Difference between Selenium IDE RC Grid and Webdriver

Do you know the exact Difference between Selenium IDE RC  Grid and Webdriver and which version is used in industry?

Today we will discuss the differences in details in tabular format, one of the blog has shared the view in very structured manner. I will share the link at the end of this post.

Before comparison just have a quick history about all.

Selenium has started in 2004 and it is designed by ThoughtWorks (they have one of the branch in Koramangala Bangalore). Later on it is OpenQA has taken and maintaining till now.

Selenium Started in with Add-on in firefox which is known as Selenium IDE and trust me it is one of the most popular Addon in market.

It gives you flexibility to convert your code into multiple languages and use it accordingly.

Now you will get plenty of Add-on  that you can integrate with Selenium IDE itself and you can use it.

Some plugin that I used  with IDE is screenshot on failure, highlighter, export code into excel and  so on.

***But Selenium IDE has some of the limitation like***

1- It is only available in Firefox so we can record your script in firefox only.

2- Selenium IDE does not have good reporting feature which generally we use for reporting to managers and lead or Team.

3- Selenium IDE does not support parallel execution which is one of the most important features of Automation.

4- Selenium IDE does not support remote execution as well.

**Selenium RC – Selenium 1**

Selenium team was totally aware of all this limitations so in 2007 they came with another version which covers almost all the limitation of Selenium IDE.

Selenium team released new version called Selenium RC. Here RC stands for remote control.

Selenium RC is not another tool or plugin it is just library which contains several packages classes interfaces and methods.

Selenium RC had so many features like

1- It supports almost all browser which is available in the market.

2- We can easily integrate Selenium RC with TestNG which makes Selenium more powerfull.

3-  Selenium RC having very good reporting feature with the help of TestNG.

4- Now we can do execution in parallel which the help of TestNG. We can pass multiple parameters, data driven and all the features which TestNG have.

5- Selenium RC came with Remote execution which we can achieve through Selenium grid.

Selenium Grid 1.0

It is very big topic to explain but I will give a quick intro about Grid.

Selenium Grid has node and hub concept which works in Client-Server architecture concept.

We can create one central hub which can connect multiple Node .

Node can be different machine like Windows, Linux, Mac, Android, IOS and so on.

I will create another post with video which will cover all this in details.

Limitations of Selenium RC

It seems like Selenium RC was very powerfull and does not have any limitations but no Selenium RC also had some limitations which were

1- It generally interact with Server it means everytime you run script it will send request to server then server will communicate with browsers.

Performance was an issue here.

2- Since 2010 Mobile industry is leading in market so Selenium RC was  not capable to perform.

***Note- Selenium RC officially deprecated now in market.***

[](https://i0.wp.com/learn-automation.com/wp-content/uploads/2015/08/Cross-browser-testing-in-Selenium-Webdriver.png?ssl=1)

Selenium Webdriver- Selenium 2

Selenium Webdriver came in 2011 and It supports all the feature of Selenium RC and additionally they have following benefits.

 1- They removed server part from it so performance not an issue in Webdriver. It means simply write your code and it will directly communicate with browsers.

2- Selenium Webdriver supports Mobile Automation as well which make Selenium more powerfull. Currently Selendroid and Appium are present which allow us to automate IOS and Android application.

Soon Webriver will launch API for other platforms too.

First test:

**package** package1;

**import** org.openqa.selenium.WebDriver;

//It contains the WebDriver class to instantiate a new browser

**import** org.openqa.selenium.firefox.FirefoxDriver;

//It contains the FirefoxDriver class to instantiate a Firefox driver

**public** **class** FirstSelenium {

**public** **static** **void** main(String args[]) {

System.***out***.print("hai");

System.*setProperty*("webdriver.gecko.driver","D://Selenium Training//Selenium Environment Files//geckodriver.exe");

//Instantiation of driver object. To launch Firefox browser

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

//Declaration of variables

String url = "http://softwaretestingmaterial.com";

String expectedTitle = "Software Testing Material - A site for Software Testers";

String actualTitle = **null**;

//To open URL "https://softwaretestingmaterial.com". This is what we have assigned to the variable named 'url'.

driver.get(url);

// maximize the browser window

driver.manage().window().maximize();

//To get the actual value of the title. getTitle method used to get the page title

actualTitle = driver.getTitle();

//Using if-else condition to compare the Expected Title and Actual Title. As per the below lines of code (if-else condition).

**if** (actualTitle.contentEquals(expectedTitle)){

//'system.out.println' prints the output

System.***out***.println("Expected Value is "+expectedTitle);

System.***out***.println("Actual Value is "+actualTitle);

System.***out***.println("Test Passed");

} **else** {

System.***out***.println("Expected Value is "+expectedTitle);

System.***out***.println("Actual Value is "+actualTitle);

System.***out***.println("Test Failed");

}

//'close' method is used to close the browser window

driver.close();

//To run the script - Go to menu bar - click on Run - Run or use shortcut key Ctrl+F11

//You could see the output in the console as shown below:

//Expected Value is Software Testing Material - A site for Software Testers

//Actual Value is Software Testing Material - A site for Software Testers

//Test Passed

}

}

Internet exploler:

System.setProperty("webdriver.ie.driver", "D://Selenium Environment//IEDriverServer\_x64\_2.53.1//IEDriverServer.exe");

//Initialize InternetExplorerDriver Instance.

                  WebDriver driver = new InternetExplorerDriver();

driver.get("https://www.softwaretestingmaterial.com/software-testing-interview-questions-free-ebook/");

chrome driver:

System.*setProperty*("webdriver.chrome.driver","C:\\Users\\HP\\Downloads\\chromedriver-win64\\chromedriver.exe");

//Instantiation of driver object. To launch Firefox browser

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

Firefox:

System.*setProperty*("webdriver.gecko.driver","D://Selenium Training//Selenium Environment Files//geckodriver.exe");

//Instantiation of driver object. To launch Firefox browser

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

First Example:

**package** package1;

**import** org.openqa.selenium.WebDriver;

//It contains the WebDriver class to instantiate a new browser

**import** org.openqa.selenium.firefox.FirefoxDriver;

//It contains the FirefoxDriver class to instantiate a Firefox driver

**public** **class** FirstSelenium {

**public** **static** **void** main(String args[]) {

System.***out***.print("hai");

System.*setProperty*("webdriver.gecko.driver","C:\\Users\\HP\\Downloads\\geckodriver-v0.33.0-win32\\geckodriver.exe");

//Instantiation of driver object. To launch Firefox browser

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

//Declaration of variables

String baseUrl = "http://demo.guru99.com/test/newtours/";

String expectedTitle = "Welcome: Mercury Tours";

String actualTitle = "";

// launch Fire fox and direct it to the Base URL

driver.get(baseUrl);

// get the actual value of the title

actualTitle = driver.getTitle();

/\*

\* compare the actual title of the page with the expected one and print

\* the result as "Passed" or "Failed"

\*/

**if** (actualTitle.contentEquals(expectedTitle)){

System.***out***.println("Test Passed!");

} **else** {

System.***out***.println("Test Failed");

}

//close Fire fox

driver.close();

}

}

Different types of Locators in Selenium are as follows:

i. [ID](https://www.softwaretestingmaterial.com/how-to-locate-element-by-id-locator/)  
ii. [Name](https://www.softwaretestingmaterial.com/how-to-locate-element-by-name-locator/)  
iii. [Class Name](https://www.softwaretestingmaterial.com/how-to-locate-element-by-class-name-locator/)  
iv. [Tag Name](https://www.softwaretestingmaterial.com/how-to-locate-element-by-tag-name-locator/)  
v. [Link Text & Partial Link Text](https://www.softwaretestingmaterial.com/how-to-locate-element-by-link-text-and-partial-link-text-locator/)  
vi. [CSS Selector](https://www.softwaretestingmaterial.com/css-selector-selenium-webdriver-tutorial/)  
vii. [XPath](https://www.softwaretestingmaterial.com/how-to-locate-element-by-xpath-locator/)

Locating elements in WebDriver is done by using the method “**findElement(By.*locator*())**“.

We use above mentioned method to locate elements but the locator() part will be replaced with the locator names. Mentioned few examples below for reference.

findElement(By.id("IdName"))

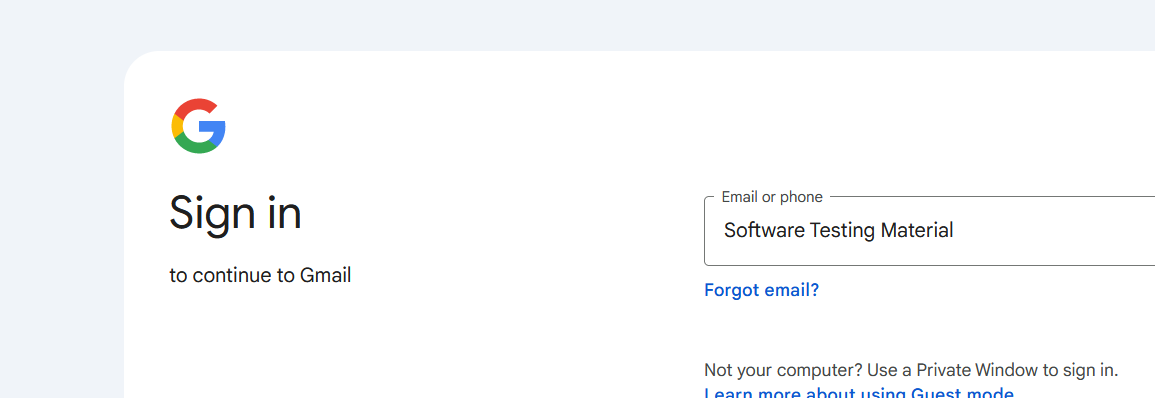
findElement(By.className("ClassName"))

**ID Locator:**

ID’s are unique for each element so it is common way to locate elements using ID Locator. As per W3C, ID’s are supposed to be unique on a page and it makes ID’s are the most reliable locator. ID locators are the fastest and safest locators out of all locators.

id = id of the element

findElement(By.id("IdName"))



Code:

**package** package1;

**import** org.openqa.selenium.By;

**import** org.openqa.selenium.WebDriver;

**import** org.openqa.selenium.firefox.FirefoxDriver;

**public** **class** byId {

**public** **static** **void** main (String [] args){

// Open browser

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

// Open Application

driver.get("https://www.gmail.com");

// Locate the element using ID locator and enters test value "Software Testing Material"

driver.findElement(By.*id*("identifierId")).sendKeys("Software Testing Material");

}

}

**Name Locator:**

We sometimes use **Name** locator to identify the elements on our webpage. Locating elements using Name is same as locating elements using ID locator.

These are not unique on a page. If there are multiple elements with the same Name locator then the first element on the page is selected. Test may fail, if another element with the same Name locator is present on the web page or added by the developers in the later stages.

Name = Name of the element

findElement(By.name("Name"))

Tagname:

**package** package1;

**import** java.util.List;

**import** org.openqa.selenium.By;

**import** org.openqa.selenium.WebDriver;

**import** org.openqa.selenium.WebElement;

**import** org.openqa.selenium.firefox.FirefoxDriver;

**public** **class** bytagName {

**public** **static** **void** main(String args[]) {

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

// Open Application

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

List <WebElement> list = driver.findElements(By.*tagName*("a"));

System.***out***.println("Number of links: "+list.size());

**for**(**int** i = 0; i < list.size(); i++){

System.***out***.println(list.get(i).getText());

}

driver.close();

}

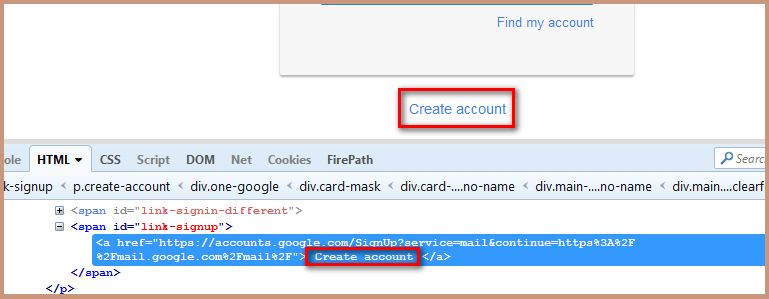
}

Link Text Locator:

If there are multiple elements with the same link text then the first one will be selected. This Link Text locator works only on links (hyperlinks) so it is called as Link Text locator.

Syntax:

findElement(By.linkText("LinkText"))

[](https://www.softwaretestingmaterial.com/wp-content/uploads/2016/10/Link-Text.png)

<span id="link-signup">

<a href="https://accounts.google.com/SignUp?service=mail&continue=https%3A%2F%2Fmail.google.com%2Fmail%2F"> Create account </a>

</span>

**Value to be added in the By.linkText method:**

findElement(By.linkText("Create account"))

**Script:**

package seleniumTutorial;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.firefox.FirefoxDriver;

public class Locators {

public static void main (String [] args){

WebDriver driver = new FirefoxDriver();

driver.get("<a href="https://www.gmail.com/" target="\_blank" data-saferedirecturl="https://www.google.com/url?hl=en&amp;q=https://www.gmail.com&amp;source=gmail&amp;ust=1475225889620000&amp;usg=AFQjCNGWiJEPM95P3VkSISyTSrTuSE2t5A">https://www.gmail.<wbr />com</a>");

driver.findElement(By.linkText("Create account")).click();

}

}

Partial link text:

**package** package1;

**import** org.openqa.selenium.By;

**import** org.openqa.selenium.WebDriver;

**import** org.openqa.selenium.firefox.FirefoxDriver;

**public** **class** bypartialLinktext {

**public** **static** **void** main (String [] args){

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

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

driver.findElement(By.*partialLinkText*("Gma")).click();

driver.close();

}

}

**CSS Selector Locator:**

There is a debate on the performance between CSS Locator and XPath locator and the debate on the performance of CSS and XPath locator is out of scope of this post. Most of the automation testers believe that using CSS selector makes the execution of script faster compared to XPath locator. This locator is always the best way to locate elements on the page.

Following are the some of the mainly used formats of CSS Selectors.

* Tag and ID

findElement(By.cssSelector(tag#id))

* Tag and Class

findElement(By.cssSelector(tag.class))

* Tag and Attribute

findElement(By.cssSelector(tag[attribute=value]))

* Tag, Class and Attribute

findElement(By.cssSelector(tag.class[attribute=value]))

Following are some of the mainly used formats of CSS Selectors.

* Tag and ID
* Tag and Class
* Tag and Attribute
* Tag, Class, and Attribute
* Sub-String Matches
  + Starts With (^)
  + Ends With (**$**)
  + Contains (**\***)
* Child Elements
  + Direct Child
  + Sub-child
  + nth-child

Refer to [W3C CSS-Selectors](http://www.w3.org/TR/CSS/#selectors) for a list of generally available CSS Selectors

**Tag and ID:**

CSS ID Selector.

**Syntax:**

css=tag#id

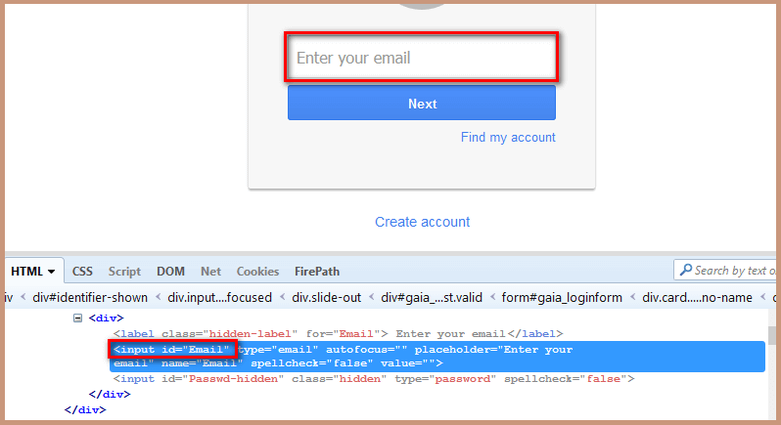
Open **Mozilla Firefox** and navigate to **Gmail**application.

