What is Selenium?

Selenium is an open source automation testing tool for testing only web based applications. It cannot test your desktop application. Selenium supports multiple browsers. It’s easy to use, strong and is very flexible. You can work on many operating systems and you can code in many programming languages using selenium.

**Languages supported by Selenium:**

* Java
* C#
* Ruby
* Pyton
* PHP
* Pearl

**Browsers supported by Selenium:**

* Mozilla
* IE
* Google chrome
* Opera

The language which you use is independent of the language in which your application is made. For example if your application is made in C#, then you can use selenium with any languages mentioned above to test it. You need to know at least one programming languages mentioned above in order to learn selenium.

**Operating systems Supported by Selenium:**

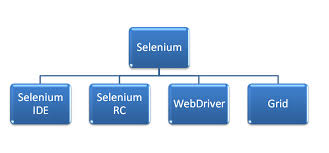
* Windows
* Mac
* Linux
* Unix
* Android

**Features of Selenium:**

* Open source
* Works on multiple browsers and multiple operating systems as compared to other tools in market.
* You can develop selenium code and make it run parallel on multiple machines using different browsers.
* Support for Android and IPhone Testing.
* Selenium IDE is a simple tool which comes as an add on in Firefox and is easy to use. It has the record and run feature which is very strong.
* You can also extend the functionality/scope of IDE with the help of many plugins available
* Selenium RC is the older version of selenium and is supporting all the languages mentioned above
* Web driver is the latest version of selenium and is very strong. It’s removed lots of drawbacks in RC and introduced many more new features in selenium.
* Object oriented data driven or hybrid testing framework can be made very easily.
* You can use open source frameworks such as junit, testng, nuint etc and can write selenium test cases in them

**Components of Selenium:**

Selenium tool has 4 components



1. Selenium IDE:

Is an add on to Mozilla Firefox and Chrome. It’s got a strong feature of record and playback. It supports HTML as source language. It only works on Mozilla Firefox browser.

1. Selenium RC:

This is the older version of selenium. It works on multiple browsers. RC can be implemented in any one of the programming languages mentioned above.

1. Selenium Web Driver:

Is the new version of selenium. It also works on multiple browsers. Its removed many drawbacks and issues in Selenium RC. It also supports Android and IPhone Testing.

1. Selenium Grid:

Grid is used to run test cases parallel on multiple machines and browsers. It is used for Compatibility Testing

**Selenium IDE:**

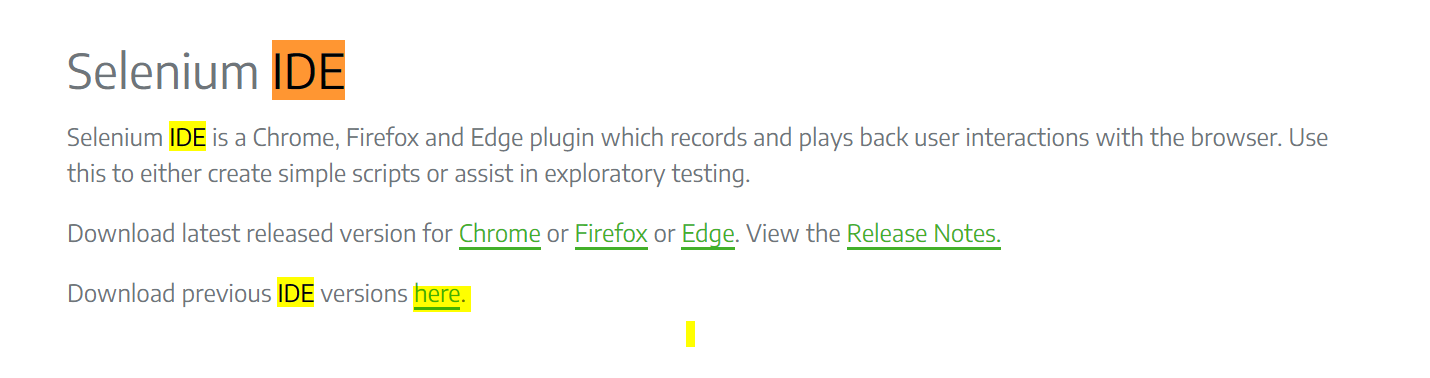
Installs as an add on in Mozilla. Only runs in Mozilla. It’s got a strong feature of record and run. You can also extend IDE functionality with the help of user extensions. You can also parameterize your test cases using IDE.

Features:

* Easy record and playback
* Intelligent field selection will use IDs, names, or XPath as needed
* Auto complete for all common Selenium commands
* Debug and set breakpoints
* Save tests as HTML, Ruby scripts, or any other format
* Option to automatically assert the title of every page

**Installation of IDE:**

1. Install Firefox Browser OR Chrome and open Firefox / Chrome browser
2. Go to site <https://www.selenium.dev/downloads/>
3. Click on the link latest or latest version under Selenium IDE
4. Click on Install and once installed restart browser



Selenium identifies Objects / Web Elements in AUT (Application Under Test) using below properties

* Id
* Name
* Link
* Xpath
* CSS(Cascading Style Sheets)

1. Id: Selenium uses object id to identify the objects in the application.

syn: id = id name

1. Name: Selenium uses object name to identify the objects in the application.

syn: name= name of the element

1. LinkText syn: This is a simple method of locating a hyperlink in your web page by using the text of the link. If two links with the same text are present, then the first match will be used.
2. Xpath:

XPath is the language used for locating nodes in an XML document. As HTML can be an implementation of XML (XHTML), Selenium users can leverage this powerful language to target elements in their web applications.

XPath extends beyond the simple methods of locating by id or name attributes, and opens up all sorts of new possibilities such as locating the second radio button on the page.

Reasons for using XPath is when I don’t have a suitable id or name attribute for the element I wish to locate. You can use XPath to either locate the element in absolute or relative to an element that does not have an id or name attribute.

XPath locators can also be used to specify elements via attributes other than id and name.

