered collection, Set – unordered collection, Map – {key,value})
- ✓ File Handling
- ✓ Exception Handling

# Before we begin...

## Introduction to Maven:

Maven is a project management and comprehension tool that provides developers a complete build lifecycle framework. Maven provides developers ways to manage the following –

- *Builds*
- *Documentation*
- *Reporting*
- ***Dependencies***
- *SCMs*
- *Releases*
- *Distribution*
- *Mailing list*

Using Maven for dependency management:

- Creating & Configuring a Maven project
- Setting up the Maven project for Selenium

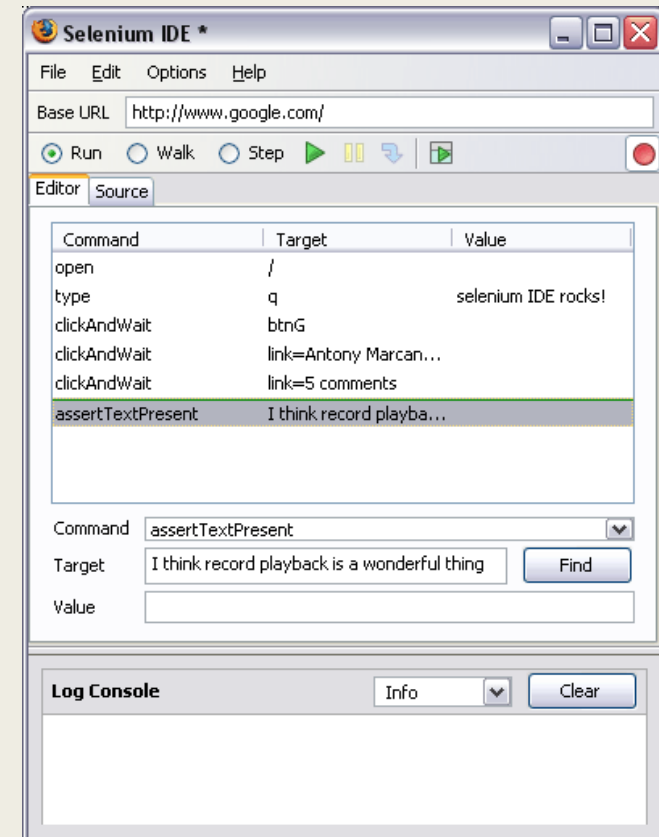
# Selenium IDE

The *Firefox extension* for Selenium

**Selenium IDE** is an integrated development environment for Selenium scripts. It is implemented as a Firefox extension, and allows you to record, edit, and debug tests. Selenium IDE includes the entire Selenium Core, allowing you to easily and quickly record and play back tests in the actual environment that they will run in.

Let's get rolling!

- ✓ Downloading & installing Selenium IDE
- ✓ Selenium IDE user interface
- ✓ Record & Playback
- ✓ "Selenese"
- ✓ Managing Test Cases & Test Suites
- ✓ Debug & set breakpoints
- ✓ Export Selenium code from Selenium IDE
- ✓ user-extensions.js
- ✓ Rollup
- ✓ UI Map
- ✓ Plugins/extensions





# Selenium WebDriver

The starting point for testing all web applications begins with a browser. Selenium WebDriver provides support for all major browsers such as Firefox, Google Chrome, Internet Explorer, Safari and Opera. Calls are directed to the browser by using the browser's native support for automation. Actions are performed on **WebElements** through **WebDriver**.

WebElements are your buttons, text boxes, checkboxes, drop down menus, radio buttons, hyperlinks etc.

The building block of Selenium WebDriver is locating a WebElement and performing an action on the WebElement.

Before we begin, let's create a Java project and configure it for Selenium WebDriver.

