**Selenium Webdriver**

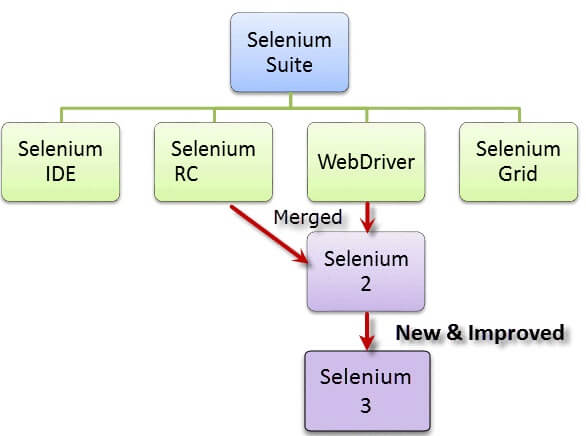
**Introducing Selenium**

Selenium is a set of different software tools each with a different approach to supporting test automation. Most Selenium QA Engineers focus on the one or two tools that most meet the needs of their project, however learning all the tools will give you many different options for approaching different test automation problems. The entire suite of tools results in a rich set of testing functions specifically geared to the needs of testing of web applications of all types. These operations are highly flexible, allowing many options for locating UI elements and comparing expected test results against actual application behavior. One of Selenium’s key features is the support for executing one’s tests on multiple browser platforms.

**Brief History of The Selenium Project**

Selenium first came to life in 2004 when Jason Huggins was testing an internal application at ThoughtWorks. Being a smart guy, he realized there were better uses of his time than manually stepping through the same tests with every change he made. He developed a Javascript library that could drive interactions with the page, allowing him to automatically rerun tests against multiple browsers. That library eventually became Selenium Core, which underlies all the functionality of Selenium Remote Control (RC) and Selenium IDE. Selenium RC was ground-breaking because no other product allowed you to control a browser from a language of your choice.

In 2006 an engineer at Google named Simon Stewart started work on a project he called WebDriver. Google had long been a heavy user of Selenium, but testers had to work around the limitations of the product. Simon wanted a testing tool that spoke directly to the browser using the ‘native’ method for the browser and operating system, thus avoiding the restrictions of a sandboxed Javascript environment. The WebDriver project began with the aim to solve the Selenium’ pain-points.



Selenium runs in [many browsers](https://www.seleniumhq.org/about/platforms.html#browsers) and [operating systems](https://www.seleniumhq.org/about/platforms.html#operating-systems) amd can be can be controlled by many [programming languages](https://www.seleniumhq.org/about/platforms.html#programming-languages) and [testing frameworks](https://www.seleniumhq.org/about/platforms.html#testing-frameworks).

**Selenium IDE,** a Firefox/Chrome add-on that you can only use in creating relatively simple test cases and test suites.

**Selenium Remote Control**, also known as Selenium 1, which is the first Selenium tool that allowed users to use programming languages in creating complex tests.

**WebDriver**, the newer breakthrough that allows your test scripts to communicate directly to the browser, thereby controlling it from the OS level.

**Selenium Grid** is also a tool that is used with Selenium RC/Selenium server to execute parallel tests across different browsers and operating systems.

**Brief Introduction WebDriver**

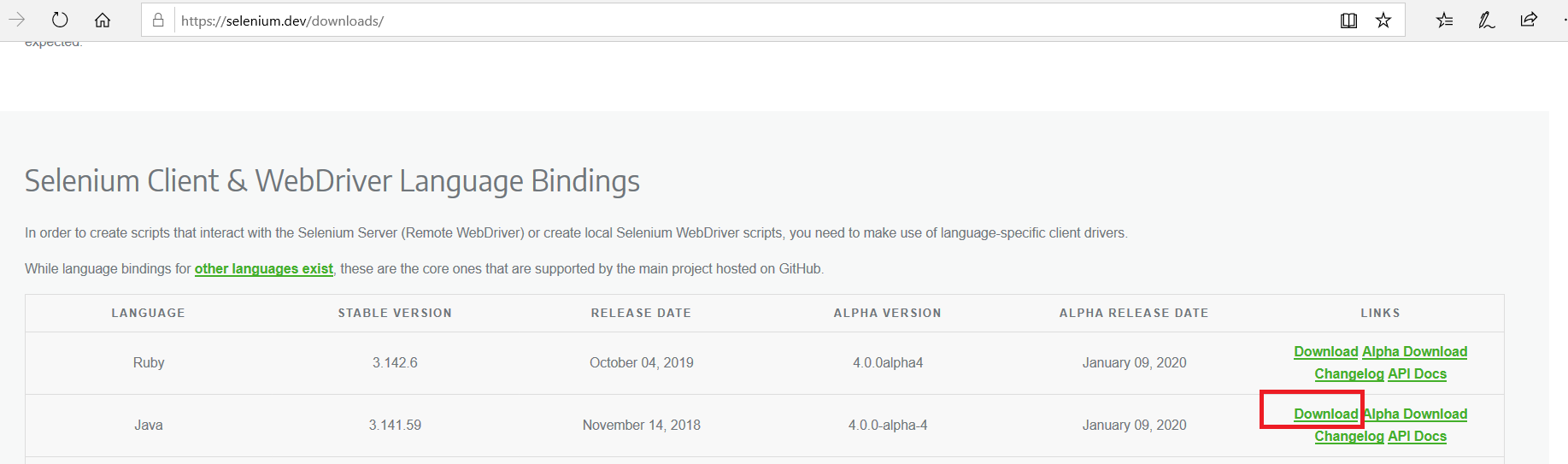
The WebDriver proves itself to be better than both Selenium IDE and Selenium RC in many aspects. It implements a more modern and stable approach in automating the browser's actions. WebDriver, unlike Selenium RC, does not rely on JavaScript for Automation. It controls the browser by directly communicating with it.

The supported languages are the same as those in Selenium .

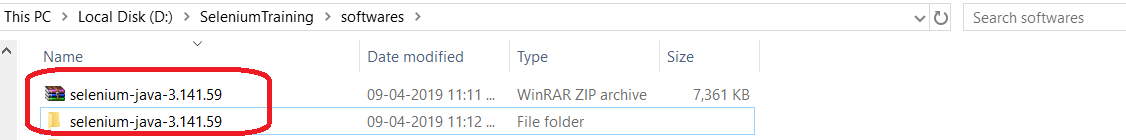
* Java
* C#
* Python
* Perl
* Ruby
* PHP etc

**WebDriver Integration with Eclipse:**

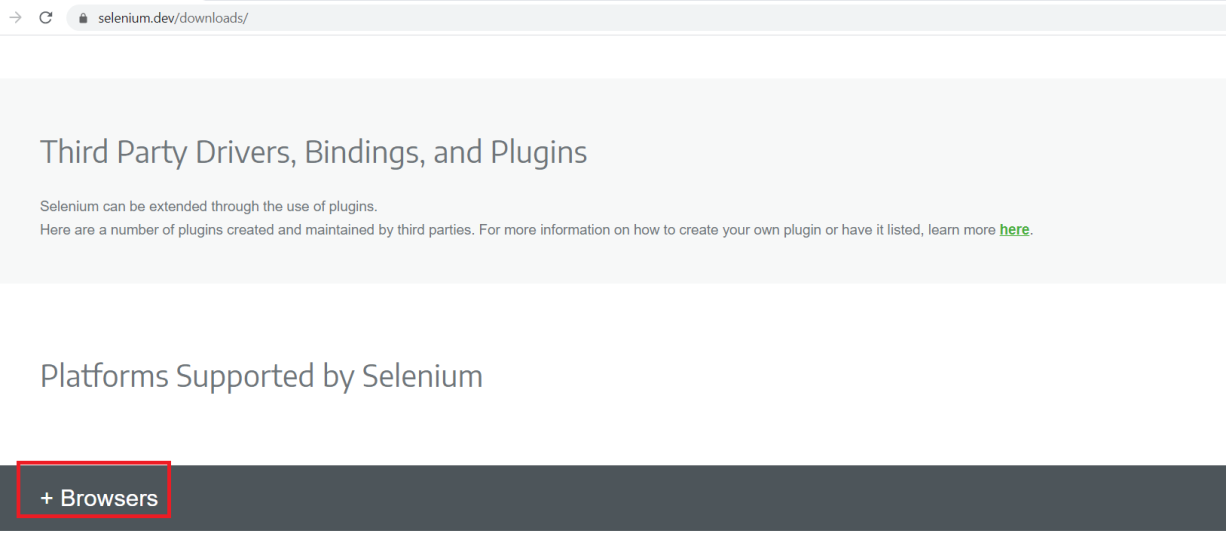
1. Download Webdriver from <https://www.selenium.dev/downloads/> for Java language Bindings to a local folder

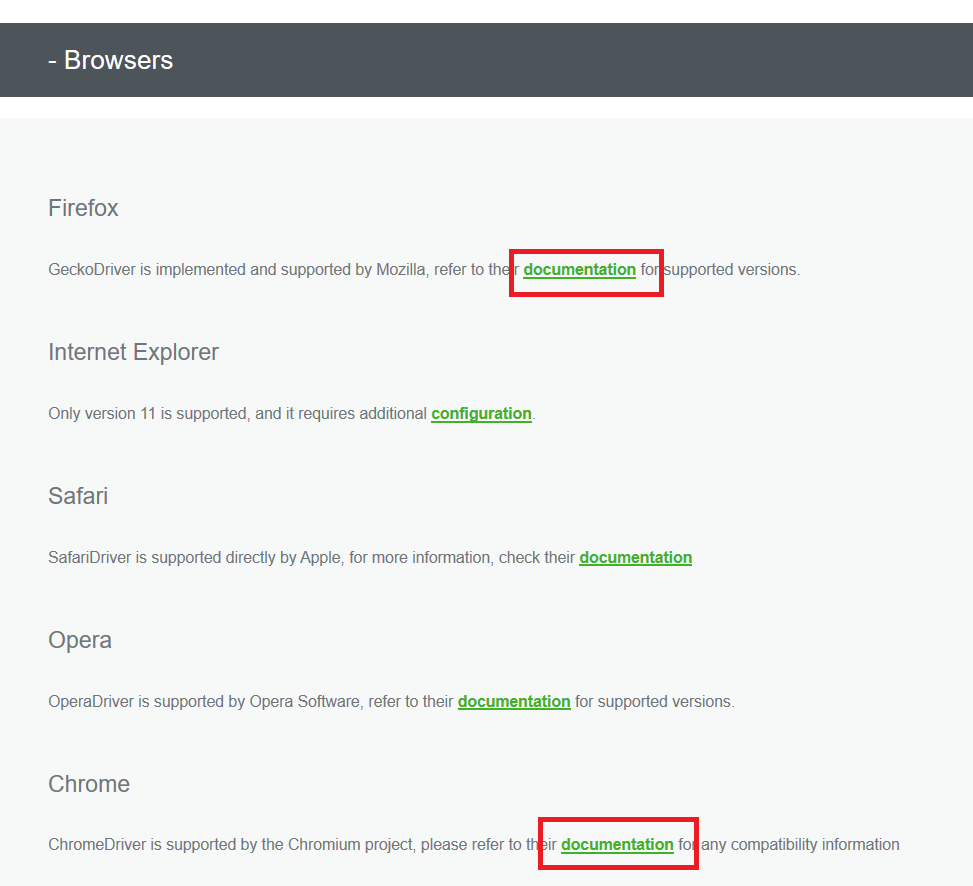


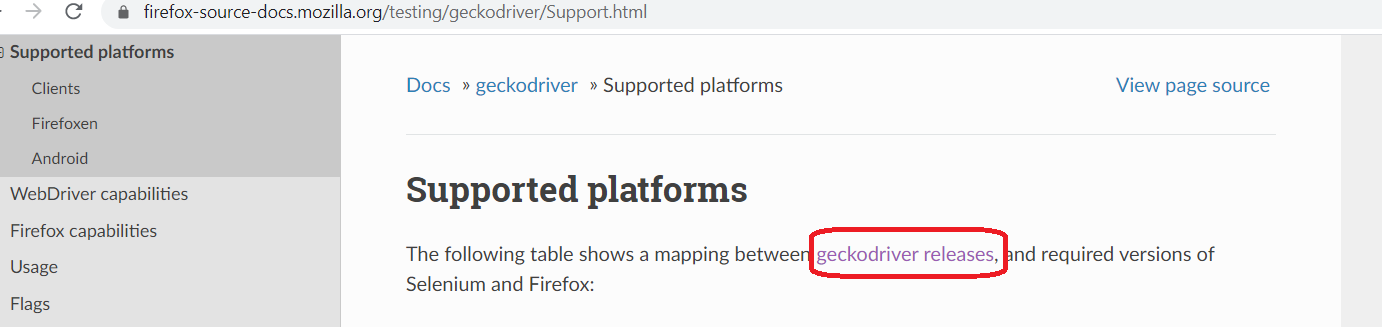
1. Unzip the folder like below



1. Download Mozilla/Firefox Geckodriver and Google Chrome Drivers from <https://www.selenium.dev/downloads/> based on your browser version and bit type(32-bit/64bit)

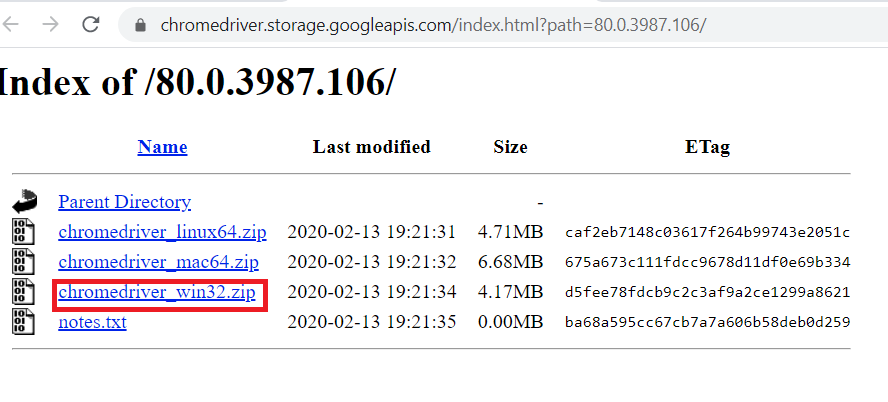




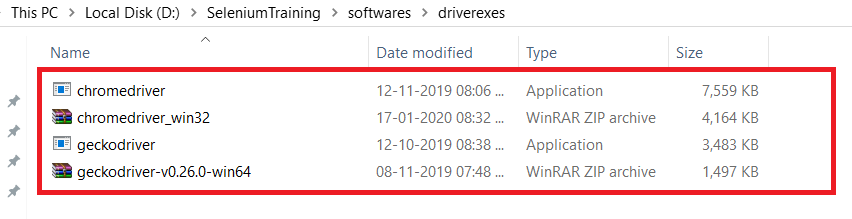


****

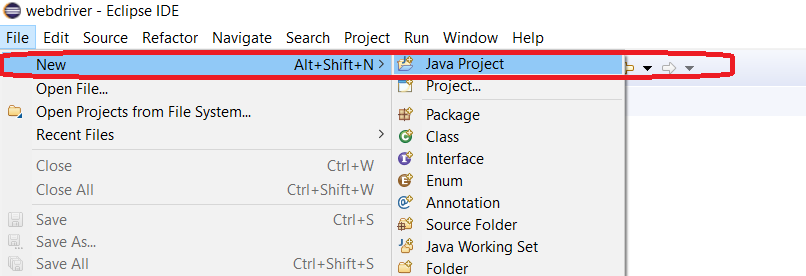
****

****

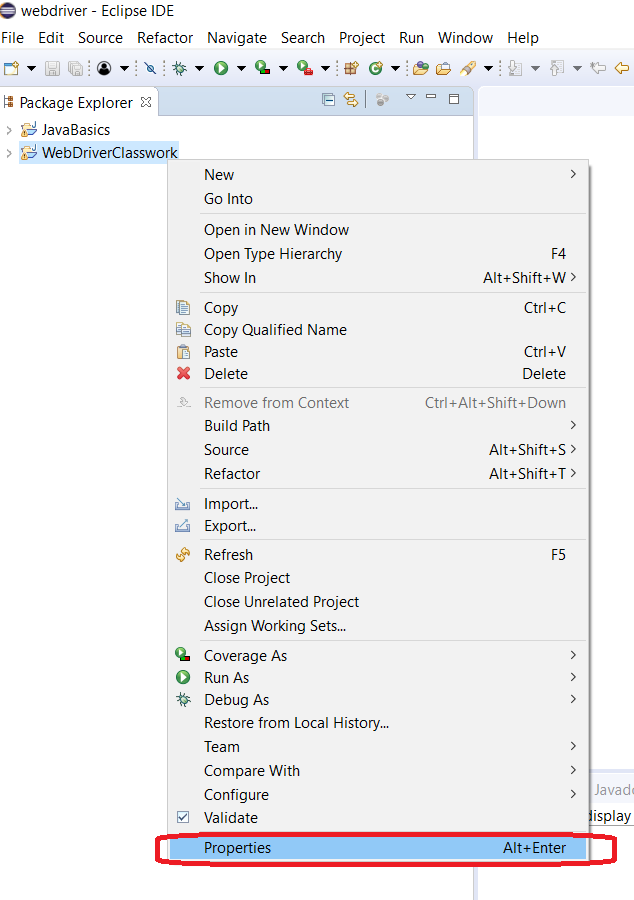
1. Copy and Unzip Firefox Geckodriver and Chrome driver



