Contents

About Selenium Webdriver

Selenium is a portable software-testing framework for web applications. Selenium provides a playback (formerly also recording) tool for authoring tests without the need to learn a test scripting language (Selenium IDE). It also provides a test domain-specific language (Selenese) to write tests in a number of popular programming languages, including C#, Groovy, Java, Perl, PHP, Python, Ruby and Scala. The tests can then run against most modern web browsers. Selenium deploys on Windows, Linux, and macOS platforms. It is open-source software, released under the Apache 2.0 license: web developers can download and use it without charge.

Installation for different browsers:

Firefox

Mozilla runs on Gecko browser engine.Gecko browser engine was developed by Mozilla foundation as a part of Mozilla browser.

Gecko Driver is the link between your tests in Selenium and the Firefox browser. GeckoDriver is a proxy for using W3C WebDriver-compatible clients to interact with Gecko-based browsers i.e. Mozilla Firefox in this case. As Selenium 3 will not have any native implementation of FF, we have to direct all the driver commands through Gecko Driver. Gecko Driver is an executable file that you need to have in one of the system path before starting your tests. Firefox browser implements the WebDriver protocol using an executable called GeckoDriver.exe. This executable starts a server on your system. All your tests communicate to this server to run your tests. It translates calls into the Marionette automation protocol by acting as a proxy between the local and remote ends.

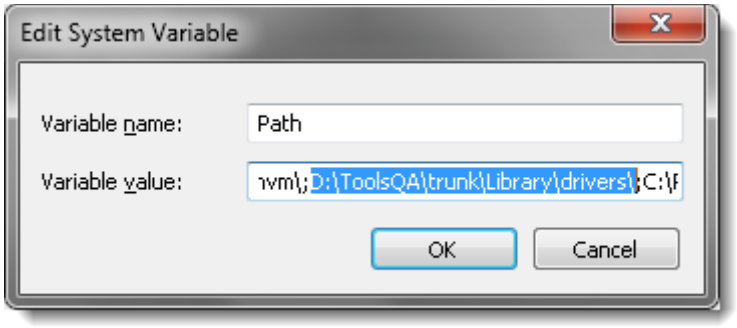
Download Gecko Driver

https://github.com/mozilla/geckodriver/releases

Set System Properties for Gecko Driver

```
System.setProperty("webdriver.gecko.driver","Path to geckodriver.exe");
```

Set property in Environment Variables



Set Desired Capabilities of the Marionette/Gecko Driver

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

capabilities.setCapability("marionette", true);

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

http://toolsqa.com/selenium-webdriver/how-to-use-geckodriver/

Chrome

Chrome browser implements the WebDriver protocol using an executable called ChromeDriver.exe. This executable starts a server on your system.

Chrome Driver Server
https://sites.google.com/a/chromium.org/chromedriver/

http://chromedriver.storage.googleapis.com/index.html

Launching Chrome Browser using Selenium WebDriver

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

http://toolsqa.com/selenium-webdriver/running-tests-in-chrome-browser/

IE

Download latest version server

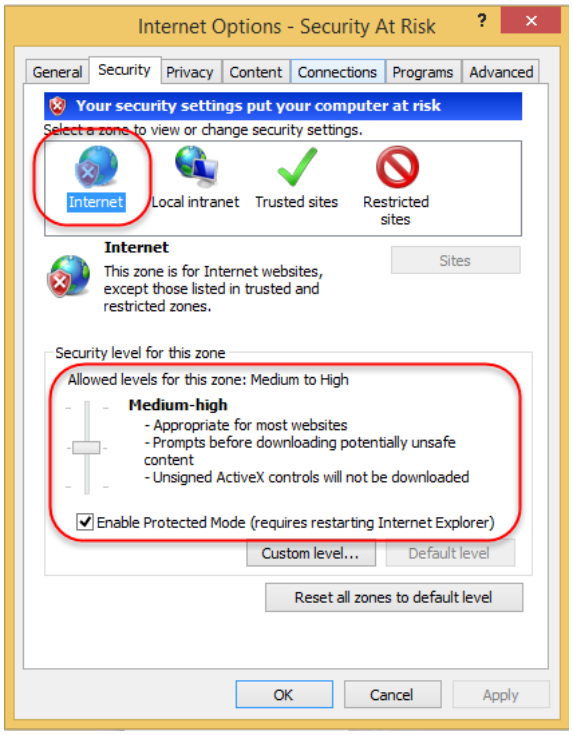
https://docs.seleniumhq.org/download/

```
driver = new InternetExplorerDriver();
```

If see issue some thing like 'Unexpected error launching Internet Explorer' below, You have to set 'Enable protected mode' option in all levels with same value.

