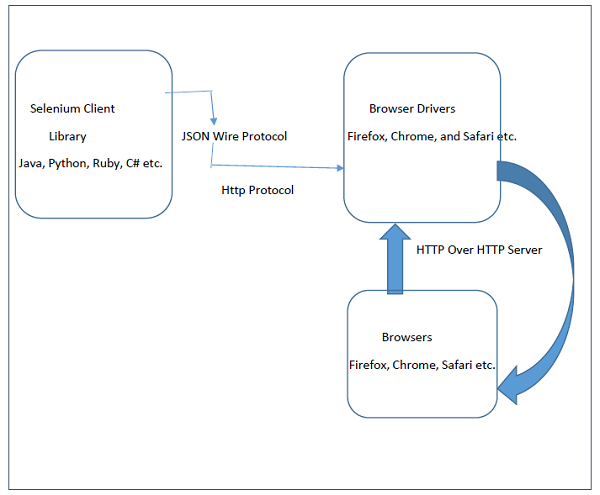
**What is the Selenium Web Driver Architecture?**

Selenium Web Driver architecture in a simplified diagram is described below:



Let us now understand the Selenium Web Driver Architecture. Selenium WebDriver API enables interaction between browsers and browser drivers. This architecture consists of four layers namely the Selenium Client Library, JSON Wire Protocol, Browser Drivers and Browsers.

* Selenium Client Library consists of languages like Java, Ruby, Python, C# and so on. After the test cases are triggered, entire Selenium code will be converted to Json format.
* JSON stands for Javascript Object Notation. It takes up the task of transferring information from the server to the client. JSON Wire Protocol is primarily responsible for transfer of data between HTTP servers. Generated Json is made available to browser drivers through http Protocol.
* Each browser has a specific browser driver. Browser drivers interact with its respective browsers and execute the commands by interpreting Json which they received from the browser. As soon as the browser driver gets any instructions, they run them on the browser. Then the response is given back in the form of HTTP response.

Let’s consider the following block of code −

WebDriver driver = new ChromeDriver();

driver.get ([“https://www.tutorialspoint.com/index.htm“](https://www.tutorialspoint.com/index.htm));

Once we run this block of code, the entire code will be converted with the help of JSON Wire Protocol over HTTP as a URL. The converted URL will be fed to the ChromeDriver.

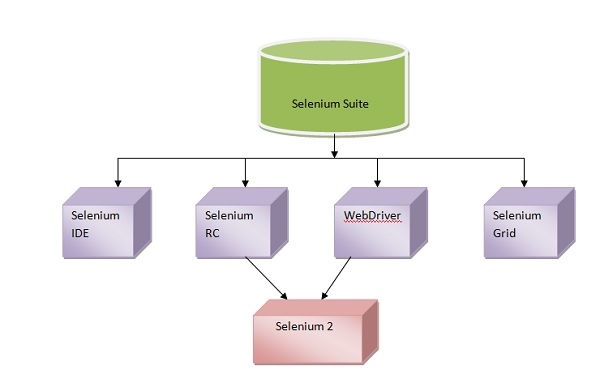
The browser driver utilizes HTTP server to get the request from HTTP. As the browser driver gets the URL, it passes the request to its browser via HTTP. It will trigger the event of executing the Selenium instructions on the browser.

Now if the request is that of POST, it will trigger an action on the browser. If it’s a GET request, then the response will be produced at the browser end. Finally it will be passed over HTTP to the browser driver. The browser driver will in turn send it to the UI via JSON Wire Protocol.

# What is selenium web driver?

Selenium Webdriver is a framework that allows automation testing. It allows testing across various browsers. It can execute multiple tests over multiple browsers on multiple OS. Web Driver makes it possible to write a test script in Linux and run it in Windows. There are multiple programming languages that are supported by Web Driver such as Java, Python, Ruby, .Net, PHP to create test scripts.

Selenium Web driver is derived from −



In Selenium2, integration of Web Driver was considered which was designed to address a few limitations of Selenium RC.