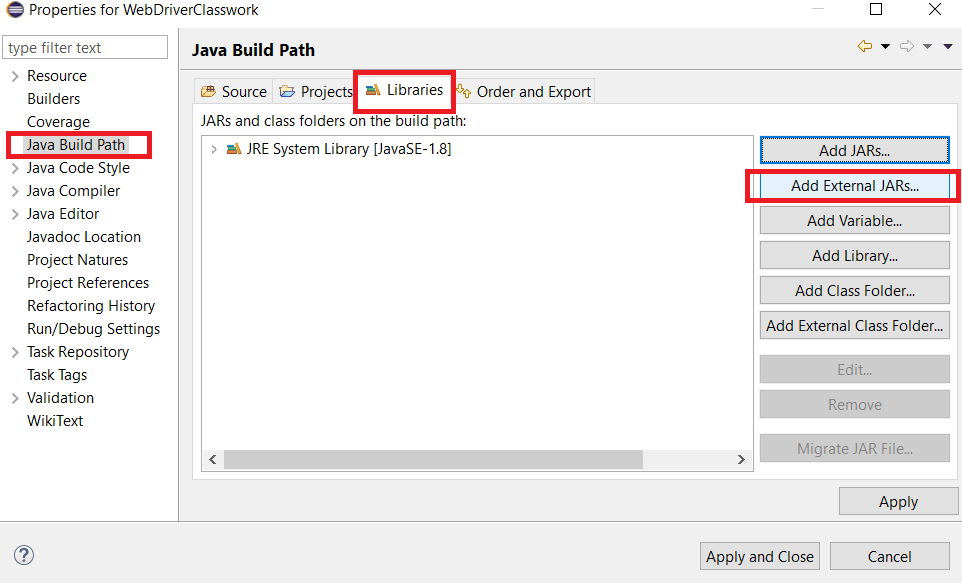
1. Launch Eclipse and Create a Project/Use an exising project



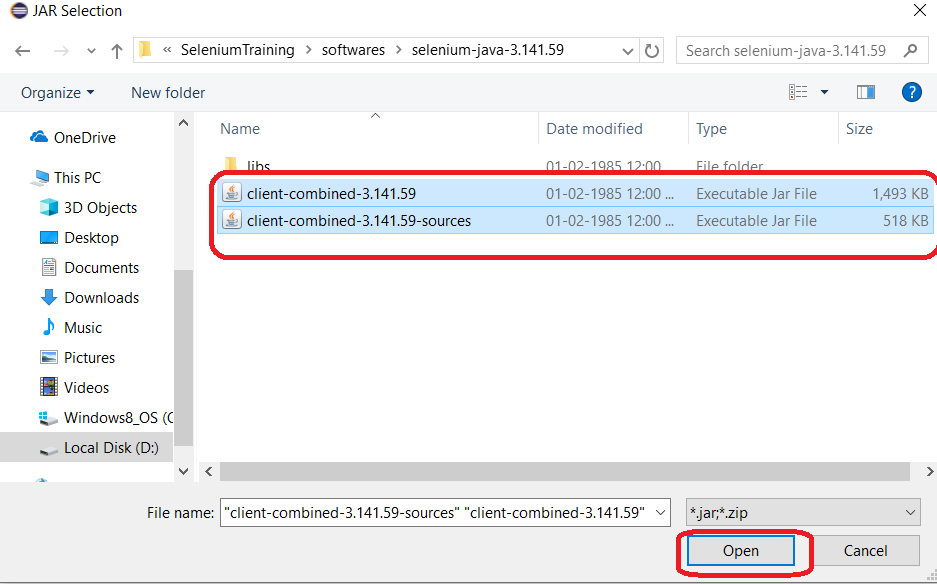
1. Select Project and rightclick to Select “Project Properties”

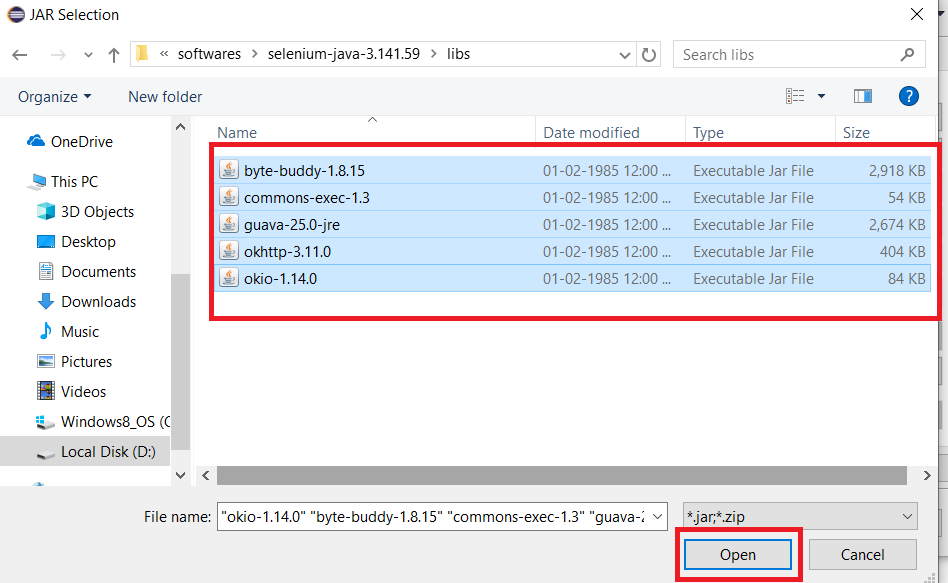


1. In Properties Dialog, Choose JavaBuildPath->Libraries->Add External JARS

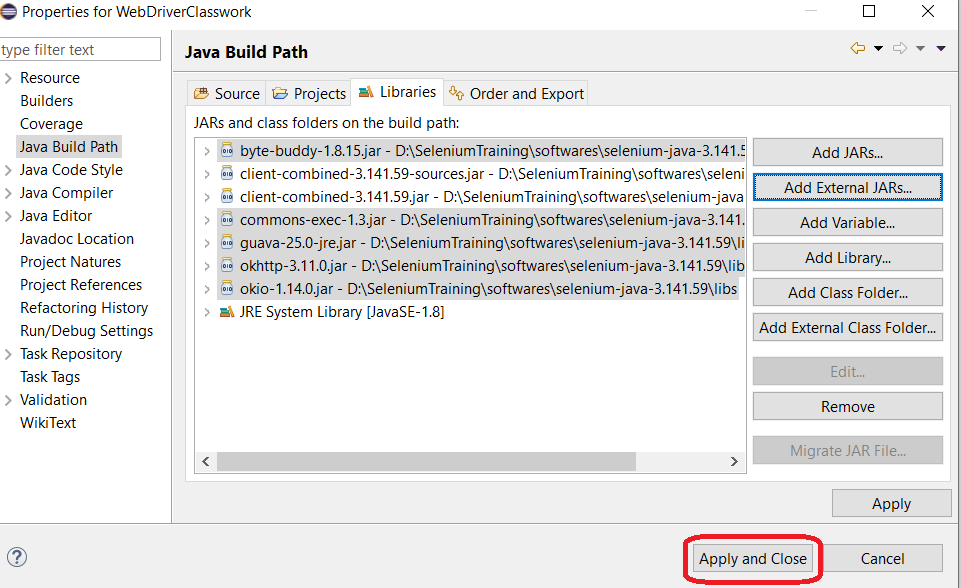


1. Select webdriver download folder and select all Jars and open. Repeat the same for libs folder also





1. Click on Apply and Close on Properties Dialog and your Eclipse is inetegrated with Selenium Webdriver libraries



**Selenium 3 Important Features**

Some of the most important features of Selenium 3 are ..

* Minimum version of Java required to run Selenium 3 is Java 8+
* You would need to use GeckoDriver if you want to run your scripts in Firefox versions greater than 47.0.1. Firefox 47.0.1 and before would not need GeckoDriver
* Apple has come up with its own SafariDriver to let you run your tests in Safari on Mac

**WebDriver Interface**

The main interface to use for testing, which represents an idealised web browser. The methods in this interface fall into three categories:

* Control of the browser itself
* Selection of [WebElement](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/WebElement.html)s
* Debugging aids

Key methods are [get(String)](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/WebDriver.html#get-java.lang.String-), which is used to load a new web page, and the various methods similar to [findElement(By)](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/WebDriver.html#findElement-org.openqa.selenium.By-), which is used to find [WebElement](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/WebElement.html)s.

Note that all methods that use XPath to locate elements will throw a RuntimeException should there be an error thrown by the underlying XPath engine.

Useful Methods

|  |  |  |
| --- | --- | --- |
| **Modifier and Type** | **Method** | **Description** |
| void | close() | Close the current window, quitting the browser if it's the last window currently open. |
| WebElement | findElement(**By** by) | Find the first WebElement using the given method. |
| java.util.List<WebElement> | findElements(**By** by) | Find all elements within the current page using the given mechanism. |
| void | get(java.lang.String url) | Load a new web page in the current browser window. |
| java.lang.String | getCurrentUrl() | Get a string representing the current URL that the browser is looking at. |
| java.lang.String | getPageSource() | Get the source of the last loaded page. |
| java.lang.String | getTitle() | The title of the current page. |
| java.lang.String | getWindowHandle() | Return an opaque handle to this window that uniquely identifies it within this driver instance. |
| java.util.Set<java.lang.String> | getWindowHandles() | Return a set of window handles which can be used to iterate over all open windows of this WebDriver instance by passing them to switchTo().WebDriver.Options.window() |
| WebDriver.Options | manage() | Gets the Option interface |
| WebDriver.Navigation | navigate() | An abstraction allowing the driver to access the browser's history and to navigate to a given URL. |
| void | quit() | Quits this driver, closing every associated window. |
| WebDriver.TargetLocator | switchTo() | Send future commands to a different frame or window. |

Usage :<webdriver>.<method()>

Example: driver.quit();

**Webdriver Navigation**

Webdriver navigation used to perform browser navigation functions like back, refresh, forward etc.

|  |  |  |
| --- | --- | --- |
| **Modifier and Type** | **Method and** | **Description** |
| void | back() | Move back a single "item" in the browser's history. |
| void | [forward()](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/WebDriver.Navigation.html#forward--) | Move a single "item" forward in the browser's history. |
| void | [refresh()](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/WebDriver.Navigation.html#refresh--) | Refresh the current page |
| void | [to(java.lang.String url)](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/WebDriver.Navigation.html#to-java.lang.String-) | Load a new web page in the current browser window. |
| void | [to(java.net.URL url)](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/WebDriver.Navigation.html#to-java.net.URL-) | Overloaded version of to(String) that makes it easy to pass in a URL. |

Usage: <webdriver>.navigate().<methodname()>;

Eg: driver.navigate().foreward();

**Webdriver Options**

An interface for managing stuff you would do in a browser menu.

Useful Methods

|  |  |  |
| --- | --- | --- |
| **Modifier and Type** | **Method** | **Description** |
| void | addCookie(**Cookie** cookie) | Add a specific cookie. |
| void | [deleteAllCookies()](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/WebDriver.Options.html#deleteAllCookies--) | Delete all the cookies for the current domain. |
| void | deleteCookie(**Cookie** cookie) | Delete a cookie from the browser's "cookie jar". |
| void | [deleteCookieNamed(java.lang.String name)](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/WebDriver.Options.html#deleteCookieNamed-java.lang.String-) | Delete the named cookie from the current domain. |
| [Cookie](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/Cookie.html) | [getCookieNamed(java.lang.String name)](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/WebDriver.Options.html#getCookieNamed-java.lang.String-) | Get a cookie with a given name. |
| [java.util.Set<Cookie>](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/Cookie.html) | [getCookies()](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/WebDriver.Options.html#getCookies--) | Get all the cookies for the current domain. |

Usage: <webdriver>.manage().<methodname()>;

Eg: driver.manage().deleteAllCookies();

**Webdriver Window Options**

An interface for managing the current window.

Useful Methods

|  |  |  |
| --- | --- | --- |
| **Modifier and Type** | **Method** | **Description** |
| void | fullscreen() | Fullscreen the current window if it is not already fullscreen |
| [Point](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/Point.html) | getPosition() | Get the position of the current window, relative to the upper left corner of the screen. |
| [Dimension](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/Dimension.html) | getSize() | Get the size of the current window. |
| void | maximize() | Maximizes the current window if it is not already maximized |
| void | setPosition(**Dimension** targetPosition) | Set the position of the current window. |

Usage: <webdriver>.manage().window().<methodname()>;

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

**Webdriver Timeout Options**

An interface for managing timeout behavior for WebDriver instances.Useful Methods

Useful Methods

|  |  |  |
| --- | --- | --- |
| **Modifier and Type** | **Method** | **Description** |
| [WebDriver.Timeouts](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/WebDriver.Timeouts.html) | implicitlyWait(long time, java.util.concurrent.TimeUnit unit) | Specifies the amount of time the driver should wait when searching for an element if it is not immediately present. |
| [WebDriver.Timeouts](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/WebDriver.Timeouts.html) | pageLoadTimeout(long time, java.util.concurrent.TimeUnit unit) | Sets the amount of time to wait for a page load to complete before throwing an error. |
| [WebDriver.Timeouts](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/WebDriver.Timeouts.html) | setScriptTimeout(long time, java.util.concurrent.TimeUnit unit) | Sets the amount of time to wait for an asynchronous script to finish execution before throwing an error. |

Usage: <Wedriver>.manage().timeouts().implicitlyWait(TimeOut, TimeUnit.SECONDS);

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

**Webdriver switchTo/ Target loacator options**

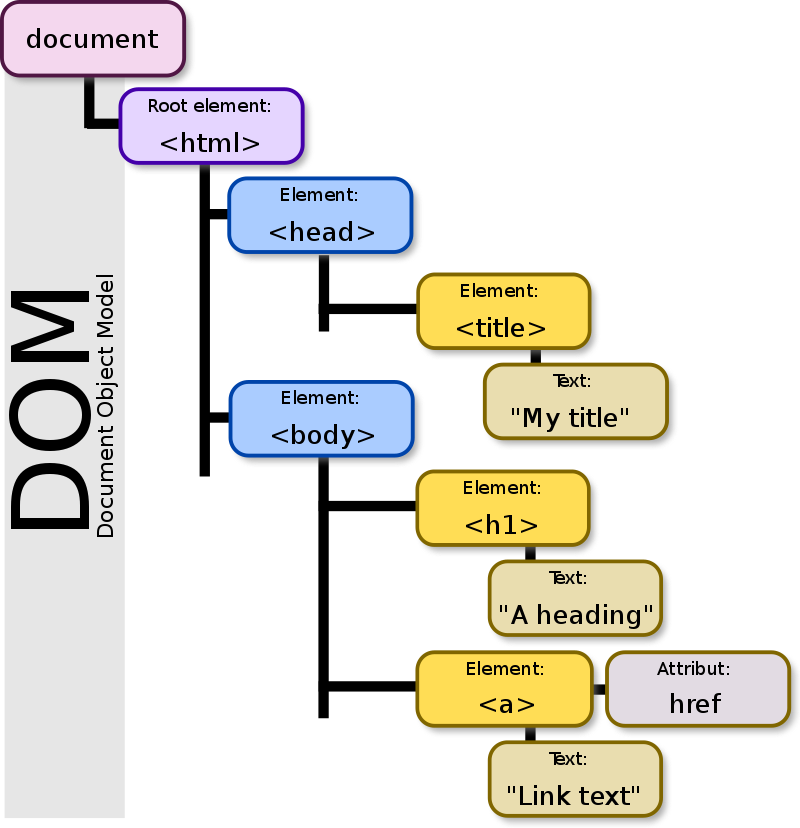
Used to locate a given frame or window.

Useful Methods

|  |  |  |
| --- | --- | --- |
| **Modifier and Type** | **Method** | **Description** |
| [WebElement](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/WebElement.html) | activeElement() | Switches to the element that currently has focus within the document currently "switched to", or the body element if this cannot be detected. |
| [Alert](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/Alert.html) | [alert()](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/WebDriver.TargetLocator.html#alert--) | Switches to the currently active modal dialog for this particular driver instance. |
| [WebDriver](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/WebDriver.html) | [defaultContent()](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/WebDriver.TargetLocator.html#defaultContent--) | Selects either the first frame on the page, or the main document when a page contains iframes. |
| [WebDriver](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/WebDriver.html) | [frame(int index)](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/WebDriver.TargetLocator.html#frame-int-) | Select a frame by its (zero-based) index. |
| [WebDriver](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/WebDriver.html) | [frame(java.lang.String nameOrId)](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/WebDriver.TargetLocator.html#frame-java.lang.String-) | Select a frame by its name or ID. |
| [WebDriver](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/WebDriver.html) | frame(**WebElement** frameElement) | [Select a frame using its previously located WebElement.](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/WebElement.html) |
| [WebDriver](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/WebDriver.html) | [parentFrame()](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/WebDriver.TargetLocator.html#parentFrame--) | Change focus to the parent context. |
| [WebDriver](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/WebDriver.html) | [window(java.lang.String nameOrHandle)](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/WebDriver.TargetLocator.html#window-java.lang.String-) | Switch the focus of future commands for this driver to the window with the given name/handle. |

**Document Object Model (DOM)**

The **Document Object Model (DOM)** is a [cross-platform](https://en.wikipedia.org/wiki/Cross-platform) and [language](https://en.wikipedia.org/wiki/Programming_language)-independent [application programming interface](https://en.wikipedia.org/wiki/Application_programming_interface) that treats an [HTML](https://en.wikipedia.org/wiki/HTML), [XHTML](https://en.wikipedia.org/wiki/XHTML), or [XML](https://en.wikipedia.org/wiki/XML) document as a [tree structure](https://en.wikipedia.org/wiki/Tree_structure) wherein each [node](https://en.wikipedia.org/wiki/Node_(computer_science)) is an [object](https://en.wikipedia.org/wiki/Object_(computer_science)) representing a part of the document. The DOM represents a document with a logical tree. Each branch of the tree ends in a node, and each node contains objects.