1. Absolute x path: if x path is calculated from beginning of the root of HTML tag to locate then it is known as absolute x path. ex: //Html/body/H1/input. Absolute XPaths contain the location of all elements from the root (html) and as a result chances to fail if slightest adjustments to the application.

2. Relative x path: if x path is calculated with nearest unique value then it is known as relative x path. ex: xpath=//@frames/id=submit. By finding a nearby element with an id or name attribute (ideally a parent element) you can locate your target element based on the relationship. This is much less likely to change and can make your tests more robust.

Since only xpath locators start with “//”, it is not necessary to include the xpath= label when specifying an XPath locator.

Syn:

a. //tagname[@atriburtname='value'] ex: //input[@type='Submit']

b. //parenttagname[@parentattributevalue='value']/childtagname

c. Xpath position: this can be used to positions of element. Example link will always change but other link displays with different name but position is same. In this case if you want to click on any element based on the location you need to use this. Syn: //parenttag/parenttag/childtag

1. <html>

2. <body>

3. <form id="login">

4. <input name="name" type="text" />

5. <input name="password" type="password" />

6. <input name="continue" type="submit" value="Login" />

7. <input name="continue" type="button" value="Reset" />

8. </form>

9. </body>

10. <html>

xpath=/html/body/form[1] - Absolute path (would break if the HTML was changed only slightly)

//form[1] - First form element in the HTML

xpath=//form[@id='login'] - The form element with attribute named ‘id’ and the value ‘loginForm’

xpath=//form[input/@name='field1'] - First form element with an input child element with attribute named ‘name’ and the value ‘name’

//input[@name='field1'] - First input element with attribute named ‘name’ and the value ‘username’

//form[@id='login']/input[1] - First input child element of the form element with attribute named ‘id’ and the value ‘login’

//input[@name='continue'][@type='button'] - Input with attribute named ‘name’ and the value ‘continue’ and attribute named ‘type’ and the value ‘button’

//form[@id='login']/input[4] - Fourth input child element of the form element with attribute named ‘id’ and value ‘login’

6. CSS syn:

CSS (Cascading Style Sheets) is a language for describing the rendering of HTML and XML documents. CSS uses Selectors for binding style properties to elements in the document. These Selectors can be used by Selenium as another locating strategy.

<html>

<body>

<form id="login">

<input class="required" name="name" type="text" />

<input class="required passfield" name="password" type="password" />

<input name="continue" type="submit" value="Login" />

<input name="continue" type="button" value="Reset" />

</form>

</body>

<html>

css=form#login

css=input[name="name"]

css=input.required[type="text"]

css=input.passfield

css=#login input[type="button"]

**Selenium Commands – “Selenese”:**

Selenium commands called as selenese. These are the set of commands that run your tests. Sequences of commands are a test script.

Selenium commands are three

* Actions
* Assessors
* Assertions

Actions: are commands that manipulate the application. They do things like “click this button” and “select value”. If action fails to perform, or has an error, the execution of the current test is stopped. Actions may be called with the “AndWait” suffix, e.g. “TypeAndWait”. This suffix tells that the action will cause the browser to make a call to the server, and that tool should wait for a new page to load.

Assessors: are used to store the results in variables, e.g. “storebrowsertitle”. They are also used to automatically generate Assertions.

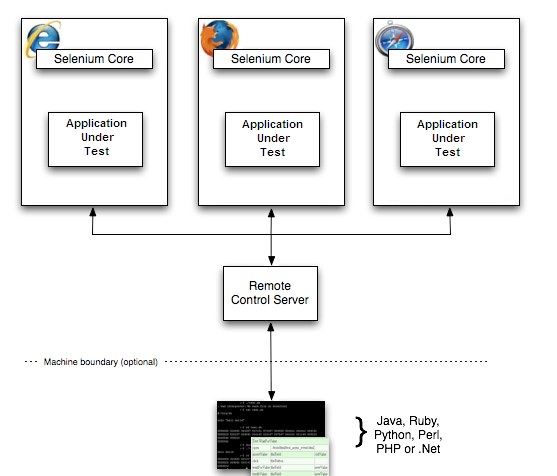
Assertions: Selenium Assertions can be used in 3 modes: “assert”, “verify”, and ” waitFor”. For example, you can “assertText”, “verifyText” and “waitForText”. When an “assert” failed, the test is aborted. When a “verify” failed, the test will continue execution, by logging the failure.

“waitFor” commands wait for certain condition to become true (which is useful for testing Ajax applications). They will succeed immediately if the condition is true. However, they will fail and stop the test if the condition does not become true within the timeout setting which is configurable parameter in IDE.

**Selenium RC:**

RC is the older version of selenium. It works on multiple browsers & operating systems.

RC can be implemented in any one of the programming languages mentioned above.



All you’re coding passes to RC server and this server converts it to Java Script commands. These Java Script commands are sent to AUT.

Selenium RC injects JavaScript into the page to drive the interaction.

Because of its JavaScript based automation engine and the security limitations browsers apply to JavaScript, different things became impossible to do.

Since RC is deprecated we will not discuss much on this and will spend more time on Web Driver.

No pre-requisite knowledge is required to learn Web Driver. You can still learn Web Driver directly without knowledge in RC.

**Selenium Web Driver:**

Web Driver is a tool for automating web application testing. It aims to provide a friendly API that’s easy to explore and understand, easier to use than the Selenium-RC (1.0) API, which will help to make your tests easier to read and maintain.

**Web Driver Compared to Selenium-RC:**

Selenium-Web Driver makes direct calls to the browser using each browser’s native support for automation.

Selenium-RC ‘injected’ JavaScript functions into the browser when the browser was loaded and then used its JavaScript to drive the AUT within the browser.

Web Driver does not use this technique. Again, it drives the browser directly using the browser’s built in support for automation.

Web Driver is faster compared to RC as itdirectly interacts with the browser

Web Driver supports HtmlUnit Driver whereas RC not supports HtmlUnit.