Selenium was introduced with the following new features −

It can test dynamic websites where the content of pages changes by a mouse click.

By Web Driver, mobile automation is also feasible.

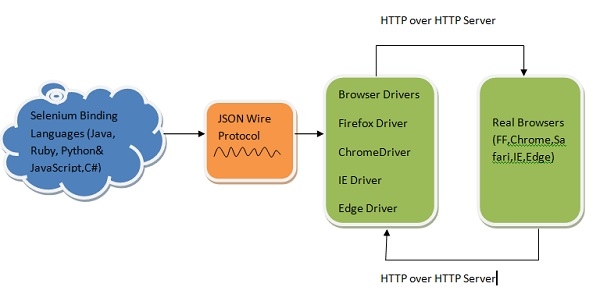
Web driver comes with another advantage which is its use on HTMLUnit browsers(HTMLUnit browser are headless browsers which means these are invisible to the user, in simple words they have no GUI), due to these testing on HTMLUnit browser is faster as these browsers save the time needed to load the page elements. This results in less execution time for test cases.

Browser vendors will ship their own WebDriver implementation, hence are tightly coupled to the browser giving a better testing experience. Modern Browsers such as Edge, Safari are introduced with the WebDrivers shipped by their vendors

Let's have a look at how Selenium WebDriver work −

WebDriver makes direct calls to the browser using each browser’s native support for automation. It uses browser driver for this communication

WebDriver contributes its object-oriented API for Document Object Model (DOM) interaction and browser control.



When any Automated Script is executed, for every Selenium command, HTTP Request is created and sent to the browser to the driver. The browser driver uses an HTTP Server for getting HTTP Requests. This HTTP Server determines the steps needed for the implementation of the Selenium command. The execution status which is run on the browser is sent back to HTTP Server, which sends the status back to the automation script.

# First Selenium Webdriver Script: JAVA Sample Code Example

package newproject;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.firefox.FirefoxDriver;

//comment the above line and uncomment below line to use Chrome

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

public class PG1 {

public static void main(String[] args) {

// declaration and instantiation of objects/variables

System.setProperty("webdriver.gecko.driver","C:\\geckodriver.exe");

WebDriver driver = new FirefoxDriver();

//comment the above 2 lines and uncomment below 2 lines to use Chrome

//System.setProperty("webdriver.chrome.driver","G:\\chromedriver.exe");

//WebDriver driver = new ChromeDriver();

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();

}

}

**Explaining the code**

**Importing Packages**

To get started, you need to import following two packages:

1. **org.openqa.selenium.\***– contains the WebDriver class needed to instantiate a new browser loaded with a specific driver
2. **org.openqa.selenium.firefox.FirefoxDriver**– contains the FirefoxDriver class needed to instantiate a Firefox-specific driver onto the browser instantiated by the WebDriver class

If your test needs more complicated actions such as accessing another class, taking browser screenshots, or manipulating external files, definitely you will need to import more packages.

## **Instantiating objects and variables**

Normally, this is how a driver object is instantiated.

First Selenium Webdriver Script: JAVA Code Example

## **Launching a Browser Session**

WebDriver’s **get()** method is used to launch a new browser session and directs it to the URL that you specify as its parameter.

First Selenium Webdriver Script: JAVA Code Example

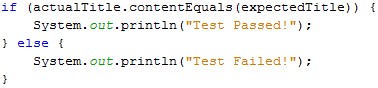
## **Get the Actual Page Title**

The WebDriver class has the **getTitle()** method that is always used to obtain the page title of the currently loaded page.

First Selenium Webdriver Script: JAVA Code Example

## **Compare the Expected and Actual Values**

This portion of the code simply uses a basic Java if-else structure to compare the actual title with the expected one.



## **Terminating a Browser Session**

The “**close()**” method is used to close the browser window.