1. Open Internet Explorer browser--> Select Internet Options from Tools menu

2. Select Security Tab --> Select Enable Protected Mode option -- > Check the default Zone level for 'Internet'. If you look at the screen shot below, security level for this zone is selected as 'Allowed level for this zone : Medium to High.' and 'Enable Protected Mode' option is Checked.



https://www.seleniumeasy.com/selenium-tutorials/how-to-run-webdriver-in-ie-browser

How to build interaction process

1. Configure browser capabilities
2. Run browser
3. Navigate to the page
4. Find element
5. Interact with element
6. Good practice to find element right before interaction
7. Use waits instead sleep
8. Do verifications needed
9. Close the browser

How to interact with browser

Get Command

get(String arg0) : void – This method Load a new web page in the current browser window. Accepts String as a parameter and returns nothing.

Command – driver.get(appUrl);

Where appUrl is the website address to load. It is best to use a fully qualified URL.

```
driver.get( "http://www.google.com" );

//Or can be written as

String URL = "http://www.DemoQA.com";
driver.get( URL );
```

Get Title Command

getTitle() : String – This method fetches the Title of the current page. Accepts nothing as a parameter and returns a String value.

Command – driver.getTitle();

As the return type is String value, the output must be stored in String object/variable.

```
driver.getTitle();

//Or can be used as

String Title = driver.getTitle();
```

Get Current URL Command

getCurrentUrl() : String – This method fetches the string representing the Current URL which is opened in the browser. Accepts nothing as a parameter and returns a String value.

Command – driver.getCurrentUrl();

As the return type is String value, the output must be stored in String object/variable.

```
driver.getCurrentUrl();

//Or can be written as

String CurrentUrl = driver.getCurrentUrl();
```

Get Page Source Command

getPageSource() : String – This method returns the Source Code of the page. Accepts nothing as a parameter and returns a String value.

Command – driver.getPageSource();

As the return type is String value, the output must be stored in String object/variable.

```
driver.getPageSource();

//Or can be written as

String PageSource = driver.getPageSource();
```

Close Command

close() : void – This method Close only the current window the WebDriver is currently controlling. Accepts nothing as a parameter and returns nothing.

Command – driver.close();

Quit the browser if it's the last window currently open.

```
driver.close();
```

Quit Command

quit() : void – This method Closes all windows opened by the WebDriver. Accepts nothing as a parameter and returns nothing.

Command – driver.quit();

Close every associated window.

```
driver.quit();
```

Navigate To Command

to(String arg0) : void – This method Loads a new web page in the current browser window. It accepts a String parameter and returns nothing.

Command – driver.navigate().to(appUrl);

It does exactly the same thing as the driver.get(appUrl) method. Where appUrl is the website address to load. It is best to use a fully qualified URL.

```
driver.navigate().to("http://www.DemoQA.com");
```

Forward Command

forward() : void – This method does the same operation as clicking on the Forward Button of any browser. It neither accepts nor returns anything.

Command – driver.navigate().forward();

Takes you forward by one page on the browser's history.

```
driver.navigate().forward();
```

Back Command

back() : void – This method does the same operation as clicking on the Back Button of any browser. It neither accepts nor returns anything.

Command – driver.navigate().back();

Takes youback by one page on the browser's history.

```
driver.navigate().back();
```

Refresh Command

refresh() : void – This method Refresh the current page. It neither accepts nor returns anything.