# Writing code in WebDriver

//WebDriver Packages and Classes

```
import org.openqa.selenium.WebDriver;  
import org.openqa.selenium.firefox.FirefoxDriver;
```

//WebDriver Object and Methods

//instantiating a Firefox driver object

```
WebDriver driver = new FirefoxDriver();
```

//Launch the webapp in the browser

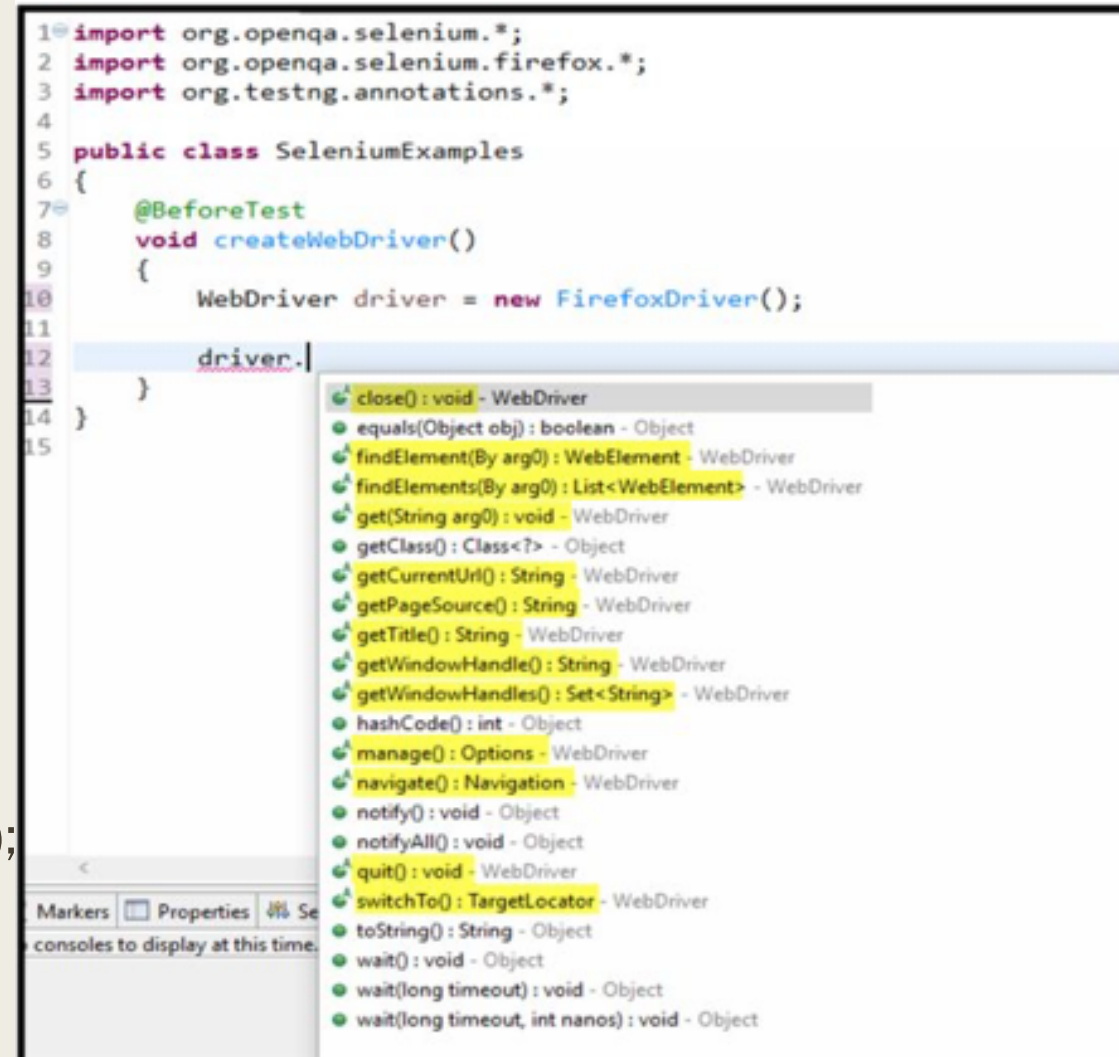
```
driver.get("<webapp url>");
```

//Find and Perform Actions on WebElements

```
WebElement element = driver.findElement
```

```
(By.locatorType("<element id>"));
```

```
element.click();
```



The screenshot shows an IDE with a Java file named `SeleniumExamples`. The code includes imports for Selenium and TestNG, and a `@BeforeTest` method `createWebDriver()` that instantiates a `FirefoxDriver`. The cursor is positioned at the end of the `driver` object, and a dropdown menu displays a list of methods available on the `WebDriver` interface. The methods listed include `close()`, `equals()`, `findElement()`, `findElements()`, `get()`, `getClass()`, `getCurrentUrl()`, `getPageSource()`, `getTitle()`, `getWindowHandle()`, `getWindowHandles()`, `hashCode()`, `manage()`, `navigate()`, `notify()`, `notifyAll()`, `quit()`, `switchTo()`, `toString()`, `wait()`, `wait(long timeout)`, and `wait(long timeout, int nanos)`.

```
1 import org.openqa.selenium.*;  
2 import org.openqa.selenium.firefox.*;  
3 import org.testng.annotations.*;  
4  
5 public class SeleniumExamples  
6 {  
7     @BeforeTest  
8     void createWebDriver()  
9     {  
10         WebDriver driver = new FirefoxDriver();  
11  
12         driver.  
13     }  
14 }  
15
```