**What is Locator?**

The locator can be termed as an address that identifies a web element uniquely within the webpage. Locators are the HTML properties of a web element which tells the Selenium about the web element it needs to perform the action on.

**WebElement Interface**

Represents an HTML element. Generally, all interesting operations to do with interacting with a page will be performed through this interface.

All method calls will do a freshness check to ensure that the element reference is still valid. This essentially determines whether or not the element is still attached to the DOM. If this test fails, then StaleElementReferenceException is thrown, and all future calls to this instance will fail.

The most common WebElements are:

* Text box
* Button
* Drop Down
* Hyperlink
* Check Box
* Radio Button
* TextArea

Etc

Useful Methods

|  |  |  |
| --- | --- | --- |
| **Modifier and Type** | **Method** | **Description** |
| void | clear() | If this element is a text entry element, this will clear the value. |
| void | click() | Click this element. |
| WebElement | findElement(**By** by) | Find the first WebElement using the given method. |
| java.util.List<WebElement> | findElements(**By** by) | Find all elements within the current context using the given mechanism. |
| java.lang.String | getAttribute(java.lang.String name) | Get the value of the given attribute of the element. |
| java.lang.String | getCssValue(java.lang.String propertyName) | Get the value of a given CSS property. |
| Point | getLocation() | Where on the page is the top left-hand corner of the rendered element? |
| Rectangle | getRect() | Returns the location and size of the rendered element |
| Dimension | getSize() | Returns the size of the element on the page. |
| java.lang.String | getTagName() | Get the tag name of this element. |
| java.lang.String | getText() | Get the visible (i.e. not hidden by CSS) text of this element, including sub-elements. |
| boolean | isDisplayed() | Is this element displayed or not? This method avoids the problem of having to parse an element's "style" attribute. |
| boolean | isEnabled() | Is the element currently enabled or not? This will generally return true for everything but disabled input elements. |
| boolean | isSelected() | Determine whether or not this element is selected or not. |
| void | sendKeys(java.lang.CharSequence... keysToSend) | Use this method to simulate typing into an element, which may set its value. |
| void | submit() | If this current element is a form, or an element within a form, then this will be submitted to the remote server. |

Example:

WebElement usernameElmnt= driver.findElement(By.id(“username”));

usernameElmnt.click();

**Class By**

Mechanism used to locate elements within a document. In order to create your own locating mechanisms, it is possible to subclass this class and override the protected methods as required, though it is expected that all subclasses rely on the basic finding mechanisms provided through static methods of this class

Useful Methods

|  |  |  |
| --- | --- | --- |
| **Modifier and Type** | **Method** | Description |
| static By | className(java.lang.String className) | Find elements based on the value of the "class" attribute. |
| static By | cssSelector(java.lang.String cssSelector) | Find elements via the driver's underlying W3C Selector engine. |
| static By | id(java.lang.String id) | Find elements based on the value of the "id" attribute. |
| static By | linkText(java.lang.String linkText) | Find elements based on the value of the "linkText" attribute. |
| static By | name(java.lang.String name) | Find elements based on the value of the "name" attribute. |
| static By | partialLinkText(java.lang.String partialLinkText) | Find elements based on the value of the "Partial LinkText" attribute. |
| static By | tagName(java.lang.String tagName) | Find elements based on the value of the "tagName" attribute. |
| static By | xpath(java.lang.String xpathExpression) | Find elements based on the value of the "xpath" |

Example:

driver.findElement(By.className(“textboxcolor”));

driver.findElement(By.id(“username”));

driver.findElement(By.name("password")).sendKeys("mypassword");

driver.findElement(By.linkText("Sign In")).click();

**FindElements**

Find Elements method takes in By object as the parameter and returns the list of web elements. It returns an empty list if there are no elements found using the given locator.

Syntax:

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

Example:

List<WebElement> inputList = driver.findElements(By.xpath("//input"));

Differernces between FindElement and FindEelemnts

|  |  |
| --- | --- |
| **Find Element** | **Find Elements** |
| Returns the first most web element if there are multiple web elements found with the same locator | Returns a list of web elements |
| Throws exception NoSuchElementException if there are no elements matching the locator strategy | Returns an empty list if there are no web elements matching the locator strategy |
| It will only find one web element | It will find a collection of elements whose match the locator strategy. |
| Not Applicable | Each Web element is indexed with a number starting from 0 just like an array |

**CSS Selector as a Locator:**

CSS Selector is the combination of an element selector and a selector value which identifies the web element within a web page. The composite of element selector and selector value is known as Selector Pattern.

Selector Pattern is constructed using HTML tags, attributes and their values. The central theme behind the procedure to create CSS Selector and Xpath are very much similar underlying the only difference in their construction protocol.

CSS Selector: ID

In this sample, we would access “Username” text box present in the login form as shown below.

The Username textbox has an ID attribute whose value is defined as “username”. Thus ID attribute and its value can be used to create CSS Selector to access the email textbox.



Locate/inspect the web element (“Username” textbox ) and notice that the HTML tag is “input” and value of ID attribute is “username” and both of them collectively make a reference to the “Username Textbox”. Hence the above data would be used to create CSS Selector.

Syntax:

css=<HTML tag><#><Value of ID attribute>

* **HTML tag** – It is the tag which is used to denote the web element which we want to access.
* **#**– The hash sign is used to symbolize ID attribute. It is mandatory to use hash sign if ID attribute is being used to create CSS Selector.
* **Value of ID attribute** – It is the value of an ID attribute which is being accessed.
* The value of ID is always preceded by a hash sign.

Example: css for username is input#username

CSS Selector: Class

Locating an element using Class as a CSS Selector is very much similar to using ID, the lone difference lies in their syntax formation.

Syntax:

css=<HTML tag><.><Value of Class attribute>

* **.**– The dot sign is used to symbolize Class attribute. It is mandatory to use dot sign if a Class attribute is being used to create CSS Selector.
* **The value of Class** is always preceded by a dot sign.

Example: css for username is input.textboxcolor

CSS Selector: Attribute

Locating an element using Attribute as a CSS Selector is very much similar to using ID/Class, the lone difference lies in their syntax formation.

Syntax:css=<HTML tag><[attribute=Value of attribute]>

* **Attribute** – It is the attribute we want to use to create CSS Selector. It can value, type, name etc. It is recommended to choose an attribute whose value uniquely identifies the web element.
* **Value of attribute** – It is the value of an attribute which is being accessed.

Example: css for username is input[type=’text’]

CSS Selector: ID/Class and attribute

ID /Class attribute, type attribute and their values can be used to create CSS Selector to access the designated web element.

Syntax:

css=<HTML tag><. Or #><value of Class or ID attribute><[attribute=Value of attribute]>

css for username is input#username[type=’text’]

css for username is input.textboxcolor[type=’text’]

Two or more attributes can also be furnished for exampe:

Example: css=input.textboxcolor[type=’text’][name=’username’]

CSS Selector: Sub-string

CSS in Selenium allows matching a partial string and thus deriving a very interesting feature to create CSS Selectors using substrings. There are three ways in which CSS Selectors can be created based on the mechanism used to match the substring.

**Types of mechanisms**

All the underneath mechanisms have symbolic significance.

* Match a prefix
* Match a suffix
* Match a sub string

***Match a prefix***

It is used to correspond to the string with the help of a matching prefix.

Syntax

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

* **^**– Symbolic notation to match a string using prefix.
* **Prefix**– It is the string based on which match operation is performed. The likely string is expected to start with the specified string.

Example: css for username is input[name^=’user’]

***Match a suffix***

It is used to correspond to the string with the help of a matching suffix.

Syntax

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

* **$**– Symbolic notation to match a string using suffix.
* **The suffix** – It is the string based on which match operation is performed. The likely string is expected to ends with the specified string.

css for username is input[name$=’name’]

***Match a sub string***

It is used to correspond to the string with the help of a matching sub string.

Syntax

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

* **\*** – Symbolic notation to match a string using sub string.
* **Sub string** – It is the string based on which match operation is performed. The likely string is expected to have the specified string pattern.

css for username is input[name\*=’ser’]

**Using XPath as a Locator**

Xpath is used to locate a web element based on its XML path. XML stands for Extensible Markup Language and is used to store, organize and transport arbitrary data. It stores data in a key-value pair which is very much similar to HTML tags. Both being the markup languages and since they fall under the same umbrella, xpath can be used to locate HTML elements.

The fundamental behind locating elements using Xpath is the traversing between various elements across the entire page and thus enabling a user to find an element with the reference of another element.

***Xpath can be created in two ways:***

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.

The key characteristic of XPath is that it begins with the single forward slash(/) ,which means you can select the element from the root node.

For example: /html/body/table/tbody/tr/td/input[@id=’username’]

* The success rate of finding an element using Xpath is too high. Xpath can find relatively all the elements on a web page. Thus, Xpaths can be used to locate elements having no id, class or name.
* Creating a valid Xpath is a tricky and complex process.
* While creating xpath, the user should be aware of the various rules and protocols.

Relative Xpath

For Relative Xpath the path starts from the middle of the HTML DOM structure. It starts with the double forward slash (//), which means it can search the element anywhere at the webpage.

You can start from the middle of the HTML DOM structure and no need to write long xpath.

Syntax:

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.

Example: //input[@class= ‘textboxcolor’]

**XPath axes and Functions**

XPath axes search different nodes in XML/HTML 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 like child, parent, ancestor, sibling, preceding, self, etc.

Basic XPath

XPath expression select nodes or list of nodes on the basis of attributes like **ID , Name, Classname**, etc.

Contains()

Contains() is a method used in XPath expression. It is used when the value of any attribute changes dynamically.The contain feature has an ability to find the element with partial text

Example: Xpath=//\*[contains(@type,' text')]

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 AND expression, two conditions are used, both conditions should be true to find the element. It fails to find element if any one condition is false.

Example: Xpath=//input[@type='text' and @name='username']

Xpath=//input[@type='text' or @name='username']

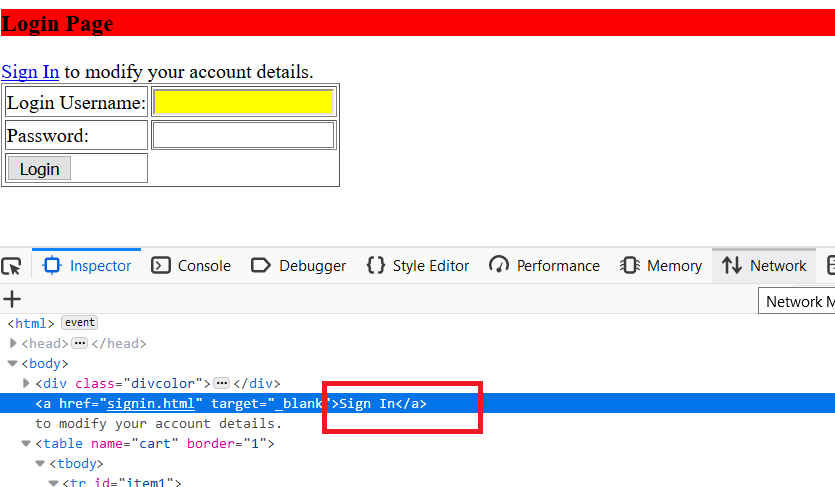
Starts-with function

Start-with function finds the element whose attribute value changes on refresh or any operation on the webpage.In this expression, match the starting text of the attribute is used to find the element whose attribute changes dynamically. You can also find the element whose attribute value is static (not changes).

Example: Xpath=// input[starts-with(@id,'user')]

Text()

In this expression, with text function, we find the element with exact text match as shown below.



Example: Xpath=//a[text()='Sign In']

*Note:* *.(dot) also used to read text from the webelement in place of text().*

Xpath=//a[.='Sign In']

Not()

not() is a used to negate the attribute value. It is used when the value of any attribute changes dynamically and you may be looking for attribute value shall not contain . This can be used with any conjunction of other axes like not(contains()) etc

Example: //input[not(@type='hidden')]

As per above xpath expression, it selects input boxes on page which does not have type ‘hidden’.

*Note:* *not() expects your expression to be in paranthesis. So not() is always followed by paranthesis.*

 Following

Selects all elements in the document of the current node( )

Example: Xpath=//td[text()='Login Username:']//following::input

Example: Xpath=//td[text()='Login Username:']/following::input[1]

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

Following-sibling

Select the following sibling of the context node.

Example: Xpath = //td[text()='Login Username:']/following-sibling::td/input[@id='username']

Preceding

Select all nodes that come before the current node.

In the below expression, it identifies all the td elements elements before username input element

Example: Xpath = //input[@id='username']/preceding::td

Ancestor

The ancestor axis selects all ancestors element (grandparent, parent, etc.) of the current node .

Example: Xpath=//input[@id='username']/ancestor::tr

Parent

Selects the parent of the current node.

Example: Xpath = //input[@id='username']/parent::td

*Note:* ***..(dot dot)*** *also used to refer to immediate parent of the webelemnt*

Example: Xpath = //input[@id='username']/..

Self

Selects the current node or 'self' means it indicates the node

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

*Note: In above xpath \* represents any element type*

Child

Selects all children elements of the current node .

Example: Xpath=//td/child::input

Descendant

Selects the descendants of the current node.

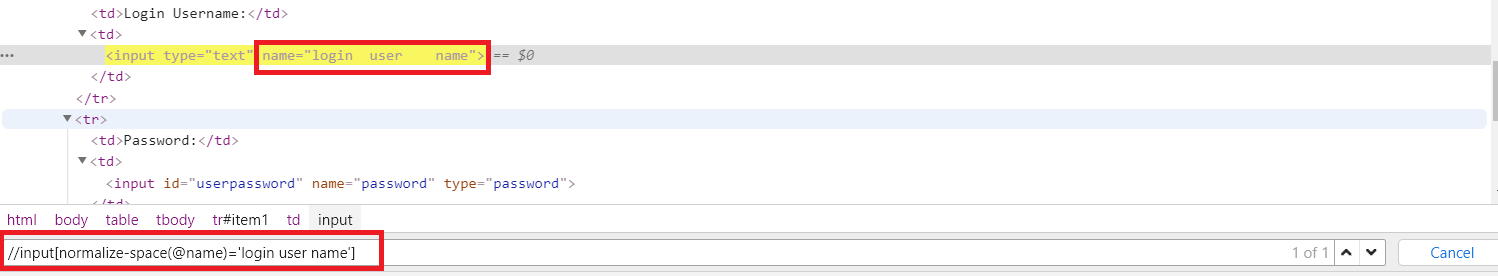
In the below expression, it identifies all the element descendants to current element ( tr) which means down under the node (child node , grandchild node, etc.).

Example: xpath = //tr/descendant::input

*Note: If multiple elements selected for any xpath, you can select the particular element by using its index like [1],[2] etc. Here 1 represent first element displayed, 2 resprenents 2nd element displayed etc. And this can be used with all above axes and element types.*

Normalize-sapce function

The normalize-space function strips leading and trailing white-space from a string, replaces sequences of whitespace characters by a single space, and returns the resulting string. This also removes all new lines and tabs present in a string



Eg: //input[normalize-space(@name)=’login user name’]

**WebElements and common Methods**

Input Box

Input box generally availble in 2 types, one is of Type Text and other is of type password.

Useful methods with Examples

WebElement userId = driver.findElement(By.xpath(“//input[@name='userName']”));

click() – to click on input box

Example: userId.click();

clear() – to clear the existing content from input box

Example: userId.clear();

sendKeys(char[]) - to write a value to Text

Example: userId.sendKeys(“admin”);

getAttribute(<property>) - returns the attribute associated with the webelement

Example: String attributeValue = userId.getAttribute("value");

Button/Image/Link

The buttons/Images/Links can be accessed using the click() method.

Useful methods with Examples

WebElement SignIn = driver.findElement(By.xpath(“//button[@name='SignIn']”));

click() – to click on Button/Link/Image

Example: SignIn.click();

getAttribute(<property>) - returns the attribute associated with the webelement

Example: String textValue = SignIn.getAttribute("text");

Submit Button

Submit buttons are used to submit the entire form to the server. We can either use the click () method on the web element like a normal button as we have done above or use the submit () method on any web element in the form or on the submit button itself.

When submit() is used, WebDriver will look up the DOM to know which form the element belongs to, and then trigger its submit function.

Useful methods with Examples

WebElement SignIn = driver.findElement(By.xpath(“//button[@type='Submit']”));

click() – to click on submit button

Example: SignIn.click();

submit () – to submit the form

Example: SignIn.submit();

getAttribute(<property>) - returns the attribute associated with the webelement

Example: String textValue = SignIn.getAttribute("type");

Radio Button

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

WebElement roundtrip = driver.findElement(By.xpath(“//input[(@type='radio') and (@value=’roundtrip’)]”));

click() – to click on submit button

Example: roundtrip.click();

getAttribute(<property>) also can be used to get any attribute value.

Check Box

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

isSelected() method is used to know whether the Checkbox is toggled on or off. Returns true if the checkbox is selected, else return false.

Example:

WebElement checkMeanType = driver.findElement(By.xpath(“//input[(@type='checkbox') and (@value=’chinese’)]”));

checkMeanType.click();

checkMeanType.isSelected(); – this returns true if check box selected, else returns false

getAttribute(<property>) also can be used to get any attribute value.

Select Option /Drop-Down Box

For handling dropdowns, Selenium already provides [Select class](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/support/ui/Select.html) that has some predefined method which help is a lot while working with Dropdown.

Select class can be find in below package. So you must import this to use for select.