Command – driver.navigate().refresh();

Perform the same function as pressing F5 in the browser.

```
driver.navigate().refresh();
```

How to locate elements

Locators types

There are 8 explicit locators: id, name, identifier, dom, xpath, link, css and ui that Selenium's commands support.

ID Locator

Ids are the most preferred way to locate elements on a page, as each id is supposed to be unique which makes ids a very faster and reliable way to locate elements.

With this strategy, the first element with the id attribute value matching the location will be returned. If no element has a matching id attribute, a NoSuchElementException will be raised.

Example: If an element is given like this:


```
<form name="loginForm">Login
Username: <input id="username" type="text" name="login" />
Password: <input id="password" type="password" name="password" />
<input type="submit" name="signin" value="SignIn" /></form>
```

You can easily choose the element with the help of ID locator from the above example:

id = "username"

id = "password"

Use the same in your Selenium test script as well:

```
WebElement elementUsername = driver.findElement( By.id("username"));

WebElement elementPassword = driver.findElement(By.id("password"));
```

Even though this is a great locator, obviously it is not realistic for all objects on a page to have ids. In some cases developers make it having non-unique ids on a page or auto-generate the ids, in both cases it should be avoided.

Name Locator

This is also an efficient way to locate an element with name attribute, after Ids give it your second preference but likewise Ids, name attributes don't have to be unique in a page.

With this strategy, the first element with the name attribute value matching the location will be returned. If no element has a matching name attribute, a NoSuchElementException will be raised.

Example: Let's take the above example:

You can easily choose the element with the help of Name locator from the above example:

name = "login"

name = "password"

Use the same in your Selenium test script as well:

```
WebElement elementUsername = driver.findElement(By.name("login"));

WebElement elementPassword = driver.findElement(By.name("password"));
```

Identifier Locator

This selects the element with the specified @id attribute. If no match is found, then it tries to select the first element whose @name attribute is id.

Example: Valid locator for above example is:

identifier = "password"

First it will try to find the locator with id = "password" if it exists, otherwise it would target name = "password".

Link Locator

With this you can find elements of "a" tags(Link) with the link names. Use this when you know link text used within an anchor tag.

Example: If an element is given like this:

```
<a href="link.html">Name of the Link</a>
```

To click this hyperlink using the anchor tag's text, you can use the link locator:

link="Name of the Link"

Use the same in your Selenium test script as well:

```
WebElement element=driver.findElement(By.linkText("Name of the Link"));
```

DOM Locator

The DOM strategy works by locating elements that matches the JavaScript expression referring to an element in the DOM of the page. DOM stands for Document Object Model. DOM is convention for representing objects in HTML documents.

Example: To select the username from the above example you can use the following ways:

document.forms[0].elements[0]

document.forms['loginForm'].elements['username']

document.forms['loginForm'].username

document.getElementById('username')

XPath Locator

While DOM is the recognized standard for navigation through an HTML element tree, XPath is the standard navigation tool for XML; and an HTML document is also an XML document (xHTML). XPath is used everywhere where there is XML.

Example: To select the username from the above example you can use the following ways:

xpath=//*[@id='username']

xpath=//input[@id='username']

xpath=//form[@name='loginForm']/input[1]

xpath=//*[@name='loginForm']/input[1]

CSS Selector

Selectors are patterns that match against elements in a tree, and as such form one of several technologies that can be used to select nodes in an XML document. Selectors have been optimized for use with HTML and XML, and are designed to be usable in performance-critical code.

WebElement form4 = driver.findElement(By.cssSelector("form.login"));