Open F**irebug** and inspect the **Enter your email** input box. Take a note of its Tag and ID. Follow the below screenshot to do so.

Copy the below mentioned script and execute in your system.

**[×](https://go.ezodn.com/ads/charity/proxy?p_id=c4003a32-3d09-4b7e-6325-635325c1abbd&d_id=74963&imp_id=2472888478589312&c_id=1079&l_id=10016&url=https%3A%2F%2Fwww.directrelief.org%2Femergency%2Fhurricane-ian%2F&ffid=1&co=IN)**

Ezoic

[](https://www.softwaretestingmaterial.com/wp-content/uploads/2016/10/TagId.png)

<div>

<label class=”hidden-label” for=”Email”> Enter your email</label>

<input id=”Email” type=”email” autofocus=”” placeholder=”Enter your email” name=”Email” spellcheck=”false” value=””> <input id=”Passwd-hidden” class=”hidden” type=”password” spellcheck=”false”>

</div>

**Value to be added in the By.cssSelector method:**

css=input#Email

**Script:**

package seleniumTutorial;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.firefox.FirefoxDriver;

public class Locators {

public static void main (String [] args){

WebDriver driver = new FirefoxDriver();

driver.get("https://www.gmail.com");

// Here Tag = input and Id = Email

driver.findElement(By.cssSelector("input#Email")).sendKeys("Software Testing Material");

}

}

**Tag and Class:**

If multiple elements have the same HTML tag and class, then the first one will be recognized.

**Syntax:**

css=tag.class

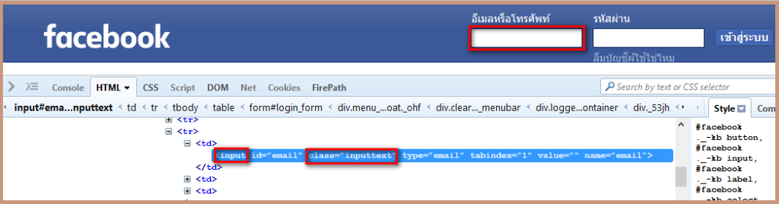
Open **Mozilla Firefox** and navigate to **Facebook**application.

Open F**irebug** and inspect the **Email** input box. Take a note of its Tag and Class. Follow the below screenshot to do so.

Copy the below mentioned script and execute in your system.

**[×](https://go.ezodn.com/ads/charity/proxy?p_id=c4003a32-3d09-4b7e-6325-635325c1abbd&d_id=74963&imp_id=3933415610666189&c_id=1079&l_id=10016&url=https%3A%2F%2Fwww.directrelief.org%2Femergency%2Fhurricane-ian%2F&ffid=1&co=IN)**

Ezoic



<td>

<input id=”email“ class=”inputtext“ type=”email“ tabindex=”1“ value=”” name=”email“>

</td>

**Value to be added in the *By.cssSelector* method:**

css=input.inputtext

**Script:**

package seleniumTutorial;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.firefox.FirefoxDriver;

public class Locators {

public static void main (String [] args){

WebDriver driver = new FirefoxDriver();

driver.get("https://www.facebook.com/");

// Here Tag = input and Class = email

driver.findElement(By.cssSelector("input.inputtext")).sendKeys("Software Testing Material");

}

}

**Tag and Attribute:**

If multiple elements have the same HTML tag and attribute, then the first one will be recognized. It acts in the same way of locating elements using CSS selectors with the same tag and class.

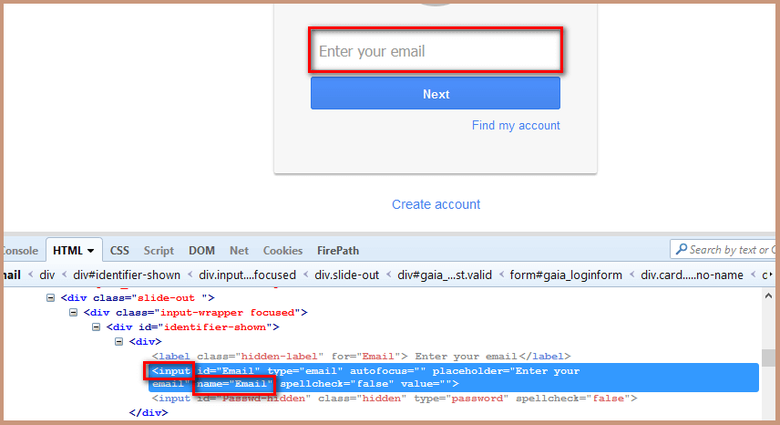
**Syntax:**

css=tag[attribute=value]

Open **Mozilla Firefox** and navigate to **Gmail**application.

Open F**irebug** and inspect the **Enter your email** input box. Take a note of its Tag and Attribute. Follow the below screenshot to do so.

Copy the below mentioned script and execute in your system.



<div>

<label class=”hidden-label“ for=”Email“> Enter your email</label>

<input id=”Email“ type=”email“ autofocus=”” placeholder=”Enter your email“ name=”Email“ spellcheck=”false“ value=””> <input id=”Passwd-hidden“ class=”hidden“ type=”password“ spellcheck=”false“>

</div>

**Value to be added in the *By.cssSelector* method:**

css=input[name=Email]

**Script:**

package seleniumTutorial;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.firefox.FirefoxDriver;

public class Locators {

public static void main (String [] args){

WebDriver driver = new FirefoxDriver();

driver.get("https://www.gmail.com");

// Here Tag = input and Id = Email

driver.findElement(By.cssSelector("input[name=Email]")).sendKeys("Software Testing Material");

}

}

**Tag, Class And Attribute:**

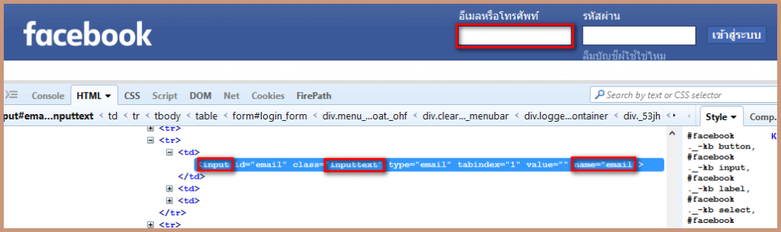
**Syntax:**

css=tag.class[attribute=value]

Open **Mozilla Firefox** and navigate to **Facebook**application.

Open **Firebug** and inspect the **Email** input box. Take a note of its Tag, Class and Attribute. Follow the below screenshot to do so.

Copy the below mentioned script and execute in your system.



<td>

<input id="email" class="inputtext" type="email" tabindex="1" value="" name="email">

</td>

**Value to be added in the *By.cssSelector* method:**

css=input.inputtext[name=email]

**Script:**

package seleniumTutorial;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.firefox.FirefoxDriver;

public class Locators {

public static void main (String [] args){

WebDriver driver = new FirefoxDriver();

driver.get("https://www.facebook.com/");

// Here Tag = input and Class = email

driver.findElement(By.cssSelector("input.inputtext[name=email]")).sendKeys("Software Testing Material");

}

}

**SUB-STRING MATCHES:**

CSS in Selenium has an interesting feature of allowing partial string matches using **^**, **$**, and **\***.

Have a look on the below mentioned HTML

<input="Employee\_ID\_001">

**Starts with (^):**

To select the element, we would use *^* which means ‘*starts with*’

**Syntax:**

css=<HTML tag><[attribute^=prefix of the string]>

**Value to be added in the *By.cssSelector* method:**

css=input[id^='Em']

Add the below step in the script to find the element and write a text as “hi”

driver.findElement(By.cssSelector("input[id^='Em']")).sendKeys("hi");

**Ends with ($):**

To select the element, we would use *$* which means ‘*ends with’*

**Syntax:**

css=<HTML tag><[attribute$=suffix of the string]>

**Value to be added in the *By.cssSelector* method:**

css=input[id$='001']

Add the below step in the script to find the element and write a text as “hi”

driver.findElement(By.cssSelector("input[id$='001']")).sendKeys("hi");

**Contains (\*):**

To select the element, we would use ***\**** which means ‘*sub-string*’

**Syntax:**

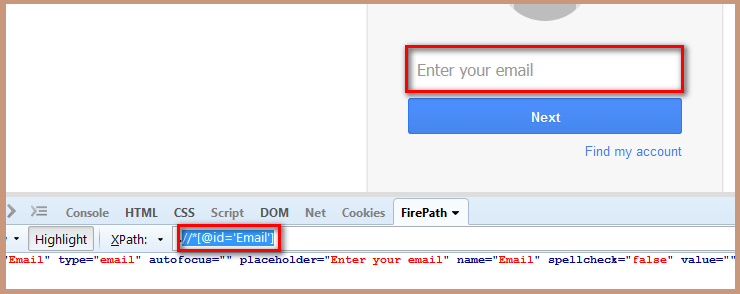
css=<HTML tag><[attribute\*=sub string]>

**Value to be added in the *By.cssSelector* method:**

css=input[id\*='id']

Add the below step in the script to find the element and write a text as “hi”

driver.findElement(By.cssSelector("input[id\*='id']")).sendKeys("hi");

[](https://www.softwaretestingmaterial.com/wp-content/uploads/2016/10/XPath.png)

**Value to be added in the By.xpath method:**

findElement(By.xpath("//\*[@id='Email']"))

**Script:**

package seleniumTutorial;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.firefox.FirefoxDriver;

public class Locators {

public static void main (String [] args){

WebDriver driver = new FirefoxDriver();

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

driver.findElement(By.xpath("//\*[@id='Email']")).sendKeys("Software Testing Material");

}

}

**What are the Types of XPath?**

There are two types of XPath in Selenium

1. Absolute XPath
2. Relative XPath

**#1. Absolute XPath**

**Absolute XPath** starts from the root node and ends with desired descendant element’s node. It starts with the top HTML node and ends with the input node. It starts with a single forward-slash(/) as shown below.

**Absolute XPath Example**

/html[1]/body[1]/div[2]/header[1]/div[1]/div[1]/div[1]/div[1]/div[1]/div[1]/div[1]/div[1]/div[2]/div[1]/nav[1]/div[1]/ul[1]/li[5]/a[1]

**Another example**

/html/body/div/div/form/table/tbody/tr/td/input

**Relative XPath** starts from any node in between the HTML page to the current element’s node(last node of the element). It starts with a double forward-slash(//) as shown below.

**Relative XPath Example**

//li[@id='menu-item-4335']//a[normalize-space()='Free Resources']

**Another example**

//input@id='email']

**What is Dynamic XPath in Selenium & How to Find XPath?**

Sometimes, we may not identify the element using the locators such as id, class, name, etc. In those cases, we use *XPath* to find an element on the web page.**What are XPath axes**?

XPath axes are the methods used to find dynamic elements that would otherwise be impossible using standard XPath methods, which do not include an ID, Classname, Name, or other identifiers.

Axes methods are used to find those elements, which dynamically change on refresh or any other operations. There are few axes methods commonly used in Selenium Webdriver like child, parent, ancestor, sibling, preceding, self, etc.

**Using Multiple Attribute**

**Syntax:**

//<HTML tag>[@attribute\_name1='attribute\_value1'][@attribute\_name2='attribute\_value2]

or

//\*[@attribute\_name1='attribute\_value1'][@attribute\_name2='attribute\_value2]

**XPath based on above HTML:**

//input[@id='Email'][@name='Email']

or

//\*[@id='Email'][@name='Email']

**#5. Using AND**

**Syntax:**

//<HTML tag>[@attribute\_name1='attribute\_value1' and @attribute\_name2='attribute\_value2]

or

//\*[@attribute\_name1='attribute\_value1' and @attribute\_name2='attribute\_value2]

**XPath based on above HTML:**

//input[@id='Email' and @name='Email']

or

//\*[@id='Email' and @name='Email']

**#6. Using OR**

**Syntax:**

//<HTML tag>[@attribute\_name1='attribute\_value1' or @attribute\_name2='attribute\_value2]

or

//\*[@attribute\_name1='attribute\_value1' or @attribute\_name2='attribute\_value2]

**XPath based on above HTML:**

//input[@id='Email' or @name='Email']

or

//\*[@id='Email' or @name='Email']

**#7. Using contains()**

Contains() is a method used to identify an element that changes dynamically and when we are familiar with some part of the attribute value of that element.

**Syntax:**

//<HTML tag>[contains(@attribute\_name,'attribute\_value')]

or

//\*[contains(@attribute\_name,'attribute\_value')]

**XPath based on above HTML:**

//input[contains(@id,'Email')]

or

//\*[contains(@id,'Email')]

or

//input[contains(@name,'Email')]

or

//\*input[contains(@name,'Email')]

#**8. Using starts-with()**

starts-with() method is used to identify an element when we are familiar with the value of the attribute (starting with the specified text) of an element.

**Syntax:**

//<HTML tag>[starts-with(@attribute\_name,'attribute\_value')]

or

//\*[starts-with(@attribute\_name,'attribute\_value')]

**XPath based on above HTML:**

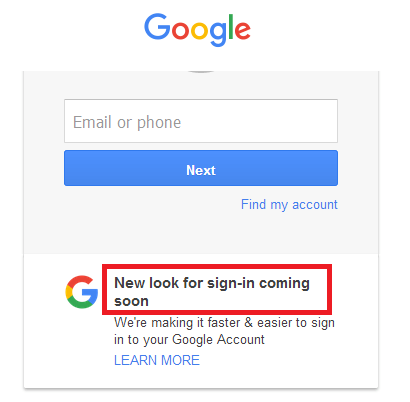
//input[starts-with(@id,'Ema')]

or

//\*[starts-with(@id,'Ema')]

**#9. Using text()**

This mechanism is used to locate an element based on the text available on a webpage

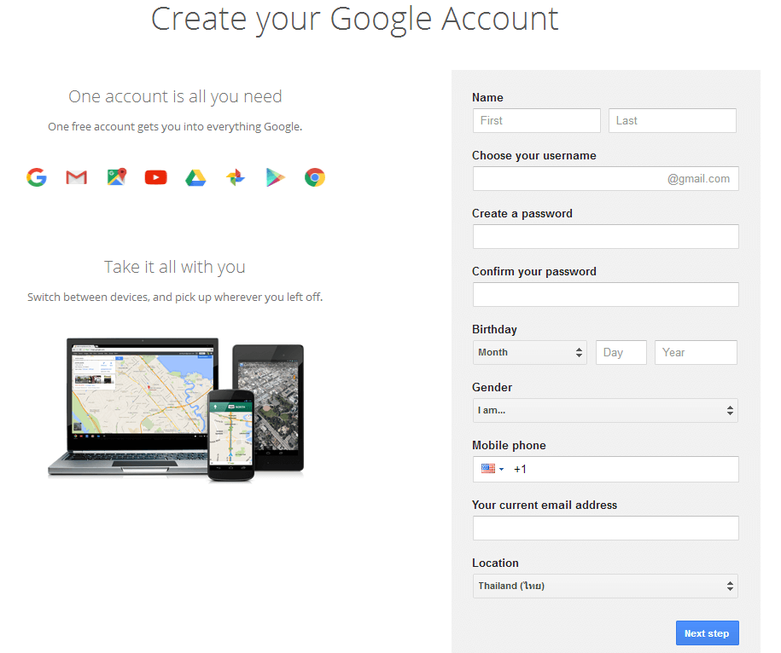
[](https://www.softwaretestingmaterial.com/wp-content/uploads/2017/04/Gmail_New_Look_Text.png)

As per the above image, we could identify the elements text based on the below xpath.

//\*[text()='New look for sign-in coming soon']

**#10. Using last()**

Selects the last element (of mentioned type) out of all input element present

[](https://www.softwaretestingmaterial.com/wp-content/uploads/2017/04/Registration_Page.png)

To identify the element (last text field ) “Your current email address”, we could use the below xpath.

findElement(By.xpath("(//input[@type='text'])[last()]"))

To identify the element “Year”, we could use the below xpath.