First Selenium Webdriver Script: JAVA Code Example

## **Terminating the Entire Program**

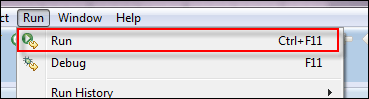
If you use this command without closing all browser windows first, your whole Java program will end while leaving the browser window open.

First Selenium Webdriver Script: JAVA Code Example

## **Running the Test**

There are two ways to execute code in Eclipse IDE.

1. On Eclipse’s menu bar, click **Run > Run.**
2. Press **Ctrl+F11** to run the entire code.



# Locators in Selenium IDE: CSS Selector | DOM | XPath | ID

## **What are Locators?**

Locator is a command that tells Selenium IDE which GUI elements ( say Text Box, Buttons, Check Boxes etc) its needs to operate on.  Identification of correct GUI elements is a prerequisite to creating an automation script. But accurate identification of GUI elements is more difficult than it sounds. Sometimes, you end up working with incorrect GUI elements or no elements at all!  Hence, Selenium provides a number of Locators to precisely locate a GUI element

The different types of CSS Locator in Selenium IDE are:

## **Locating by ID**

This is the most common way of locating elements since ID’s are supposed to be unique for each element.

**Target Format:**id=id of the element

## **Locating by Name**

Locating elements by name are very similar to locating by ID, except that we use the **“name=”** prefix instead.

**Target Format:**name=name of the element

## **Locating by Link Text**

This type of CSS locator in Selenium applies only to hyperlink texts. We access the link by prefixing our target with “link=” and then followed by the hyperlink text.

## **Locating by CSS Selector**

CSS Selectors in Selenium are string patterns used to identify an element based on a combination of HTML tag, id, class, and attributes. Locating by CSS Selectors in Selenium is more complicated than the previous methods, but it is the most common locating strategy of advanced Selenium users because it can access even those elements that have no ID or name.

CSS Selectors in Selenium have many formats, but we will only focus on the most common ones.

* Tag and ID
* Tag and class
* Tag and attribute
* Tag, class, and attribute
* Inner text

## **Locating by XPath**

XPath is the language used when locating XML (Extensible Markup Language) nodes. Since HTML can be thought of as an implementation of XML, we can also use[XPath](https://www.guru99.com/xpath-selenium.html)in locating HTML elements.

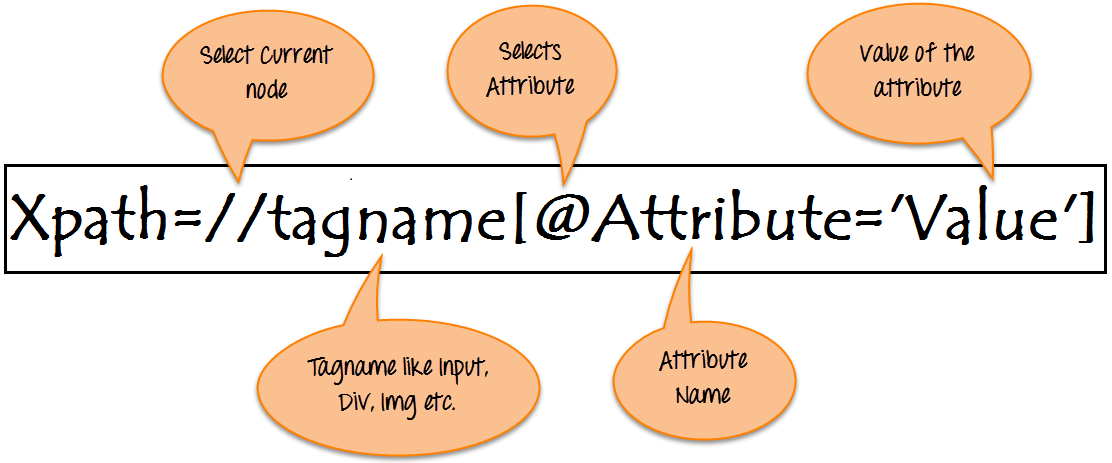
**Advantage:** It can access almost any element, even those without class, name, or id attributes.