**Web Driver and the Selenium-Server:**

You may, or may not, need the Selenium Server, depending on how you intend to use Selenium-Web Driver. If you will be only using the Web Driver API you do not need the Selenium-Server. If your browser and tests are running on the same machine, and your tests only use the Web Driver API, then you do not need to run the Selenium-Server; Web Driver will run the browser directly.

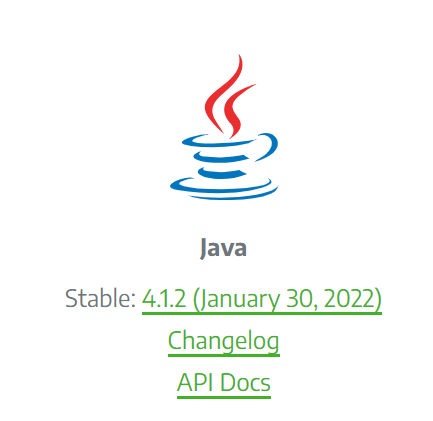
There are some reasons to use the Selenium-Server with Selenium-Web Driver.

* If you use Selenium-Grid to distribute your tests over multiple machines or virtual machines (VMs).
* If you want to connect to a remote machine which has a particular browser version that is not on there in current machine.
* If you are not using the Java bindings (i.e. Python, C#, or Ruby) and would like to use HtmlUnit Driver

**Installation of Web Driver:**

1. First install the JDK

2. Download Java Client Driver from <https://www.selenium.dev/downloads/>



### 3. Click on latest Java Client Driver link under “Selenium Client & WebDriver Language Bindings”

4. Save Zip file to local system

5. Extract files to some location in your system.

**Configure Eclipse IDE with Web Driver:**

1. Open Eclipse

2. Select default work space or modify and select

13. Create new java project

4. Create new java class

5. Right click on **project** name select Build Path and click on Configure Build Path, click on Classpath

6. Click on the Libraries tab and click Add External JARs

7. Navigate to extracted path from where you have extracted webdriver (step 5 above) web driver

8. Add the JAR file inside and outside the lib folder

9. Finally click Apply & Close

**Web Driver’s Drivers:**

Web Driver is the key **interface** against which tests are written, but there are several implementations.

These include:

* HtmlUnit Driver
* Firefox Driver
* Internet Explorer Driver
* Chrome Driver
* Opera Driver
* iOS Driver
* Microsoft Edge

Interface webdriver

{

Void get(String URL);

Void findElement();

}

Class FirefoxDriver implements webdriver

{

Void get(String URL)  
 {

Code to open firefox browser

}

Void findElement()

{

}

}

Class ChromeDriver implements webdriver

{

Void get(String URL)

{

Cod to open chrome browser

}

Void findElement()

{

}

}

Steps to download Chromedriver.exe files

1. Navigate to website <https://chromedriver.chromium.org/downloads>
2. Check your chrome browser version
3. Based on your browser version click on link, example if you have 99 version of chrome browser then use same link of the version example then download 99 version. Click on this 99 version
4. Click on the zip file based on your operating system. Example if you have windows then click on Chromedriver\_win32.zip file and save in your local machine
5. You need to unzip the file and save

**Opening a web page using Web Driver:**

get() or navigate() are the methods which are used to open the web page in the browser.

Example if I want to open google page: driver.get("http://www.google.com");

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

driver.navigate().forward(): Which is equal to clicking on “forward” button in the browser.

driver.navigate().back(): Which is equal to clicking on “back” button in the browser.

driver is nothing but an instance of the Web Driver

Difference between driver.get and driver.navigate: driver.navigate tracks the history whereas driver.get doesn’t track any history.

**Locating UI Elements (WebElements):**

Locating elements in WebDriver can be done using

* WebDriver instance itself or
* On a WebElement.

Each of the language bindings expose a “Find Element” and “Find Elements” method. The first returns a WebElement object otherwise it throws an exception. The second one returns a list of WebElements, it can return an empty list if no DOM elements match the query.

The “Find” method take a locator or query object called “By”. “By” strategies are listed below.

**By ID:**

This is the most efficient and preferred way to locate an element. Common pitfalls that UI developers make is having non-unique id’s on a page or auto-generating the id, both should be avoided. A class on an html element is more appropriate than an auto-generated id.

Example of how to find an element that looks like this:

<div id="userId">...</div>

WebElement element = driver.findElement(By.id("userId"));

**By Class Name:**

“Class” in this case refers to the attribute on the DOM element. Often in practical use there are many DOM elements with the same class name, thus finding multiple elements becomes the more practical option over finding the first element.

Example of how to find an element that looks like this:

<div class="cheese"><span>Cheddar</span></div><div class="cheese"><span>Gouda</span></div>

List<WebElement> cheeses = driver.findElements(By.className("cheese"));

**By Link Text:**

Find the link element with matching visible text.

Example of how to find an element that looks like this:

<a href="http://www.google.com/search?q=cheese">cheese</a>>

WebElement cheese = driver.findElement(By.linkText("cheese"));

**Simple example using web driver:**

package org.openqa.selenium.example;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.firefox.FirefoxDriver

**public** **class** WebExample {

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

System.*setProperty*("webdriver.chrome.driver","C:\\Ashok\\chromedriver.exe");

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

String URL="https://www.google.com/";

//driver.get(URL);

driver.navigate().to(URL);

driver.navigate().refresh();

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

driver.findElement(By.name("btnk")).click();

driver.close();

**Popup Dialogs:**

Starting with Selenium 2.0 beta 1, there is built in support for handling popup dialog boxes. After you’ve triggered an action that opens a popup, you can access the alert with the following:

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

**Synchronization in Web Driver:**

There are three 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.
3. Fluent Wait—Used to set the waiting time but it will keep checking for element present in given interval of times. As soon as object found it continue of execution of code.

Note: We have Thread.sleep() method which also provides same mechanism but it is not dynamic and your scrip will go sleep for specified time.

Implicit Wait:

* It is simple 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.