**[last()-1]** – Selects the last but one element (of mentioned type) out of all input element present

findElement(By.xpath("(//input[@type='text'])[last()-1]"))

**#11. Using position()**

Selects the element out of all input element present depending on the position number provided

In below given xpath, [@type=’text’] will locate text field and function [position()=2] will locate text filed which is located on 2nd position from the top.

findElement(By.xpath("(//input[@type='text'])[position()=2]"))

or

findElement(By.xpath("(//input[@type='text'])[2]"))

**#12. Using index**

By providing the index position in the square brackets, we could move to the nth element. Based on the below xpath, we could identify the Last Name field.

findElement(By.xpath("//label[2]"))

**#11. Using position()**

Selects the element out of all input element present depending on the position number provided

In below given xpath, [@type=’text’] will locate text field and function [position()=2] will locate text filed which is located on 2nd position from the top.

findElement(By.xpath("(//input[@type='text'])[position()=2]"))

or

findElement(By.xpath("(//input[@type='text'])[2]"))

**#12. Using index**

By providing the index position in the square brackets, we could move to the nth element. Based on the below xpath, we could identify the Last Name field.

findElement(By.xpath("//label[2]"))

**#13. Using following XPath axes**

By using this we could select everything on the web page after the closing tag of the current node

xpath of the FirstName field is as follows

//\*[@id='FirstName']

To identify the input field of type text after the FirstName field, we need to use the below xpath.

//\*[@id='FirstName']/following::input[@type='text']

Here I used, *following* xpath axes and *two colons* and then specified the required tag (i.e., input)

To identify just the input field after the FirstName field, we need to use the below xpath.

//\*[@id='FirstName']/following::input

**#14. Using preceding XPath axes**

Selects all nodes that appear before the current node in the document, except ancestors, attribute nodes, and namespace nodes

xpath of the LastName field is as follows

//\*[@id='LastName']

To identify the input field of type text before the LastName field, we need to use the below xpath.

//\*[@id='LastName']//preceding::input[@type='text']"

Here I used, *preceding* xpath axes and *two colons* and then specified the required tag (i.e., input).

**XPath axes methods**

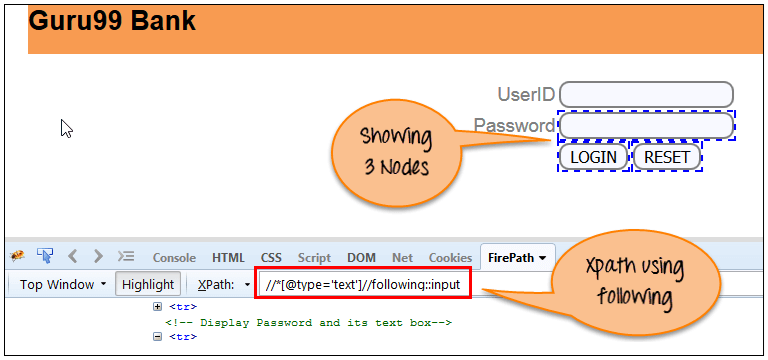
These XPath axes methods are used to find the complex or dynamic elements. Below we will see some of these methods.

For illustrating these XPath axes method, we will use the Guru99 bank demo site.

**1) Following**

Selects all elements in the document of the current node( ) [ UserID input box is the current node] as shown in the below screen.

Xpath=//\*[@type='text']//following::input

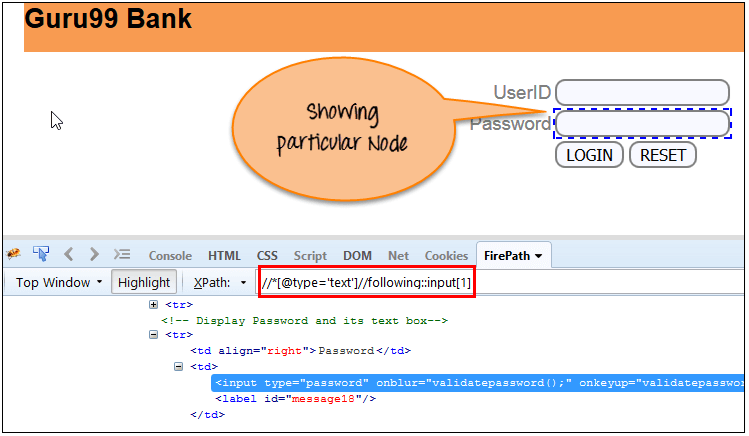
[](https://www.guru99.com/images/3-2016/032816_0758_XPathinSele12.png)

There are 3 “input” nodes matching by using “following” axis- password, login and reset button. If you want to focus on any particular element then you can use the below XPath method:

Xpath=//\*[@type='text']//following::input[1]

You can change the XPath according to the requirement by putting [1],[2]…………and so on.

With the input as ‘1’, the below screen shot finds the particular node that is ‘Password’ input box element.

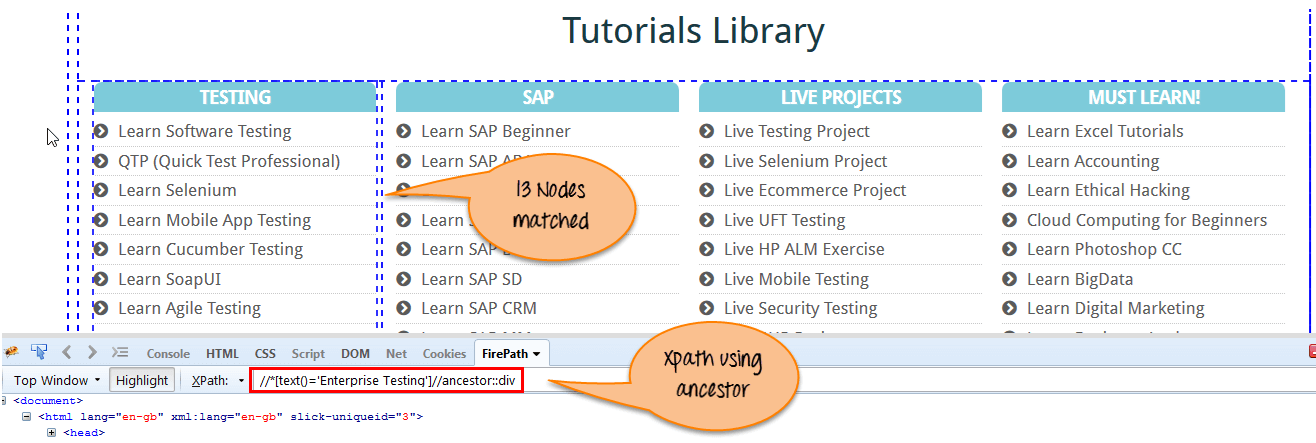
[](https://www.guru99.com/images/3-2016/032816_0758_XPathinSele13.png)

**2) Ancestor**

The ancestor axis selects all ancestors element (grandparent, parent, etc.) of the current node as shown in the below screen.

In the below expression, we are finding ancestors element of the current node(“ENTERPRISE TESTING” node).

Xpath=//\*[text()='Enterprise Testing']//ancestor::div

[](https://www.guru99.com/images/3-2016/032816_0758_XPathinSele14.png)

There are 13 “div” nodes matching by using “ancestor” axis. If you want to focus on any particular element then you can use the below XPath, where you change the number 1, 2 as per your requirement:

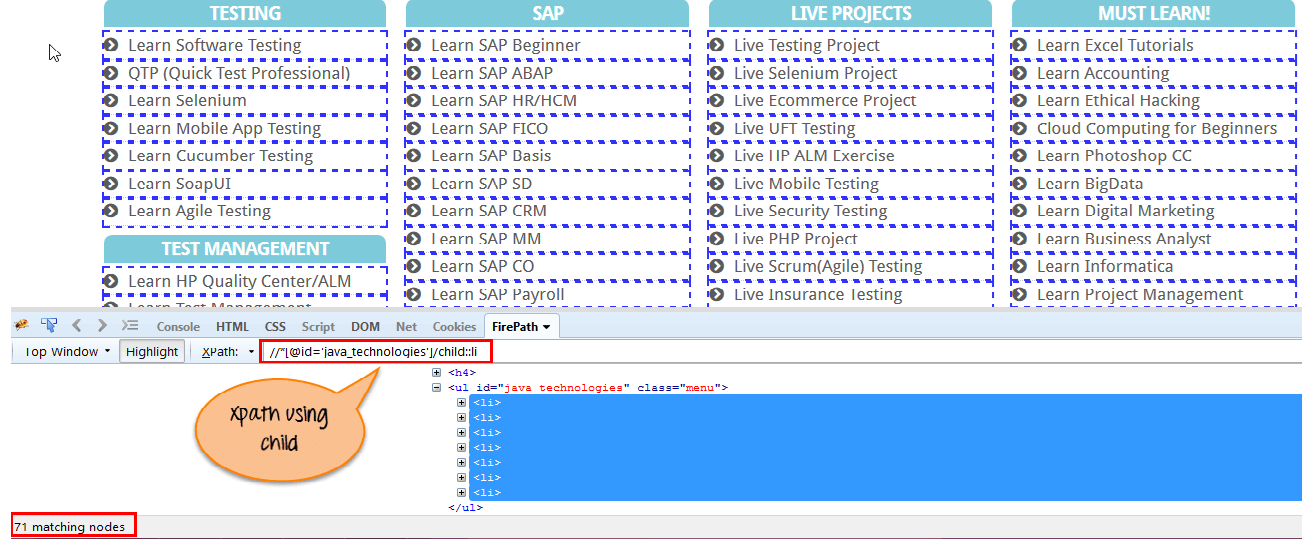
Xpath=//\*[text()='Enterprise Testing']//ancestor::div[1]

You can change the XPath according to the requirement by putting [1], [2]…………and so on.

**3) Child**

Selects all children elements of the current node (Java) as shown in the below screen.

Xpath=//\*[@id='java\_technologies']//child::li

[](https://www.guru99.com/images/3-2016/032816_0758_XPathinSele15.png)

There are 71 “li” nodes matching by using “child” axis. If you want to focus on any particular element then you can use the below xpath:

Xpath=//\*[@id='java\_technologies']//child::li[1]

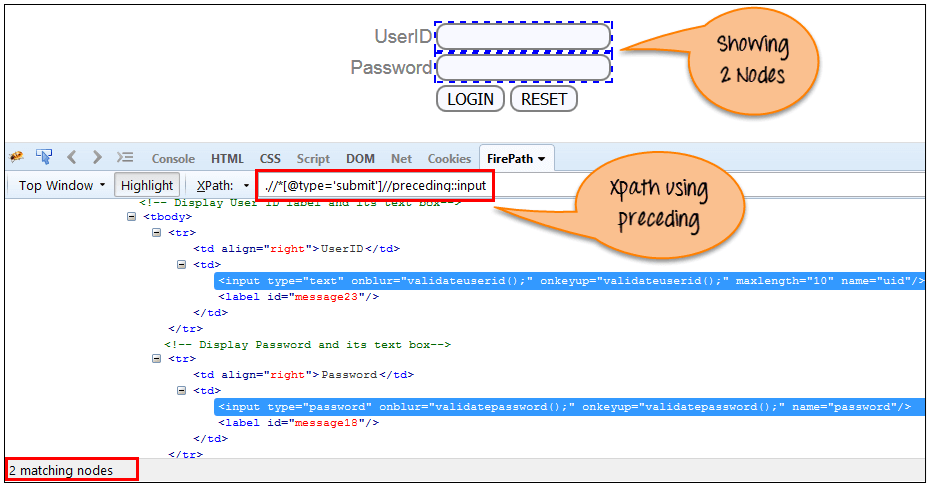
You can change the xpath according to the requirement by putting [1],[2]…………and so on.

**4) Preceding**

Select all nodes that come before the current node as shown in the below screen.

In the below expression, it identifies all the input elements before “LOGIN” button that is **Userid** and **password** input element.

Xpath=//\*[@type='submit']//preceding::input

[](https://www.guru99.com/images/3-2016/032816_0758_XPathinSele16.png)

There are 2 “input” nodes matching by using “preceding” axis. If you want to focus on any particular element then you can use the below XPath:

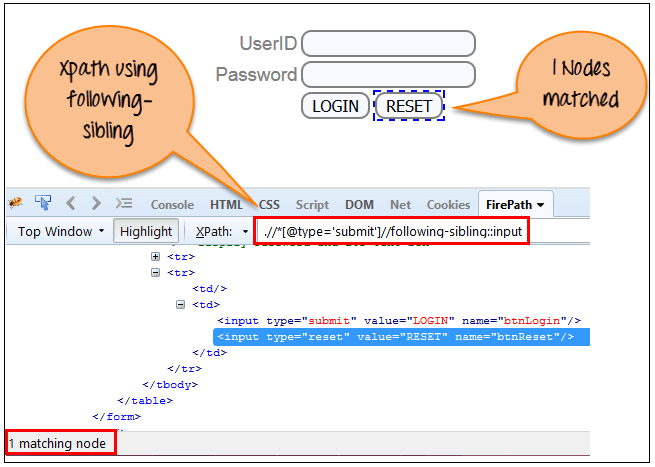
Xpath=//\*[@type='submit']//preceding::input[1]

You can change the xpath according to the requirement by putting [1],[2]…………and so on.

**5) Following-sibling**

Select the following siblings of the context node. Siblings are at the same level of the current node as shown in the below screen. It will find the element after the current node.

xpath=//\*[@type='submit']//following-sibling::input

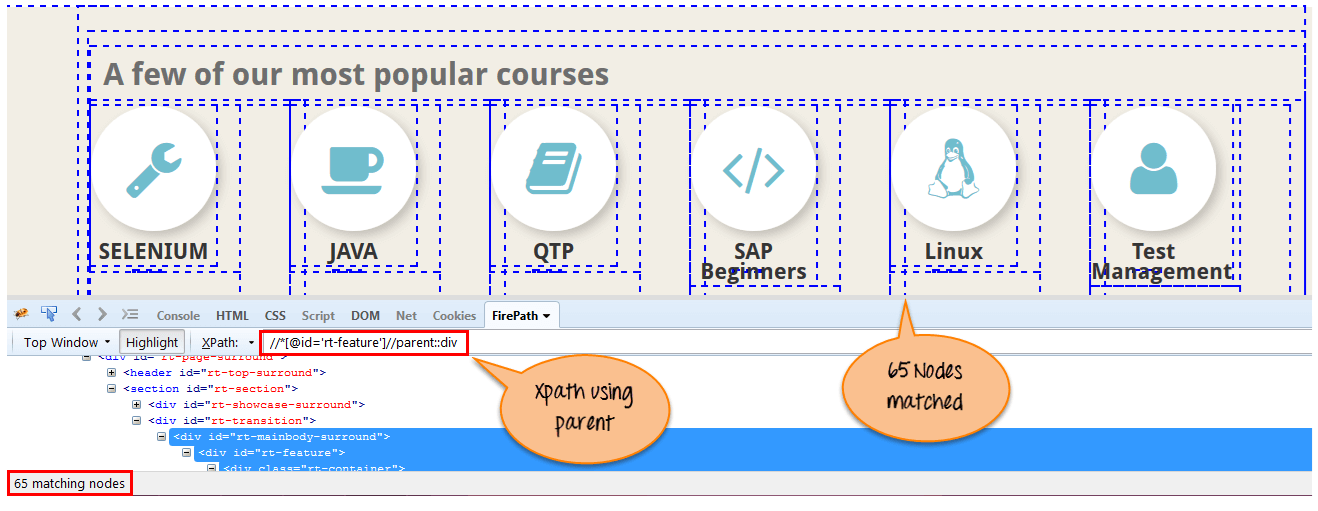
[](https://www.guru99.com/images/3-2016/032816_0758_XPathinSele17.png)

One input nodes matching by using “following-sibling” axis.

**6) Parent**

Selects the parent of the current node as shown in the below screen.