**Disadvantage:** It is the most complicated method of identifying elements because of too many different rules and considerations.

## **What is XPath in Selenium?**

**XPath in Selenium** is an XML path used for navigation through the HTML structure of the page. It is a syntax or language for finding any element on a web page using XML path expression. XPath can be used for both HTML and XML documents to find the location of any element on a webpage using HTML DOM structure.

The basic format of XPath in selenium is explained below with screen shot.



Basic Format of XPath

**Syntax for XPath selenium:**

XPath contains the path of the element situated at the web page. Standard XPath syntax for creating XPath is.

Xpath=//tagname[@attribute='value']

* **// :** Select current node.
* **Tagname:**Tagname of the particular node.
* **@:** Select attribute.
* **Attribute:** Attribute name of the node.
* **Value:** Value of the attribute.

To find the element on web pages accurately there are different types of locators:

|  |  |
| --- | --- |
| **XPath Locators** | **Find different elements on web page** |
| **ID** | To find the element by ID of the element |
| **Classname** | To find the element by Classname of the element |
| **Name** | To find the element by name of the element |
| **Link text** | To find the element by text of the link |
| **XPath** | XPath required for finding the dynamic element and traverse between  various elements of the web page |
| **CSS path** | CSS path also locates elements having no name, class or ID. |

## **Types of X-path**

There are two types of XPath:

**1) Absolute XPath**

**2) Relative XPath**

### **Absolute XPath:**

It is the direct way to find the element, but the disadvantage of the absolute XPath is that if there are any changes made in the path of the element then that XPath gets failed.

**Absolute XPath:** /html/body/div[2]/div[1]/div/h4[1]/b/html[1]/body[1]/div[2]/div[1]/div[1]/h4[1]/b[1]

### **Relative Xpath:**

**Relative Xpath** starts from the middle of HTML DOM structure. It starts with double forward slash (//). It can search elements anywhere on the webpage, means no need to write a long xpath and you can start from the middle of HTML DOM structure. Relative Xpath is always preferred as it is not a complete path from the root element.

Relative XPath: //div[@class='featured-box cloumnsize1']//h4[1]//b[1]

**What are XPath axes.**

XPath axes search different nodes in XML document from current context node. XPath Axes are the methods used to find dynamic elements, which otherwise not possible by normal XPath method having no ID , Classname, Name, etc.

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](https://www.guru99.com/introduction-webdriver-comparison-selenium-rc.html) like child, parent, ancestor, sibling, preceding, self, etc.

## **How To Write Dynamic XPath In Selenium WebDriver**

### **1) Basic XPath:**

XPath expression select nodes or list of nodes on the basis of attributes like **ID , Name, Classname**, etc. from the XML document as illustrated below.

Xpath=//input[@name='uid']

Some more basic xpath expressions:

Xpath=//input[@type='text']

Xpath= //label[@id='message23']

Xpath= //input[@value='RESET']

Xpath=//\*[@class='barone']

Xpath=//a[@href='http://demo.guru99.com/']

Xpath= //img[@src='//guru99.com/images/home/java.png']

### **2) Contains():**

Contains() is a method used in XPath expression. It is used when the value of any attribute changes dynamically, for example, login information.

The contain feature has an ability to find the element with partial text as shown in below XPath example.

In this example, we tried to identify the element by just using partial text value of the attribute. In the below XPath expression partial value ‘sub’ is used in place of submit button. It can be observed that the element is found successfully.

Complete value of ‘Type’ is ‘submit’ but using only partial value ‘sub’.

Xpath=//\*[contains(@type,'sub')]

### **3) Using OR & AND:**

In OR expression, two conditions are used, whether 1st condition OR 2nd condition should be true. It is also applicable if any one condition is true or maybe both. Means any one condition should be true to find the element.

In the below XPath expression, it identifies the elements whose single or both conditions are true.