**org.openqa.selenium.support.ui.Select**

Example: Select mealSelect = new Select(driver.findElement(By.xpath("//select[contains(@name,'meal')]")));

mealSelect.selectByVisibleText("Hindu");

Commonly used methods:

|  |  |
| --- | --- |
| **Method** | **Purpose** |
| *boolean isMultiple()* | Whether this select element supports selecting multiple options at the same time? This is done by checking the value of the "multiple" attribute. |
| java.util.List<WebElement> getOptions() | All selected options belonging to this select tag |
| WebElement getFirstSelectedOption() | The first selected option in this select tag (or the currently selected option in a normal select). |
| void selectByVisibleText(java.lang.String text) | Select all options that display text matching the argument |
| void selectByIndex(int index) | Select the option at the given index. This is done by examining the "index" attribute of an element, and not merely by counting. |
| void selectByValue(java.lang.String value) | Select all options that have a value matching the argument. |
| void deselectAll() | Clear all selected entries. This is only valid when the SELECT supports multiple selections. |
| void deselectByValue(java.lang.String value) | Deselect all options that have a value matching the argument. |
| void deselectByIndex(int index) | Deselect the option at the given index. This is done by examining the "index" attribute of an element, and not merely by counting. |
| void deselectByVisibleText(java.lang.String text) | Deselect all options that display text matching the argument. |

Text Area

Textarea used to provide multiple lines of text.

Useful methods with Examples

WebElement userId = driver.findElement(By.xpath(“//input[@name='address']”));

click() – to click on TextArea box

Example: userId.click();

clear() – to clear the existing content from TextArea box

Example: userId.clear();

sendKeys(char[]) - to write a value to Textarea

Example: userId.sendKeys(“adress1 \n Address 2 \n City \n State”);

Note: Here “\n” represnts new Line

getAttribute(<property>) - returns the attribute associated with the webelement

Example: String attributeValue = userId.getAttribute("value");

**Handling KeyBoard and Mouse Events with Actions**

Handling keyboard and mouse events are done using the **Advanced User Interactions API**. It contains the **Actions** class and the **Action** interface that are needed when executing these events. The following are the most commonly used keyboard and mouse events provided by the Actions class.

Class Actions

The user-facing API for emulating complex user gestures. Use this class rather than using the Keyboard or Mouse directly.Implements the builder pattern: Builds a CompositeAction containing all actions specified by the method calls.

|  |  |  |
| --- | --- | --- |
| **Modifier and Type** | **Method Name** | **Description** |
| [Action](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Action.html) | [build()](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html#build--) | Generates a composite action containing all actions so far, ready to be performed (and resets the internal builder state, so subsequent calls to build() will contain fresh sequences). |
| [Actions](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html) | [click()](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html#click--) | Clicks at the current mouse location. |
| [Actions](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html) | click(WebElement target) | Clicks in the middle of the given element. |
| [Actions](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html) | [clickAndHold()](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html#clickAndHold--) | Clicks (without releasing) at the current mouse location. |
| [Actions](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html) | clickAndHold(WebElement target) | Clicks (without releasing) in the middle of the given element. |
| [Actions](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html) | [contextClick()](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html#contextClick--) | Performs a context-click at the current mouse location. |
| [Actions](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html) | contextClick(WebElement target) | Performs a context-click at middle of the given element. |
| [Actions](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html) | [doubleClick()](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html#doubleClick--) | Performs a double-click at the current mouse location. |
| [Actions](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html) | doubleClick(WebElement target) | Performs a double-click at middle of the given element. |
| [Actions](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html) | dragAndDrop(WebElement source, WebElement target) | A convenience method that performs click-and-hold at the location of the source element, moves to the location of the target element, then releases the mouse. |
| [Actions](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html) | dragAndDropBy(WebElement source, int xOffset, int yOffset) | A convenience method that performs click-and-hold at the location of the source element, moves by a given offset, then releases the mouse. |
| [Actions](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html) | [keyDown(java.lang.CharSequence key)](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html#keyDown-java.lang.CharSequence-) | Performs a modifier key press. |
| [Actions](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html) | keyDown(WebElement target, java.lang.CharSequence key) | Performs a modifier key press after focusing on an element. |
| [Actions](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html) | [keyUp(java.lang.CharSequence key)](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html#keyUp-java.lang.CharSequence-) | Performs a modifier key release. |
| [Actions](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html) | keyUp(WebElement target, java.lang.CharSequence key) | Performs a modifier key release after focusing on an element. |
| [Actions](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html) | [moveByOffset(int xOffset, int yOffset)](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html#moveByOffset-int-int-) | Moves the mouse from its current position (or 0,0) by the given offset. |
| [Actions](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html) | moveToElement(WebElement target) | Moves the mouse to the middle of the element. |
| [Actions](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html) | moveToElement(WebElement target, int xOffset, int yOffset) | Moves the mouse to an offset from the top-left corner of the element. |
| void | [perform()](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html#perform--) | A convenience method for performing the actions without calling build() first. |
| [Actions](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html) | [release()](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html#release--) | Releases the depressed left mouse button at the current mouse location. |
| [Actions](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html) | release(WebElement target) | Releases the depressed left mouse button, in the middle of the given element. |
| [Actions](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html) | [sendKeys(java.lang.CharSequence... keys)](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html#sendKeys-java.lang.CharSequence...-) | Sends keys to the active element. |
| [Actions](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html) | sendKeys(WebElement target, java.lang.CharSequence... keys) | Equivalent to calling: Actions.click(element).sendKeys(keysToSend). This method is different from WebElement.sendKeys(CharSequence...) - see sendKeys(CharSequence...) |

Interface Action

Interface representing a single user-interaction action. It has a method .

[perform](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Action.html#perform--)()  - to perform an action build by Actions

Useful Actions with Examples

For all examples given below, we use below one

**Actions myActions = new Actions();**

**click()**

*To click on a element*

WebElement signImg = driver.findElement(By.xpath(“//img[@name=’login’]”));

Action myclick = myActions.**click**(signImg).build();

myclick.perform();

*To click at the current Mouse location*

Action myclick = myActions.**click**().build();

myclick.perform();

**doubleclick()**

*To double click on a element*

WebElement signImg = driver.findElement(By.xpath(“//img[@name=’login’]”));

Action mydblclick = myActions.**doubleClick**(signImg).build();

mydblclick.perform();

*To double click at the current Mouse location*

Action mydblclick = myActions.**doubleClick**().build();

mydblclick.perform();

**contextClick() (right click)**

*To context click on a element*

WebElement signImg = driver.findElement(By.xpath(“//img[@name=’login’]”));

Action mycontextclick = myActions.**contextClick**(signImg).build();

mycontextclick.perform();

*To context click at the current Mouse location*

Action mycontextclick = myActions.**contextClick**().build();

mycontextclick.perform();

**moveToElement (mouse over)**

*To move mouse on to a Webelement*

WebElement signImg = driver.findElement(By.xpath(“//img[@name=’login’]”));

Action myMoveToElmnt = myActions.**moveToElement**(signImg).build();

myMoveToElmnt.perform();

*To move mouse on to a Webelement with offset Coordinates*

Action myMoveToElmnt = myActions.**moveToElement**(signInImage, 5, 5).build();

myMoveToElmnt.perform();

**sendKeys (Using Keys)**

*To send keys to the current active element(for eg: Tab)*

Action myTab = myActions.**sendKeys**(Keys.TAB).build();

myTab.perform();

*To send keys to the web element(for eg: Tab)*

WebElement un = driver.findElement(By.xpath(“//input[@name=’userName’]”));

Action myTab = myActions.**sendKeys**(un, Keys.TAB).build();

myTab.perform();

**sendKeys (Using Keys Chord/Combination of Keys)**

*To send keys to the current active element (for eg: CTRL + A)*

Action myKeys = myActions.**sendKeys**(Keys.chord(Keys.CONTROL,"a")).build();

myKeys.perform();

*To send keys to the web element(for eg: CTRL + A)*

WebElement un = driver.findElement(By.xpath(“//input[@name=’userName’]”));

String myChords = Keys.chord(Keys.CONTROL,"a");

Action myKeys = myActions.**sendKeys**(un, myChords).build();

myKeys.perform();

**keyDown()**

*To send keys to the current active element(for eg: SHIFT)*

Action mykeys = myActions.**keyDown**(Keys.SHIFT).build();

mykeys.perform();

*To send keys to the web element(for eg: SHIFT)*

WebElement un = driver.findElement(By.xpath(“//input[@name=’userName’]”));

Action mykeys = myActions.**keyDown**(un, Keys.SHIFT).build();

mykeys.perform();

**keyUp()**

*Note: KeyUp(key release) happens only when KeyDown happened on webelement.*

*To send keys to the current active element(for eg: SHIFT)*

Action mykeys = myActions.**keyUp**(Keys.SHIFT).build();

mykeys.perform();

*To send keys to the web element(for eg: SHIFT)*

WebElement un = driver.findElement(By.xpath(“//input[@name=’userName’]”));

Action mykeys = myActions.**keyUp**(un, Keys.SHIFT).build();

mykeys.perform();

**Building Multiple Key Actions**

Multiple Keys Actions can be build and performed on an weblement . All the key actions would executed sequentially as per the oder .

*Due to KeyUp and KeyDown deals with a single Key, it suggested to use both together to proform the correct Key action. This way you can build more than one key action.For example in the below code, we are typing usename in CAPITAL lettes by using KeyDown and Keyup of SHIFT key.*

WebElement un = driver.findElement(By.xpath(“//input[@name=’userName’]”));

Action mykeys = myActions.**keyDown**(un, Keys.SHIFT)

.**sendKeys**("mercury")

         .**keyUp**(text, Keys.SHIFT)

.**build**();

Mykeys.perform();

Example with few more Actions added to above

WebElement un = driver.findElement(By.xpath(“//input[@name=’userName’]”));

Action mykeys = myActions.**keyDown**(un, Keys.SHIFT)

.**sendKeys**("mercury")

         .**keyUp**(text, Keys.SHIFT)

.**doubleClick**()

.**contextClick**()

.**build**();

Mykeys.perform();

**dragAndDrop()**

This is used to drag and drop an element in to a target element or offset values

*To drag and drop an element to target element*

WebElement srcElmt = driver.findElement(By.xpath("//img[[@id='drag1']”));

WebElement tgElmt = driver.findElement(By.xpath("//div[@id='div1']"));

Action drgDrop = myActions.**dragAndDrop**(srcElmt, tgElmt).build();

drgDrop.perform();

**dragAndDropBy()**

*To drag and drop an element to offset values*

WebElement srcElmt = driver.findElement(By.xpath("//img[[@id='drag1']”));

Action drgDrop = myActions.**dragAndDropBy**(srcElmt, 200, 300).build();

drgDrop.perform();

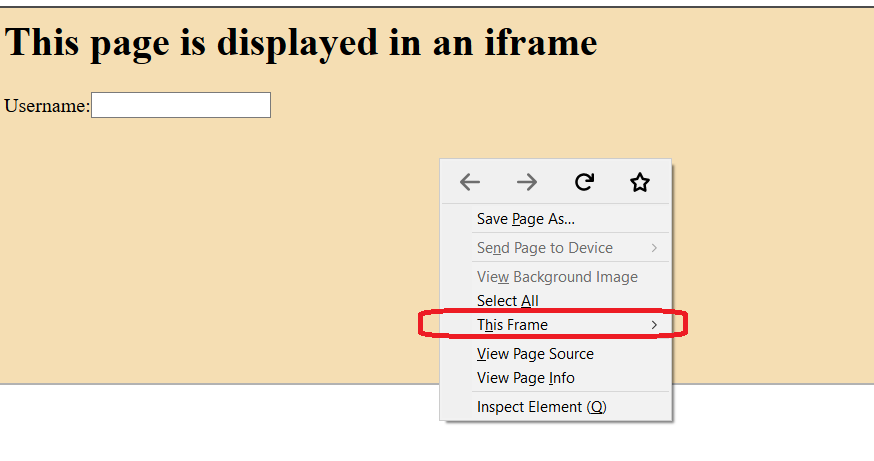
**Handing Frames**

Frame is a web page which is embedded in another web page or an HTML document embedded inside another HTML document.

The IFrame is often used to insert content from another source. The <**iframe**> tag specifies an inline frame.

Easy ways of identifying the frame

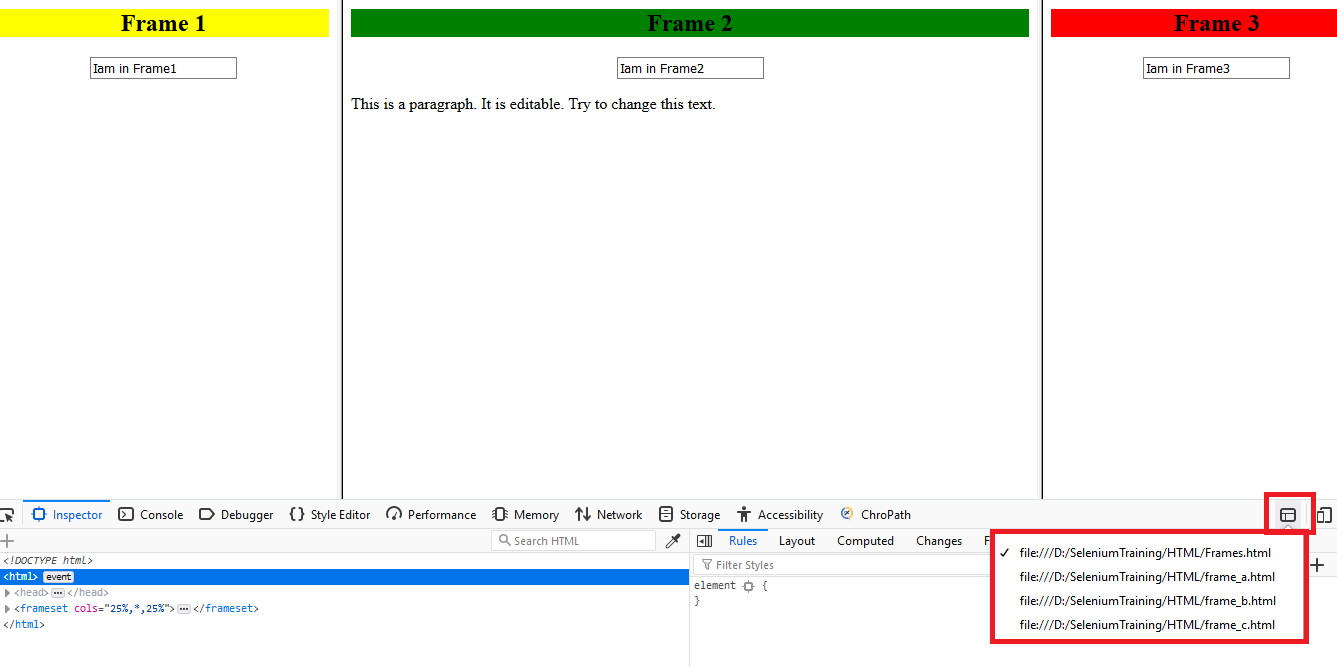
1. When you are object/page eleemnt.Right click on the Page, if you are seeing “This Frame” means that the web eleemnt on different frame/iframe.

****

1. When inspect using Inspector of Development tools(F12 or Ctrl+Shift+I) of browser, you can find/search for frame/iframe to know the frames available in the page.

****

1. Check on the select frames icon on the right side of inspector tool, you would see all frames/iframes of page displayed

****

Switching between frames

To work with different frames/iFrames on a page we have to switch between these Frames/iFrames. To Switch between frames/iFrames we have to use the driver’s **switchTo().frame** command.

We can switch over the elements in frames using 3 ways.

1. By Index
2. By Name or Id
3. By Web Element

**Switch by index:**

Index is one of the attributes for the frame/Iframe through which we can switch to it.

Index of the frame/iframe starts with '0'.

Suppose if there are 3 frames in page, we can switch to the iframe by using index.

[Syntax: driver.switchTo().frame](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/WebDriver.TargetLocator.html#frame-int-)(int index);

Example: driver.switchTo().frame(0);

driver.switchTo().frame(1);

**Switch by Name or ID:**

Name and ID are attributes of frame/iframe through which we can switch to it.

Syntax: driver.switchTo().[frame](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/WebDriver.TargetLocator.html#frame-java.lang.String-)(java.lang.String nameOrId);

Example: driver.switchTo().frame("name of element");

driver.switchTo().frame("id of element");

**Switch by Web Element:**

We can even switch to the iframe using web element .

Syntax : driver.switchTo().[frame](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/WebDriver.TargetLocator.html#frame-org.openqa.selenium.WebElement-)([WebElement](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/WebElement.html) frameElement);

Example:

WebElement mywe = driver.findElement(By.xpath(“//frame[contains(@id,’frame1’)]”));

driver.switchTo().[frame](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/WebDriver.TargetLocator.html#frame-org.openqa.selenium.WebElement-)(mywe);

**Finding Total Frames in a page:**

To find Total frames which are visible at the Page level, we cause driver.findElements();

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

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

Switching Back to Page :

If you are switched to any frame of your page, to switch back to Page level, you can use following command.

driver.switchTo().defaultContent();

This will ensure you switch to Page level.