- `close() : void - WebDriver`
- `equals(Object obj) : boolean - Object`
- `findElement(By arg0) : WebElement - WebDriver`
- `findElements(By arg0) : List<WebElement> - WebDriver`
- `get(String arg0) : void - WebDriver`
- `getClass() : Class<?> - Object`
- `getCurrentUrl() : String - WebDriver`
- `getPageSource() : String - WebDriver`
- `getTitle() : String - WebDriver`
- `getWindowHandle() : String - WebDriver`
- `getWindowHandles() : Set<String> - WebDriver`
- `hashCode() : int - Object`
- `manage() : Options - WebDriver`
- `navigate() : Navigation - WebDriver`
- `notify() : void - Object`
- `notifyAll() : void - Object`
- `quit() : void - WebDriver`
- `switchTo() : TargetLocator - WebDriver`
- `toString() : String - Object`
- `wait() : void - Object`
- `wait(long timeout) : void - Object`
- `wait(long timeout, int nanos) : void - Object`

# WebDriver basic commands

driver.

- get("url") – load the web application

- getTitle() – returns the title of the current page as shown in the browser's title bar

- getCurrentUrl() – returns the url of the current page as shown in the browser's address bar

- getPageSource() – fetches the page source

- close() – closes the active browser window associated with the WebDriver

- quit() – closes all the browser associated with the WebDriver, i.e. shuts down

WebDriver

- findElement(By locator) – locates & returns the first WebElement matching the specified locator

- findElements(By locator) – locates & returns all WebElements matching the specified locator

# Browser Navigations

`driver.navigate().`

`to("url")` – loads a new web page

`forward()` – go forward in the browser

`back()` – go back in the browser

`refresh()` – refresh the browser

`driver.manage().window()`

`.maximize()` – Maximize browser window

`.setPosition()` – Move the browser window

`.getPosition()` – Get the browser position

`.setSize()` – Resize the browser window

`.getSize()` – Returns the current browser window size

`.fullscreen()` – Switch to fullscreen mode

# Element Locators

By.

className	<code>&lt;button class="button0" id="login" type="submit"&gt;</code>
cssSelector	<code>input#username</code>
id	<code>&lt;input id="username" name="name" type="text"&gt;</code>
linkText	<code>&lt;a href="/login"&gt;User Login&lt;/a&gt;</code>
name	<code>&lt;input id="username" name="name" type="text"&gt;</code>
partialLinkText	<code>&lt;a href="/login"&gt;User Login&lt;/a&gt;</code>
tagName	<code>&lt;input id="username" name="name" type="text"&gt;</code>
xpath	

→ Absolute xpath (/)      `html/body/input`

→ Relative xpath (//)      `//input[@id='username']`

# XPath Syntax

Expression	Description
<i>nodename</i>	Selects all nodes with the name " <i>nodename</i> "
/	Selects from the root node
//	Selects nodes in the document from the current node that match the selection no matter where they are
.	Selects the current node
..	Selects the parent of the current node
@	Selects attributes

Wildcard	Description
*	Matches any element node
@*	Matches any attribute node
node()	Matches any node of any kind

By using the | operator in an XPath expression you can select several paths.

# XPath Predicates

Predicates are used to find a specific node or a node that contains a specific value. Predicates are always embedded in square brackets.