Xpath=//\*[@type='submit' or @name='btnReset']

### **4) Xpath Starts-with**

**XPath starts-with()** is a function used for finding the web element whose attribute value gets changed on refresh or by other dynamic operations on the webpage. In this method, the starting text of the attribute is matched to find the element whose attribute value changes dynamically. You can also find elements whose attribute value is static (not changes).

Xpath=//label[starts-with(@id,'message')]

### **5) XPath Text() Function**

The **XPath text() function** is a built-in function of selenium webdriver which is used to locate elements based on text of a web element. It helps to find the exact text elements and it locates the elements within the set of text nodes. The elements to be located should be in string form.

In this expression, with text function, we find the element with exact text match as shown below. In our case, we find the element with text “UserID”.

Xpath=//td[text()='UserID']

### **6) XPath axes methods:**

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

### **a) 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

### **b) 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

### **c) Child:**

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

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

### **d) 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

### **e) 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

### **f) Parent:**

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

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

### **g) Self:**

One node matching by using “self ” axis. It always finds only one node as it represents self-element.

Xpath =//\*[@type='password']//self::input

### **h) 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

## **Common Commands**

## **Instantiating Web Elements**

Instead of using the long “driver.findElement(By.locator())” syntax every time you will access a particular element, we can instantiate a WebElement object for it. The WebElement class is contained in the “org.openqa.selenium.\*” package.

First Selenium Webdriver Script: JAVA Code Example

## **Clicking on an Element**

Clicking is perhaps the most common way of interacting with web elements**. The click() method is used to simulate the clicking of any element.** The following Selenium Java example shows how click() was used to click on Mercury Tours’  “Sign-In” button.

First Selenium Webdriver Script: JAVA Code Example

Following things must be noted when using the click() method.

* **It does not take any parameter/argument.**
* The method **automatically waits for a new page to load** if applicable.
* The element to be clicked-on, **must be visible** (height and width must not be equal to zero).

## **Get Commands**

Get commands fetch various important information about the page/element. Here are some important “get” commands you must be familiar with.

|  |  |
| --- | --- |
| **Commands** | **Usage** |
| **get()**  Sample usage: | * It automatically opens a new browser window and fetches the page that you specify inside its parentheses. * It is the counterpart of Selenium IDE’s “open” command. * The parameter must be a **String** object. |
| **getTitle()**  Sample usage: | * Needs no parameters * Fetches the title of the current page * Leading and trailing white spaces are trimmed * Returns a null string if the page has no title |
| **getPageSource()**  Sample usage: | * Needs no parameters * Returns the **source code of the page** as a String value |
| **getCurrentUrl()**  Sample usage: | * Needs no parameters * Fetches the string representing the **current URL** that the browser is looking at |
| **getText()**  Sample usage: | * Fetches the **inner text** of the element that you specify |

## Navigate commands

These commands allow you to  refresh,go-into and switch back and forth between different web pages.

|  |  |
| --- | --- |
| **navigate().to()**  Sample usage: | * It automatically **opens a new browser window and fetches the page** that you specify inside its parentheses. * **It does exactly the same thing as the get() method.** |
| **navigate().refresh()**  Sample usage: | * Needs no parameters. * It **refreshes** the current page. |
| **navigate().back()**  Sample usage: | * Needs no parameters * Takes you **back by one page** on the browser’s history. |
| **navigate().forward()**  Sample usage: | * Needs no parameters * Takes you **forward by one page** on the browser’s history. |

## **Closing and Quitting Browser Windows**

|  |  |
| --- | --- |
| **close()**  Sample usage: | * Needs no parameters * **It closes only the browser window that WebDriver is currently** * **controlling**. |
| **quit()**  Sample usage: | * Needs no parameters * **It closes all windows that WebDriver has opened.** |

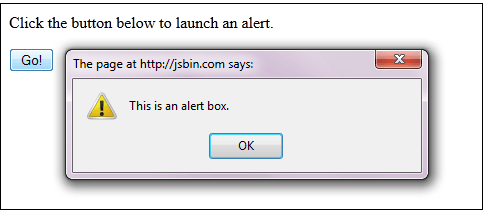
## **Switching Between Frames**

To access GUI elements in a Frame, we should first direct WebDriver to focus on the frame or pop-up window first before we can access elements within them.