Switching to Parent Frame :

If You are switched to any of the frame, you can switch to its Parent frame using the following command.

driver.switchTo().parentFrame();

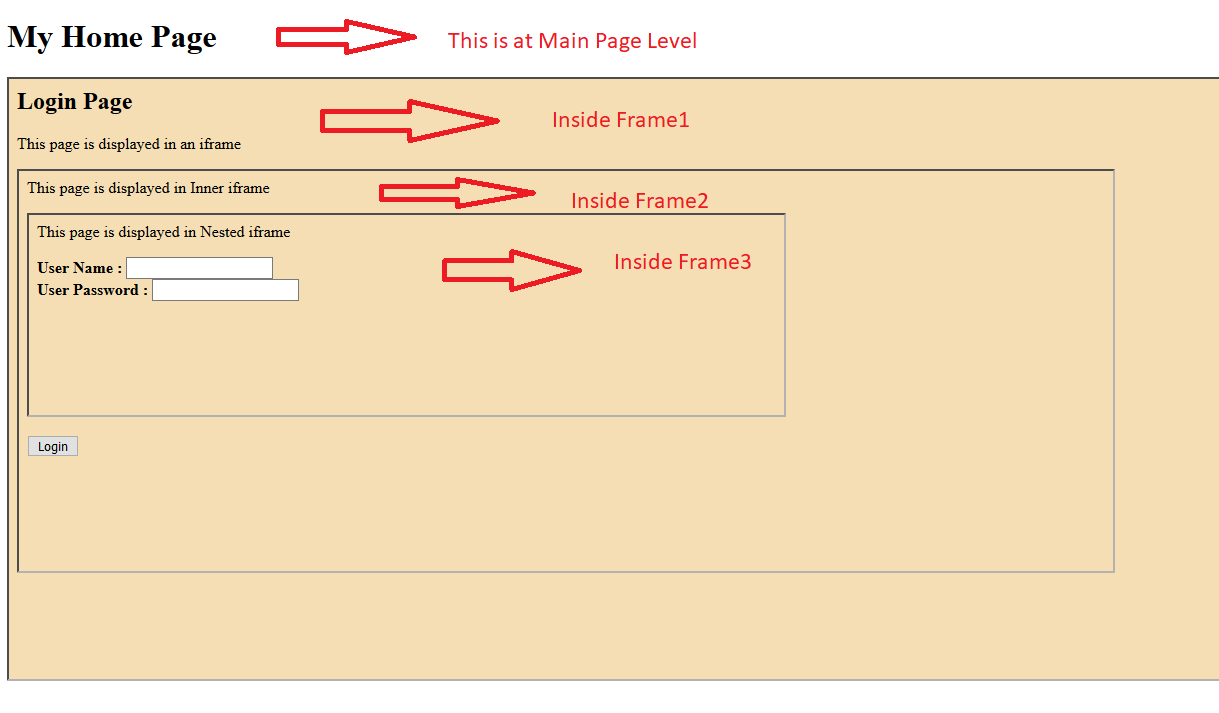
This will ensure you switch to Parent frame of the Current Frame. This will switch to Page, if there is no Parent frame.

Nested Frames(Frames inside Frames)

When one frame is insdie of another frame, we call them as nested frames.

Switching to nested frames should not be done like nornal frame switch directly from the page. It should be done thru parent.

For example: Observe the below screenshot which has Frame1 at Page level, Frame1 contains inner frame called frame2, which has Login button and another inner frame called Frame3. Frame3 contains username and password input fields.



Assume a scenario here, to Enter usename and password at Frame3 and Click on Login button at Frame2 and switch back to Main Page. Here are the steps to be followed.

1. From Main page Switch to Frame1 (As Frame2 and Frame3 nested frames, these are not accessible directly from Main Page).
2. Once Switched to Frame1, switch to Frame2 (As Frame3 is nested frame, it is not directly accessble to Frame1)
3. Once switced to Frame2, switch to Frame3 (As Frame3 is inner frame of Frame2, it can directly access Frame3)
4. Once Switch to Frame3, Enter Username and Password details
5. Now to go Frame2 Login button, switch to Parent fo Frame 3 which is Frame 2
6. Once you switched back to Frame2, Click On Login button.
7. To witch to back Main page, swtich to Defautlt content.

driver.switchTo().frame(0); //Switch To first frame

driver.switchTo().frame(0); //Switch To second frame

driver.switchTo().frame(0); //Switch To third frame

driver.findElement(By.xpath("//input[@name='userName']")).sendKeys("Admin");

driver.findElement(By.xpath("//input[@name='usePwd']")).sendKeys("password");

driver.switchTo().parentFrame();//Switch To seond frame

driver.findElement(By.xpath("//button[@name='Login']")).click(); // Login Click

driver.switchTo().defaultContent(); //Switch to Main

**Handling of Alerts/Popups**

Alert is a small message box which displays on-screen notification to give the user some kind of information or ask for permission to perform certain kind of operation. It may be also used for warning purpose.

**Simple Alert**

It displays some information or warning on the screen. It displays an alert box with a specified message and an OK button.

**Confirmation Alert**

It displays a dialog box with a specified message, along with an OK and a Cancel button. A confirm alert is often used if you want the user to verify or accept something.

**Prompt Alert**

It is a dialog box that prompts the visitor for input. A prompt box is often used if you want the user to input a value.

***Note:*** *The alert popup takes the focus away from the current window, and forces the browser to read the message.*

Alert interface provides the below few methods which are widely used in Selenium Webdriver.

**void dismiss()** - To click on the 'Cancel' button of the alert.

Usage : driver.switchTo().alert().dismiss();

**void accept()**- To click on the 'OK' button of the alert.

Usage : driver.switchTo().alert().accept();

**String getText()** - To capture the alert message.

Usage : driver.switchTo().alert().getText();

**void sendKeys(String stringToSend)** - To send some data/text to alert box.

Usage : driver.switchTo().alert().sendKeys("Name");

Example: Sample code for handling prompt popup with passing of a value

driver.findElement(By.xpath(prmptXpath)).click();

String myStr = driver.switchTo().alert().getText();

System.out.println("Prompt text displayed : "+ myStr); // Message displayed

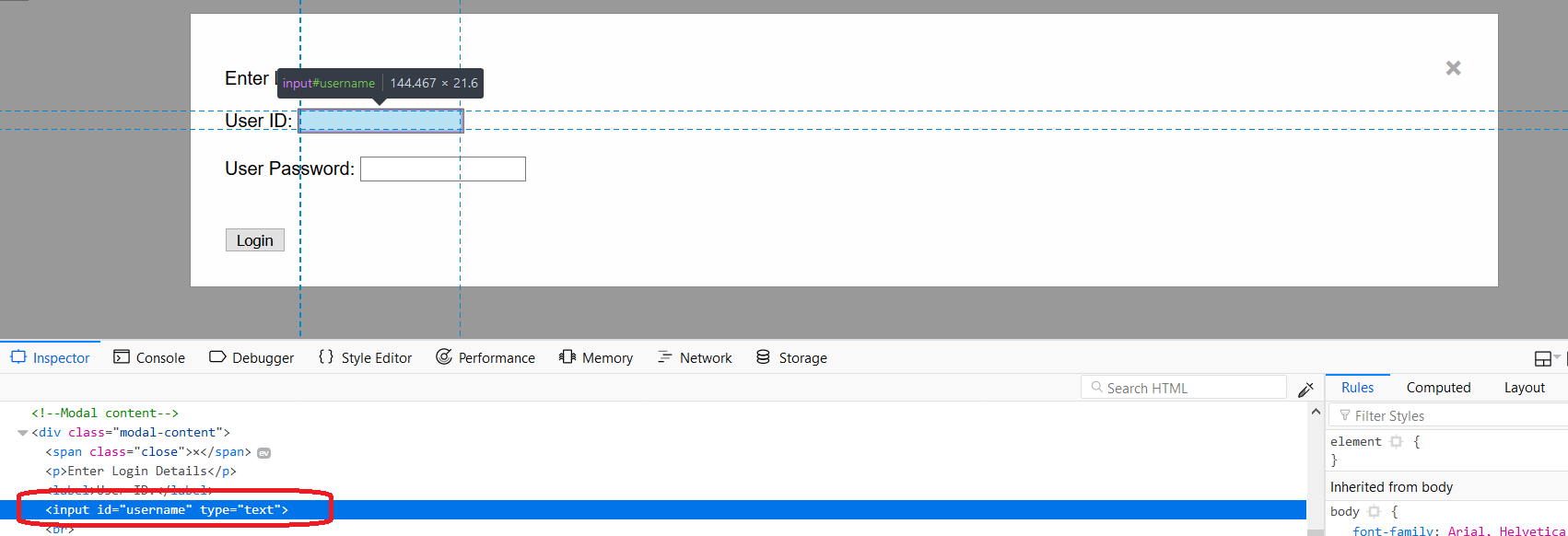
driver.switchTo().alert().sendKeys("H2kInfosys"); // Send expected text

driver.switchTo().alert().accept(); // Click on Ok button of poupup

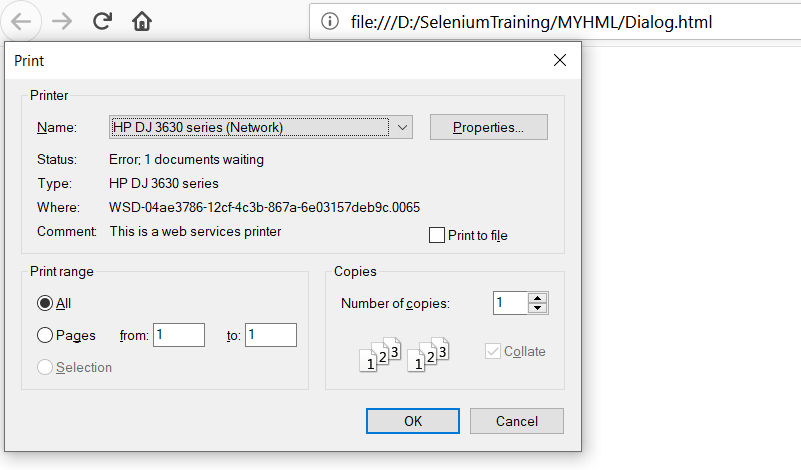
Handling of Web Dialogs

Web page dialogs, which not alert types, can be handled with selenium like any other web element.

For instance observe the Dialog in screenshot below. You are able to Inspect that dialog elements alike page elements and you can handle the same way we handle page elements.



***Note:*** *Any window based alerts/dialogs like Print Screen as shown below, can not be handled by Selenium. To handle them you require to use like Robot or Autoit etc.*

**

**Handling Multiple Browser Windows**

There are many cases, where a application displays multiple windows when you open a website. Those are may be advertisements or may be a kind of information showing on popup windows.

Selenium webdriver using Switch To methods which will allow us to switch control from one window to another window.If you have to switch between tabs then also you have to use the same approach.

There is only one way you can get multiple windows via Selenium web driver, that is by clicking on a link which opens the page in a new browser window. Selenium web driver keeps a track of how many windows it opened during a session. Which means that it will not keep track of any browser windows which are  **Opened manually** or **Opened by a previous session of Selenium Webdriver** . By session of selenium WebDriver means the duration from the time we instantiate a WebDriver instance to the time we kill it via WebDriver Quit or by manually killing the process.

Useful methods

**String getWindowHandle() -** Return an opaque handle to current window that uniquely identifies it within this driver instance. This can be used to switch to this window at a later time.

Usage : String mainHandleId = driver.getWindowHandle();

**Set<java.lang.String> getWindowHandles() -** Return a set of window handles which can be used to iterate over all open windows of this WebDriver instance.

Usage : Set<String> handles = driver.getWindowHandles();

Iterator<String> it = handles.iterator();

while(it.hasNext()){

winHandle = it.next();

System.out.println("Window Handle : " + winHandle );

}

**WebDriver.SwitchTo().window(String windowHandle) -** In order to shift focus from Parent Window to any child window we have to use the following command on WebDriver.

Usage : driver.switchTo().window(winHandle);

**WebDriver.Close()  -** command to close the current window on which the focus is present.

Usage : driver.close();

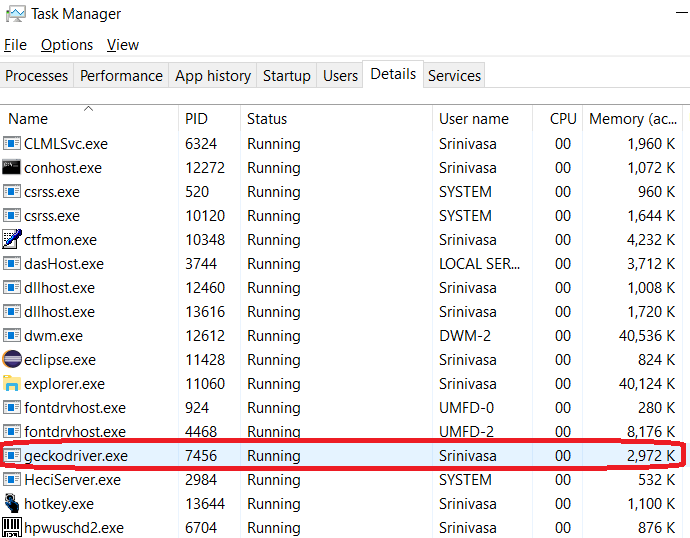
**WebDriver.quit() *-*** will close all the windows opened in the session. This command basically shuts down the driver instance and any further commands to WebDriver results in exception.

Usage : driver.quit();

**Killing of geckodriver and Chromedriver (.exe)**

When you close or quit browser under test, Browser will be closed. However selenium does not automatically close the geckodriver or chromedriver which started along with Browser launch.

You could find them in Task Manager of your machine like below.

****

To kill those tasks use the Task kill command like below

Runtime.getRuntime().exec("taskkill /F /IM geckodriver.exe");

Runtime.getRuntime().exec("taskkill /F /IM geckodriver.exe \*32");//for 32-bit exe

Runtime.getRuntime().exec("taskkill /F /IM chromedriver.exe");

Runtime.getRuntime().exec("taskkill /F /IM chromedriver.exe \*32"); //for 32-bit exe

When using the TASKKILL command, /F means to terminate the process forcefully. /IM means the image name, i.e. the process name. If you want to kill using the process ID (PID), you have to use /PID instead of /IM.

**JavaScriptExecutor**

Java Script executor is an interface of Selenium WebDriver that has the functionality similar to that of Java Script and can interact with HTML DOM elements. Instead of using driver.findElement method of the Selenium WebDriver we can use JavaScriptExecutor Interface to perform similar action on the Page.

It provides advantages over FindElement method while handling tricky XPath as well as finding element which are sometimes hidden.

Selenium supports javaScriptExecutor without an extra plugin or add-on. You just need to import (org.openqa.selenium.JavascriptExecutor) in the script as to use JavaScriptExecutor.

Methods

**executeAsyncScript()**

With Asynchronous script, your page renders more quickly. Instead of forcing users to wait for a script to download before the page renders. This function will execute an asynchronous piece of JavaScript in the context of the currently selected frame or window in Selenium.

**executeScript()**

This method executes JavaScript in the context of the currently selected frame or window in Selenium. The script used in this method runs in the body of an anonymous function (a function without a name). We can also pass arguments to it.

The script can return values. Data types returned are

* Boolean
* Long
* String
* List
* WebElement.

Syntax:

JavascriptExecutor js = (JavascriptExecutor) driver;

js.executeScript(Script,Arguments);

Useful Code Snippets

JavascriptExecutor js = ((JavascriptExecutor)driver);

* **Rerfreshing the Browser**

            js.executeScript("history.go(0)");

* **Fetching the URL of the Website. Tostring() change object to name**

String url = js.executeScript("return document.URL;").toString(); System.out.println("URL of the site = "+url);

* **Fetching the Title of the Website.**

String TitleName = js.executeScript("return document.title;").toString()

System.out.println("Title of the page = "+TitleName);

* **Navigate to new Page/Launch new url**

js.executeScript("window.location = 'http://h2kinfosys.com/'");

* **Click on an WebElement**

        js.executeScript("arguments[0].click();",element);

* **Entering Text to WebElement**

js.executeScript(“arguments[0].value=’mercury’;”, we);

* **Fetching Text from WebElement**

String text = (String) js.executeScript(“return arguments[0].text;”, we);

* **Fetching Inner HTML from WebElement**

String text = (String) js.executeScript(“return arguments[0].innerHTML;”, we);

* **Scroll down bottom of the page**

js.executeScript("window.scrollBy(0,document.body.scrollHeight)");

* **Vertical scroll down by pixels**

js.executeScript("window.scrollBy(0,500)"); //This will scroll by 500 pixels

* **Scroll the page till the element is found**

js.executeScript("arguments[0].scrollIntoView();", Element);

**Finding Web Element X Y Coordinates(Point):**

Any web element has its own position on page known as x y coordinates. x y coordinates of a web element is measured in x and y pixels. x pixels means the horizontal position of an element on a page from the left side and y pixels means the vertical position of an element on a page from the top.

Usage:

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

//Used points class to get x and y coordinates of element.

Point point = element.getLocation();

int xcord = point.getX();

int ycord = point.getY();

System.out.println("Position of the webelement from left side is "+xcord +" pixels" + “ and from top” + ycord + “pixel”);

**Explicit Wait**

Explicit waits are confined to a particular web element. Explicit Wait is code you define to wait for a certain condition to occur before proceeding further in the code.

The explicit wait is an intelligent kind of wait, but it can be applied only for specified elements. Explicit wait gives better options than that of an implicit wait as it will wait for dynamically loaded Ajax elements.

Once we declare explicit wait we have to use "**ExpectedCondtions**" or we can configure how frequently we want to check the condition using **Fluent Wait**. While implementing we are using **Thread.Sleep()**generally it is not recommended to use

Explicit wait is of two types:

* WebDriverWait
* FluentWait

WebdriverWait

We create a reference variable “wait” for WebDriverWait class and instantiate it using WebDriver instance and maximum wait time for the execution to layoff. The maximum wait time quoted is measured in “seconds”.

Syntax : WebDriverWait wait = new WebDriverWait(<driver>, <wait time in seconds>);

Example : WebDriverWait wait = new WebDriverWait(driver,30);

Expected Condition

We use the “wait” reference variable of WebDriverWait class created in the previous step with ExpectedConditions class and an actual condition which is expected to occur. Therefore, as soon as the expected condition occurs, the program control would move to the next execution step instead of forcefully waiting for the entire 30 seconds.