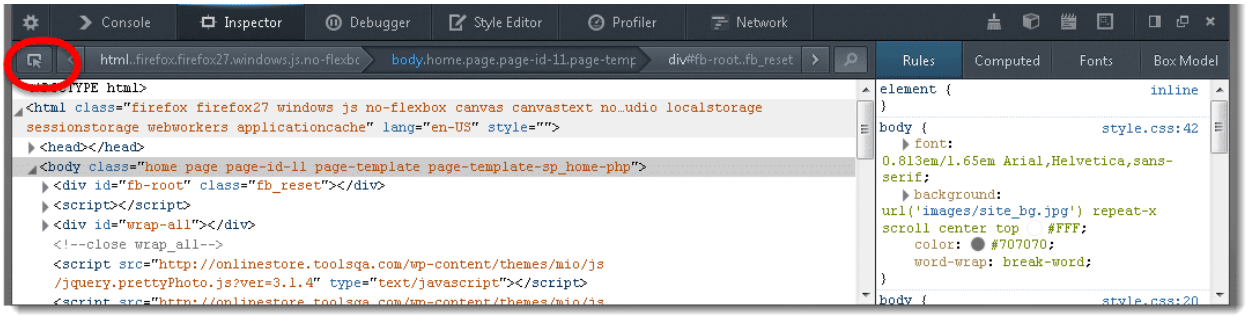
WebElement form5 = driver.findElement(By.cssSelector("#login-form"));

http://toolsqa.com/selenium-webdriver/locators/

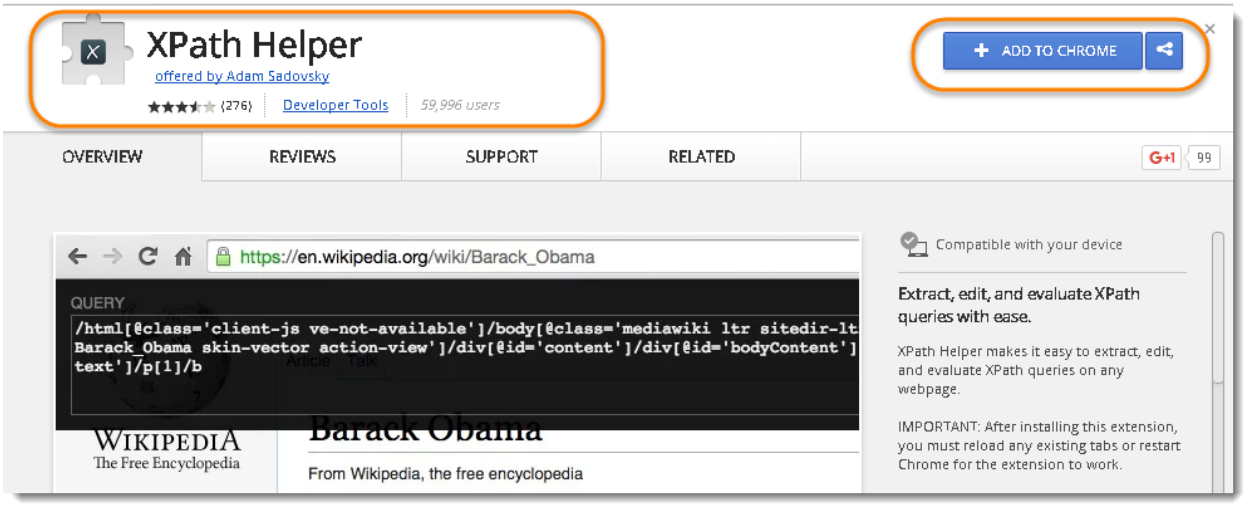
http://toolsqa.com/selenium-webdriver/finding-elements-using-browser-inspector/