Driver.switchTo.frame();

## **Switching Between Pop-up Windows**

WebDriver allows pop-up windows like alerts to be displayed, unlike in Selenium IDE. To access the elements within the alert (such as the message it contains), we must use the **“switchTo().alert()”** method. In the code below, we will use this method to access the alert box and then retrieve its message using the **“getText()”** method, and then automatically close the alert box using the **“switchTo().alert().accept()”** method.



## **Waits**

There are two kinds of waits.

1. Implicit wait – used to set the default waiting time throughout the program
2. Explicit wait – used to set the waiting time for a particular instance only

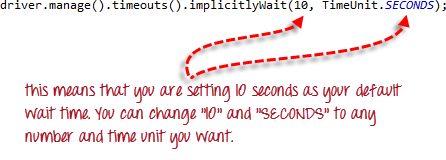
## **Implicit Wait**

* It is simpler to code than Explicit Waits.
* It is usually declared in the instantiation part of the code.
* You will only need one additional package to import.

To start using an implicit wait, you would have to import this package into your code.

First Selenium Webdriver Script: JAVA Code Example

Then on the instantiation part of your code, add this.



## **Explicit Wait**

**Explicit waits are done using the WebDriverWait and ExpectedCondition classes**. For the following Selenium WebDriver example, we shall wait up to 10 seconds for an element whose id is “username” to become visible before proceeding to the next command. Here are the steps.

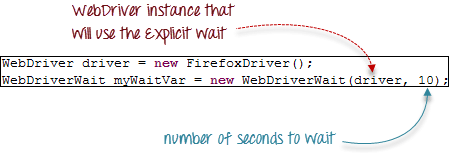
**Step 1**

Import these two packages:

**First Selenium Webdriver Script: JAVA Code Example**

**Step 2**

Declare a WebDriverWait variable. In this example, we will use “myWaitVar” as the name of the variable.



**Step 3**

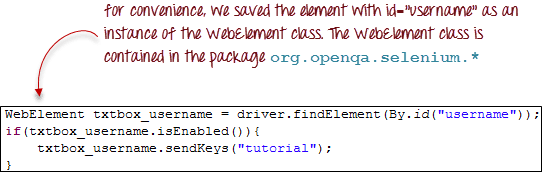
Use myWaitVar with ExpectedConditions on portions where you need the explicit wait to occur. In this case, we will use explicit wait on the “username” (Mercury Tours HomePage) input before we type the text “tutorial” onto it.

First Selenium Webdriver Script: JAVA Code Example

## **Conditions**

Following  methods are used  in conditional and looping operations —

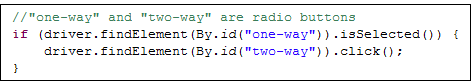
* **isEnabled()** is used when you want to verify whether a certain element is enabled or not before executing a command.



* **isDisplayed()** is used when you want to verify whether a certain element is displayed or not before executing a command.

First Selenium Webdriver Script: JAVA Code Example

* **isSelected()** is used when you want to verify whether a certain **check box, radio button, or option in a drop-down box** is selected. It does not work on other elements.



## **Using ExpectedConditions**

The ExpectedConditions class offers a wider set of conditions that you can use in conjunction with WebDriverWait’s until() method.

Below are some of the most common ExpectedConditions methods.

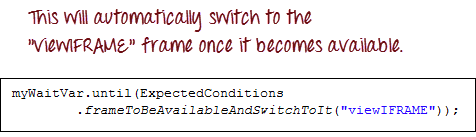
* **alertIsPresent()**– waits until an alert box is displayed.

First Selenium Webdriver Script: JAVA Code Example

* **elementToBeClickable()** – Waits until an element is visible and, at the same time, enabled. The sample Selenium Code below will wait until the element with id=”username” to become visible and enabled first before assigning that element as a WebElement variable named “txtUserName”.