Usage :

wait.until(ExpectedConditions.visibilityOfElementLocated(By.xpath("//input[@name='userName']")));

WebElement userId = driver.findElement(By.xpath("//input[@name='userName']"));

wait.until(ExpectedConditions.elementToBeClickable(userId));

The following are the Expected Conditions that can be used in Explicit Wait

* alertIsPresent()
* elementSelectionStateToBe()
* elementToBeClickable()
* elementToBeSelected()
* frameToBeAvaliableAndSwitchToIt()
* invisibilityOfTheElementLocated()
* invisibilityOfElementWithText()
* presenceOfAllElementsLocatedBy()
* presenceOfElementLocated()
* textToBePresentInElement()
* textToBePresentInElementLocated()
* textToBePresentInElementValue()
* titleIs()
* titleContains()
* visibilityOf()
* visibilityOfAllElements()
* visibilityOfAllElementsLocatedBy()
* visibilityOfElementLocated()

Fluent Wait

The fluent wait is used to tell the web driver to wait for a condition, as well as the frequency with which we want to check the condition before throwing an "ElementNotVisibleException" exception.

FluentWait uses two parameters mainly – *timeout value*and *polling frequency*.

**Frequency:**Setting up a repeat cycle with the time frame to verify/check the condition at the regular interval of time.

Syntax:

Wait wait = new FluentWait(WebDriver reference)

.withTimeout(Duration)

.pollingEvery(Duration)

.ignoring(Exception.class);

Example:

Wait<WebDriver> wait = new FluentWait<WebDriver>(driver)

.withTimeout(Duration.ofSeconds(60))

.pollingEvery(Duration.ofSeconds(5))

.ignoring(NoSuchElementException.class);

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

public WebElement apply(WebDriver driver) {

WebElement element = driver.findElement(By.xpath("//input[@name='userName']"));

return element;

}

});

**Capturing Screenshots**

Screenshots are desirable for bug analysis. Selenium can automatically take screenshots during execution. You need to type cast WebDriver instance to TakesScreenshot.

Steps for Screenshot capture

* Convert web driver object to TakeScreenshot

TakesScreenshot scrShot =(TakesScreenshot)webdriver;

* Call getScreenshotAs method to create image file

File SrcFile=scrShot.getScreenshotAs(OutputType.FILE);

* Copy file to Desired Location

Example: Assume this for URL : http:// newtours.demoaut.com

Once launched, we want take screenhot of login page

TakesScreenshot scrShot =((TakesScreenshot)driver);

File scrFile = scrShot.getScreenshotAs(OutputType.FILE);

Path srcFilePath = Paths.get(scrFile.getAbsolutePath());

Path destDir = Paths.get("D:\\SeleniumTraining\\screenshots\\LoginPage.png");

Files.copy(srcFilePath, destDir, StandardCopyOption.REPLACE\_EXISTING);

**Capturing Page Source HTML Code:**

Using selenium, we can capture the HTML source of a Page/Frame using webdriver. This is useful for debugging pruposes. Page source will be captured for the window/frame of the webdriver it currently points to.

Usage:

String pageSource = driver.getPageSource();

Path path = Paths.get("D:\\SeleniumTraining\\pagesource\\mercSource.html");

Files.write(path, pageSource.getBytes());

**Page Object Model(POM):**

**Page Object Model** is a design pattern (*Design patterns in java are best practices which are used to resolve some known issues.* ) which has become popular in test automation for enhancing test maintenance and reducing code duplication. A page object is an object-oriented class that serves as an interface to a page of your AUT(Application Under Test).

The tests then use the methods of this page object class whenever they need to interact with the UI of that page, the benefit is that if the UI changes for the page, the tests themselves don’t need to be changed, only the code within the page object needs to change.Subsequently all changes to support that new UI are located in one place.

Why we need POM?

Increasing automation test coverage can result in unmaintainable project structure, if [locators](https://www.protechtraining.com/content/selenium_tutorial-locators) are not managed in right way. This can happen due to duplication of code or mainly due to duplicated usage of locators. Updating element-locators in duplicated code will consume a lot of time to only adjust locators, while this time can be consumed to increase test coverage. We can save this time by using **Page Object Model** in our test automation framework.

Advantages of Page Object Model:

According to **Page Object Model**, we should keep our tests and element locators separately, this will keep code clean and easy to understand and maintain.

* The Page Object approach makes test automation framework programmer friendly, more durable and comprehensive.
* Another important advantage is our *Page Object Repository is Independent of Automation Tests*. Keeping separate repository for page objects helps us to use this repository for different purposes with different frameworks like, we are able to integrate this repository with other tools like [**JUnit**](https://junit.org/)/[**TestNG**](http://testng.org/) etc.
* Test cases become short and optimized as we are able to reuse page object methods in the **POM** classes.
* Any change in UI can easily be implemented, updated and maintained into the Page Objects and Classes.

What is Page Factory:

We have seen that ‘Page Object Model’ is a way of representing an application in a test framework. For every ‘page’ in the application, we create a Page Object to reference the ‘page’ whereas a ‘Page Factory’ is one way of implementing the ‘Page Object Model’.

**@FindBy** annotation is used in Page Objects in Selenium tests to specify the object location strategy for a WebElement or a list of WebElements. Using the PageFactory, these WebElements are usually initialized when a Page Object is created.

***Sytntax :***

*CLASS\_NAME*: **@FindBy**(className = "classname")

*CSS:* **@FindBy**(css = "css")

*ID*: **@FindBy**(id = "id")

*LINK\_TEXT:* **@FindBy**(linkText= "text")

*NAME:* **@FindBy**(name= "name")

*PARTIAL\_LINK\_TEXT*: **@FindBy**(partialLinkText= "text")

*TAG\_NAME:* **@FindBy**(tagName="tagname")

*XPATH:* **@FindBy**(xpath="xpath")

**Sample Page Object Class:**

public class HomePage {

protected WebDriver driver; //Driver Declaration

public HomePage(WebDriver driver){ // constructor

this.driver =driver;

}

/\* Page Elements\*/

@FindBy(xpath="//input[@name='userName']")

private WebElement userId;

@FindBy(xpath="//input[contains(@name,'password')]")

private WebElement userPwd;

@FindBy(xpath="//input[(@type='image') and (@name='login')]")

private WebElement signImg;

/\*\*

\* Method to verify Title of the Page

\* @param expTitle Expected Title

\* @return true if pass, else false

\*/

public boolean verifyTitle(String expTitle) throws Exception{

boolean retFlag = false;

/\* Get Title of the page\*/

String actTitle = driver.getTitle();

/\* Verify Title\*/

if ( actTitle.equalsIgnoreCase(expTitle)) {

System.out.println("Found expected Title : "+ retFlag);

retFlag = true;

} else {

System.out.println("Page Title Mismatch. Expected : "+

expTitle + " Actual/Displayed : "+ actTitle);

retFlag = false;

}

return retFlag;

}

/\*\*

\* Method to Logon to Application

\* @param username User Name

\* @param password User Password

\* @return Empty String if Pass, else Error String

\*/

public String login(String username, String password) throws Exception{

if (!userId.isDisplayed()) {

return "Failed to Locate Username inputbox";

}

userId.sendKeys(username);

userPwd.sendKeys(password);

signImg.click();

return "";

}

}

Implementing Page Factory:

public class MercuryTestCase {

public WebDriver driver ;

public String browser = "FF";

public String baseUrl = "http://newtours.demoaut.com";

public String expTitle="Welcome: Mercury Tours";

public String username = "mercury";

public String userwd = "mercury";

HomePage homeObj=null;

@BeforeClass

public void launch() throws Exception {

System.setProperty("webdriver.gecko.driver","D:\\SeleniumTraining\\software

s\\driverexes\\geckodriver.exe");

System.setProperty("webdriver.chrome.driver","D:\\SeleniumTraining\\softwar

es\\driverexes\\chromedriver.exe");

if(browser.equalsIgnoreCase("FF")) {

System.out.println("Instantiating Firefox Driver");

driver = new FirefoxDriver();

} else {

System.out.println("Instantiating Chrome Driver");

driver = new ChromeDriver();

}

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

System.out.println("Setting temeout Settings..");

driver.manage().timeouts().pageLoadTimeout(60, TimeUnit.SECONDS);

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

driver.get(baseUrl);

}

@Test(priority=0)

public void loginTest() throws Exception {

boolean rFlag = false;

String rMsg = "";

/\* Instantiating HomePage Object\*/

**homeObj = PageFactory.initElements(driver,HomePage.class);**

/\* Call Page Object Method Verify Title\*/

rFlag = homeObj.verifyTitle(expTitle);

Assert.assertTrue(rFlag, " Failed to Verify Title");

/\* Call Login\*/

rMsg = homeObj.login(username, userwd);

Assert.assertTrue(rMsg.isEmpty(), " Failed to Login");

}

}

Getting list of WebElements using @FindBy

You can also get List of WebElement using **@FindBy**. For instance, if you want to get all input fields on page you can use @FindBy like below mentioned code.

Example:

@FindBy(xpath="//input")

private List<**WebElement**> inputs;

**Apache POI**

Apache POI is a 100% open source library provided by Apache Software Foundation. Most of the small and medium scale application developers depend heavily on Apache POI. It supports all the basic features of Excel libraries.

What is Apache POI?

Apache POI(Poor Obfuscation Implementation) is a popular API that allows programmers to create, modify, and display MS Office files using Java programs. It is an open source library developed and distributed by Apache Software Foundation to design or modify Microsoft Office files using Java program. It contains classes and methods to decode the user input data or a file into MS Office documents.

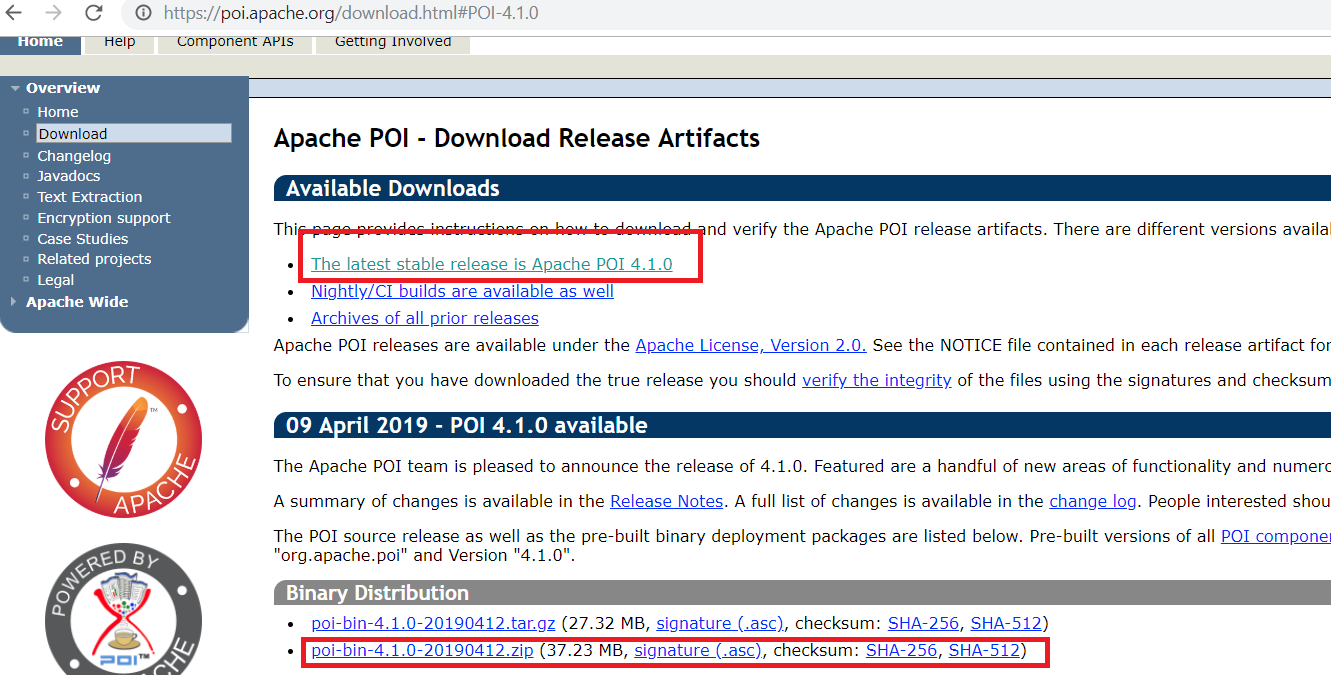
Components of Apache POI

Apache POI contains classes and methods to work on all OLE2 Compound documents of MS Office. The list of components of this API is given below.

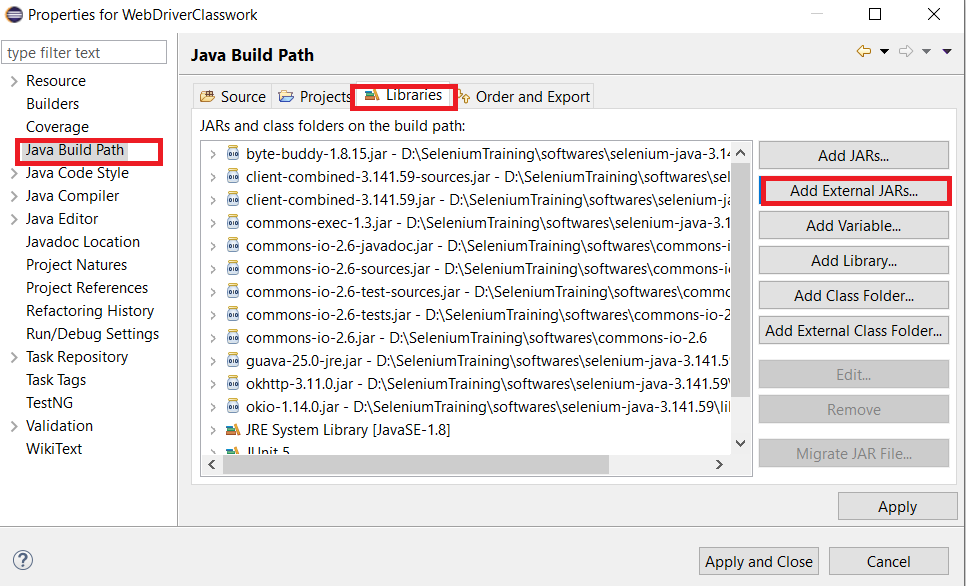
* **POIFS** (Poor Obfuscation Implementation File System) − This component is the basic factor of all other POI elements. It is used to read different files explicitly.
* **HSSF** (Horrible Spreadsheet Format) − It is used to read and write xls format of MS-Excel files.
* **XSSF** (XML Spreadsheet Format) − It is used for xlsx file format of MS-Excel.
* **HPSF** (Horrible Property Set Format) − It is used to extract property sets of the MS-Office files.
* **HWPF** (Horrible Word Processor Format) − It is used to read and write doc extension files of MS-Word.
* **XWPF** (XML Word Processor Format) − It is used to read and write docx extension files of MS-Word.
* **HSLF** (Horrible Slide Layout Format) − It is used for read, create, and edit PowerPoint presentations.
* **HDGF** (Horrible DiaGram Format) − It contains classes and methods for MS-Visio binary files.
* **HPBF**(Horrible PuBlisher Format) − It is used to read and write MS-Publisher files.

Download and intstall Apache POI with Eclipse

**Step1:** Download the latest verson of Apache POI API from the location <https://poi.apache.org/download.html> to the local drive. Choose latest stable Apache Release and download as shown below.Download and extract the zip.



**Step2:** Launch Eclipse and select the project, click on Right click on the Project and Select Properties. In properties Dialog, Click on Java BuildPath->Library->Add External JARs button



**Step3:** Select and Add all libraries of Aapache POI from extracted folder and click on Apply and Close button.

Workbook

This is the super-interface of all classes that create or maintain Excel workbooks. It belongs to the org.apache.poi.ss.usermodel package. The two classes that implement this interface are

* **HSSFWorkbook** − This class has methods to read and write Microsoft Excel files in .xls format. It is compatible with MS-Office versions 97–2003.
* **XSSFWorkbook** − This class has methods to read and write Microsoft Excel and OpenOffice xml files in .xls or .xlsx format. It is compatible with MS-Office versions 2007 or later.

**Useful methods**

*createSheet() -* Creates an XSSFSheet for this workbook, adds it to the sheets, and returns the high level representation.

*createSheet(String sheetname)* - Creates a new sheet for this Workbook and returns the high level representation.

*createCellStyle() -* Creates a new XSSFCellStyle and adds it to the workbook's style table.

Sheet

Sheet is an interface under the org.apache.poi.ss.usermodel package and it is a super-interface of all classes that create high or low level spreadsheets with specific names. The most common type of spreadsheet is worksheet, which is represented as a grid of cells.

* **HSSFSheet** - This is a class under the org.apache.poi.hssf.usermodel package. It can create excel spreadsheets and it allows to format the sheet style and sheet data.
* **XSSFSheet** - This is a class which represents high level representation of excel spreadsheet. It is under org.apache.poi.hssf.usermodel package.

**Useful methods**

*addMergedRegion(CellRangeAddress region)* - Adds a merged region of cells (hence those cells form one).

*autoSizeColumn(int column)* - Adjusts the column width to fit the contents.

*iterator()* - This method is an alias for rowIterator() to allow foreach loops

*addHyperlink(XSSFHyperlink hyperlink)* - Registers a hyperlink in the collection of hyperlinks on this sheet

Row

This is an interface under the org.apache.poi.ss.usermodel package. It is used for high-level representation of a row of a spreadsheet. It is a super-interface of all classes that represent rows in POI library.

* **XSSFRow** - This is a class under the org.apache.poi.xssf.usermodel package. It implements the Row interface, therefore it can create rows in a spreadsheet.