Path Expression	Result
/bookstore/book[1]	Selects the first book element that is the child of the bookstore element.
/bookstore/book[last()]	Selects the last book element that is the child of the bookstore element
/bookstore/book[last()-1]	Selects the last but one book element that is the child of the bookstore element
/bookstore/book[position()<3]	Selects the first two book elements that are children of the bookstore element
//title[@lang]	Selects all the title elements that have an attribute named lang
//title[@lang='en']	Selects all the title elements that have a "lang" attribute with a value of "en"
/bookstore/book[price>35.00]	Selects all the book elements of the bookstore element that have a price element with a value greater than 35.00
/bookstore/book[price>35.00]/title	Selects all the title elements of the book elements of the bookstore element that have a price element with a value greater than 35.00

<bookstore>

<book>

<title lang="en">Harry Potter</title>

<price>29.99</price>

</book>

<book>

<title lang="en">Learning XML</title>

<price>39.95</price>

</book>

</bookstore>

# XPath Axes

An axis defines a node-set relative to the current node. (*axisname::node[predicate]*)

AxisName	Result
ancestor	Selects all ancestors (parent, grandparent, etc.) of the current node
ancestor-or-self	Selects all ancestors (parent, grandparent, etc.) of the current node and the current node itself
attribute	Selects all attributes of the current node
child	Selects all children of the current node
descendant	Selects all descendants (children, grandchildren, etc.) of the current node
descendant-or-self	Selects all descendants (children, grandchildren, etc.) of the current node and the current node itself
following	Selects everything in the document after the closing tag of the current node
following-sibling	Selects all siblings after the current node
namespace	Selects all namespace nodes of the current node
parent	Selects the parent of the current node
preceding	Selects all nodes that appear before the current node in the document, except ancestors, attribute nodes and namespace nodes
preceding-sibling	Selects all siblings before the current node
self	Selects the current node



# XPath Axes

Example	Result
child::node1	Selects all node1 nodes that are children of the current node
attribute::att1	Selects the att1 attribute of the current node
child::*	Selects all element children of the current node
attribute::*	Selects all attributes of the current node
child::text()	Selects all text node children of the current node
child::node()	Selects all children of the current node
descendant::node1	Selects all node1 descendants of the current node
ancestor::node1	Selects all node1 ancestors of the current node
ancestor-or-self::node1	Selects all node1 ancestors of the current node - and the current as well if it is a node1 node
child::* / child::node1	Selects all node1 grandchildren of the current node

# XPath Operators

Below is a list of the operators that can be used in XPath expressions:

Operator	Description	Example
	Computes two node-sets	//book   //cd
+	Addition	6 + 4
-	Subtraction	6 - 4
*	Multiplication	6 * 4
div	Division	8 div 4
=	Equal	price=9.80
!=	Not equal	price!=9.80
<	Less than	price<9.80
<=	Less than or equal to	price<=9.80
>	Greater than	price>9.80
>=	Greater than or equal to	price>=9.80
or	or	price=9.80 or price=9.70
and	and	price>9.00 and price<9.90
mod	Modulus (division remainder)	5 mod 2

# More XPath functions

- `contains()` - `//div[contains(@id,'button')]`
- `starts-with()` - `//div[starts-with(@id,'but')]`
- `ends-with()` - `//div[ends-with(@id,'ton')]`
- `text()` - `//a[text()='welcome']`

# CSS Locators

- ID - #            e.g. input#username
- Class - .        e.g. input.textfield
- Attribute       e.g. input[type='submit']
- Match a prefix       e.g. input[type^='sub']
- Match a suffix       e.g. input[type\$='mit']
- Match a sub string   e.g. input[type\*='ubm']
- Inner text       e.g. a:contains('Sign In'), div:contains('^abc\$')

# More jQuery selectors for CSS

- `div:first` – first div element
- `div:last` – last div element
- `ul li:first` – first li element under first ul element
- `ul li:first-child` – first li element of every ul element
- `ul li:last-child` – last li element of every ul element
- `tr:even` – all even tr elements
- `tr:odd` – all odd tr elements
- `nth-child(n)`
- `div:parent()`
- `div:parents()`
- `div:parentsUntil(input)`
- `div:children()`

# Using Patterns in Locators

abc]	A single character of: a, b, or c
[^abc]	Any single character except: a, b, or c
[a-z]	Any single character in the range a-z
[a-zA-Z]	Any single character in the range a-z or A-Z
^	Start of line
\$	End of line
\A	Start of string
\Z	End of string

.	Any single character
\s	Any whitespace character
\S	Any non-whitespace character
\d	Any digit
\D	Any non-digit
\w	Any word character (letter, number, underscore)
\W	Any non-word character
\b	Any word boundary

(...)	Capture everything enclosed
(a b)	a or b
a?	Zero or one of a
a*	Zero or more of a
a+	One or more of a
a{3}	Exactly 3 of a
a{3,}	3 or more of a
a{3,6}	Between 3 and 6 of a

xpath =  
//div[**matches**(@id,'che.\*boxes')]

# Handy tools

## Firefox

- In-built Web inspector
- Firebug
- Firepath
- WebDriver Element Locator

## Internet Explorer

- Developer tools

## Chrome

- In-built Web Inspector
- Xpath Helper

# WebElement

...is everything you see on your web page and interact with.