Useful Tools

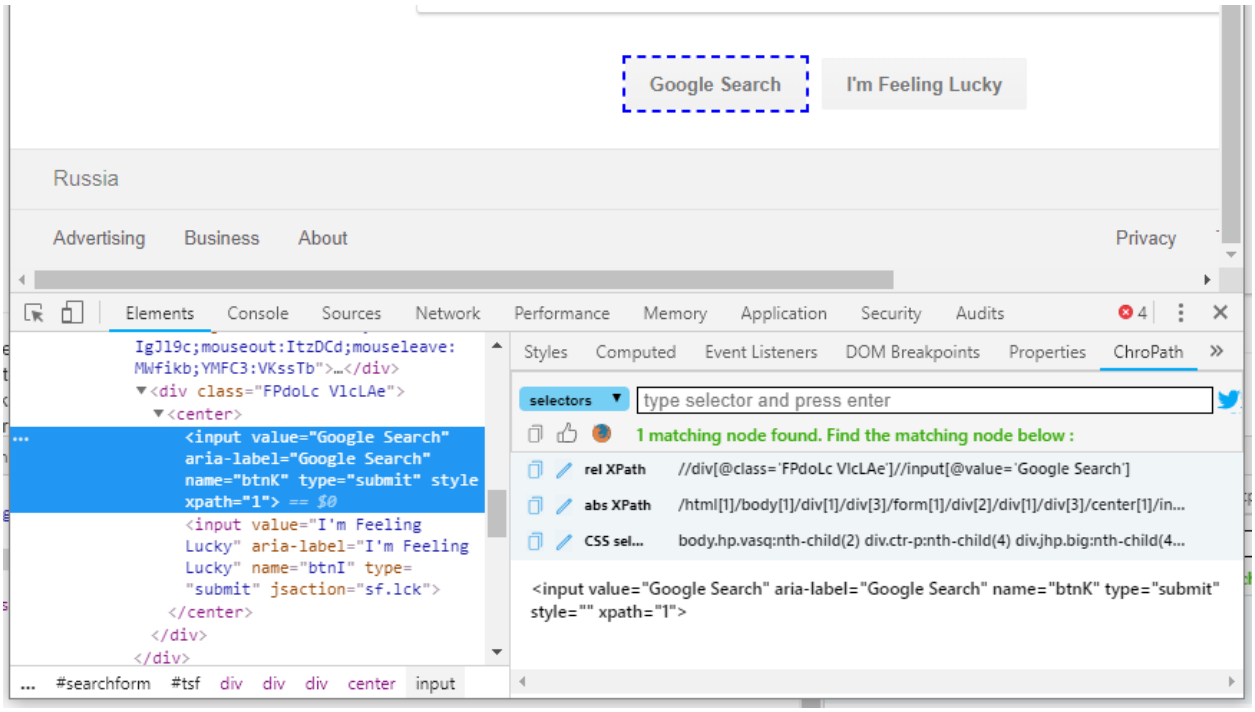
Finding Elements using Browser Inspector



XPath Helper Plugin for Chrome Browser



ChroPath Helper Plugin for Chrome Browser



How to interact with elements

Click to the visible element

driver.findElement(By.cssSelector("input[type='button']")).click();

Type a text in the form

driver.findElement(By.id("email")).sendKeys("name@abc.com");

Submit the form

driver.findElement(By.id("submit")).submit();

How to Select Option from DropDown using Selenium Webdriver

Command	Description
selectByVisibleText() deselectByVisibleText()	selects/deselects an option by its displayed text
selectByValue() deselectByValue()	selects/deselects an option by the value of its "value" attribute
selectByIndex() deselectByIndex()	selects/deselects an option by its index
isMultiple()	returns TRUE if the drop-down element allows multiple selection at a time; FALSE if otherwise
deselectAll()	deselects all previously selected options

http://toolsqa.com/selenium-webdriver/dropdown-multiple-select-operations/

Get Attribute values using Webdriver

There are cases where you want to get the attributes values and then perform any action.

In the below, if you see, button tag which has multiple attributes 'name', 'id', 'class' and 'aria-label' and has values for each attribute. To get the attribute value using selenium webdriver, we can use 'element.getAttribute(attributeName)'.

If we try to get the attribute value that doesn't exists for the tag, it will return null value.

```
<button name="btnK" id="gbqfba" aria-label="Google Search" class="gbqfba"><span id="gbqfsa">Google Search</span></button>
```

```
WebElement googleSearchBtn = driver.findElement(bySearchButton);
System.out.println("Name of the button is:- " +googleSearchBtn.getAttribute("name"));

System.out.println("Id of the button is:- "+ googleSearchBtn.getAttribute("id"));

System.out.println("Class of the button is:- "+
googleSearchBtn.getAttribute("class"));

//Will return null value as the 'status' attribute doesn't exists
System.out.println("Invalid Attribute status of the button is:- "+
googleSearchBtn.getAttribute("status"));

System.out.println("Label of the button is:- "+ googleSearchBtn.getAttribute("aria-label"));
```

After executing the above code the output should look something like below:

Name of the button is:- btnK

Id of the button is:- gbqfba

Class of the button is: gbqfba

Invalid Attribute status of the button is: null

Label of the button is: Google Search

How to wait element loading

Explicit and Implicit Waits

Waiting is having the automated task execution elapse a certain amount of time before continuing with the next step. You should choose to use Explicit Waits or Implicit Waits.