**Useful methods**

*createRow(int rowIndex)* - Creates new row and returns it.

*setHeight(short height) -* Sets the height in short units.

Cell

This is an interface under the org.apache.poi.ss.usermodel package. It is a super-interface of all classes that represent cells in the rows of a spreadsheet.

Cells can take various attributes such as blank, numeric, date, error, etc.

* **XSSFCell** - This is a class under the org.apache.poi.xssf.usermodel package. It implements the Cell interface. It is a high-level representation of cells in the rows of a spreadsheet.

There are many other classes under Cell inetrface like XSSFCellStyle, XSSFColor, XSSFFont etc

*createCell(int columnIndex)* - Creates new cell within the row and returns it.

Below are some of the fields of the XSSFCell class along with Cell Type & Description.

CELL\_TYPE\_BLANK - Represents blank cell

CELL\_TYPE\_BOOLEAN - Represents Boolean cell (true or false)

CELL\_TYPE\_ERROR - Represents error value on a cell

CELL\_TYPE\_FORMULA - Represents formula result on a cell

CELL\_TYPE\_NUMERIC - Represents numeric data on a cell

CELL\_TYPE\_STRING - Represents string (text) on a cell

**Useful methods**

*setCellStyle(CellStyle style)* - Sets the style for the cell.

*setCellType(int cellType)* - Sets the type of cells (numeric, formula, or string).

*setCellValue(boolean value)* - Sets a boolean value for the cell.

*setCellValue(Calendar value)* - Sets a date value for the cell.

*setCellValue(double value)* - Sets a numeric value for the cell.

*setCellValue(String str) -* Sets a string value for the cell.

*setHyperlink(Hyperlink hyperlink)* - Assigns a hyperlink to this cell.

Woking with WorkBook:

Example: Write To Workbook

public class WriteToExcelDemo {

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

//Create blank workbook

HSSFWorkbook workbook = new HSSFWorkbook();

//Create a blank sheet

HSSFSheet spreadsheet = workbook.createSheet("UserInfo ");

//Create row object

HSSFRow row;

//Data to be written

String[][] dataArray = {{"USERNAME", "PASSWORD"},

{"merury", "mercury"},

{"Srini", "password"},

{"myuser", "mypassword"},

{"testuser", "testpassword"}

};

//Iterate over data and write to sheet

for(int i =0; i < dataArray.length; i++) {

row = spreadsheet.createRow(i);

for(int j = 0; j < dataArray[i].length; j++ ) {

Cell cell = row.createCell(j);

cell.setCellValue(dataArray[i][j]);

}

}

//Write the workbook in file system

FileOutputStream out = new FileOutputStream(new

File("D:\\SeleniumTraining\\datafiles\\MyDataSheet.xls"));

workbook.write(out);

out.close();

System.out.println("MyDataSheet.xls written successfully");

}

}

Example: Read From Workbook

public class ReadFromExcel {

static HSSFRow row;

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

FileInputStream fis = new FileInputStream(new File("D:\\SeleniumTraining\\datafiles\\MyDataSheet.xls"));

HSSFWorkbook workbook = new HSSFWorkbook(fis);

HSSFSheet spreadsheet = workbook.getSheetAt(0);

Iterator < Row > rowIterator = spreadsheet.iterator();

while (rowIterator.hasNext()) {

row = (HSSFRow) rowIterator.next();

Iterator <Cell> cellIterator = row.cellIterator();

while ( cellIterator.hasNext()) {

Cell cell = cellIterator.next();

System.out.print(cell.toString() + "\t\t");

}

System.out.println();

}

fis.close();

}

}

**Reading Data from Database(MySQL) using JDBC**

Connecting to Database

The interface for accessing relational databases from Java is *Java Database Connectivity (JDBC)*. Via JDBC you create a connection to the database, issue database queries and update as well as receive the results.

JDBC provides an interface which allows you to perform SQL operations independently of the instance of the used database. To use JDBC, you require the database specific implementation of the JDBC driver.

[MySQL JDBC driver](https://www.vogella.com/tutorials/MySQLJava/article.html#jdbcdriver)

To connect to MySQL from Java, you have to use the JDBC driver from MySQL. The MySQL JDBC driver is called *MySQL Connector/J*. You can download the latest MySQL JDBC driver under the Project you are working in Eclipse from the following location.

<http://dev.mysql.com/downloads/connector/j> or <https://mvnrepository.com/artifact/mysql/mysql-connector-java> based on version of MySQL you are using

Add MySQL-Connecter-java jar to the Project.

[Creating](https://www.vogella.com/tutorials/MySQLJava/article.html#jdbcdriver) DataProvider using JDBC

This example uses a database Table called “Login” “with “username” and “pwd” as table fields under Schema/Database called “DEMO”

**public class JDBCDataProvider {**

private Connection connect = null;

private Statement statement = null;

private ResultSet resultSet = null;

**@DataProvider(name="dbData")**

public Object[][] readDataBase() throws Exception {

String[][] dataArray;

try {

/\* This will load the MySQL driver \*/

Class.forName("com.mysql.cj.jdbc.Driver");

/\* Setup the connection with the DB \*/

connect = DriverManager.getConnection("**jdbc:mysql://localhost/DEMO?user=TRAINING&password=training**");

/\* Statements allow to issue SQL queries to the database \*/

statement = connect.createStatement();

/\* Result set get the result of the SQL query \*/

resultSet = statement.executeQuery("select \* from DEMO.LOGIN");

/\* Get Number of Rows and Columns\*/

resultSet.last();

int rows = resultSet.getRow();

System.out.println("No.of Rows:" + rows);

int cols = resultSet.getMetaData().getColumnCount();

System.out.println("No.of Cols:" + cols);

/\* Declare size of return values of 2D array\*/

dataArray = new String[rows][cols];

int r = 0;

/\* Read Result Set into Double Dimension array\*/

resultSet.beforeFirst();

while (resultSet.next()) {

String username = resultSet.getString("username");

String userpwd = resultSet.getString("pwd");

System.out.println("User: " + username);

dataArray[r][0]=username; //0, 1, 2, 3, 4, 5

System.out.println("Password: " + userpwd);

dataArray[r][1]=userpwd;//0, 1, 2, 3, 4, 5

r++;

}

/\* Return Data\*/

return dataArray;

} catch (Exception e) {

throw e;

} finally {

close();

}

}

/\* Close all DB Objects \*/

private void close() {

try {

if (resultSet != null) {

resultSet.close();

}

if (statement != null) {

statement.close();

}

if (connect != null) {

connect.close();

}

} catch (Exception e) {

}

}

}

Calling Data Provider in Testcase File:

public class JDBCDemo {

public String baseUrl = "http://newtours.demoaut.com";

public String browser = "FF";

public WebDriver driver ;

@BeforeClass

public void launch() throws Exception {

System.setProperty("webdriver.gecko.driver","D:\\SeleniumTraining\\softwares\\driverexes\\geckodriver.exe");

System.setProperty("webdriver.chrome.driver","D:\\SeleniumTraining\\softwares\\driverexes\\chromedriver.exe");

if(browser.equalsIgnoreCase("FF")) {

System.out.println("Instantiating Firefox Driver");

driver = new FirefoxDriver();

} else {

System.out.println("Instantiating Chrome Driver");

driver = new ChromeDriver();

}

Thread.sleep(2000);

System.out.println("Maximizing window");

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

Thread.sleep(2000);

System.out.println("Setting temeout Settings..");

driver.manage().timeouts().pageLoadTimeout(60, TimeUnit.SECONDS);

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

driver.get(baseUrl);

System.out.println(" Broswer launched for URI:"+ baseUrl);

Thread.sleep(5000);

}

@Test(priority=0, dataProvider = "dbData", dataProviderClass=JDBCDataProvider.class)

public void login(String username, String password) throws Exception {

boolean rFlag = true;

String userNameXapth = "//input[@name='userName']";

String userPwdXpath="//input[contains(@name,'password')]";

String signInXpath= "//input[(@type='image') and (@name='login')]";

WebElement userId = driver.findElement(By.xpath(userNameXapth));

WebElement userPwd = driver.findElement(By.xpath(userPwdXpath));

WebElement signInImage = driver.findElement(By.xpath(signInXpath));

/\* Verify Username field displayed\*/

rFlag = userId.isDisplayed();

if(!rFlag) {

WebElement homeLink = driver.findElement(By.xpath("//a[text()='Home']"));

homeLink.click();

}

Assert.assertTrue(rFlag, "Failed to locate Username");

Thread.sleep(2000);

userId.sendKeys(username);

userPwd.sendKeys(password);

System.out.println("Entered Username and password");

Thread.sleep(3000);

signInImage.click();

System.out.println("Clicked on SignIn image");

Thread.sleep(3000);

WebElement signOutLink = driver.findElement(By.xpath("//a[text()='SIGN-OFF']"));

boolean signoutFlag = signOutLink.isDisplayed();

Assert.assertTrue(signoutFlag, "Signout Link not displayed");

signOutLink.click();

WebElement homeLink = driver.findElement(By.xpath("//a[text()='Home']"));

homeLink.click();

Thread.sleep(5000);

}

**Logging in Selenium**

*Logging* is the process of writing log messages during the execution of automation scripts . This logging allows you to report and persist error and warning messages as well as info messages and will be useful while debugging test scripts.

Sometimes logging is considered to be an overhead upon the existing script creation mechanism but experts consider it to be one of the best practices if used in the accurate proportion because of the following advantages:

* Grants a complete understanding of test suites execution
* Log messages can be stored in external files for post-execution scrutiny
* Logs are an exceptional assistant in debugging the program execution issues and failures
* Logs can also be reviewed to ascertain the application’s health by the stakeholders

Common logging machanism used in Selenium are Java Logging API, Log4J API etc.

**Log4J** – is an opens source Logging API from Apache. It is a fast, flexible and reliable logging framework (APIS) written in Java. Visit <https://logging.apache.org/log4j/2.x/> for more info.

**Java Logging API**

Java Logging API allows you to configure which message types are written. Individual classes can use this logger to write messages to the configured log files.

The java.util.logging package provides the logging capabilities via the Logger class.

Create Logger

To create a logger in your Java code, you can use the following snippet.

import java.util.logging.Logger;

// assumes the current class is called MyLogger

Logger log = Logger.getLogger(MyLoggerDemo.class.getName());

Logger Level

The log levels define the severity of a message. The Level class is used to define which messages should be written to the log.The following lists the Log Levels in descending order:

* SEVERE (highest)
* WARNING
* INFO
* CONFIG
* FINE
* FINER
* FINEST

In addition to that you also have the levels OFF and ALL to turn the logging off or to log everything.

For example, the following code sets the logger to the info level, which means all messages with severe, warning and info will be logged.

log.setLevel(Level.INFO);

Handler

Each logger can have access to several handlers.The handler receives the log message from the logger and exports it to a certain target.

A handler can be turned off with the setLevel(Level.OFF) method and turned on with setLevel()method. You have several standard handlers. The following are some examples.

* ConsoleHandler: Write the log message to console
* FileHandler: Writes the log message to file

Log levels INFO and higher will be automatically written to the console.

Formatter

Each handler’s output can be configured with a formatter.

Available formatters.

* SimpleFormatter: Generate all messages as text
* XMLFormatter: Generates XML output for the log messages

You can also build your own formatter.

Example of Logging for Testcases

public class LoggingDemo {

public String baseUrl = "http://newtours.demoaut.com";

public String browser = "FF";

public WebDriver driver ;

public String expTitle="Welcome: Mercury Tours";

public **Logger** **log** ;

public String currDirectory = new

File(System.getProperty("user.dir")).getAbsolutePath();

@**BeforeMethod**

public void setupLogger(**Method** method) throws Exception{

String logpath = currDirectory + File.separator + "logs" +

File.separator;

**log** = **Logger**.getLogger(logpath + method.getName()+ "\_result.txt");

FileHandler **handler** = new FileHandler(logpath + method.getName()+

"\_result.txt");

**log**.addHandler(handler);

**log**.setLevel(Level.ALL);

**log**.setUseParentHandlers(false);

**handler**.setFormatter(new SimpleFormatter());

}

@AfterMethod

public void closeHandlers() {

for(Handler h:log.getHandlers()){

h.close(); //To Close .lck(lock files)

}

}

@**Test** (enabled=true, priority=0)

public void launch() throws Exception {

System.setProperty("webdriver.gecko.driver","D:\\SeleniumTraining\\so

ftwares\\driverexes\\geckodriver.exe");

System.setProperty("webdriver.chrome.driver","D:\\SeleniumTraining\\s

oftwares\\driverexes\\chromedriver.exe");

if(browser.equalsIgnoreCase("FF")) {

**log**.info("Instantiating Firefox Driver");

driver = new FirefoxDriver();

} else {

**log**.info("Instantiating Chrome Driver");

driver = new ChromeDriver();

}

**log**.info("Maximizing window");

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

**log**.info("Setting temeouts..");

driver.manage().timeouts().pageLoadTimeout(60, TimeUnit.SECONDS);

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

driver.get(baseUrl);

**log**.info(" Broswer launched for URI:"+ baseUrl);

Thread.sleep(5000);

String pageTitle = driver.getTitle();

**log**.info(pageTitle);

Assert.assertEquals(pageTitle, expTitle, "Title is not matching");

**log**.info("Title Validation successful");

}

}

**Managing Browser Profiles**

A profile in Browser is the collection of settings, customizations, add-ons, and other personalizations that a user has made or installed into their copy of Browser.

Reasons to have multiple profiles

Web developers might want a secondary profile for testing websites, apps, or other projects on different Broswer channels. For example, you might want to have some extensions installed for Web development, but not for general-purpose Web browsing.

For QA, testing, and bug triaging contributors, you may want to have multiple development versions of Broswer installed, each with its own profile. Creating new profiles for testing can keep you from losing your preferences, bookmarks, and history. It takes little time to set up a new profile, and once it is complete, all of your Firefox versions will update separately and can be run simultaneously.

Similarly You can customize Browser profile to suit your Selenium automation requirement.

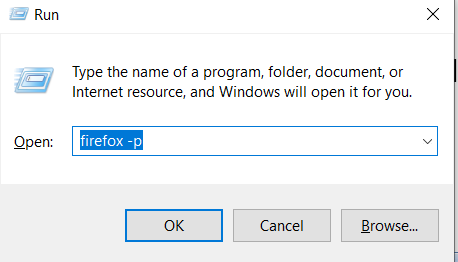
**Firefox Profile**

You cannot change the profile while Firefox is running.  Although it is possible in some cases to have multiple instances of Firefox running in different profiles, to avoid confusion, you should first exit/quit/terminate all running instances of *Firefox.*

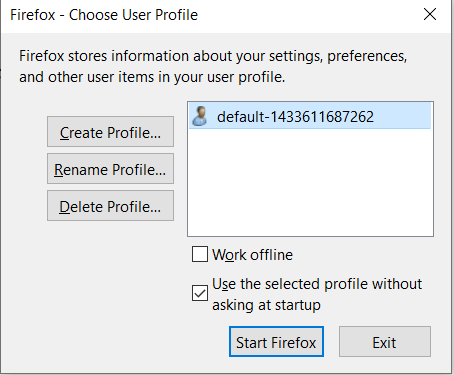
Creating a profile through the Profile Manager

Press "Windows + R" on your keyboard or launch Run dialog

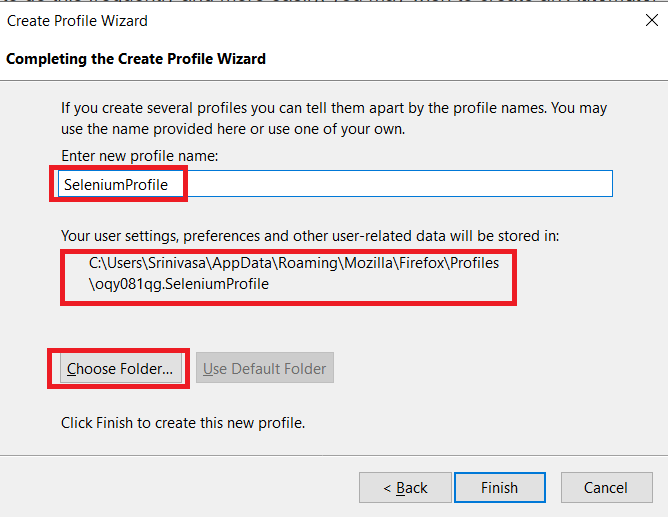
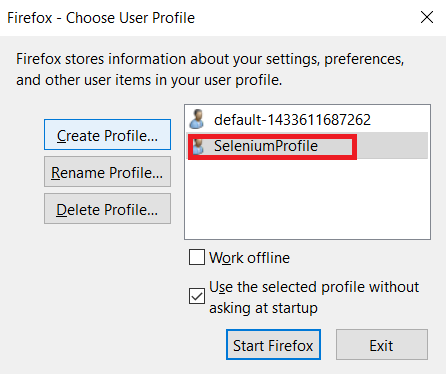
Type firefox –ProfileManager or firefox -p

****

Once you click on Ok button of Run dialog, Profile Manager window opened as shown below.