Xpath=//\*[@id='rt-feature']//parent::div

[](https://www.guru99.com/images/3-2016/032816_0758_XPathinSele18.png)

There are 65 “div” nodes matching by using “parent” axis. If you want to focus on any particular element then you can use the below XPath:

Xpath=//\*[@id='rt-feature']//parent::div[1]

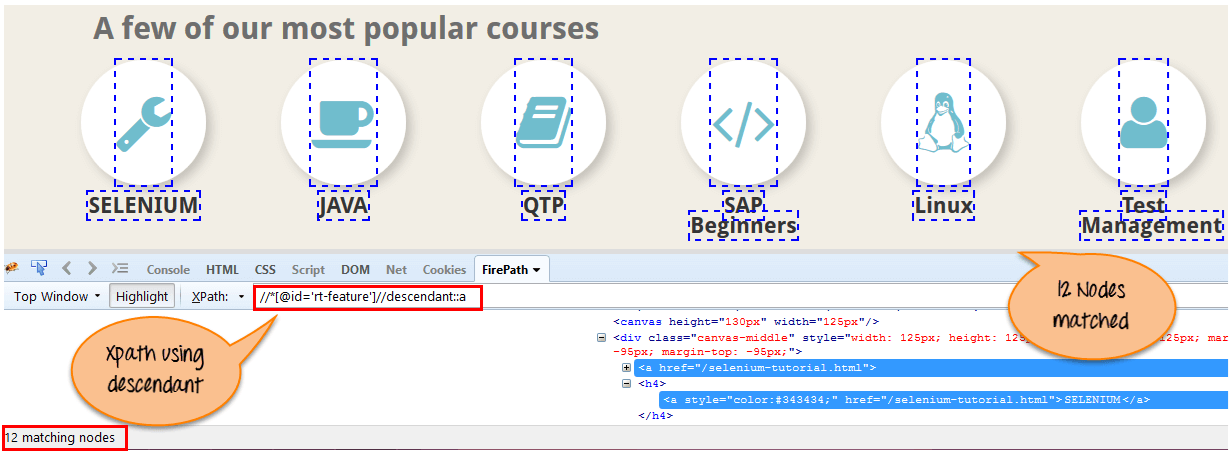
You can change the XPath according to the requirement by putting [1],[2]…………and so on.

**8) Descendant**

Selects the descendants of the current node as shown in the below screen.

In the below expression, it identifies all the element descendants to current element ( ‘Main body surround’ frame element) which means down under the node (child node , grandchild node, etc.).

Xpath=//\*[@id='rt-feature']//descendant::a

[](https://www.guru99.com/images/3-2016/032816_0758_XPathinSele20.png)

There are 12 “link” nodes matching by using “descendant” axis. If you want to focus on any particular element then you can use the below XPath:

Xpath=//\*[@id='rt-feature']//descendant::a[1]

You can change the XPath according to the requirement by putting [1],[2]…………and so on.

Following are some of the mainly used formats of CSS Selectors.

* Tag and ID
* Tag and Class
* Tag and Attribute
* Tag, Class, and Attribute
* Sub-String Matches
  + Starts With (^)
  + Ends With (**$**)
  + Contains (**\***)
* Child Elements
  + Direct Child
  + Sub-child
  + nth-child

Refer to [W3C CSS-Selectors](http://www.w3.org/TR/CSS/#selectors) for a list of generally available CSS Selectors

**Tag and ID:**

CSS ID Selector.

Ezoic

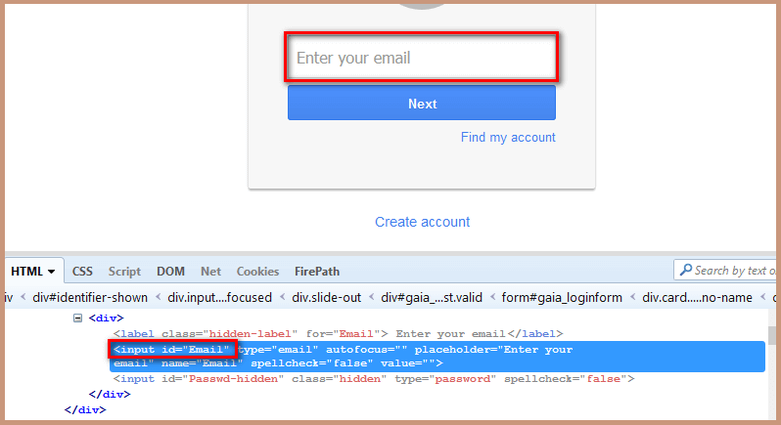
**Syntax:**

css=tag#id

Open **Mozilla Firefox** and navigate to **Gmail**application.

Open F**irebug** and inspect the **Enter your email** input box. Take a note of its Tag and ID. Follow the below screenshot to do so.

Copy the below mentioned script and execute in your system.

[](https://www.softwaretestingmaterial.com/wp-content/uploads/2016/10/TagId.png)

<div>

<label class=”hidden-label” for=”Email”> Enter your email</label>

<input id=”Email” type=”email” autofocus=”” placeholder=”Enter your email” name=”Email” spellcheck=”false” value=””> <input id=”Passwd-hidden” class=”hidden” type=”password” spellcheck=”false”>

</div>

**Value to be added in the By.cssSelector method:**

css=input#Email

**Script:**

package seleniumTutorial;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.firefox.FirefoxDriver;

public class Locators {

public static void main (String [] args){

WebDriver driver = new FirefoxDriver();

driver.get("https://www.gmail.com");

// Here Tag = input and Id = Email

driver.findElement(By.cssSelector("input#Email")).sendKeys("Software Testing Material");

}

}

**Tag and Class:**

If multiple elements have the same HTML tag and class, then the first one will be recognized.

**Syntax:**

css=tag.class

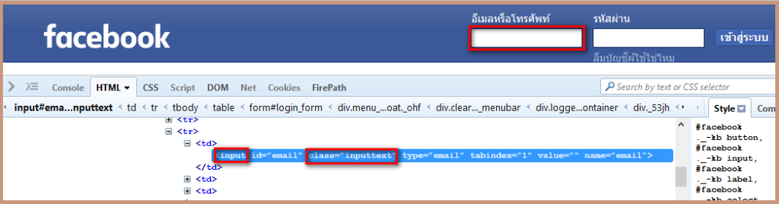
Open **Mozilla Firefox** and navigate to **Facebook**application.

Open F**irebug** and inspect the **Email** input box. Take a note of its Tag and Class. Follow the below screenshot to do so.

Copy the below mentioned script and execute in your system.

**[×](https://go.ezodn.com/ads/charity/proxy?p_id=252cf400-15a1-4b7b-4a32-15ea3f752566&d_id=74963&imp_id=8759350414024627&c_id=1079&l_id=10016&url=https%3A%2F%2Fwww.directrelief.org%2Femergency%2Fhurricane-ian%2F&ffid=1&co=IN)**

Ezoic



<td>

<input id=”email“ class=”inputtext“ type=”email“ tabindex=”1“ value=”” name=”email“>

</td>

**Value to be added in the *By.cssSelector* method:**

css=input.inputtext

**Script:**

package seleniumTutorial;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.firefox.FirefoxDriver;

public class Locators {

public static void main (String [] args){

WebDriver driver = new FirefoxDriver();

driver.get("https://www.facebook.com/");

// Here Tag = input and Class = email

driver.findElement(By.cssSelector("input.inputtext")).sendKeys("Software Testing Material");

}

}

**Tag and Attribute:**

If multiple elements have the same HTML tag and attribute, then the first one will be recognized. It acts in the same way of locating elements using CSS selectors with the same tag and class.

**[×](https://go.ezodn.com/ads/charity/proxy?p_id=252cf400-15a1-4b7b-4a32-15ea3f752566&d_id=74963&imp_id=6099671708027804&c_id=1079&l_id=10016&url=https%3A%2F%2Fwww.directrelief.org%2Femergency%2Fhurricane-ian%2F&ffid=1&co=IN)**

Ezoic

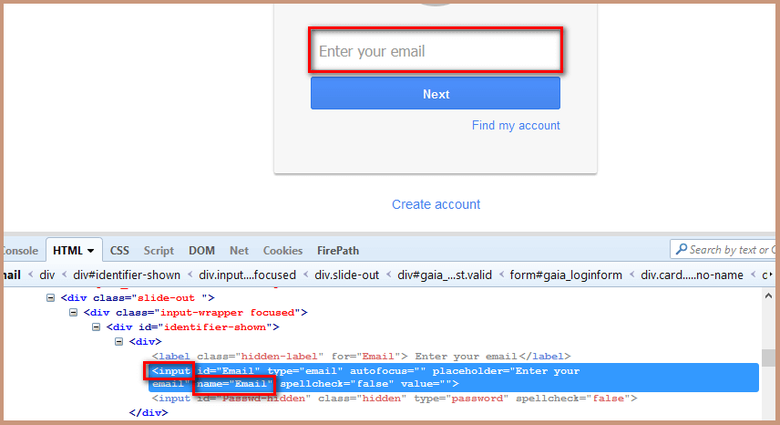
**Syntax:**

css=tag[attribute=value]

Open **Mozilla Firefox** and navigate to **Gmail**application.

Open F**irebug** and inspect the **Enter your email** input box. Take a note of its Tag and Attribute. Follow the below screenshot to do so.

Copy the below mentioned script and execute in your system.



<div>

<label class=”hidden-label“ for=”Email“> Enter your email</label>

<input id=”Email“ type=”email“ autofocus=”” placeholder=”Enter your email“ name=”Email“ spellcheck=”false“ value=””> <input id=”Passwd-hidden“ class=”hidden“ type=”password“ spellcheck=”false“>

</div>

**Value to be added in the *By.cssSelector* method:**

css=input[name=Email]

**Script:**

package seleniumTutorial;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.firefox.FirefoxDriver;

public class Locators {

public static void main (String [] args){

WebDriver driver = new FirefoxDriver();

driver.get("https://www.gmail.com");

// Here Tag = input and Id = Email

driver.findElement(By.cssSelector("input[name=Email]")).sendKeys("Software Testing Material");

}

}

**Tag, Class And Attribute:**

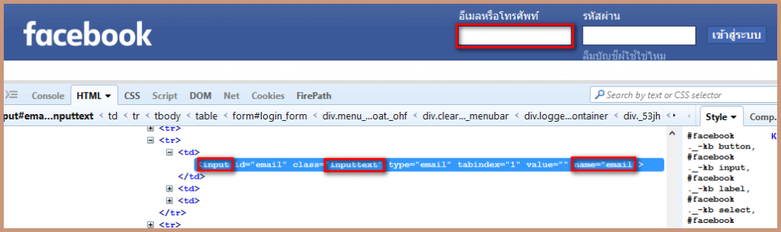
**Syntax:**

css=tag.class[attribute=value]

Open **Mozilla Firefox** and navigate to **Facebook**application.

Open **Firebug** and inspect the **Email** input box. Take a note of its Tag, Class and Attribute. Follow the below screenshot to do so.

Copy the below mentioned script and execute in your system.



<td>

<input id="email" class="inputtext" type="email" tabindex="1" value="" name="email">

</td>

**Value to be added in the *By.cssSelector* method:**

css=input.inputtext[name=email]

**Script:**

package seleniumTutorial;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.firefox.FirefoxDriver;

public class Locators {

public static void main (String [] args){

WebDriver driver = new FirefoxDriver();

driver.get("https://www.facebook.com/");

// Here Tag = input and Class = email

driver.findElement(By.cssSelector("input.inputtext[name=email]")).sendKeys("Software Testing Material");

}

}

**SUB-STRING MATCHES:**

CSS in Selenium has an interesting feature of allowing partial string matches using **^**, **$**, and **\***.

Have a look on the below mentioned HTML

<input="Employee\_ID\_001">

**Starts with (^):**

To select the element, we would use *^* which means ‘*starts with*’

**Syntax:**

css=<HTML tag><[attribute^=prefix of the string]>

**Value to be added in the *By.cssSelector* method:**

css=input[id^='Em']

Add the below step in the script to find the element and write a text as “hi”

driver.findElement(By.cssSelector("input[id^='Em']")).sendKeys("hi");

**Ends with ($):**

To select the element, we would use *$* which means ‘*ends with’*

**Syntax:**

css=<HTML tag><[attribute$=suffix of the string]>

**Value to be added in the *By.cssSelector* method:**

css=input[id$='001']

Add the below step in the script to find the element and write a text as “hi”

driver.findElement(By.cssSelector("input[id$='001']")).sendKeys("hi");

**Contains (\*):**

To select the element, we would use ***\**** which means ‘*sub-string*’

**Syntax:**

css=<HTML tag><[attribute\*=sub string]>

**Value to be added in the *By.cssSelector* method:**

css=input[id\*='id']

Add the below step in the script to find the element and write a text as “hi”

driver.findElement(By.cssSelector("input[id\*='id']")).sendKeys("hi");

**Also we can use ‘contains()’:**

driver.findElement(By.cssSelector("input:contains('id')")).sendKeys("hi");

**Locating Child Elements(Direct Child):**

<div id="buttonDiv" class="small">

<button id="submitButton" type="button" class="btn">Submit</button>

</div>

**Syntax:** parentLocator>childLocator

**CSS Locator:** div#buttonDiv>button

**Explanation:** ‘div#buttonDiv>button’ will first go to div element with id ‘buttonDiv’ and then select its child element – ‘button’

**Locating elements inside other elements (child or sub-child):**

**Syntax:** parentLocator>locator1 locator2

**CSS Locator:** div#buttonDiv button

**Explanation:** ‘div#buttonDiv button’ will first go to div element with id ‘buttonDiv’ and then select ‘button’ element inside it (which may be its child or sub child)

**Locating nth Child:**

To find nth-child css.

<ul id="automation">

<li>Selenium</li>

<li>QTP</li>

<li>Sikuli</li>

</ul>

To locate the element with text ‘QTP’, we have to use “nth-of-type”

css="ul#automation li:nth-of-type(2)"

Similarly, To select the last child element, i.e. ‘Sikuli’, we can use

css="ul#automation li:last-child"

**WebDriver Commands**

We can divide WebDriver Commands into three categories:

* Browser Specific Commands
* Browser Navigation Commands and,
* Element Specific Commands

In this post we will discuss about 25 WebDriver commands which are used frequently in Selenium Automation:

**Browser Specific Commands:**

**1. Maximize window:**

To maximize a browser window, we need to call the maximize() method of the Window interface of the driver class. Add the second line right below where you define an instance of FirefoxDriver.

**Example:**

driver = new FirefoxDriver();

driver.manage().window().maximize();

**2. Delete cookies**

Delete all the cookies for the current domain using deleteAllCookies() method.

**Example:**

driver.manage().deleteAllCookies();

Deleting the specific cookie with cookie name "--abcd"

driver.manage().deleteCookieNamed("\_\_abcd");

**3. Get command:**

The command launches a new browser and opens the specified URL in the browser instance

The command takes a single string type parameter that is usually a URL of application under test.

**Example:** driver.get("https://google.com");

**4. GetTitle:**

The command is used to retrieve the title of the webpage the user is currently working on. A null string is returned if the webpage has no title.

The command doesn’t require any parameter and returns a trimmed string value.

**Example:**String title = driver.getTitle();

**5. GetCurrentUrl:**

The command is used to retrieve the URL of the webpage the user is currently accessing

The command doesn’t require any parameter and returns a string value.

**Example:** driver.getCurrentUrl();

**package** package1;

**import** org.openqa.selenium.chrome.ChromeDriver;

**import** org.openqa.selenium.By;

**import** org.openqa.selenium.WebDriver;

**public** **class** Byxpath {

**public** **static** **void** main(String args[]) {

System.*setProperty*("webdriver.chrome.driver", "C:\\Users\\HP\\Downloads\\chromedriver-win64\\chromedriver.exe");

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

driver.get("https://www.facebook.com/");

driver.findElement(By.*xpath*("//\*[@class='inputtext \_55r1 \_6luy']")).sendKeys("Hello");

String s=driver.getCurrentUrl();

driver.close();

System.***out***.println(s);

**if**(s.contentEquals("https://www.facebook.com/")) {

System.***out***.println("Test pass");

}**else** {

System.***out***.println("test fail");

}

}

}

**7. Close:**

WebDriver’s close() method closes the web browser window that the user is currently working on or we can also say the window that is being currently accessed by the WebDriver. The command neither requires any parameter nor does it return any value.