Import package import.java.util.concurrent.TimeUnit;

Inistantiation: driver.manage().timeouts().implicitlywait(10,TimeUnit.SECONDS);

Explicit Wait:

## Condition for Explicit wait in selenium webdriver

**Condition 1-** I have a web page which has some login form and after login, it takes a lot of time to load Account page or Home page. This page is dynamic it means sometimes it takes 10 seconds to load the homepage, sometimes its 15 second and so on. In this situation, the Explicit wait can help us which will wait until specific page/page title is not present it will keep waiting.

**Condition 2-** You are working on travel application and you have filled the web form and clicked on submit button. Now you have to wait until complete data is not loaded or specific data is not loaded. In this case, again we can use Explicit wait in which we can give wait till specific or set of elements are not loaded.

**Condition 3-** There are some elements on a web page which are hidden and it will be displayed only when specific conditions get true, so we have to wait until these elements are not visible. In this case, again explicit wait will help in which we can specify wait till the element or elements are not visible.

* Using the WebDriverWait and ExpectedCondition class.
* You need to import org.openqa.seleniumsupport.ui.ExpectedConditions;

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

inistation: WebDriver driver=new FirefoxDriver();

WebDriver Wait myWaiVar=new WebDriverWait(driver,10);

Explicit Wait:

**Let us discuss a few of them at length:**

**#1) elementToBeClickable()** – The expected condition waits for an element to be clickable i.e. it should be present/displayed/visible on the screen as well as enabled.

**Sample Code**  
*wait.until(ExpectedConditions.elementToBeClickable(By.xpath(“//div[contains(text(),’COMPOSE’)]”)));*

**#2) textToBePresentInElement() –** The expected condition waits for an element having a certain string pattern.

**Sample Code**  
*wait.until(ExpectedConditions.textToBePresentInElement(By.xpath(“//div[@id= ‘forgotPass'”), “text to be found”));*

**#3) alertIsPresent()-** The expected condition waits for an alert box to appear.

**Sample Code**  
*wait.until(ExpectedConditions.alertIsPresent()) !=null);*

**#4) titleIs()** – The expected condition waits for a page with a specific title.

**Sample Code**  
*wait.until(ExpectedConditions.titleIs(“gmail”));*

**#5) frameToBeAvailableAndSwitchToIt()** – The expected condition waits for a frame to be available and then as soon as the frame is available, the control switches to it automatically.

**Sample Code**  
*wait.until(ExpectedConditions.frameToBeAvailableAndSwitchToIt(By.id(“newframe”)));*

***Fluent Wait in Selenium***

Fluent Wait in Selenium marks the maximum amount of time for Selenium WebDriver to wait for a certain condition (web element) becomes visible. It also defines how frequently WebDriver will check if the condition appears before throwing the “**ElementNotVisibleException**”.

To put it simply, Fluent Wait looks for a web element repeatedly at regular intervals until timeout happens or until the object is found.

Fluent Wait commands are most useful when interacting with web elements that can take longer durations to load. This is something that often occurs in Ajax applications.

While using Fluent Wait, it is possible to set a default polling period as needed. The user can configure the wait to ignore any exceptions during the polling period.

Fluent waits are also sometimes called smart waits because they don’t wait out the entire duration defined in the code. Instead, the test continues to execute as soon as the element is detected – as soon as the condition specified in .**until(YourCondition)** method becomes true.

**Syntax**

Wait wait = new FluentWait(WebDriver reference)

.withTimeout(timeout, SECONDS)

.pollingEvery(timeout, SECONDS)

.ignoring(Exception.class);

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

public WebElement applyy(WebDriver driver) {

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

}

}); driver

**Example of Fluent Wait Command**

//Declare and initialise a fluent wait

FluentWait wait = new FluentWait();

//Specify the timout of the wait

wait.withTimeout(5000, TimeUnit.MILLISECONDS);

//Sepcify polling time

wait.pollingEvery(250, TimeUnit.MILLISECONDS);

//Specify what exceptions to ignore

wait.ignoring(NoSuchElementException.class)

//This is how we specify the condition to wait on.

//This is what we will explore more in this chapter

wait.until(ExpectedConditions.alertIsPresent());

Drop-down:

You can get the dropdown value count using the below code.

Webelement ele = driver.findElement(By.id("Dropdown Locator Value"));

Select sel = new Select(ele);

List<WebElement> list = ele.getOptions();

System.Out.println("Drop down Value Count :" + list.size());

// To display the drop down value

for(WebElement e : list){

System.Out.println("Drop down Value :" + e.getText());

}

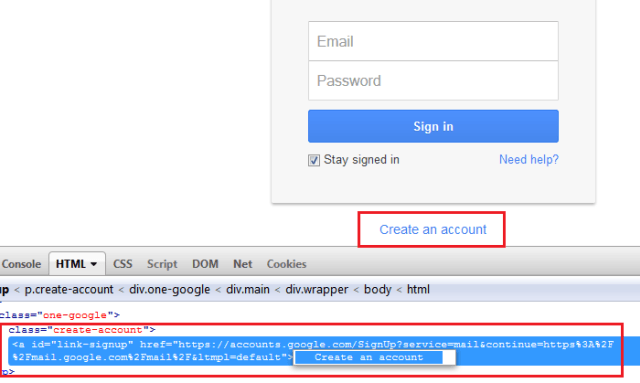
– contains()  
– starts-with()

Situations where these functions should be used:

– When we do not have complete knowledge about the web elements HTML properties  
– When the values of the attributes are dynamic i.e. changing  
– When we would like to create a list of web elements containing same partial attribute value

**Application Under Test**

Let us consider an example of “gmail.com”. We would try to find “Create an account” hyperlink using Xpath with the above mentioned functions.



**Contains()**

Contains method is used when we know about the partial attribute value or partial text associated with the web element.

**Syntax:**

To find the “Create an account” hyperlink, consider the following variations of contains() method.