WARNING: Do not mix implicit and explicit waits. Doing so can cause unpredictable wait times. For example setting an implicit wait of 10 seconds and an explicit wait of 15 seconds, could cause a timeout to occur after 20 seconds.

Explicit Waits

An explicit wait is code you define to wait for a certain condition to occur before proceeding further in the code. The worst case of this is Thread.sleep(), which sets the condition to an exact time period to wait. There are some convenience methods provided that help you write code that will wait only as long as required. WebDriverWait in combination with ExpectedCondition is one way this can be accomplished.

```
WebDriver driver = new FirefoxDriver();
driver.get("http://somedomain/url_that_delays_loading");
WebElement myDynamicElement = (new WebDriverWait(driver, 10))
        .until(
ExpectedConditions.presenceOfElementLocated(By.id("myDynamicElement")));
```

Expected Conditions

There are some common conditions that are frequently encountered when automating web browsers. Listed below are a few examples for the usage of such conditions. The Java and Python bindings include convenience methods so you don't have to code an ExpectedCondition class yourself or create your own utility package for them.

Element is Clickable - it is Displayed and Enabled.

```
WebDriverWait wait = new WebDriverWait(driver, 10);
WebElement element =
wait.until(ExpectedConditions.elementToBeClickable(By.id("someid")));
```

The ExpectedConditions package contains a set of predefined conditions to use with WebDriverWait.

Implicit Waits

An implicit wait is to tell WebDriver to poll the DOM for a certain amount of time when trying to find an element or elements if they are not immediately available. The default setting is 0. Once set, the implicit wait is set for the life of the WebDriver object instance.

```
WebDriver driver = new FirefoxDriver();
driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
driver.get("http://somedomain/url_that_delays_loading");
WebElement myDynamicElement = driver.findElement(By.id("myDynamicElement"));
```

Execute JavaScript using Selenium

It provides mechanism to execute Javascript through Selenium driver. It provides “executescript” & “executeAsyncScript” methods, to run JavaScript in the context of the currently selected frame or window.

There is no need to write separate script to execute JavaScript within the browser using Selenium WebDriver script. Just use predefined interface named ‘Java Script Executor’.

Syntax:

JavascriptExecutor js = (JavascriptExecutor) driver;

js.executeScript(Script,Arguments);

Script – The JavaScript to execute

Arguments –The arguments to the script(Optional). May be empty.

Returns –One of Boolean, Long, String, List, WebElement, or null.

Let’s see some scenarios we could handle using this Interface:

```
JavascriptExecutor js = (JavascriptExecutor) driver;
js.executeScript("document.getElementById('some id').value='someValue';");
js.executeScript("document.getElementById('Email').value='SoftwareTestingMaterial.com'
;");

//To click a Button in Selenium WebDriver using JavaScript

js.executeScript("document.getElementById('enter your element id').click();");
//or
js.executeScript("document.getElementById('enter your element id').click();");
//or
js.executeScript("arguments[0].click();", loginButton);

//To handle Checkbox
js.executeScript("document.getElementById('enter element id').checked=false;");

//To generate Alert Pop window in selenium
js.executeScript("alert('Welcome To SoftwareTestingMaterial');");

//To refresh browser window using Javascript
js.executeScript("history.go(0)");

//To get innertext of the entire webpage in Selenium
String sText =  js.executeScript("return
document.documentElement.innerText;").toString();
System.out.println(sText);

//To get the Title of our webpage
String sText =  js.executeScript("return document.title;").toString();
System.out.println(sText);

//To get the domain
String sText =  js.executeScript("return document.domain;").toString();
System.out.println(sText);


//To get the URL of a webpage
String sText =  js.executeScript("return document.URL;").toString();
System.out.println(sText);

//To perform Scroll on application using  Selenium
//Vertical scroll - down by 500  pixels
js.executeScript("window.scrollBy(0,500)");
```

```
// for scrolling till the bottom of the page we can use the code like
//js.executeScript("window.scrollBy(0,document.body.scrollHeight)");
//Vertical scroll - down by 500  pixels
js.executeScript("window.scrollBy(0,500)");
// for scrolling till the bottom of the page we can use the code like
//js.executeScript("window.scrollBy(0,document.body.scrollHeight)");

//To click on a SubMenu which is only visible on mouse hover on Menu
js.executeScript("$('ul.menus.menu-secondary.sf-js-enabled.sub-menu li').hover()");

//To navigate to different page using Javascript
js.executeScript("window.location = 'https://www.softwaretestingmaterial.com");

//To find hidden element in selenium using JavaScriptExecutor
js.executeScript("arguments[0].click();", element);
```