**Example:**

driver.close();

**8. Quit**

Unlike close() method, quit() method closes down all the windows that the program has opened. Same as close() method, the command neither requires any parameter nor does it return any value.

**Example:**

driver.quit();

**Browser Navigation Commands:**

WebDriver provides some basic Browser Navigation Commands that allows the browser to move backwards or forwards in the browser's history.

**1. Navigate To:**

Method - String to(String arg0)

In WebDriver, this method loads a new web page in the existing browser window. It accepts String as parameter and returns void. The respective command to load/navigate a new web page can be written as:

**Example:**

driver.navigate().to("www.automationtestinginsider.com");

**2. Backward:**

Method - void back()

This method enables the web browser to click on the back button in the existing browser window. It neither accepts anything nor returns anything. The respective command that takes you back by one page on the browser's history can be written as:

**Example:**

driver.navigate().back();

**3. Forward:**

Method - void forward()

This method enables the web browser to click on the forward button in the existing browser window. It neither accepts anything nor returns anything. The respective command that takes you forward by one page on the browser's history can be written as:

**Example:**

driver.navigate().forward();

**4. Refresh:**

Method - void refresh()

In WebDriver, this method refresh/reloads the current web page in the existing browser window. It neither accepts anything nor returns anything. The respective command that takes you back by one page on the browser's history can be written as:

**Example:**

driver.navigate().refresh();

**Element Specific Commands:**

**1. Clear:**

clear( ) predefined method of  Selenium 'WebDriver' Class is used to clear the text entered or displayed in the text fields

**Example:** driver.findElement(By.id("userName")).sendKeys("Admin").clear();

**2. Click:**

The click command emulates a click operation for a link, button, checkbox or radio button. In Selenium Webdriver, execute click after finding an element.

**Example:** driver.findElement(By.id("button")).click();

**4. IsSelected:**

isSelected() is the method used to verify if the web element is selected or not.

**Example:**boolean ele= driver.findElement(By.id("button")).isSelected();

**5. IsEnabled:**

isEnabled() is the method used to verify if the web element is enabled or disabled within the webpage. isEnabled() is primarily used with buttons.

**Example:** boolean ele= driver.findElement(By.id("button")).isEnabled();

**6. IsDisplayed:**

isDisplayed() is the method used to verify a presence of a web element within the webpage. The method returns a “true” value if the specified web element is present on the web page and a “false” value if the web element is not present on the web page.

**Example:** boolean ele= driver.findElement(By.id("button")).IsDisplayed();

**7. getAttribute:**

The command is used to retrieve the value of the specified attribute. The command requires a single string parameter that refers to an attribute whose value we aspire to know and returns a string value. as a result.

**Example:** driver.findElement(By.id("findID")).getAttribute("value");

**8. sendkeys:**

This is a method for sending one or more keystrokes to the active window

**Example:** driver.findElement(By.id("userName")).sendKeys("Admin")

**9. Submit:**

submit() is used to click Button in Web page. Selenium Webdriver has one special method to submit any form and that method name Is submit(). You can use .click() method to click on any button.There is no restriction for click buttons. We can use .submit() method for only submit form after click on button. That means element's type = "submit" and button should be inside <form> tag, then only submit() will work.

**10. getSize:**

It will returns the "Dimension" object. If you want to get the width and Height of the specific element on the webpage then use "getsize()" method.

**11. getCssValue:** - getCssValue method in selenium fetches the value of a CSS property of an web element.

Handle Drop Down And Multi Select List Using Selenium WebDriver:

Pause

Unmute

Current Time 6:30

/

Duration 13:43

Settings

Fullscreen

[Python Selenium Action Chains Example](https://www.humix.com/video/cgH2-QYyby2)

Share

To handle drop down and multi select list using Selenium WebDriver, we need to use ***Select***class.

The ***Select***class is a Webdriver class which provides the implementation of the HTML SELECT tag. It exposes several “Select By” and “Deselect By” type methods. We use these methods to select or deselect in the drop down list or multi select object. The ***Select***class is the part of the selenium package.

We need to import the below mentioned library.

import org.openqa.selenium.support.ui.Select;

Standard syntax of Select Class is as follows:

Select dropdown = new Select(<WebElement>);

Example:

WebElement mySelectElement = driver.findElement(By.name("dropdown"));

Select dropdown = new Select(mySelectElement);

or

Ezoic

Select dropdown = new Select(driver.findElement(By.xpath("//select[@name='dropdown']")));

Note: The ***Select***class starts with capital ‘S’.

To Handle Drop Down And Multi Select List in Selenium we use the following types of Select Methods.

**Types of Select Methods:**

i. selectByVisibleText Method  
ii. selectByIndex Method  
iii. selectByValue Method

**Types of DeSelect Methods:**  
i. deselectByVisibleText Method  
ii. deselectByIndex Method  
iii. deselectByValue Method  
iv. deselectAll Method

Let’s see one by one with a sample program:

***SelectByVisibleText* Method:**

It works based on the ‘*visible text*‘ provided by us.

Syntax:

dropdown.selectByVisibleText();

Sample program:

package softwareTestingMaterial;

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

import org.openqa.selenium.support.ui.Select;

import org.testng.annotations.Test;

public class SelectMethod {

@Test

public static void captureScreenMethod() throws Exception{

System.setProperty("webdriver.chrome.driver","D://Selenium Environment//Drivers//chromedriver.exe");

WebDriver driver = new ChromeDriver();

driver.manage().window().maximize();

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

driver.get("https://www.softwaretestingmaterial.com/sample-webpage-to-automate/");

driver.navigate().refresh();

//Once you got the select object initialised then you can access all the methods of select class.

//Identify the select HTML element:

Thread.sleep(10000);

WebElement mySelectElement = driver.findElement(By.name("dropdown"));

Select dropdown= new Select(mySelectElement);

//To select an option - selectByVisibleText, selectByIndex, selectByValue

//selectByVisibleText

dropdown.selectByVisibleText("Automation Testing");

}

}

***SelectByIndex* Method:**

It works based on the ‘*index value*‘ provided by us.

Syntax

dropdown.selectByIndex(Index);

Sample program:

package softwareTestingMaterial;

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

import org.openqa.selenium.support.ui.Select;

import org.testng.annotations.Test;

public class SelectMethod {

@Test

public static void captureScreenMethod() throws Exception{

System.setProperty("webdriver.chrome.driver","D://Selenium Environment//Drivers//chromedriver.exe");

WebDriver driver = new ChromeDriver();

driver.manage().window().maximize();

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

driver.get("https://www.softwaretestingmaterial.com/sample-webpage-to-automate/");

driver.navigate().refresh();

//Once you got the select object initialised then you can access all the methods of select class.

//Identify the select HTML element:

Thread.sleep(10000);

WebElement mySelectElement = driver.findElement(By.name("dropdown"));

Select dropdown= new Select(mySelectElement);

//To select an option - selectByVisibleText, selectByIndex, selectByValue

//selectByIndex

dropdown.selectByIndex(2); // value is QTP

}

}

***SelectByValue* Method:**

It works based on the ‘*value*‘ provided by us.

Ezoic

Syntax

dropdown.selectByValue(Value);

Sample program:

package softwareTestingMaterial;

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

import org.openqa.selenium.support.ui.Select;

import org.testng.annotations.Test;

public class SelectMethod {

@Test

public static void captureScreenMethod() throws Exception{

System.setProperty("webdriver.chrome.driver","D://Selenium Environment//Drivers//chromedriver.exe");

WebDriver driver = new ChromeDriver();

driver.manage().window().maximize();

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

driver.get("https://www.softwaretestingmaterial.com/sample-webpage-to-automate/");

driver.navigate().refresh();

//Once you got the select object initialised then you can access all the methods of select class.

//Identify the select HTML element:

Thread.sleep(10000);

WebElement mySelectElement = driver.findElement(By.name("dropdown"));

Select dropdown= new Select(mySelectElement);

//To select an option - selectByVisibleText, selectByIndex, selectByValue

//selectByValue

dropdown.selectByValue("ddmanual"); // value is Manual Testing

}

}

**DeSelect Methods With Examples:**

Ezoic

***DeselectByVisibleText* Method:**

It works based on the ‘*visible text*‘ which we provide

Syntax:

dropdown.deselectByVisibleText();

Sample program:

package softwareTestingMaterial;

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

import org.openqa.selenium.support.ui.Select;

import org.testng.annotations.Test;

public class SelectMethod {

@Test

public static void captureScreenMethod() throws Exception{

System.setProperty("webdriver.chrome.driver","D://Selenium Environment//Drivers//chromedriver.exe");

WebDriver driver = new ChromeDriver();

driver.manage().window().maximize();

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

driver.get("https://www.softwaretestingmaterial.com/sample-webpage-to-automate/");

driver.navigate().refresh();

//Once you got the select object initialised then you can access all the methods of select class.

//Identify the select HTML element:

Thread.sleep(10000);

WebElement mySelectElement = driver.findElement(By.name("multipleselect[]"));

Select dropdown= new Select(mySelectElement);

//To deselect an option

//the deselect method will throw UnsupportedOperationException if the SELECT does not support multiple selections

dropdown.selectByVisibleText("Performance Testing");

Thread.sleep(2000);

dropdown.deselectByVisibleText("Performance Testing");

}

}

***DeselectByIndex* Method:**

It works based on the ‘*index value’* which we provide

Syntax:

dropdown.deselectByIndex();

Sample program:

package softwareTestingMaterial;

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

import org.openqa.selenium.support.ui.Select;

import org.testng.annotations.Test;

public class SelectMethod {

@Test

public static void captureScreenMethod() throws Exception{

System.setProperty("webdriver.chrome.driver","D://Selenium Environment//Drivers//chromedriver.exe");

WebDriver driver = new ChromeDriver();

driver.manage().window().maximize();

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

driver.get("https://www.softwaretestingmaterial.com/sample-webpage-to-automate/");

driver.navigate().refresh();

//Once you got the select object initialised then you can access all the methods of select class.

//Identify the select HTML element:

Thread.sleep(10000);

WebElement mySelectElement = driver.findElement(By.name("multipleselect[]"));

Select dropdown= new Select(mySelectElement);

//To deselect an option

//the deselect method will throw UnsupportedOperationException if the SELECT does not support multiple selections

dropdown.selectByIndex(2);

Thread.sleep(2000);

dropdown.deselectByIndex(2);

}

}

**DeselectByValue Method:**

It works based on the ‘*value*‘ provided by us.

Syntax:

dropdown.deselectByValue();

Sample program:

package softwareTestingMaterial;

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

import org.openqa.selenium.support.ui.Select;

import org.testng.annotations.Test;

public class SelectMethod {

@Test

public static void captureScreenMethod() throws Exception{

System.setProperty("webdriver.chrome.driver","D://Selenium Environment//Drivers//chromedriver.exe");

WebDriver driver = new ChromeDriver();

driver.manage().window().maximize();

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

driver.get("https://www.softwaretestingmaterial.com/sample-webpage-to-automate/");

driver.navigate().refresh();

//Once you got the select object initialised then you can access all the methods of select class.

//Identify the select HTML element:

Thread.sleep(10000);

WebElement mySelectElement = driver.findElement(By.name("multipleselect[]"));

Select dropdown= new Select(mySelectElement);

//To deselect an option

//the deselect method will throw UnsupportedOperationException if the SELECT does not support multiple selections

dropdown.selectByValue("msagile");

Thread.sleep(2000);

dropdown.deselectByValue("msagile");

}

}

***DeselectAll* Method:**

It is to deselect all the selected options at once

Syntax:

dropdown.deselectAll( );

Sample program:

package softwareTestingMaterial;

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

import org.openqa.selenium.support.ui.Select;

import org.testng.annotations.Test;

public class SelectMethod {

@Test

public static void captureScreenMethod() throws Exception{

System.setProperty("webdriver.chrome.driver","D://Selenium Environment//Drivers//chromedriver.exe");

WebDriver driver = new ChromeDriver();

driver.manage().window().maximize();

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

driver.get("https://www.softwaretestingmaterial.com/sample-webpage-to-automate/");

driver.navigate().refresh();

//Once you got the select object initialised then you can access all the methods of select class.

//Identify the select HTML element:

Thread.sleep(10000);

WebElement mySelectElement = driver.findElement(By.name("multipleselect[]"));

Select dropdown= new Select(mySelectElement);

//To deselect an option

//the deselect method will throw UnsupportedOperationException if the SELECT does not support multiple selections

dropdown.selectByValue("msagile");

Thread.sleep(2000);

dropdown.deselectAll();

}

}

**In order to get the selected option:**

Sample program:

package softwareTestingMaterial;

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

import org.openqa.selenium.support.ui.Select;

import org.testng.annotations.Test;

public class SelectMethod {

@Test

public static void captureScreenMethod() throws Exception{

System.setProperty("webdriver.chrome.driver","D://Selenium Environment//Drivers//chromedriver.exe");

WebDriver driver = new ChromeDriver();

driver.manage().window().maximize();

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

driver.get("https://www.softwaretestingmaterial.com/sample-webpage-to-automate/");

driver.navigate().refresh();

//Once you got the select object initialised then you can access all the methods of select class.

//Identify the select HTML element:

Thread.sleep(10000);

WebElement mySelectElement = driver.findElement(By.name("multipleselect[]"));

Select dropdown= new Select(mySelectElement);

WebElement option = dropdown.getFirstSelectedOption();

System.out.println(option.getText()); //output "Selenium"

}

}

**In order to get the list of options from a dropdown:**

Sample program:

Ezoic

package softwareTestingMaterial;

import java.util.List;

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

import org.openqa.selenium.support.ui.Select;

import org.testng.annotations.Test;

public class SelectMethod {

@Test

public static void captureScreenMethod() throws Exception{

System.setProperty("webdriver.chrome.driver","D://Selenium Environment//Drivers//chromedriver.exe");

WebDriver driver = new ChromeDriver();

driver.manage().window().maximize();

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

driver.get("https://www.softwaretestingmaterial.com/sample-webpage-to-automate/");

driver.navigate().refresh();

//Once you got the select object initialised then you can access all the methods of select class.