**By Text**

–          //a[contains(text(),’Create’)]

–          //a[contains(text(),’an’)]

–          //a[contains(text(),’an account’)]

–          //a[contains(text(),’create an account’)]

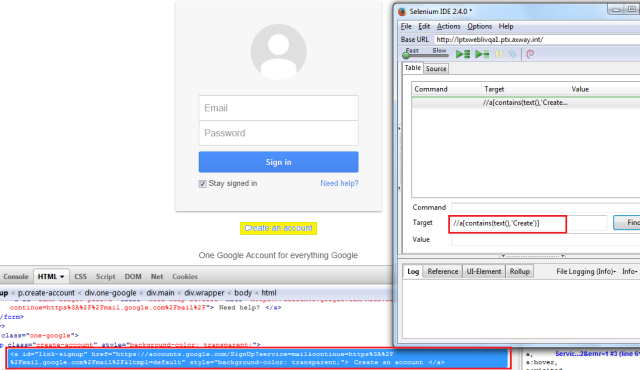
**By Attribute**

–          //a[contains(@id,’signup’)]

–          //a[contains(@id,’link-signup’)]

–          //a[contains(@id,’link)] // Take a note that two link would be highlighted, thus user would have to supply some additional attribute value in order to locate the link uniquely.

For all the above Xpaths, user would be able to find “Create an account” hyperlink successfully and the web element would be highlighted. Check out the following screenshot.



**Starts-with()**

starts-with() method is used when we know about the initial partial attribute value or initial partial text associated with the web element. User can also use this method to locate web elements those are consist of both the static(initial) and dynamic(trailing) values.

**Syntax:**

To find the “Create an account” hyperlink, consider the following variations of starts-with() method.

**By Attribute**

–          //a[starts-with(@id,’link-si’)]

–          //a[starts-with(@id,’link-sign’)]

For all the above Xpaths, user would be able to find “Create an account” hyperlink successfully and the web element would be highlighted just like previous screenshot.

Explicit Wait Example:

|  |  |
| --- | --- |
| 1 | import static org.junit.Assert.\*; |

|  |  |
| --- | --- |
| 2 | import java.util.concurrent.TimeUnit; |

|  |  |
| --- | --- |
| 3 | import org.junit.After; |

|  |  |
| --- | --- |
| 4 | import org.junit.Before; |

|  |  |
| --- | --- |
| 5 | import org.junit.Test; |

|  |  |
| --- | --- |
| 6 | import org.openqa.selenium.By; |

|  |  |
| --- | --- |
| 7 | import org.openqa.selenium.WebDriver; |

|  |  |
| --- | --- |
| 8 | import org.openqa.selenium.WebElement; |

|  |  |
| --- | --- |
| 9 | import org.openqa.selenium.firefox.FirefoxDriver; |

|  |  |
| --- | --- |
| 10 | import org.openqa.selenium.support.ui.ExpectedConditions; |

|  |  |
| --- | --- |
| 11 | import org.openqa.selenium.support.ui.WebDriverWait; |

|  |  |
| --- | --- |
| 12 |  |

|  |  |
| --- | --- |
| 13 | public class Wait\_Demonstration { |

|  |  |
| --- | --- |
| 14 |  |

|  |  |
| --- | --- |
| 15 | // created reference variable for WebDriver |

|  |  |
| --- | --- |
| 16 | WebDriver drv; |

|  |  |
| --- | --- |
| 17 | @Before |

|  |  |
| --- | --- |
| 18 | public void setup() throws InterruptedException { |

|  |  |
| --- | --- |
| 19 |  |

|  |  |
| --- | --- |
| 20 | // initializing drv variable using FirefoxDriver |

|  |  |
| --- | --- |
| 21 | drv=new FirefoxDriver(); |

|  |  |
| --- | --- |
| 22 | // launching gmail.com on the browser |

|  |  |
| --- | --- |
| 23 | drv.get("https://gmail.com"); |

|  |  |
| --- | --- |
| 24 | // maximized the browser window |

|  |  |
| --- | --- |
| 25 | drv.manage().window().maximize(); |

|  |  |
| --- | --- |
| 26 | drv.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS); |

|  |  |
| --- | --- |
| 27 | } |

|  |  |
| --- | --- |
| 28 |  |

|  |  |
| --- | --- |
| 29 | @Test |

|  |  |
| --- | --- |
| 30 | public void test() throws InterruptedException { |

|  |  |
| --- | --- |
| 31 |  |

|  |  |
| --- | --- |
| 32 | // saving the GUI element reference into a "username" variable of WebElement type |

|  |  |
| --- | --- |
| 33 | WebElement username = drv.findElement(By.id("Email")); |

|  |  |
| --- | --- |
| 34 |  |

|  |  |
| --- | --- |
| 35 | // entering username |

|  |  |
| --- | --- |
| 36 | username.sendKeys("userid"); |

|  |  |
| --- | --- |
| 37 |  |

|  |  |
| --- | --- |
| 38 | // entering password |

|  |  |
| --- | --- |
| 39 | drv.findElement(By.id("Passwd")).sendKeys("password"); |

|  |  |
| --- | --- |
| 40 |  |

|  |  |
| --- | --- |
| 41 | // clicking signin button |

|  |  |
| --- | --- |
| 42 | drv.findElement(By.id("signIn")).click(); |

|  |  |
| --- | --- |
| 43 |  |

|  |  |
| --- | --- |
| 44 | // explicit wait - to wait for the compose button to be click-able |

|  |  |
| --- | --- |
| 45 | WebDriverWait wait = newWebDriverWait(drv,30); |

|  |  |
| --- | --- |
| 46 |  |

|  |  |
| --- | --- |
| 47 | wait.until(ExpectedConditions.visibilityOfElementLocated(By.xpath("//div[contains(text(),'COMPOSE')]"))); |

|  |  |
| --- | --- |
| 48 | // click on the compose button as soon as the "compose" button is visible |

|  |  |
| --- | --- |
| 49 | drv.findElement(By.xpath("//div[contains(text(),'COMPOSE')]")).click(); |

|  |  |
| --- | --- |
| 50 | } |

|  |  |
| --- | --- |
| 51 |  |

|  |  |
| --- | --- |
| 52 | @After |

|  |  |
| --- | --- |
| 53 | public void teardown() { |

|  |  |
| --- | --- |
| 54 | // closes all the browser windows opened by web driver |

|  |  |
| --- | --- |
| 55 | drv.quit(); |

|  |  |
| --- | --- |
| 56 | } |

|  |  |
| --- | --- |
| 57 | } |

Window Based Alerts / Handling keyboard:

**Robot Class** is used in Selenium automation tests, to control over keyboard or mouse to interact with **OS windows** like selecting a file to attach from window based alerts window. Selenium Webdriver does not support handling these pop-ups/applications, so in[Java](https://www.guru99.com/java-tutorial.html) robot class was introduced which can handle OS pop-ups/applications.

|  |  |
| --- | --- |
| 1 | import java.awt.Robot;</pre> |

|  |  |
| --- | --- |
| f2 | import java.awt.event.KeyEvent; |

|  |  |
| --- | --- |
| 3 | import org.junit.After; |

|  |  |
| --- | --- |
| 4 | import org.junit.Before; |

|  |  |
| --- | --- |
| 5 | import org.junit.Test; |

|  |  |
| --- | --- |
| 6 | import org.openqa.selenium.By; |

|  |  |
| --- | --- |
| 7 | import org.openqa.selenium.WebDriver; |

|  |  |
| --- | --- |
| 8 | import org.openqa.selenium.firefox.FirefoxDriver; |

|  |  |
| --- | --- |
| 9 |  |

|  |  |
| --- | --- |
| 10 | public class DemoWindowAlert { |

|  |  |
| --- | --- |
| 11 | WebDriver driver; |

|  |  |
| --- | --- |
| 12 | @Before |

|  |  |
| --- | --- |
| 13 |  |

|  |  |
| --- | --- |
| 14 | public void setUp() |

|  |  |
| --- | --- |
| 15 | { |

|  |  |
| --- | --- |
| 16 | driver=new FirefoxDriver(); |

|  |  |
| --- | --- |
| 17 | driver.get("https://gmail.com"); |

|  |  |
| --- | --- |
| 18 | driver.manage().window().maximize(); |

|  |  |
| --- | --- |
| 19 | } |

|  |  |
| --- | --- |
| 20 |  |

|  |  |
| --- | --- |
| 21 | @Test |

|  |  |
| --- | --- |
| 22 | public void testWindowAlert() throws Exception{ |

|  |  |
| --- | --- |
| 23 |  |

|  |  |
| --- | --- |
| 24 | // enter a valid email address |

|  |  |
| --- | --- |
| 25 | driver.findElement(By.id("Email")).sendKeys("userid"); |

|  |  |
| --- | --- |
| 26 |  |

|  |  |
| --- | --- |
| 27 | // enter a valid password |

|  |  |
| --- | --- |
| 28 | driver.findElement(By.id("Passwd")).sendKeys("password"); |

|  |  |
| --- | --- |
| 29 |  |

|  |  |
| --- | --- |
| 30 | // click on sign in button |

|  |  |
| --- | --- |
| 31 | driver.findElement(By.id("signIn")).click(); |

|  |  |
| --- | --- |
| 32 | Thread.sleep(30000); |

|  |  |
| --- | --- |
| 33 |  |

|  |  |
| --- | --- |
| 34 | // click on compose button |

|  |  |
| --- | --- |
| 35 | driver.findElement(By.xpath("//div[@class='z0']//div[contains(text(),'COMPOSE')]")).click(); |

|  |  |
| --- | --- |
| 36 |  |

|  |  |
| --- | --- |
| 37 | // click on attach files icon |

|  |  |
| --- | --- |
| 38 | driver.findElement(By.xpath("//div[contains(@command,'Files')]//div[contains(@class,'aaA')]")).click(); |

|  |  |
| --- | --- |
| 39 |  |

|  |  |
| --- | --- |
| 40 | // creating instance of Robot class (A java based utility) |

|  |  |
| --- | --- |
| 41 | Robot rb =new Robot(); |

|  |  |
| --- | --- |
| 42 |  |

|  |  |
| --- | --- |
| 43 | // pressing keys with the help of keyPress and keyRelease events |

|  |  |
| --- | --- |
| 44 | rb.keyPress(KeyEvent.VK\_D); |

|  |  |
| --- | --- |
| 45 | rb.keyRelease(KeyEvent.VK\_D); |

|  |  |
| --- | --- |
| 46 | Thread.sleep(2000); |

|  |  |
| --- | --- |
| 47 |  |

|  |  |
| --- | --- |
| 48 | rb.keyPress(KeyEvent.VK\_SHIFT); |

|  |  |
| --- | --- |
| 49 | rb.keyPress(KeyEvent.VK\_SEMICOLON); |

|  |  |
| --- | --- |
| 50 | rb.keyRelease(KeyEvent.VK\_SEMICOLON); |

|  |  |
| --- | --- |
| 51 | rb.keyRelease(KeyEvent.VK\_SHIFT); |

|  |  |
| --- | --- |
| 52 |  |

|  |  |
| --- | --- |
| 53 | rb.keyPress(KeyEvent.VK\_BACK\_SLASH); |

|  |  |
| --- | --- |
| 54 | rb.keyRelease(KeyEvent.VK\_BACK\_SLASH); |

|  |  |
| --- | --- |
| 55 | Thread.sleep(2000); |
| rb.keyRelease(KeyEvent.VK\_T);  rb.keyRelease(KeyEvent.VK\_e); |  |
| rb.keyPress(KeyEvent.VK\_Enter); |  |
|  |  |