First Selenium Webdriver Script: JAVA Code Example

* **frameToBeAvailableAndSwitchToIt()**– Waits until the given frame is already available, and then automatically switches to it.



# Find Element and FindElements by XPath in Selenium WebDriver

Interaction with a web page requires a user to locate the web element. Find Element command is used to uniquely identify a (one) web element within the web page. Whereas, Find Elements command is used to uniquely identify the list of web elements within the web page.

**FindElement command syntax:**

WebElement elementName = driver.findElement(By.LocatorStrategy("LocatorValue"));

**FindElements command syntax:**

List<WebElement> elementName = driver.findElements(By.LocatorStrategy("LocatorValue"));

# How to Select CheckBox and Radio Button in Selenium WebDriver

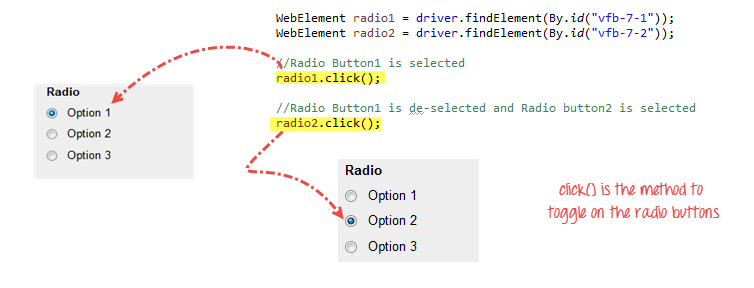
we will see how to identify the following form elements

* [Radio Button](https://www.guru99.com/checkbox-and-radio-button-webdriver.html#7)
* [Check Box](https://www.guru99.com/checkbox-and-radio-button-webdriver.html#8)

## Radio Button

Radio Buttons too can be toggled on by using the click() method.

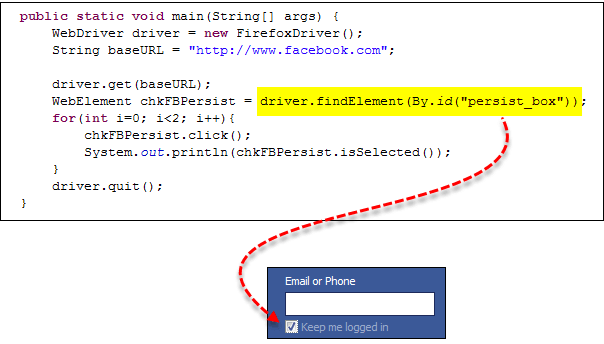
for practice, see that radio1.click() toggles on the “Option1” radio button. radio2.click() toggles on the “Option2” radio button leaving the “Option1” unselected.

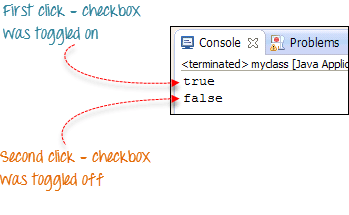


## Check Box

Toggling a check box on/off is also done using the **click()** method.

The code below will click on Facebook’s “Keep me logged in” check box twice and then output the result as TRUE when it is toggled on, and FALSE if it is toggled off.





isSelected() method is used to know whether the Checkbox is toggled on or off.

# How to Click on Image in Selenium Webdriver

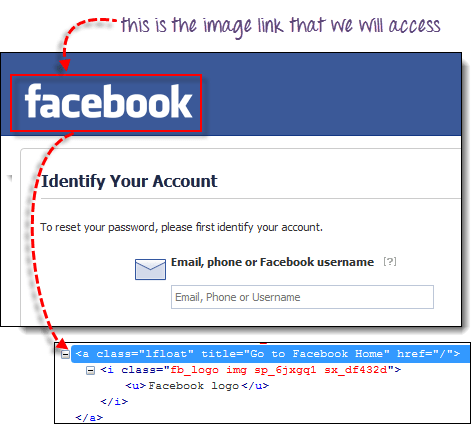
## **Accessing Image Links**

Image links are the links in web pages represented by an image which when clicked navigates to a different window or page.