WebElement interface gives you the power to interact with your web application the way a human would.

- .clear() – applicable to text fields; clears the content
- .sendKeys("keys to type") – simulates typing into an element
- .click() – simulates mouse click
- .isDisplayed()
- .isEnabled()
- .isSelected() – for checkboxes, radio buttons, select options
- .submit() – applicable to a form
- .getText()
- .getTagName()
- .getCssValue("CSS property") – fetches CSS property value
- .getAttribute("attribute name") – value of the given attribute
- .getSize()
- .getLocation()



# Handling Select element

A special 'Select' class is needed to handle dropdowns unlike the other UI element types.

```
WebElement element = driver.findElement(By.id("drop-down"));
Select oSelect = new Select(element);
```

Note: The above only works for elements with <select> tag

oSelect.

- deSelectAll() – for multi-select
- deSelectByIndex() – for multi-select
- deSelectByValue() – for multi-select
- deSelectByVisibleText() – for multi-select
- getAllSelectedOptions() – for multi-select
- getFirstSelectedOption() – for multi-select
- getOptions()
- isMultiple()
- selectByIndex()
- selectByValue()
- selectByVisibleText()

```
<select id="days" class="calendar-days"
name="weekdays">
    <option value="Sunday">Sun</option>
    <option value="Monday">Mon</option>
    <option value="Tuesday">Tue</option>
    <option value="Wednesday">Wed</option>
    <option value="Thursday">Thu</option>
    <option value="Friday">Fri</option>
    <option value="Saturday">Sat</option>
</select>
```

# Handling Tables

```
<table>
  <tbody>
    <tr>
      <th>Column 1</th>
      <th>Column 2</th>
      <th>Column 3</th>
    </tr>
    <tr>
      <td>Row1 Col1 Data</td>
      <td>Row1 Col2 Data</td>
      <td>Row1 Col3 Data</td>
    </tr>
    <tr>
      <td>Row2 Col1 Data</td>
      <td>Row2 Col2 Data</td>
      <td>Row2 Col3 Data</td>
    </tr>
  </tbody>
</table>
```

# Actions – handling advanced interactions

- Double Click
- Key Down, Key Up
- Mouse Down, Mouse Up
- Drag and Drop
- Mouse Move
- etc. etc.

```
Actions action = new Actions(driver);  
action.dragAndDrop(source,target).build().perform();
```

```
Actions builder = new Actions(driver);  
Action dragAndDrop = builder.clickAndHold(From)  
    .moveToElement(To)  
    .release(To)  
    .build();  
dragAndDrop.perform();
```

# Synchronization

- Thread.sleep() – Bad!!
- Implicit Wait – Global wait

```
driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
```

```
driver.manage().timeouts().setScriptTimeout(100,SECONDS);
```

```
driver.manage().timeouts().pageLoadTimeout(100, SECONDS);
```

- Explicit Wait – Condition based

```
WebDriverWait wait = new WebDriverWait(driver, 10);
```

```
wait.until(ExpectedConditions.elementToBeClickable(By.id(>someid>)));
```

```
(new WebDriverWait(driver, 10)).until(new ExpectedCondition<Boolean>(){  
    public Boolean apply(WebDriver d){  
        return d.getTitle().contains("Welcome");  
    }  
});
```

# Synchronization

```
// Waiting 30 seconds for an element to be present on the page, checking
```

```
// for its presence once every 5 seconds.
```

```
Wait wait = new FluentWait(driver)
```

```
    .withTimeout(30, SECONDS)
```

```
    .pollingEvery(5, SECONDS)
```

```
    .ignoring(NoSuchElementException.class);
```

```
WebElement foo = wait.until(new Function<WebDriver,WebElement>() {
```

```
    public WebElement apply(WebDriver driver) {
```

```
        return driver.findElement(By.id("foo"));
```

```
    }
```

```
});
```

# Pop-up windows

`String handle= driver.getWindowHandle();//Return a string of alphanumeric window handle`