//Identify the select HTML element:

Thread.sleep(10000);

WebElement mySelectElement = driver.findElement(By.name("multipleselect[]"));

Select dropdown= new Select(mySelectElement);

List<WebElement> options = dropdown.getOptions();

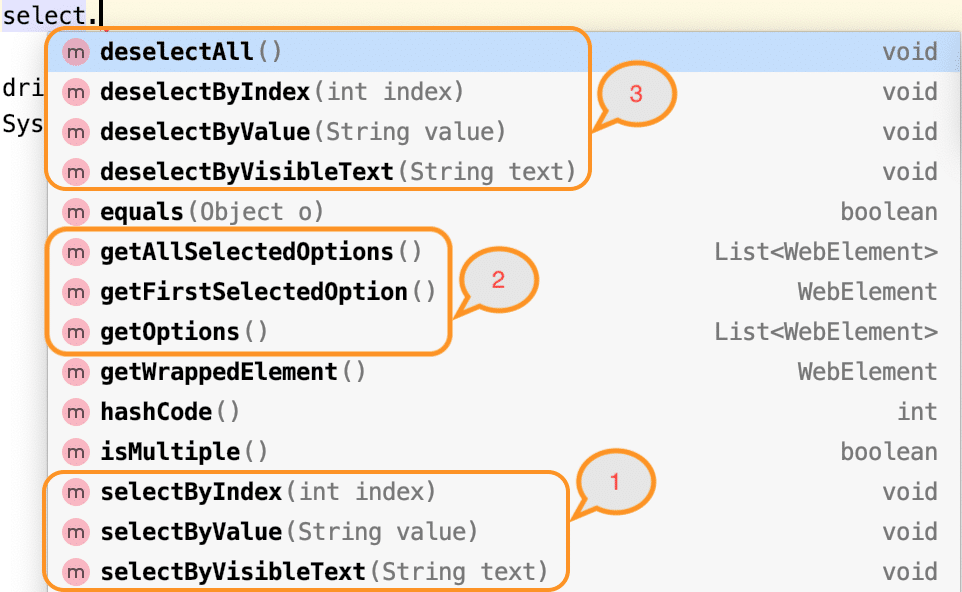
for (WebElement option : options) {

System.out.println(option.getText()); //output "Selenium", "QTP", "Manual Testing", "Automation Testing", "Performance Testing"\*/

}

}

}



***How to check whether dropdown is Multi-Select and multiselect?***

As we discussed, the Select class provides the ***"isMultiple() "*** method, which determines whether the web element in say supports multiple selections. It returns a boolean value, i.e., *True/False,* without taking any argument. It checks the attribute *'multiple'* in the *HTML* code for the web element. Consequently, it possesses the following syntax:

isMultiple(): boolean

Once you determine whether the web element is multi-select or not, you can use the Select class's various select methods on the multiple values you intend to select. The below example code shows the same-

Select oSel = new Select(driver.findElement(By.xpath(//\*[@id='cars']);

if(oSel.isMultiple()){

//Selecting multiple values by index

oSel.selectByIndex(1);

oSel.selectByIndex(2);

//Or selecting by values

oSel.selectByValue("volvo");

oSel.selectByValue("audi");

//Or selecting by visible text

oSel.selectByVisibleText("Volvo");

oSel.selectByVisibleText("Opel");

}

Alert:

To handle Browser based Alerts (Web based alert popups), we use ***Alert***Interface. The ***Alert***Interface provides some methods to handle the popups.

While running the WebDriver script, the driver control will be on the browser even after the alert generated which means the driver control will be behind the alert pop up. In order to switch the control to alert pop up, we use the following command :

driver.switchTo().alert();

Once we switch the control from browser to the alert window. We can use the ***Alert***Interface methods to do required actions such as accepting the alert, dismissing the alert, get the text from the alert window, writing some text on the alert window etc.,

Let’s see the Alert Interface Methods.

**[×](https://go.ezodn.com/ads/charity/proxy?p_id=d16123f6-d8f1-4a5b-5659-e88cb090080a&d_id=74963&imp_id=2880536464816680&c_id=1079&l_id=10016&url=https%3A%2F%2Fwww.directrelief.org%2Femergency%2Fhurricane-ian%2F&ffid=1&co=IN)**

Ezoic

We need to Import a package ***org.openqa.selenium.Alert***to handle the alerts in Selenium.

**To get a handle to the open alert:**

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

**To Click on OK button:**

alert.accept();

**To click on Cancel button**.

**[×](https://go.ezodn.com/ads/charity/proxy?p_id=d16123f6-d8f1-4a5b-5659-e88cb090080a&d_id=74963&imp_id=3081673000755120&c_id=1079&l_id=10016&url=https%3A%2F%2Fwww.directrelief.org%2Femergency%2Fhurricane-ian%2F&ffid=1&co=IN)**

Ezoic

alert.dismiss()

**To get the text which is present on the Alert**.

alert.getText();

**To enter the text into the alert box**

alert.sendkeys(String stringToSend);

**To Authenticate by passing the credentials**

alert.authenticateUsing(Credentials credentials)

**Sample Program:**

package softwareTestingMaterial;

import org.openqa.selenium.Alert;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.chrome.ChromeDriver;

import org.testng.annotations.Test;

public class AlertInterface {

@Test

public void alertWindow() throws Exception{

System.setProperty("webdriver.chrome.driver", "D:\\Selenium Environment\\Drivers\\chromedriver.exe");

WebDriver driver = new ChromeDriver();

driver.get("http://softwaretestingplace.blogspot.com/2017/03/javascript-alert-test-page.html");

driver.findElement(By.xpath("//\*[@id='content']/button")).click();

Thread.sleep(3000);

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

String print = alert.getText();

System.out.println(print);

alert.accept();

Thread.sleep(3000);

driver.findElement(By.xpath("//\*[@id='content']/button")).click();

Thread.sleep(3000);

alert.dismiss();

driver.close();

}

}

Alert example:

**package** package1;

**import** org.openqa.selenium.chrome.ChromeDriver;

**import** java.util.Iterator;

**import** java.util.Set;

**import** org.openqa.selenium.Alert;

**import** org.openqa.selenium.By;

**import** org.openqa.selenium.WebDriver;

**public** **class** alert2 {

**public** **static** **void** main(String args[]) **throws** InterruptedException {

System.*setProperty*("webdriver.chrome.driver","C:\\Users\\HP\\Downloads\\chromedriver-win64\\chromedriver.exe");

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

driver.manage().window().maximize();

// Launch Website

driver.get("https://demoqa.com/alerts");

// clicking on prompt button

driver.findElement(By.*xpath*("//\*[@id=\"confirmButton\"]")).click();

Thread.*sleep*(3000);

// accepting javascript alert

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

alert.accept();

// clicking on prompt button

driver.findElement(By.*xpath*("//\*[@id=\"confirmButton\"]")).click();

Thread.*sleep*(3000);

// Rejecting javascript alert

Alert alert1 = driver.switchTo().alert();

alert1.dismiss();

}

}

Window handle:

**package** package1;

**import** org.openqa.selenium.chrome.ChromeDriver;

**import** java.util.Iterator;

**import** java.util.Set;

**import** org.openqa.selenium.By;

**import** org.openqa.selenium.WebDriver;

**public** **class** windowHandle {

**public** **static** **void** main(String args[]) **throws** InterruptedException {

System.*setProperty*("webdriver.chrome.driver","C:\\Users\\HP\\Downloads\\chromedriver-win64\\chromedriver.exe");

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

driver.manage().window().maximize();

// Load the website

driver.get("http://seleniumpractise.blogspot.in/2017/07/multiple-window-examples.html");

// It will return the parent window name as a String

String pw=driver.getWindowHandle();

System.***out***.println(pw);

driver.findElement(By.*xpath*("//\*[@id=\"post-body-6170641642826198246\"]/a[1]")).click();

Set<String> s=driver.getWindowHandles();

**int** len=s.size();

System.***out***.println("total no of window handles is "+len);

**for**(String i:s) {

**if**(!pw.equalsIgnoreCase(i)) {

driver.switchTo().window(i);

driver.findElement(By.*xpath*("//\*[@id=\"APjFqb\"]")).sendKeys("Selenium driver");

Thread.*sleep*(3000);

driver.close();

}

}

driver.switchTo().window(pw);

System.***out***.println(driver.getWindowHandle());

}

}

Automate radio button and checkbox in selenium

When you inspect these elements via firebug and firepath you will get above html type.

The main difference between radio button and checkbox is checkbox you can select multiple but for radio button, only one selection is possible.

In Selenium we have 1 method called click() to perform click events.

This click() method you can apply with radio button, checkbox, links and sometime with dropdown as well.

Let us get started

Demo Code

WebElement ele=driver.findElement(By.id());

ele.click();

In this example I have used id only but if you want to make you script stable then you should use ***[Xpath](https://learn-automation.com/how-to-write-dynamic-xpath-in-selenium/)***and [***CSS***](https://learn-automation.com/write-dynamic-css-selector-in-selenium/)in your script.

Before performing click action, sometimes we need to verify some activity as well, take some example

* You need to verify whether radio button or checkbox is enabled.
* You need to verify whether radio button or checkbox is Displayed on UI or not.
* You need to verify whether checkbox and radio button is default selected or not.

Above validations are must used in script because automation is all about validation only. You will get these type of questions in [***interviews***](https://learn-automation.com/selenium-interview-questions-and-answers/)also.

These words looks quite big while listening but we can easily [***verify***](https://learn-automation.com/capture-error-message-in-selenium/)this using some predefined method in Selenium.

These methods are

isDisplayed();

isEnabled();

isSelected();

Therefore, you must be eager now how to use these methods in script so let us see these methods using a single program.

Program to Automate radio button and checkbox in selenium

 import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.firefox.FirefoxDriver;

public class FacebookDropdown {

     public static void main(String[] args) throws Exception {

          WebDriver driver=new FirefoxDriver();

          driver.manage().window().maximize();

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

         WebElement male\_radio\_button=driver.findElement(By.id("u\_0\_e"));

         boolean status=male\_radio\_button.isDisplayed();

         System.out.println("Male radio button is Displayed >>"+status);

          boolean enabled\_status=male\_radio\_button.isEnabled();

          System.out.println("Male radio button is Enabled >>"+enabled\_status);

        boolean selected\_status=male\_radio\_button.isSelected();

          System.out.println("Male radio button is Selected >>"+selected\_status);

          male\_radio\_button.click();

        boolean selected\_status\_new=male\_radio\_button.isSelected();

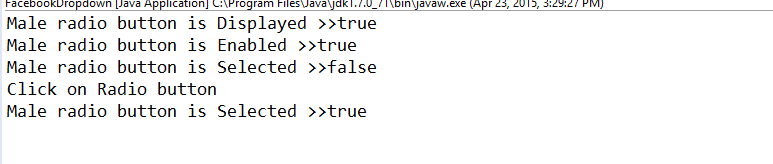
          System.out.println("Male radio button is Selected >>"+selected\_status\_new);

     }

}

**Explanation-** If you notice above scenario before click Selected status was false but after click status changed to TRUE.

**Check below image for output.**

[](https://i0.wp.com/learn-automation.com/wp-content/uploads/2015/04/radio-button-in-Selenium1.png?ssl=1)

Ex:

**package** package1;

**import** org.openqa.selenium.chrome.ChromeDriver;

**import** java.util.Iterator;

**import** java.util.List;

**import** java.util.Set;

**import** org.openqa.selenium.By;

**import** org.openqa.selenium.WebDriver;

**import** org.openqa.selenium.WebElement;

**public** **class** radio {

**public** **static** **void** main(String args[]) **throws** InterruptedException {

System.*setProperty*("webdriver.chrome.driver","C:\\Users\\HP\\Downloads\\chromedriver-win64\\chromedriver.exe");

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

driver.manage().window().maximize();

driver.get("https://seleniumpractise.blogspot.com/2016/08/how-to-automate-radio-button-in.html");

List<WebElement> l=driver.findElements(By.*xpath*("//input[@type='radio' and @name='lang']"));

System.***out***.println("the options in radio is");

**for**(WebElement i:l) {

String s=i.getAttribute("value");

System.***out***.println(s);

**if**(s.contentEquals("RUBY")) {

i.click();

}

}

List<WebElement> l1=driver.findElements(By.*xpath*("//input[@type='checkbox' and @name='lang']"));

System.***out***.println("the options in checkbox is");

**for**(WebElement i:l1) {

String s=i.getAttribute("id");

System.***out***.println(s);

**if**(s.contentEquals("code")) {

i.click();

}

i.click();

}

Thread.*sleep*(4000);

driver.close();

}

}

**Mouse Actions in Selenium WebDriver**

Mouse actions in Selenium WebDriver provide a mechanism for automating low-level elementary interactions such as mouse clicks, mouse hover, mouse button actions; as well as complex low-level interactions such as mouse hover, drag & drop, click & hold, and more.

Here is a brief list of mouse actions that are provided by the Action Class in Selenium:

* **click() method**

**click()**– Clicks on the current mouse position

**click​(WebElement web\_element)** – Clicks in the middle of the given WebElement which is passed to the method

* **doubleClick() method**

**doubleClick()** – Double clicks on the current mouse position

**doubleClick(WebElement web\_element)** – Double clicks in the middle of the given WebElement which is passed to the method

* **clickAndHold() method**

**clickAndHold()** – Clicks without releasing on the current mouse position

**clickAndHold(WebElement web\_element)** – Clicks without releasing in the middle of the WebElement which is passed to the method

* **contextClick() method**

**contextClick()** – Performs a context click operation on the current mouse position

**contextClick(WebElement web\_element)** – Performs a context click operation at the middle of the WebElement which is passed to the method

* **dragAndDrop() method**

**dragAndDrop​(WebElement source\_elem, WebElement target\_elem)** – Perform a click and hold operation in the middle of the source element (i.e. source\_elem), moves to the location of the target element (i.e. target\_elem), and release the mouse. On the successful execution of this method, the source element is dragged and dropped at the place where the target element is located

**dragAndDropBy​(WebElement elem\_source, int x\_Offset, int y\_Offset)** – Perform a click and hold operation in the middle of the source element (i.e. source\_elem) and move by a given offset available as x\_Offset and y\_Offset

[Read How To Drag And Drop In Selenium With Python?](https://www.lambdatest.com/blog/drag-and-drop-in-selenium-python/)

* **moveToElement() method**

**moveToElement​(WebElement elem\_target)** – Move the mouse to the middle of the element (i.e. elem\_target) which is passed to the method

**moveToElement​(WebElement target, int x\_Offset, int y\_Offset)** – Move to an offset (x\_Offset and y\_Offset) from the WebElement’s in center viewpoint

* **moveByOffset() method**

**moveByOffset​(int x\_Offset, int y\_Offset)** – Move the mouse from the current position by the offset which is passed to the method

* **release() method**

**release()** – Release the depressed left mouse button which is pressed at the current mouse position  
release(WebElement elem\_target) – Release the depressed left mouse button which is pressed in the middle of the given WebElement (i.e. elem\_target)  
click(), doubleClick(), contextClick(), moveToElement(), and release() are overloaded methods in the Action Class and the actions performed depends on the parameters passed to the methods.

|  |  |
| --- | --- |
| **keyDown(modifier\_key)** | Performs a modifier key press. Does not release the modifier key – subsequent interactions may assume it’s kept pressed.  **Parameters**:  modifier\_key – any of the modifier keys (Keys.ALT, Keys.SHIFT, or Keys.CONTROL) |
| **keyUp(modifier \_key)** | Performs a key release.  **Parameters**:  modifier\_key – any of the modifier keys (Keys.ALT, Keys.SHIFT, or Keys.CONTROL) |
| **moveByOffset(x-offset, y-offset)** | Moves the mouse from its current position (or 0,0) by the given offset.  **Parameters**:  x-offset- horizontal offset. A negative value means moving the mouse left.  y-offset- vertical offset. A negative value means moving the mouse down. |

Keys:

**package** package1;

**import** org.openqa.selenium.chrome.ChromeDriver;

**import** org.openqa.selenium.interactions.Actions;

**import** java.util.Iterator;

**import** java.util.Set;

**import** org.openqa.selenium.By;

**import** org.openqa.selenium.Keys;

**import** org.openqa.selenium.WebDriver;

**public** **class** KeysDemo {

**public** **static** **void** main(String[] args) **throws** InterruptedException {

// **TODO** Auto-generated method stub

System.*setProperty*("webdriver.chrome.driver","C:\\Users\\HP\\Downloads\\chromedriver-win64\\chromedriver.exe");

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

driver.manage().window().maximize();

driver.get("https://demo.guru99.com/test/newtours/");

Actions act=**new** Actions(driver);

act.keyDown(driver.findElement(By.*xpath*("/html/body/div[2]/table/tbody/tr/td[2]/table/tbody/tr[4]/td/table/tbody/tr/td[2]/table/tbody/tr[2]/td[3]/form/table/tbody/tr[4]/td/table/tbody/tr[2]/td[2]/input")),Keys.***SHIFT***).sendKeys(driver.findElement(By.*xpath*("/html/body/div[2]/table/tbody/tr/td[2]/table/tbody/tr[4]/td/table/tbody/tr/td[2]/table/tbody/tr[2]/td[3]/form/table/tbody/tr[4]/td/table/tbody/tr[2]/td[2]/input")),"Ram").perform();

Thread.*sleep*(3000);

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

driver.manage().window().maximize();

act.contextClick(driver.findElement(By.*linkText*("తెలుగు"))).sendKeys(Keys.***ARROW\_DOWN***).sendKeys(Keys.***ARROW\_DOWN***).sendKeys(Keys.***ENTER***).build().perform();