|  |  |
| --- | --- |
|  |  |

|  |  |
| --- | --- |
| 57 | rb.keyPress(KeyEvent.VK\_P); |

|  |  |
| --- | --- |
| 58 | rb.keyRelease(KeyEvent.VK\_P); |

|  |  |
| --- | --- |
| 59 |  |

|  |  |
| --- | --- |
| 60 | rb.keyPress(KeyEvent.VK\_I); |

|  |  |
| --- | --- |
| 61 | rb.keyRelease(KeyEvent.VK\_I); |

|  |  |
| --- | --- |
| 62 |  |

|  |  |
| --- | --- |
| 63 | rb.keyPress(KeyEvent.VK\_C); |

|  |  |
| --- | --- |
| 64 | rb.keyRelease(KeyEvent.VK\_C); |

|  |  |
| --- | --- |
| 65 | Thread.sleep(2000); |

|  |  |
| --- | --- |
| 66 |  |

|  |  |
| --- | --- |
| 67 | rb.keyPress(KeyEvent.VK\_ENTER); |

|  |  |
| --- | --- |
| 68 | rb.keyRelease(KeyEvent.VK\_ENTER); |

|  |  |
| --- | --- |
| 69 | Thread.sleep(2000); |

|  |  |
| --- | --- |
| 70 | } |

|  |  |
| --- | --- |
| 71 |  |

|  |  |
| --- | --- |
| 72 | @After |

|  |  |
| --- | --- |
| 73 | public void tearDown() |

|  |  |
| --- | --- |
| 74 | { |

|  |  |
| --- | --- |
| 75 | driver.quit(); |

|  |  |
| --- | --- |
| 76 | } |

|  |  |
| --- | --- |
| 77 | } |

**Reading data from Excel:**

# How to Read/Write Data from Excel File: Selenium POI

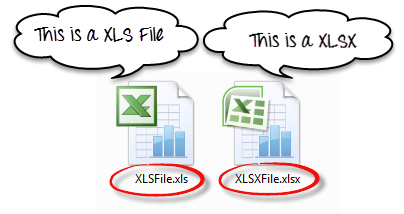
File IO is a critical part of any software process. We frequently create a file, open it & update something or delete it in our Computers. Same is the case with Selenium Automation. We need a process to manipulate files with Selenium.

Java provides us different classes for File Manipulation with Selenium. In this tutorial, we are going to learn how can we read and write on Excel file with the help of Java IO package and Apache POI library.

## Apache POI in Selenium

The **Apache POI in Selenium** is a widely used API for selenium data driven testing. It is a POI library written in Java that gives users an API for manipulating Microsoft documents like .xls and .xlsx. Users can easily create, modify and read/write into excel files. POI stands for “Poor Obfuscation Implementation.”

### How to handle excel file using POI (Maven POM Dependency)

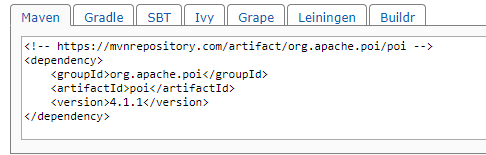


To Read and Write Excel file in Java, Apache provides a very famous library POI. This library is capable enough to read and write both**XLS** and**XLSX** file format of Excel.

To read**XLS** files, an**HSSF** implementation is provided by POI library.

To read**XLSX, XSSF** implementation of**POI library** will be the choice. Let’s study these implementations in detail.