Selenium Grid

The Selenium Grid is a testing tool which allows us to run our tests on different machines against different browsers. It is a part of the Selenium Suite which specialise in running multiple tests across different browsers, operating system and machines. 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.

There are two main elements to Selenium Grid — a hub, and nodes.

What is a Hub?

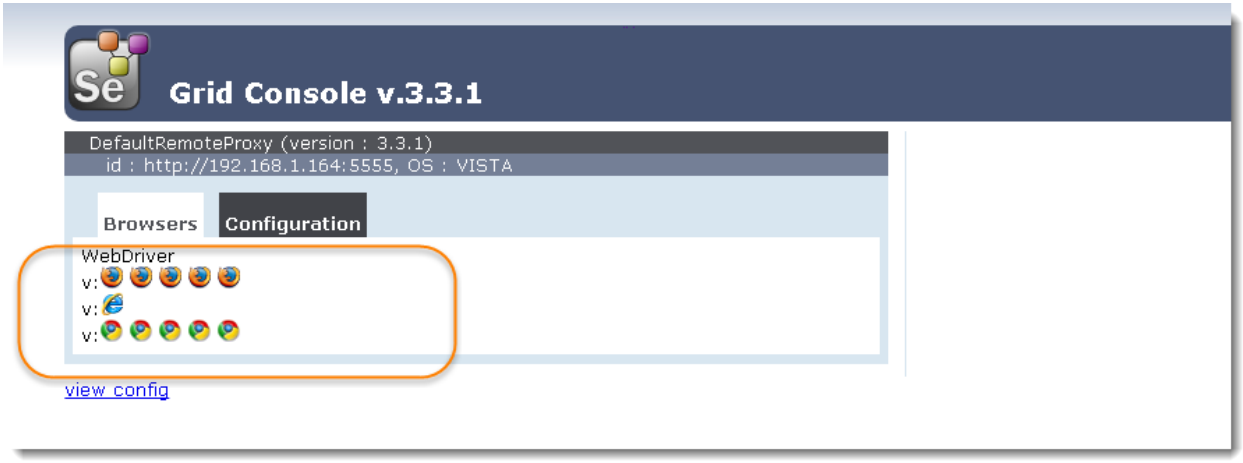
In Selenium Grid, the hub is a computer which is the central point where we can load our tests into. Hub also acts as a server because of which it acts as a central point to control the network of Test machines. The Selenium Grid has only one hub and it is the master of the network. When a test with given DesiredCapabilities is given to Hub, the Hub searches for the node witch matches the given configuration. For example, you can say that you want to run the test on Windows 10 and on Chrome browser with verision XXX. 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.

What is a Node?

In Selenium Grid, a node is referred to a Test Machine which opts to connect with the Hub. This test machine will be used by Hub to run tests on. A Grid network can have multiple nodes. A node is supposed to have different platforms i.e. different operating system and browsers. The node does not need the same platform for running as that of hub.

How it works?

First you need to create a hub. Then you can connect (or "register") nodes to that hub. 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).



http://toolsqa.com/selenium-webdriver/selenium-grid-how-to-easily-setup-a-hub-and-node/