`Set<String> handle= driver.getWindowHandles();//Return a set of window handle`

`driver.switchTo().window("windowName");`

Or

```
for (String handle : driver.getWindowHandles()) {  
    driver.switchTo().window(handle);  
}
```

Or

```
driver.findElement(By.id("id of the link which opens new window")).click();  
//wait till two windows are not opened  
waitForNumberOfWindowsToEqual(2);//this method is for wait  
Set handles = driver.getWindowHandles();  
firstWinHandle = driver.getWindowHandle();  
handles.remove(firstWinHandle);  
String winHandle=handles.iterator().next();  
if (winHandle!=firstWinHandle){  
    //To retrieve the handle of second window, extracting the handle which does not match to first window handle  
    secondWinHandle=winHandle; //Storing handle of second window handle  
    //Switch control to new window  
    driver.switchTo().window(secondWinHandle);
```

# iFrames

```
driver.switchTo().frame(index);  
driver.switchTo().frame(name/id);  
driver.switchTo().frame(frameElement);  
driver.switchTo().defaultContent();
```

Finding count of frames:

```
    //By executing a java script  
    JavascriptExecutor exe = (JavascriptExecutor) driver;  
    Integer numberOfFrames = Integer.parseInt(exe.executeScript("return  
window.length").toString());  
System.out.println("Number of iframes on the page are " + numberOfFrames);  
  
    //By finding all the web elements using iframe tag  
    List<WebElement> iframeElements = driver.findElements(By.tagName("iframe"));  
    System.out.println("The total number of iframes are " + iframeElements.size());
```

# Alerts

Types of Alerts:

- Simple alert
- Confirmation alert – Ok/Cancel
- Prompt alert – Yes/No?, enter some text

```
Alert alert = driver.switchTo().alert();
```

```
alert.accept();
```

```
alert.dismiss();
```

```
alert.getText();
```

```
alert.sendKeys();
```



# Taking Screenshots

- Using TakesScreenshot interface
- Using Augmenter
- Using EventFiringWebDriver

# Firefox Profiles

- Creating new Firefox Profile
- Using an existing Firefox Profile
- Usage

# Desired Capabilities

Basically, the DesiredCapabilities help to set properties for the WebDriver. You can use this to customize & configure a browser session by making use of the various key-value pairs.

Learn [more](#)

# Browser Cookies

Managing cookies through WebDriver:

- Fetching a cookie(s)
- Deleting cookie(s)
- Adding a cookie

# WebDriver Events

**EventFiringWebDriver** to wrap the webdriver around to throw events.

**WebDriverEventListener** to catch the webdriver events.

# Cross browser testing

Making use of Third Party Browser Drivers to run webdriver code on different browsers.



# Selenium Grid

With ***Selenium Grid*** you can create a network of connected test machines (also called nodes). This network of test machines is controlled by a Hub, using which you can run your tests on different connected nodes. Each node is basically a computer (even a virtual machine) with a combination of Operating system and Browsers. This enables us to create a network of test machines with varying combinations of Operating system and browsers. Using Selenium Grid you can run tests on a variety of Operating System and Browser combinations. You can connect to it with Selenium Remote by specifying the browser, browser version, and operating system you want. You specify these values through *Selenium Remote's Capabilities*.

Step 1: Start a Hub (*java -jar selenium-server-standalone-2.53.1.jar -role hub*) – *port:4444*

Step 2: Register Nodes to the Hub (*java -jar selenium-server-standalone-2.53.1.jar -role node -hub http://<hub IP>:4444/grid/register*) – *port:5555*

Nodes are where your tests will run, and the hub is responsible for making sure your tests end up on the right one (e.g., the machine with the operating system and browser you specified in your test). Hub will try to find a machine in the Grid which matches the criterion and will run the test on that Machine. If there is no match, then hub returns an error. There should be only one hub in a Grid.

Usage:

- *When we want to run our tests against multiple browsers, the multiple versions of browsers and the browsers running on different operating system.*
- *It is also used to reduce the time taken by the test suite to complete a test pass by running tests in parallel.*

# Selenium Grid

## Step 3: Use RemoteWebDriver

You can use RemoteWebDriver the same way you would use WebDriver locally. The primary difference is that RemoteWebDriver needs to be configured so that it can run your tests on a separate machine.