}

}

Similarly, we can use Keys enum for different non-text keys and pass them to the sendKeys method. The following table has an entry for each of the non-text key present in a keyboard.

|  |  |
| --- | --- |
| **Keyboard’s Key** | **Keys enum’s value** |
| Arrow Key – Down | Keys.ARROW\_DOWN |
| Arrow Key – Up | Keys.ARROW\_UP |
| Arrow Key – Left | Keys.ARROW\_LEFT |
| Arrow Key – Right | Keys.ARROW\_RIGHT |
| Backspace | Keys.BACK\_SPACE |
| Ctrl Key | Keys.CONTROL |
| Alt key | Keys.ALT |
| DELETE | Keys.DELETE |
| Enter Key | Keys.ENTER |
| Shift Key | Keys.SHIFT |
| Spacebar | Keys.SPACE |
| Tab Key | Keys.TAB |
| Equals Key | Keys.EQUALS |
| Esc Key | Keys.ESCAPE |
| Home Key | Keys.HOME |
| Insert Key | Keys.INSERT |
| PgUp Key | Keys.PAGE\_UP |
| PgDn Key | Keys.PAGE\_DOWN |
| Function Key F1 | Keys.F1 |
| Function Key F2 | Keys.F2 |
| Function Key F3 | Keys.F3 |
| Function Key F4 | Keys.F4 |
| Function Key F5 | Keys.F5 |
| Function Key F6 | Keys.F6 |
| Function Key F7 | Keys.F7 |
| Function Key F8 | Keys.F8 |
| Function Key F9 | Keys.F9 |
| Function Key F10 | Keys.F10 |
| Function Key F11 | Keys.F11 |
| Function Key F12 | Keys.F12 |

2)

**package** package1;

**import** org.openqa.selenium.chrome.ChromeDriver;

**import** org.openqa.selenium.interactions.Actions;

**import** java.util.Iterator;

**import** java.util.Set;

**import** org.openqa.selenium.By;

**import** org.openqa.selenium.WebDriver;

**public** **class** contextclick {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

System.*setProperty*("webdriver.chrome.driver","C:\\Users\\HP\\Downloads\\chromedriver-win64\\chromedriver.exe");

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

driver.manage().window().maximize();

driver.get("https://demo.guru99.com/test/newtours/");

Actions act=**new** Actions(driver);

act.contextClick(driver.findElement(By.*linkText*("Home"))).perform();

}

}

3)

**package** package1;

**import** org.openqa.selenium.chrome.ChromeDriver;

**import** org.openqa.selenium.interactions.Actions;

**import** java.util.Iterator;

**import** java.util.Set;

**import** org.openqa.selenium.By;

**import** org.openqa.selenium.WebDriver;

**public** **class** click {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

System.*setProperty*("webdriver.chrome.driver","C:\\Users\\HP\\Downloads\\chromedriver-win64\\chromedriver.exe");

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

driver.manage().window().maximize();

driver.get("https://demo.guru99.com/test/newtours/");

Actions act=**new** Actions(driver);

act.click(driver.findElement(By.*linkText*("Home"))).perform();

}

}

Frames:

**package** package1;

**import** java.util.List;

**import** java.util.concurrent.TimeUnit;

**import** org.openqa.selenium.By;

**import** org.openqa.selenium.WebDriver;

**import** org.openqa.selenium.WebElement;

**import** org.openqa.selenium.chrome.ChromeDriver;

**public** **class** Frames {

**public** **static** **void** main(String[] args) **throws** InterruptedException {

// **TODO** Auto-generated method stub

System.*setProperty*("webdriver.chrome.driver","C:\\Users\\HP\\Downloads\\chromedriver-win64\\chromedriver.exe");

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

driver.manage().window().maximize();

driver.get("https://demo.guru99.com/test/guru99home/");

// navigates to the page consisting an iframe

String pw=driver.getWindowHandle();

driver.manage().window().maximize();

driver.switchTo().frame("a077aa5e"); //switching the frame by ID

System.***out***.println("\*\*\*\*\*\*\*\*We are switch to the iframe\*\*\*\*\*\*\*");

driver.findElement(By.*xpath*("html/body/a/img")).click();

//Clicks the iframe

System.***out***.println("\*\*\*\*\*\*\*\*\*We are done\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

driver.switchTo().window(pw);

Thread.*sleep*(3000);

driver.switchTo().defaultContent();

// driver.manage().timeouts().implicitlyWait(100, TimeUnit.SECONDS);

**int** size = driver.findElements(By.*tagName*("iframe")).size();

**try** {

**for**(**int** i=0; i<=size; i++){

driver.switchTo().frame(i);

**int** total=driver.findElements(By.*xpath*("html/body/a/img")).size();

System.***out***.println(total);

driver.switchTo().parentFrame();

}

}**catch**(Exception e) {

System.***out***.println("no frame");

}

}

}

Screenshots:

**package** package1;

**import** java.io.File;

**import** java.io.IOException;

**import** org.apache.commons.io.FileUtils;

**import** org.openqa.selenium.By;

**import** org.openqa.selenium.OutputType;

**import** org.openqa.selenium.TakesScreenshot;

**import** org.openqa.selenium.WebDriver;

**import** org.openqa.selenium.chrome.ChromeDriver;

**public** **class** Screenshotdemo {

**void** screenshot() **throws** IOException {

System.*setProperty*("webdriver.chrome.driver","C:\\Users\\HP\\Downloads\\chromedriver-win64\\chromedriver.exe");

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

driver.manage().window().maximize();

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

driver.findElement(By.*xpath*(".//\*[@id='email']")).sendKeys("Learn automation");

TakesScreenshot ts=(TakesScreenshot)driver;

File source=ts.getScreenshotAs(OutputType.***FILE***);

FileUtils.*copyFile*(source,**new** File("./screenshots/facebook.png"));

driver.quit();

}

**public** **static** **void** main(String args[]) **throws** IOException {

Screenshotdemo o=**new** Screenshotdemo();

o.screenshot();

}

}

Implicitly Wait():

**package** package1;

**import** java.util.concurrent.TimeUnit;

**import** org.openqa.selenium.By;

**import** org.openqa.selenium.WebDriver;

**import** org.openqa.selenium.WebElement;

**import** org.openqa.selenium.chrome.ChromeDriver;

**public** **class** ImplicitWait {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

System.*setProperty*("webdriver.chrome.driver","C:\\Users\\HP\\Downloads\\chromedriver-win64\\chromedriver.exe");

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

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

String eTitle = "Demo Guru99 Page";

String aTitle = "" ;

// launch Chrome and redirect it to the Base URL

driver.get("http://demo.guru99.com/test/guru99home/" );

//Maximizes the browser window

driver.manage().window().maximize() ;

//get the actual value of the title

aTitle = driver.getTitle();

//compare the actual title with the expected title

**if** (aTitle.equals(eTitle))

{

System.***out***.println( "Test Passed") ;

}

**else** {

System.***out***.println( "Test Failed" );

}

//close browser

driver.close();

}

}

Wait:

driver.manage().timeouts().pageLoadTimeout(30, TimeUnit.***SECONDS***);

driver.manage().timeouts().setScriptTimeout(30, TimeUnit.***SECONDS***);

driver.manage().timeouts().scriptTimeout(30,TimeUnit.***SECONDS***);

example2:

**package** package1;

**import** java.util.concurrent.TimeUnit;

**import** org.openqa.selenium.By;

**import** org.openqa.selenium.WebDriver;

**import** org.openqa.selenium.chrome.ChromeDriver;

**public** **class** boostrappopups {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

System.*setProperty*("webdriver.chrome.driver","C:\\Users\\HP\\Downloads\\chromedriver-win64\\chromedriver.exe");

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

driver.manage().window().maximize();

driver.get("https://www.goibibo.com/flights/?utm\_source=google&utm\_medium=cpc&utm\_campaign=DF-Brand-EM&utm\_adgroup=Only%20Goibibo&utm\_term=!SEM!DF!G!Brand!RSA!108599293!6504095653!617695092667!e!goibibo!c!&gad\_source=1&gclid=Cj0KCQjwwO20BhCJARIsAAnTIVRwxOLlpDgCMQDFiiTBE5Tumjul9LzoaD1w6IpxlQZTE9zFIlDUXMsaAvQOEALw\_wcB");

driver.manage().timeouts().implicitlyWait(30, TimeUnit.***SECONDS***);

driver.findElement(By.*xpath*("//\*[@id=\"root\"]/div[1]/div/header/div/div/div")).click();

driver.manage().timeouts().implicitlyWait(30, TimeUnit.***SECONDS***);

driver.findElement(By.*xpath*("//input[@name='phone']")).sendKeys("1234567890");

driver.manage().timeouts().implicitlyWait(30, TimeUnit.***SECONDS***);

driver.findElement(By.*xpath*("/html/body/div[2]/div[2]/div/div/div[2]/span/span")).click();

}

}

Boosttrap popouts handle:

**package** package1;

**import** java.util.concurrent.TimeUnit;

**import** org.openqa.selenium.By;

**import** org.openqa.selenium.WebDriver;

**import** org.openqa.selenium.chrome.ChromeDriver;

**public** **class** boostrappopups {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

System.*setProperty*("webdriver.chrome.driver","C:\\Users\\HP\\Downloads\\chromedriver-win64\\chromedriver.exe");

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

driver.manage().window().maximize();

driver.get("https://www.goibibo.com/flights/?utm\_source=google&utm\_medium=cpc&utm\_campaign=DF-Brand-EM&utm\_adgroup=Only%20Goibibo&utm\_term=!SEM!DF!G!Brand!RSA!108599293!6504095653!617695092667!e!goibibo!c!&gad\_source=1&gclid=Cj0KCQjwwO20BhCJARIsAAnTIVRwxOLlpDgCMQDFiiTBE5Tumjul9LzoaD1w6IpxlQZTE9zFIlDUXMsaAvQOEALw\_wcB");

driver.manage().timeouts().implicitlyWait(30, TimeUnit.***SECONDS***);

driver.findElement(By.*xpath*("//\*[@id=\"root\"]/div[1]/div/header/div/div/div")).click();

driver.manage().timeouts().implicitlyWait(30, TimeUnit.***SECONDS***);

driver.findElement(By.*xpath*("//input[@name='phone']")).sendKeys("1234567890");

driver.manage().timeouts().implicitlyWait(30, TimeUnit.***SECONDS***);

driver.findElement(By.*xpath*("/html/body/div[2]/div[2]/div/div/div[2]/span/span")).click();

}

}

ExplicitWait:

**package** package1;

**import** java.time.Duration;

**import** java.util.concurrent.TimeUnit;

**import** org.openqa.selenium.By;

**import** org.openqa.selenium.WebDriver;

**import** org.openqa.selenium.WebElement;

**import** org.openqa.selenium.chrome.ChromeDriver;

**import** org.openqa.selenium.support.ui.ExpectedConditions;

**import** org.openqa.selenium.support.ui.WebDriverWait;

**public** **class** DemoExplicit {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

System.*setProperty*("webdriver.chrome.driver","C:\\Users\\HP\\Downloads\\chromedriver-win64\\chromedriver.exe");

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

driver.manage().window().maximize();

driver.manage().timeouts().pageLoadTimeout(30, TimeUnit.***SECONDS***);

driver.get("https://seleniumpractise.blogspot.com/2016/08/how-to-use-explicit-wait-in-selenium.html");

driver.findElement(By.*xpath*("//button")).click();

WebDriverWait d = **new** WebDriverWait(driver, Duration.*ofSeconds*(7000));

d.until(ExpectedConditions.*visibilityOfElementLocated*(By.*xpath*("//p[text()='WebDriver']")));

WebElement o=driver.findElement(By.*xpath*("//p[text()='WebDriver']"));

**if**(o.isDisplayed()) {

System.***out***.println("pass");

}**else** {

System.***out***.println("fail");

}

}

}

In order to declare explicit wait, one has to use “ExpectedConditions”. The following Expected Conditions can be used in Explicit Wait.

alertIsPresent()

elementSelectionStateToBe()

elementToBeClickable()

elementToBeSelected()

frameToBeAvaliableAndSwitchToIt()

invisibilityOfTheElementLocated()

invisibilityOfElementWithText()

presenceOfAllElementsLocatedBy()

presenceOfElementLocated()

textToBePresentInElement()

textToBePresentInElementLocated()

textToBePresentInElementValue()

titleIs()

titleContains()

visibilityOf()

visibilityOfAllElements()

visibilityOfAllElementsLocatedBy()

visibilityOfElementLocated()

**Syntax:**

WebDriverWait wait= new WebDriverWait(driver, 5);

        WebElement ele=wait.until(ExpectedConditions.visibilityOfElementLocated(By.xpath(“xpath")));

**Headless Browser Testing Using HTMLUnitDriver**  
HTML UnitDriver is the most light weight and fastest implementation headless browser for of WebDriver. It is based on HtmlUnit. It is known as Headless Browser Driver. It is same as Chrome, IE, or FireFox driver, but it does not have GUI so one cannot see the test execution on screen.  
**Steps:**  
1. You need Eclipse with Selenium installed  
2. Download the HTMLUnit Driver. Add the jar to your project  
https://github.com/SeleniumHQ/htmlunit-driver/releases  
  
**Program:**

**import** **org.openqa.selenium.By**;

**import** **org.openqa.selenium.WebDriver**;

**import** **org.openqa.selenium.htmlunit.HtmlUnitDriver**;

**import** **org.testng.annotations.Test**;

**public** **class** **HTMLUnitTest** {

**@Test**

**public** **void** **headLessBrowserTest**() {

WebDriver driver = **new** HtmlUnitDriver();

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

driver.findElement(By.name("q")).sendKeys("Test");

String titleString=driver.getTitle();

System.out.println("Page title is: "+titleString);

String urlString=driver.getCurrentUrl();

System.out.println("Page URL is: "+urlString);

driver.quit();

}

}

**Output:**

Page title **is:** Google

Page URL **is:** **https:**//www.google.com/

**PASSED:** headLessBrowserTest

Write a program which will capture System Date/ Time

In this example I am taking time as well so I have added HH:MM:SS also in format

package demo;

import java.text.DateFormat;

import java.text.SimpleDateFormat;

import java.util.Date;

public class GetDateinJava {

public static void main(String[] args) {

// Create object of SimpleDateFormat class and decide the format

DateFormat dateFormat = new SimpleDateFormat("MM/dd/yyyy HH:mm:ss");

//get current date time with Date()

Date date = new Date();

// Now format the date

String date1= dateFormat.format(date);

// Print the Date

System.out.println("Current date and time is " +date1);

}

}

Calender:

package package1;

// TODO Auto-generated method stub

import java.util.List;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.firefox.FirefoxDriver;

public class Calendar {

public static void main(String[] args)

{

// Set the driver path

System.setProperty("webdriver.firefox.marionette","G:\\Selenium\\Firefox driver\\geckodriver.exe");

// start firefox

WebDriver driver=new FirefoxDriver();

// start application

driver.get("http://seleniumpractise.blogspot.com/2016/08/how-to-handle-calendar-in-selenium.html");

// click on date picker so that we can get the calendar in view

driver.findElement(By.id("datepicker")).click();

// this will find all matching nodes in calendar

//List<WebElement> allDates=driver.findElements(By.xpath("//table[@class='ui-datepicker-calendar']//td"));

// Click on date 28 or any other date of your choice

driver.findElement(By.xpath("//a[text()='28']")).click();

}

}

Javascriptexecutor:

**package** package1;

**import** org.openqa.selenium.WebDriver;

**import** org.openqa.selenium.chrome.ChromeDriver;

**import** org.openqa.selenium.JavascriptExecutor;

**public** **class** jsexecutor {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

System.*setProperty*("webdriver.chrome.driver","C:\\Users\\HP\\Downloads\\chromedriver-win64\\chromedriver.exe");

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

driver.manage().window().maximize();