If you are using Maven in your project, the Maven dependency will be



<dependency>

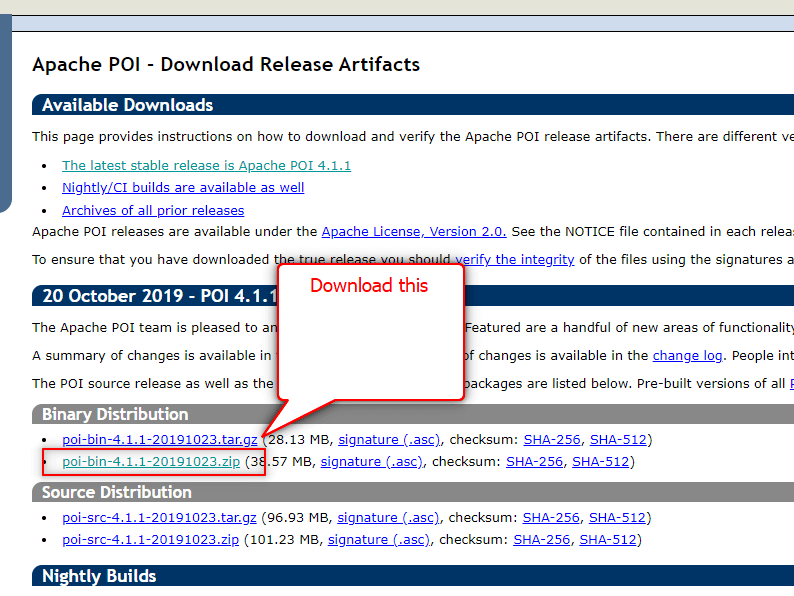
<groupId>org.apache.poi</groupId>

<artifactId>poi</artifactId>

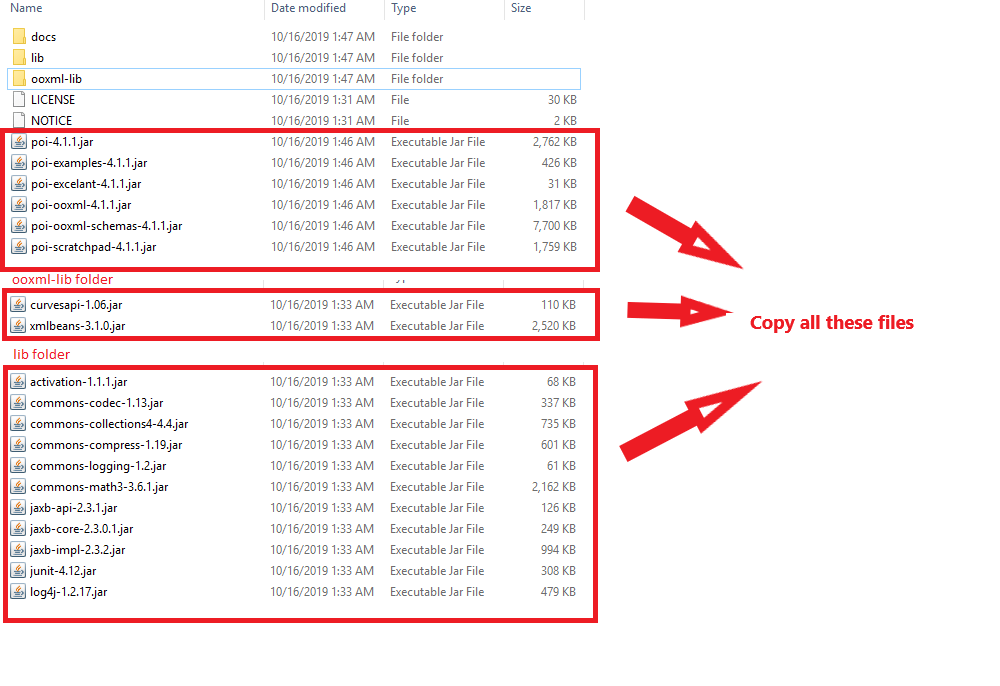
<version>4.1.1</version>

</dependency>

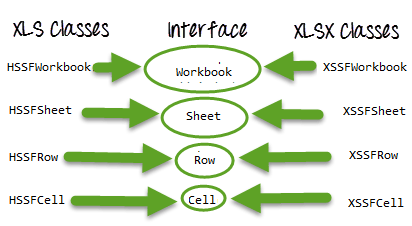
Or you can simply download the latest version POI jars from [http://poi.apache.org/download.html](https://poi.apache.org/download.html) & download the latest zip file



When you download the zip file for this jar, you need to unzip it and add these all jars to the class path of your project.



## Classes and Interfaces in POI:

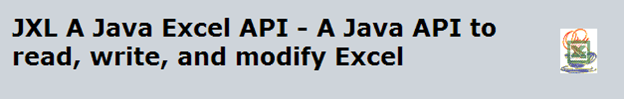


Classes and Interfaces in Apache POI

Following is a list of different Java Interfaces and classes in **POI** for reading **XLS** and **XLSX** file-

* **Workbook**: XSSFWorkbook and HSSFWorkbook classes implement this interface.
* **XSSFWorkbook**: Is a class representation of XLSX file.
* **HSSFWorkbook**: Is a class representation of XLS file.
* **Sheet**: XSSFSheet and HSSFSheet classes implement this interface.
* **XSSFSheet**: Is a class representing a sheet in an XLSX file.
* **HSSFSheet**: Is a class representing a sheet in an XLS file.
* **Row**: XSSFRow and HSSFRow classes implement this interface.
* **XSSFRow**: Is a class representing a row in the sheet of XLSX file.
* **HSSFRow**: Is a class representing a row in the sheet of XLS file.
* **Cell**: XSSFCell and HSSFCell classes implement this interface.
* **XSSFCell**: Is a class representing a cell in a row of XLSX file.
* **HSSFCell:** Is a class representing a cell in a row of XLS file.

## Excel Manipulation using JXL API



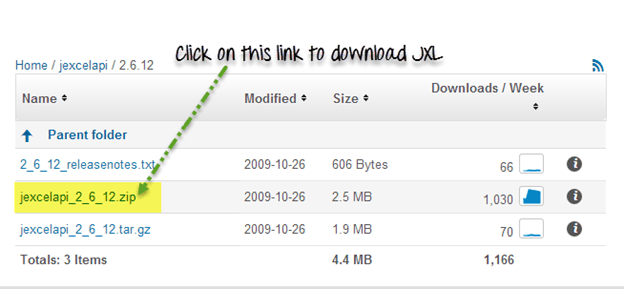
JXL is also another famous jar to read Excel file in Java and writing files. Nowadays, POI is used in most of the projects, but before POI, JXL was only Java API for Excel manipulation. It is a very small and simple API for excel reading in Selenium.

S*uggestion is not to use JXL in any new project because the library is not in active development from 2011 and lack of the features in compare to POI API.*

Download JXL:

If you want to work with JXL, you can download it from this link

<https://sourceforge.net/projects/jexcelapi/files/jexcelapi/2.6.12/>



You can also get demo example inside this zipped file for JXL.

Some of the features:

* JXL is able to read Excel file in Selenium for 95, 97, 2000, XP, 2003 workbook.
* We can work with English, French, Spanish, German.
* Copying a Chart and image insertion in Excel is possible

Drawback:

* We can write Excel 97 and later only (writing in Excel 95 is not supported).
* JXL does not support XLSX format of excel file.
* It Generates spreadsheet in Excel 2000 format.

Before knowing what is Soft Assert, first let’s see what is an Assert and what is the disadvantage in using Assert and why we are moving to Soft Assert.

There are two types of Assert:

1. Hard Assert
2. Soft Assert

When an assert fails the test script stops execution unless handled in some form. We call general assert as Hard Assert

**Hard Assert** – Hard Assert throws an *AssertException* immediately when an assert statement fails and test suite continues with next *@Test*

The disadvantage of Hard Assert – It marks method as fail if assert condition gets failed and the remaining statements inside the method will be aborted.

To overcome this we need to use Soft Assert. Let’s see what is Soft Assert.

**Soft Assert** – Soft Assert collects errors during *@Test*. Soft Assert does not throw an exception when an assert fails and would continue with the next step after the assert statement.

If there is any exception and you want to throw it then you need to use *assertAll()* method as a last statement in the @Test and test suite again continue with next *@Test* as it is.

We need to create an object to use Soft Assert which is not needed in Hard Assert.

Here I took two methods namely *softAssert()*and*hardAssert()*.

In the *softAssert()* method, I have used SoftAssert class and intentionally passing value false in the assertTrue() method to make it fail

In the*hardAssert()*method, I simply used Assert and intentionally passing parameter value false in the assertTrue() method to make it fail