There are two parts to RemoteWebDriver: a server(hub) and a client(node):

- The RemoteWebDriver server is a component that listens on a port for various requests from a RemoteWebDriver. Once it receives the requests, it forwards them to any of the following: Firefox Driver, IE Driver, or Chrome Driver, whichever is asked.
- The language-binding client libraries that serve as a RemoteWebDriver. The client, as it used to when executing tests locally, translates your test script requests to JSON payload and sends them across to the RemoteWebDriver server using the JSON wire protocol.

```
String grid = "http://<hub IP>:4444/wd/hub";
```

```
DesiredCapabilities cap = DesiredCapabilities.firefox();
```

```
driver = new RemoteWebDriver(new URL(grid), cap);
```

```
driver.navigate().to(URL);
```

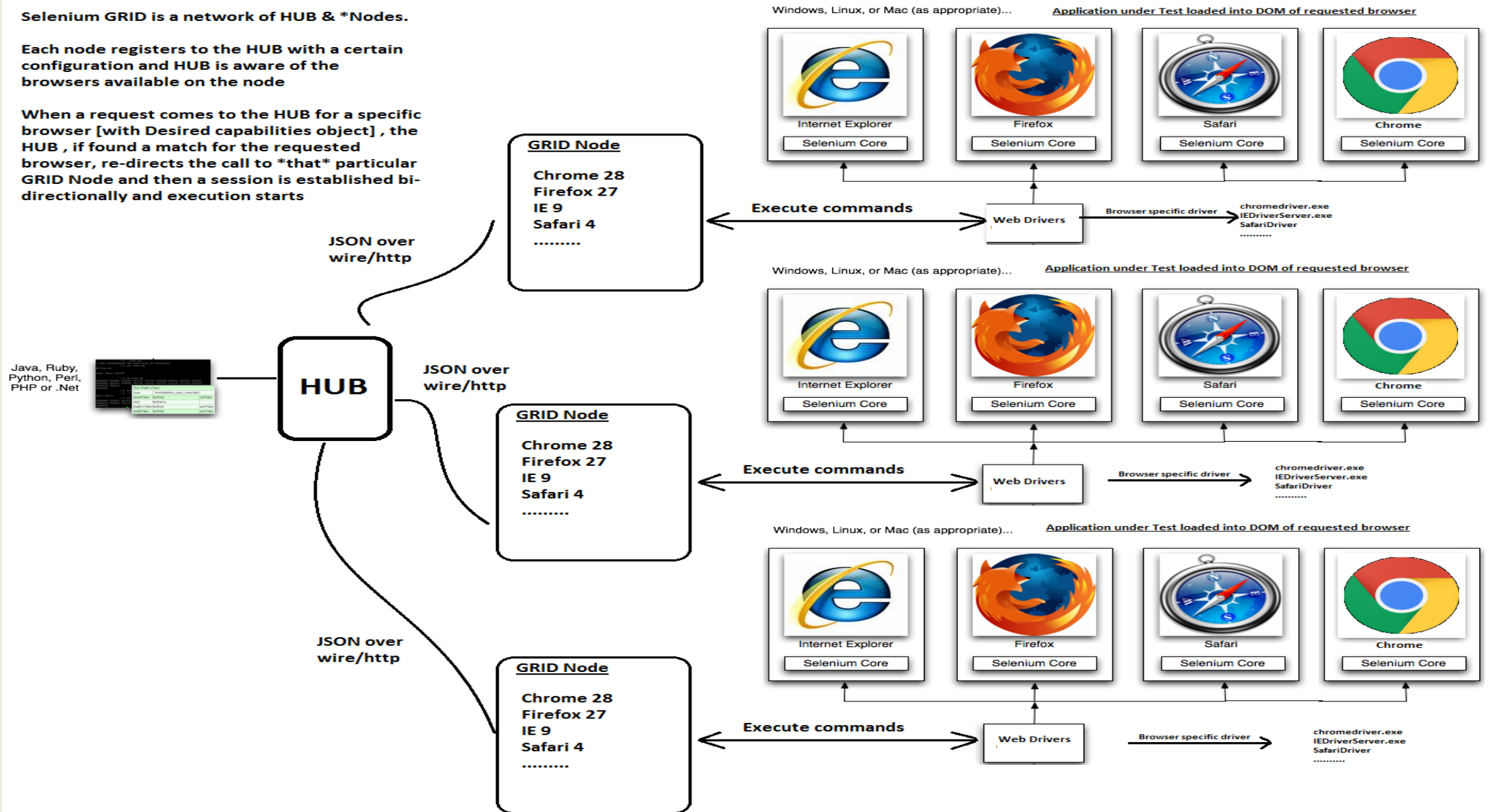


# Selenium Grid

Selenium GRID is a network of HUB & \*Nodes.