Since they are images, we cannot use the By.linkText() and By.partialLinkText() methods because image links basically have no link texts at all.

In this case, we should resort to using either By.cssSelector or By.xpath. The first method is more preferred because of its simplicity.

In the example below, we will access the “Facebook” logo on the upper left portion of Facebook’s Password Recovery page.



We will use By.cssSelector and the element’s “title” attribute to access the image link. And then we will verify if we are taken to Facebook’s homepage.

# How to Select Value from DropDown using Selenium Webdriver

## Select Class in Selenium

The **Select Class in Selenium** is a method used to implement the HTML SELECT tag. The html select tag provides helper methods to select and deselect the elements. The Select class is an ordinary class so New keyword is used to create its object and it specifies the web element location.

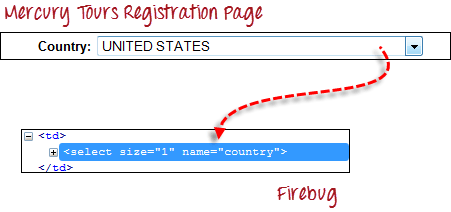
## Select Option from Drop-Down Box

Following is a step by step process on how to select value from dropdown in Selenium:

Before handling dropdown in Selenium and controlling drop-down boxes, we must do following two things:

1. Import the package **org.openqa.selenium.support.ui.Select**
2. Instantiate the drop-down box as an object, Select in Selenium WebDriver

As an example, go to Mercury Tours’ Registration page (http://demo.guru99.com/test/newtours/register.php) and notice the “Country” drop-down box there.



**Step 1**

Import the “Select” package.

How to Select Option from DropDown using Selenium Webdriver

**Step 2**

Declare the drop-down element as an instance of the Select class. In the example below, we named this instance as “drpCountry”.

How to Select Option from DropDown using Selenium Webdriver

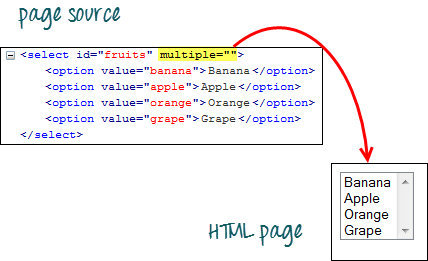
**Step 3**

We can now start controlling “drpCountry” by using any of the available Select methods to select dropdown in Selenium. The sample code below will select the option “ANTARCTICA.”

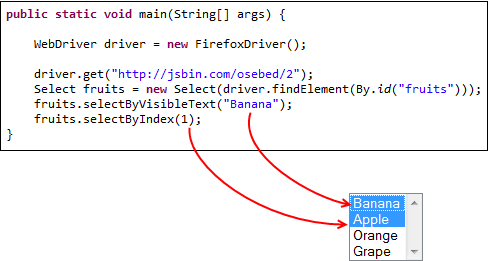
How to Select Option from DropDown using Selenium Webdriver

## Selecting Items in a Multiple SELECT elements

We can also use the **selectByVisibleText()** method in selecting multiple options in a multi SELECT element the base URL. It contains a drop-down box that allows multiple selections at a time.



The code below will select the first two options using the selectByVisibleText() method.



## Select Methods

The following are the most common methods used on Selenium dropdown list.

**selectByVisibleText()**and  
**deselectByVisibleText()**

*Example:*

How to Select Option from DropDown using Selenium Webdriver

**selectByValue()** and  
**deselectByValue()**

*Example:*

How to Select Option from DropDown using Selenium Webdriver

How to Select Option from DropDown using Selenium Webdriver

**selectByIndex()** and **deselectByIndex()**

*Example:*

How to Select Option from DropDown using Selenium Webdriver

**isMultiple()**

*Example:*

*How to Select Option from DropDown using Selenium Webdriver*

**deselectAll()**

*Example:*

*How to Select Option from DropDown using Selenium Webdriver*