JavascriptExecutor js = (JavascriptExecutor)driver;

driver.manage().window().maximize();

driver.get("https://www.browserstack.com/users/sign\_in");

js.executeScript("document.getElementById('user\_email\_login').value='rbc@xyz.com';");

js.executeScript("document.getElementById('user\_password').value='password';");

//js.executeScript("document.getElementById('user\_submit').click();");

//js.executeScript("alert('enter correct login credentials to continue');");

js.executeScript("window.scrollBy(0,200)");

}

}

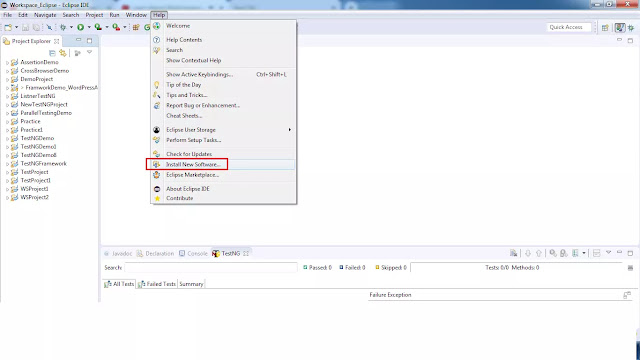
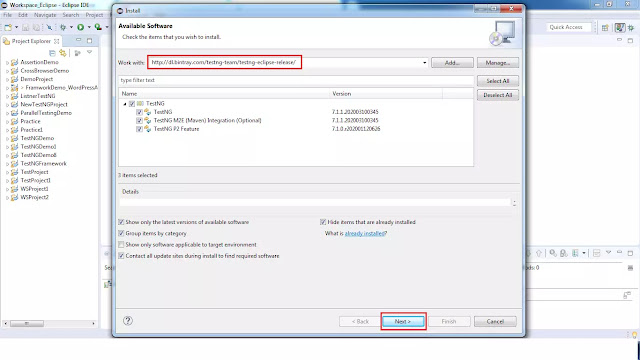
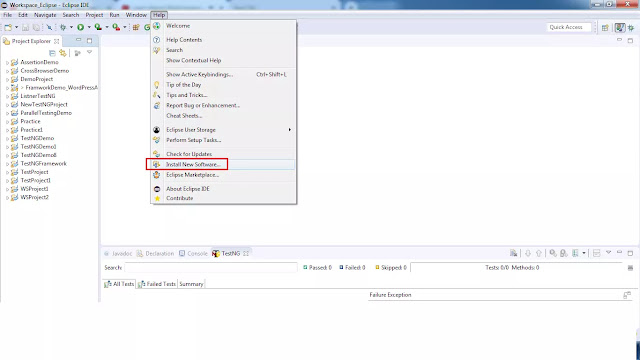
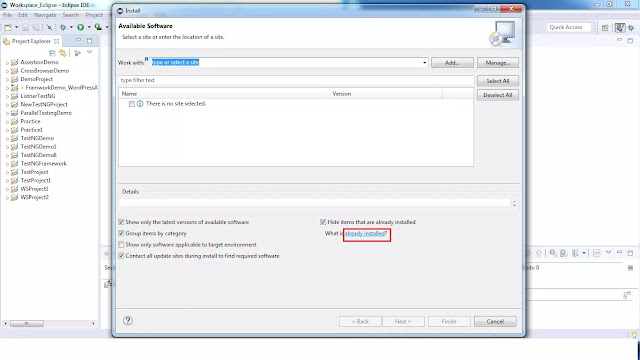
**What is TestNG?**

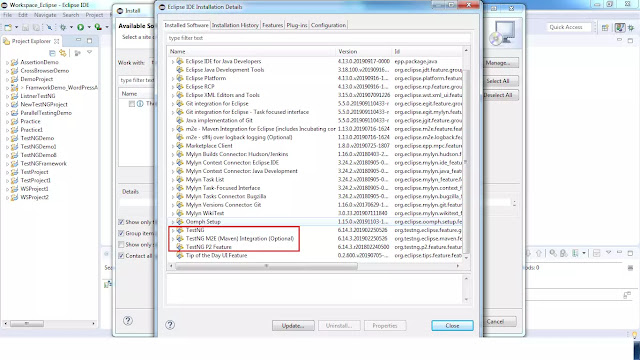
* TestNG is an automation testing framework.
* NG stands for "Next Generation".
* Java unit testing framework.
* TestNG is an advance framework designed in a way to leverage the benefits by both the developers and testers.
* TestNG is a testing framework inspired from JUnit and NUnit but introducing some new functionalities that make it more powerful and easier to use.

**TestNG Features:**

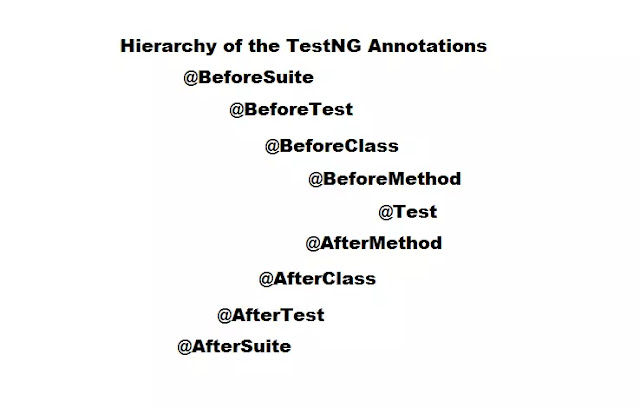
* TestNG simplifies the way the tests are coded. There is no more need for a static main method in our tests.
* Support for annotations (@).
* Using TestNG you can generate a proper report  and can see passed, failed, skipped test results.
* Test case Prioritization.
* Support for Data Driven Testing using Data providers.
* Multiple test cases can be grouped more easily by converting them into testng.xml file.
* The same test case can be executed multiple times without loops just by using keyword called 'invocation count.'.
* Cross browser testing.
* Parallel Testing is possible using TestNG.

**Advantages of TestNG over Junit:**

* There are three major advantages of TestNG over Junit.
* More Annotations than JUnit and Annotations are easier to understand
* Test cases can be grouped more easily.
* Parallel testing is possible.
* The testing framework can be easily integrated with tools like Maven, Jenkins, etc.
* Install TestNG step by step
* There are two ways you can install TestNG in Eclipse.
* First Way on installing Eclipse is using "Install new software" option.
* Second way is using "Eclipse Market Place". - This option will be available in new versions of eclipse.
* Lets have a look at the first version:
* **Step 1:**
* In Eclipse, on top menu bar, Under Help Menu, Click on "Install new Software" in help window.
* [](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEjhWqd1FNHanIPEYYT1Eenx20ft7uwoBwL0zlvGrTRcmJaHySsLikmFLZVOAb22rr1wM00Tf3cfMwZUr8UQP14mSztC9le2HRKtDC06ntH8yTJmZFLv_e0jPjq8-Ad6F8UJaoR6VRR836sJ/s1600/testng1.webp)
* **Step 2:**
* Enter the URL (http://dl.bintray.com/testng-team/testng-eclipse-release/) at Work With field and click on "Add" button. Add the URL and later click on next.
* [](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEjH6LdFuCcgT0cHYEmvdj0ybaR6ppZ0CTCBSA6msFT5sxQV7xhEotCqCD2Io4dRSaTQuWUoRnRwii7ftnayhOOl1EgIrSBDz1KhUD2NAw6lj4WUT8XNYESXwNt7nRmLGFHZIpveMzjpsfCw/s1600/testng2.webp)
* **Step 3:**
* It will check for the requirement and dependencies before starting the installation. Once the above step is done, it will ask you to review the installation details. If your are ready or OK to install TestNG, click on "Next" to continue..Later Accept the Terms of the license agreement and Click on "Finish" button. If there is any problem with the requirements/dependencies, it will ask you to install them first before continuing with TestNG. Most of the cases it will successfully get installed nothing to worry about it.
* It will take few minutes to get installed.
* Verify TestNG installed or not?
* **Step 4:**In Eclipse, on top menu bar, Under Help Menu, Click on "Install new Software" in help window.
* [](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEi_-O4X-XfR4-JWQOtpIlxvNX4ZvVeExmAGlI6oZJuAFb2v07flYLWzDSgCufEaTfnKtwMcjg3r4xB8oDbDohNPNrbt4yIHlNI2EFH6MgTUNWvleSuoiRnVFj5n6wwTuFhyha7KZjD8-Jo-/s1600/testng1.webp)
* **Step 5:**Click on already installed?
* [](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEjnEI6-8Fa-Nd_yNPtllciSPyxHvicu1Cp9rhmWgHYg8EMqXYnbAgM3xxjYi6QUR4UAfOqXRa3gywnYrS-ATqFR1y_rhonZ2jMP14feiZI2Fjzo7p1z4i5-zDX9d0o9fg6vQzHuNz4SCmng/s1600/testng3.webp)
* **Step 6:**You can verify if the TestNG is available as per the below screen shot.

[](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEhEvWC5XC0vUmDtAbNR7K_K-aWWxXI4stuhXuY9HAazHDM5Oc3TOT8fQuunDOgio40d_iJpQRcDUWUIEjkZlHqOz_-iGEzGqR6A9EtLy-pJc0If1srWyh6IXbqwD741WsuaDunfhnuYlj7d/s1600/testng4.webp)

**List of TestNG Annotations**  
 Below are the different annotations used in TestNG  
  
@BeforeSuite: The annotated method will be run only once before all tests in this suite have run.  
  
@AfterSuite: The annotated method will be run only once after all tests in this suite have run.  
  
@BeforeTest: The annotated method will be run before any test method belonging to the classes inside the <test> tag is run.  
  
@AfterTest: The annotated method will be run after all the test methods belonging to the classes inside the <test> tag have run.  
  
@BeforeClass: The annotated method will be run only once before the first test method in the current class is invoked.  
  
@AfterClass: The annotated method will be run only once after all the test methods in the current class have run.  
  
@BeforeMethod: The annotated method will be run before each test method.  
  
@AfterMethod: The annotated method will be run after each test method.  
  
@BeforeGroups - The @BeforeGroups annotated method run only once for a group before the execution of all test cases belonging to that group.  
  
@AfterGroups - The @AfterGroups annotated method run only once for a group after the execution of all test cases belonging to that group.

**Different Attributes in TestNG**

**1. priority attribute :**  
  
In TestNG "Priority" is used to schedule the test cases. When there are multiple test cases, we want to execute test cases in order. Like First we need to execute a test case "Registration" before login.  
In order to achive, we use need to add annotation as @Test(priority=??). The default value will be zero for priority.  
If you don't mention the priority, it will take all the test cases as "priority=0" and execute  
alphabetically.  
Lower priorities will be scheduled first.  
**For example** if you give the priority like @Test(priority=0), @Test(priority=-1), @Test(priority=1), @Test(priority=2)  
Then test will be executed in following order: @Test(priority=-1), @Test(priority=0), @Test(priority=1), @Test(priority=2)  
  
**2. description attribute:** gives information about the test it is attached to.  
**Example:**  
@Test(description='Title Test')  
public void titleTest(){  
   String actualTitle = driver.getTitle();  
   String expectedTitle = "OrangeHRM";  
   Assert.assertEquals(actualTitle, expectedTitle);  
}  
  
**3. timeOut:** maximum number of milliseconds for a test run.  
**Example:**5000 milliseconds (5 seconds) time allotted to run below test method.  
@Test(timeOut=5000)  
public void testCase(){  
  
}

1. **dependsOnMethods:** specify when you want to run a test, only after another test has run successfully, making the second test’s run dependent on the first test’s successful outcome.  
   **Example:**  
   @Test(priority = 1)  
    public void testcase3() {  
     String actualUrl = driver.getCurrentUrl();  
     String expectedURL = "https://opensource-demo.orangehrmlive.com/index.php/dashboard";  
     Assert.assertEquals(actualUrl, expectedURL);  
    }  
   @Test(priority = 2, dependsOnMethods = "testcase3")  
     public void testcase4() {  
     String actualTitle = driver.getTitle();  
     String expectedTitle = "OrangeHRM";  
     Assert.assertEquals(actualTitle, expectedTitle);  
    }  
   **5. enabled:** this attribute has boolean values, and by default is ‘true’. Only worth specifying explicitly when you don’t want a certain test method or class to be run, by setting the attribute to false.  
   **Example:** Below test case will not be executed as specified enabled = false.  
   @Test(enabled = false)  
     public void testcase4() {  
     String actualTitle = driver.getTitle();  
     String expectedTitle = "OrangeHRM";  
     Assert.assertEquals(actualTitle, expectedTitle);  
    }  
   **6. groups:** useful for grouping together tests that relate to the same functionality, are of the same importance or are of the same type.  
     
   **7. alwaysRun**- If set to true, this test method will always be run even if it depends on a method that failed.  
   **Example:** Below testcase4 will always execute even it depends on a method that fails (testcase3)  
     
   @Test(priority = 1)  
    public void testcase3() {  
     String actualUrl = driver.getCurrentUrl();  
     String expectedURL = "https://opensource-demo.orangehrmlive.com/index.php/dashboard";  
     Assert.assertEquals(actualUrl, expectedURL);  
    }  
     
   @Test(priority = 2, dependsOnMethods = "testcase3", alwaysRun = true)  
     public void testcase4() {  
     String actualTitle = driver.getTitle();  
     String expectedTitle = "OrangeHRM";  
     Assert.assertEquals(actualTitle, expectedTitle);  
    }  
   **8. invocationcount** - The number of times this method should be invoked.  
   **Example:** Below testcase3 will execute 3 times.  
   @Test(priority = 4, description = "Verify URL", invocationCount = 3)  
     public void testcase3() {  
     String actualUrl = driver.getCurrentUrl();  
     String expectedURL = "https://opensource-demo.orangehrmlive.com/index.php/dashboard";  
     Assert.assertEquals(actualUrl, expectedURL);  
    }

**9. How to ignore test case** - We can ignore test at test, method and package levels  
**Example:**  
**1. Test Level** - Below test method will be ignored as @Ignore specified at Test Level  
@Ignore  
@Test(priority = 4, description = "Verify URL", invocationCount = 3)  
  public void testcase3() {  
  String actualUrl = driver.getCurrentUrl();  
  String expectedURL = "https://opensource-demo.orangehrmlive.com/index.php/dashboard";  
  Assert.assertEquals(actualUrl, expectedURL);  
 }  
  
**2. Class Level** - All test methods will be ignored for this classe as @Ignore specified at class Level  
  
@Ignore  
public class OrangeHRMTest1 {  
 WebDriver driver;  
  
  @Test(priority = 1)  
  public void testcase1() {  
  driver.get("https://opensource-demo.orangehrmlive.com/index.php/auth/login");  
  boolean img = driver.findElement(By.xpath("//\*[@id=\"divLogo\"]/img")).isDisplayed();  
  Assert.assertTrue(img);  
 }  
  
  @Test(priority = 2)  
  public void testcase2() {  
  String actualTitle = driver.getTitle();  
  String expectedTitle = "OrangeHRM";  
  Assert.assertEquals(actualTitle, expectedTitle);  
 }  
   
}  
  
**3. Package Level** - All test classes will be ignored under this package  
Example  
@org.testng.annotations.Ignore  
package testNGPackage;  
  
**10. @Test annotation at Class Level** - TestNG has great feature to define annotations on a class instead of a each test method.  
If say suppose there are 10 test methods, where adding @Test on a class level is simpler than adding @Test for each method.  
When we make class level @Test annotation, all the public methods of this class will become test methods even if they are not annotated.  
We can still define @Test annotation on of the method if we want to add any attributes to particular test method.  
  
**Example:**  
  
@Test  
public class OrangeHRMTest {  
 WebDriver driver;  
  
 @BeforeClass  
 public void setup() {  
 System.setProperty("webdriver.chrome.driver",  
 "C:\\Users\\Hitendra\\Downloads\\chromedriver\_win32\\chromedriver.exe");  
 driver = new ChromeDriver();  
 driver.manage().window().maximize();  
 }  
public void testCase1(priority=1) {  
 }  
public void testCase2(priority=2) {  
 }  
}