Each node registers to the HUB with a certain configuration and HUB is aware of the browsers available on the node

When a request comes to the HUB for a specific browser [with Desired capabilities object] , the HUB , if found a match for the requested browser, re-directs the call to \*that\* particular GRID Node and then a session is established bi-directionally and execution starts



# Selenium Grid

## Useful parameters

- -port
- -browser browserName=,maxInstances=,version=,platform=
- -maxSession (max. no. of browsers that can run in parallel on a node)
- -timeout (node timeout, default=300s)
- -Dwebdriver.ie.driver=
- -Dwebdriver.chrome.driver=

...[more](#)

<http://localhost:5555/selenium-server/driver?cmd=shutDownSeleniumServer>

<http://localhost:4444/lifecycle-manager?action=shutdown>

# Headless testing

## ■ [HtmlUnitDriver](#)

- *Fastest implementation of WebDriver*
  - *Uses a Java based "HtmlUnit" browser*
  - *htmlUnit browser uses Rhino JS engine (not used by any real browser ☹)*
  - *No screenshot support*
  - *Platform independent*
- ## ■ PhantomJsDriver
- *Good JS support*
  - *Comes with the standard DOM API*
  - *Screenshot supported*
  - *Platform independent*

**Caution:** *The headless browsers have advanced enough to emulate any browser to a great extent, however, still not completely. Hence, it is not recommended to run all your tests in a headless browser. JavaScript is one area where you would want to be really careful before using a Headless browser. JavaScript are implemented differently by different browsers, including, headless browsers.*

## ■ [Xvfb](#) (X virtual frame buffer)

- *For Unix environment with no display support; hence no browsers*
- *Is an in-memory display server; enables you to run GUI applications without a display*
- *Pre-requisites: \$ apt-get install xvfb firefox*
- *Start Xvfb: \$ Xvfb :1 -screen 0 1024x768x24 &*
- *Set the display: \$ export DISPLAY=:1 or Guide firefox to use this virtual display*
- *Test firefox: & firefox*
- *Run the WebDriver test normally on a real browser*

# JUnit

- Is an unit testing framework; can be leveraged to do automation runs
- Annotations:
  - *@BeforeClass*
  - *@AfterClass*
  - *@Before*
  - *@After*
  - *@Test*
  - *@RunWith* – *to change the runner and not use the default JUnit runner*
  - *@Suite.SuiteClasses*
- A very simple XML report

# TestNg

- A testing framework inspired from JUnit and NUnit but is more powerful
- Annotations:
  - *@BeforeSuite*
  - *@AfterSuite*
  - *@BeforeTest*
  - *@AfterTest*
  - *@BeforeGroups*
  - *@AfterGroups*
  - *@BeforeClass*
  - *@AfterClass*
  - *@BeforeMethod*
  - *@AfterMethod*
  - *@Test*
- Tests can be grouped & prioritized
- Ability to produce HTML reports & Logs
- Parallel testing
- Data-driven testing

# JUnit vs. TestNg

Description	TestNG	JUnit 4
Test annotation	@Test	@Test
Executes before the first test method is invoked in the current class	@BeforeClass	@BeforeClass
Executes after all the test methods in the current class	@AfterClass	@AfterClass
Executes before each test method	@BeforeMethod	@Before
Executes after each test method	@AfterMethod	@After
annotation to ignore a test	@Test(enabled=false)	@ignore
annotation for exception	@Test(expectedExceptions = ArithmeticException.class)	@Test(expected = ArithmeticException.class)
timeout	@Test(timeout = 1000)	@Test(timeout = 1000)
Executes before all tests in the suite	@BeforeSuite	n/a
Executes after all tests in the suite	@AfterSuite	n/a
Executes before a test runs	@BeforeTest	n/a
Executes after a test runs	@AfterTest	n/a
Executes before the first test method is invoked that belongs to any of these groups is invoked	@BeforeGroups	n/a
run after the last test method that belongs to any of the groups here	@AfterGroups	n/a