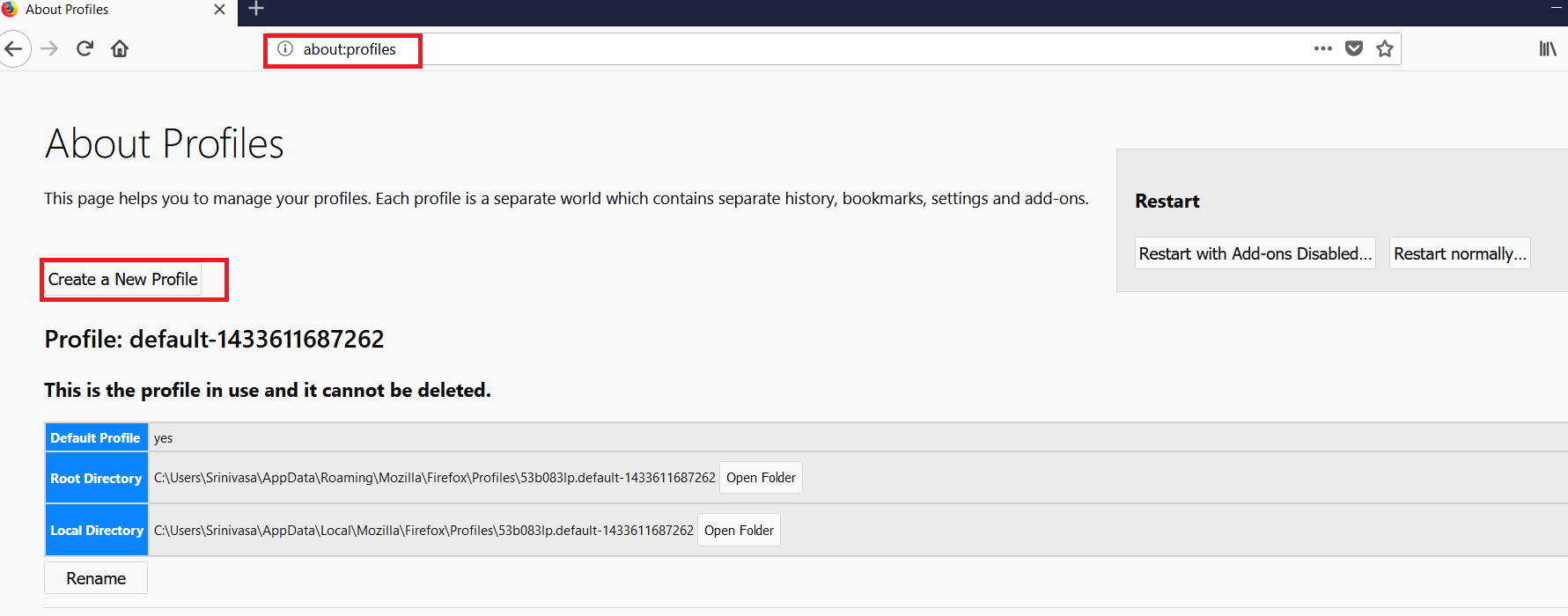
1. To start the Create Profile Wizard, click "Create Profile..." in the Profile Manager.
2. Click Next and enter the name of the profile. Use a profile name that is descriptive, such as your personal name. This name is not exposed to the Internet.
3. You can also choose where to store the profile on your computer. To select storage location, click Choose Folder....
4. If you choose your folder location for the profile, select a new or empty folder. If you choose a folder that isn't empty, and you later remove the profile and choose the "Delete Files" option, everything inside that folder will be deleted.
5. To create the new profile, click Finish as shown below.

Creating a profile through the Firefox browser

You can create a new Firefox profile directly from the browser.

1. Type **about:profiles** into the browser URL search bar
2. On the page, click Create a New Profile button
3. Read the introduction, and click Next
4. Enter a profile name for your new Profile. Use a profile name that is descriptive, such as your personal name. This name is not exposed to the Internet.
5. Optionally, to change where the profile will be stored on your computer, click Choose Folder...
6. To create the new profile, click Finish.



Using Firefox Profile in Selenium

**FirefoxProfile Class**

This class used to create Profile for Firefox and to set the preferenes to the profile.

Sample Usage:

FirefoxProfile profile = new **FirefoxProfile**(new File

"D:\\SeleniumTraining\\Code\\webdriver\\TestAutoamtionFramework\\profiles\\"));

profile.setPreference("browser.download.directory",

"D:\\SeleniumTraining\\Code\\webdriver\\TestAutoamtionFramework\\downloads");

profile.addExtension(new File(<Extension file name with path>));

**FirefoxOptions Class**

Manage firefox specific settings in a way that geckodriver can understand.

Sample Usage :

FirefoxOptions options = new FirefoxOptions()

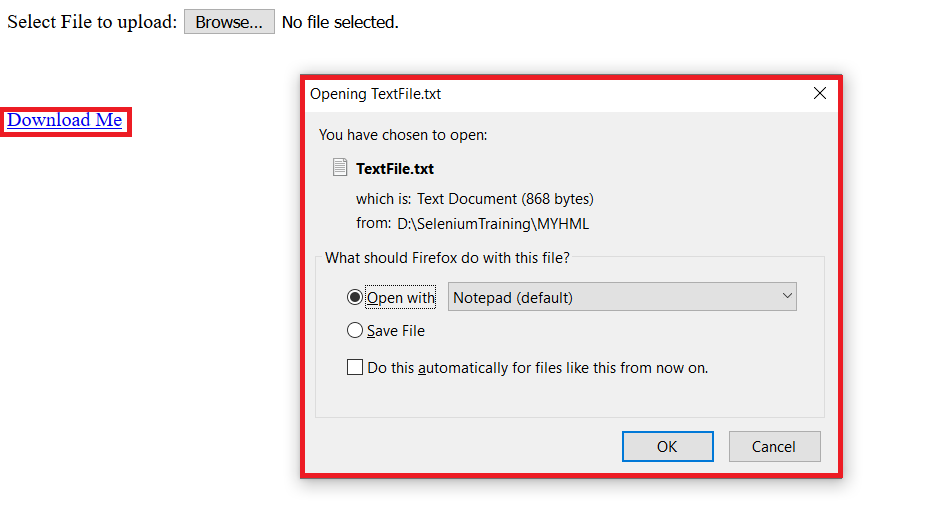
.addPreference("browser.startup.page", 1)

.addPreference("browser.startup.homepage", "https://www.google.com");

WebDriver driver = new FirefoxDriver(options);

**File Download from Application**

Classic case of applying profile and browser options is downloading a file from the application. When downloading a file from a link or button, browser may pop open dialog (like shown below) or it may download to User/Downloads folder based on the developer implementation.



Above case can be easily handled with SetPreference where we can instruct browser to dowanload all the files to the folder we want and by pass the open dialog to download file.

Example Code:

@Test

public void launch() throws InterruptedException {

System.setProperty("webdriver.gecko.driver","D:\\SeleniumTraining\\software

s\\driverexes\\geckodriver.exe");

// Creating firefox profile

FirefoxProfile profile = new FirefoxProfile();

**// Instructing firefox to use custom download location**

profile.setPreference("browser.download.folderList", 2);

**// Setting custom download directory**

profile.setPreference("browser.download.dir",

"D:\\SeleniumTraining\\Code\\webdriver\\TestAutomationFramework\\downloads\\");

**// Skipping Save As dialog box for types of files with their MIME**

profile.setPreference("browser.helperApps.neverAsk.saveToDisk",

"text/csv,application/java-archive, application/x-msexcel,application/excel,application/vnd.openxmlformats-officedocument.wordprocessingml.document,application/x-excel,application/vnd.ms-excel,image/png,image/jpeg,text/html,text/plain,application/msword,application/xml,application/vnd.microsoft.portable-executable");

// Creating FirefoxOptions to set profile

FirefoxOptions option = new FirefoxOptions();

option.setProfile(profile);

// Launching browser with desired capabilities

FirefoxDriver driver = new FirefoxDriver(option);

driver.get("file:D:/SeleniumTraining/MYHML/FileUpLoad.html");

Thread.sleep(5000);

driver.findElement(By.xpath("//a[text()='Download Me']")).click();

Thread.sleep(5000);

}

**Chrome Profile**

By default, ChromeDriver will create a new temporary profile for each session. At times you may want to set special preferences or just use a custom profile altogether. You can use the 'chrome.prefs' capability to specify preferences that will be applied after Chrome starts.

**ChromeOptions class**

Class to manage options specific to [ChromeDriver](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/chrome/ChromeDriver.html).You can create an instance of ChromeOptions, which has convenient methods for setting ChromeDriver-specific capabilities. You can then pass the ChromeOptions object into the ChromeDriver constructor:

ChromeOptions options = new ChromeOptions();

options.addExtensions(new File("/path/to/extension.crx"));

ChromeDriver driver = new ChromeDriver(options);

Since Selenium version 3.6.0, the ChromeOptions class in Java also implements the Capabilities interface, allowing you to specify other WebDriver capabilities not specific to ChromeDriver.

ChromeOptions options = new ChromeOptions();

//Capabilities

capabilities = DesiredCapabilities.chrome();

capabilities.setCapability(CapabilityType.ACCEPT\_SSL\_CERTS, true);

capabilities.setCapability(ChromeOptions.CAPABILITY, options);

// Add a ChromeDriver-specific capability.

options.addExtensions(new File("/path/to/extension.crx"));

ChromeDriver driver = new ChromeDriver(options);

Exmaple for Downaloading a file:

System.setProperty("webdriver.chrome.driver","D:\\SeleniumTraining\\softwares\\driverexes\\chromedriver.exe");

// Setting new download directory path

Map<String, Object> **prefs** = new HashMap<String, Object>();

// Use File.separator as it will work on any OS

prefs.put("download.default\_directory",

"D:\\SeleniumTraining\\Code\\webdriver\\TestAutomationFramework\\downloads\\");

// Adding capabilities to ChromeOptions

ChromeOptions **options** = new ChromeOptions();

options.**setExperimentalOption**("prefs", **prefs**);

// Launching browser with desired capabilities

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

driver.get("file:D:/SeleniumTraining/MYHML/FileUpLoad.html");

Thread.sleep(5000);

driver.findElement(By.xpath("//a[text()='Download Me']")).click();

Thread.sleep(5000);

**File Uploading**

File uploading for an input/button will be perofrmed using senkdKeys as like text input box. However the value sending should be path of the valid file.

Example:

driver.findElement(By.xpath("//input[@name='fileToUpload']")).sendKeys("D:\\SeleniumTraining\\MYHML\\chromedriver\_win32.zip");

**Selenium Grid**

Selenium-Grid allows you run your tests on different machines against different browsers in parallel. That is, running multiple tests at the same time against different machines running different browsers and operating systems. Essentially, Selenium-Grid support distributed test execution. It allows for running your tests in a *distributed test execution* environment.

When to Use It

Two reasons why you might want to use Selenium-Grid.

* To run your tests against multiple browsers, multiple versions of browser, and browsers running on different operating systems.
* To reduce the time it takes for the test suite to complete a test pass.

How Selenium-Grid Works–With a Hub and Nodes

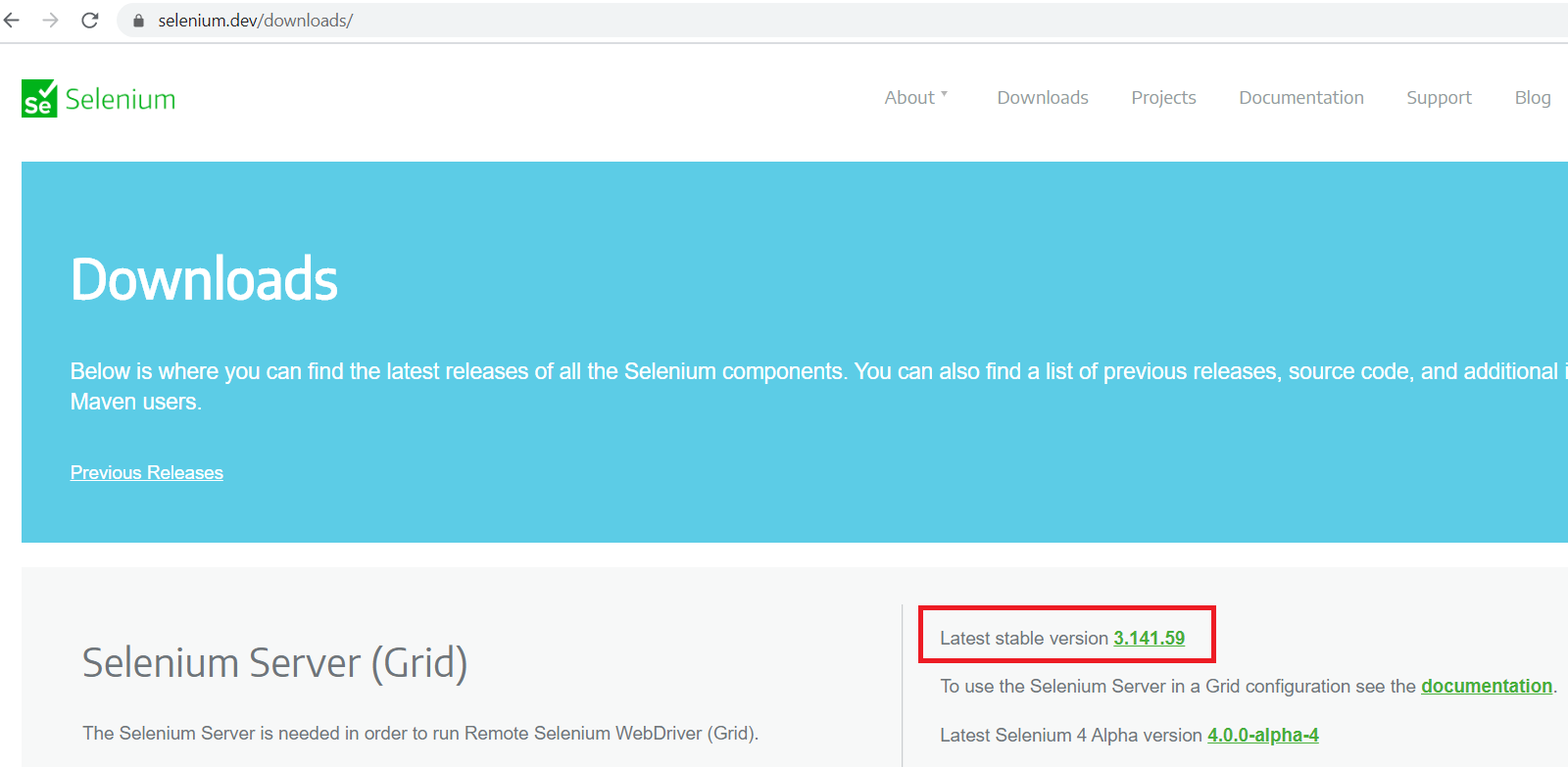
A grid consists of a single hub, and one or more nodes. Both are started using the selenium-server.jar executable. The hub receives a test to be executed along with information on which browser and ‘platform’ (i.e. WINDOWS, LINUX, etc) where the test should be run. It ‘knows’ the configuration of each node that has been ‘registered’ to the hub.

Using this information it selects an available node that has the requested browser-platform combination. Once a node has been selected, Selenium commands initiated by the test are sent to the hub, which passes them to the node assigned to that test.

The node runs the browser, and executes the Selenium commands within that browser against the application under test.

Selenium Grid Installation

Installation is simple. Download the Selenium-Server jar file from the [SeleniumHq website’s download page](http://seleniumhq.org/download/). (<https://www.selenium.dev/downloads/>)



Install it in a folder of your choice. You’ll need to be sure the java executable is on your execution path so you can run it from the command-line. If it does not run correcly, verify your system’s path variable includes the path to the java.exe.

Starting Selenium-Grid

***Starting a Hub***

To start a hub with default parameters, run the following command from a command-line shell. This will work on all the supported platforms, Windows Linux, or Mac OSX.

**java -jar selenium-server-standalone-3.141.59.jar -role hub**

This starts a hub using default parameter values. We’ll explain these parameters in folowing subsections. Note that you will likely have to change the version number in the jar filename depending on which version of the selenium-server you’re using.

You can check Hub using <http://localhost:4444/grid/console>



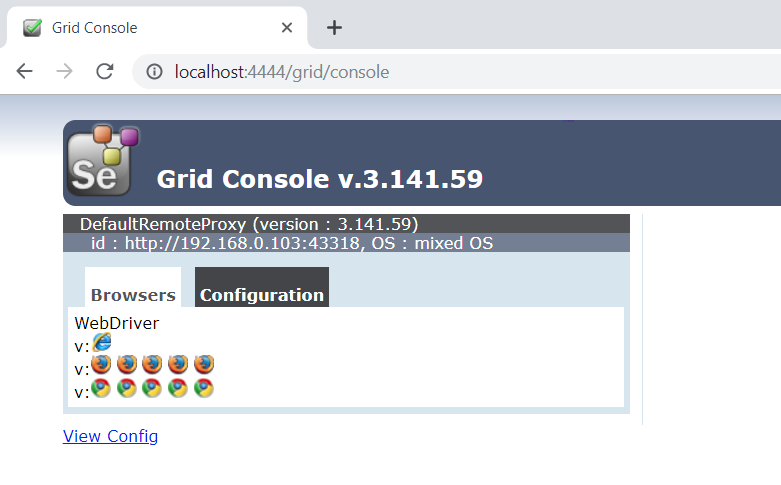
***Starting a Node***

To start a node using default parameters, run the following command from a command-line.

**java -Dwebdriver.chrome.driver="D:\SeleniumTraining\softwares\driverexes\chromedriver.exe" -Dwebdriver.gecko.driver="D:\SeleniumTraining\softwares\driverexes\geckodriver.exe" -jar selenium-server-standalone-3.141.59.jar -role node -hub** [**http://localhost:4444/grid/register**](http://localhost:4444/grid/register)

This assumes the hub has been started above using default parameters. The default port the hub uses to listen for new requests is port 4444. This is why port 4444 was used in the URL for locating the hub. Also the use of ‘localhost’ assumes your node is running on the same machine as your hub. For getting started this is probably easiest. If running the hub and node on separate machines, simply replace ‘localhost’ with the hostname of the remote machine running the hub.

WARNING: Be sure to turn off the firewalls on the machine running your hub and nodes. Otherwise you may get connection errors.

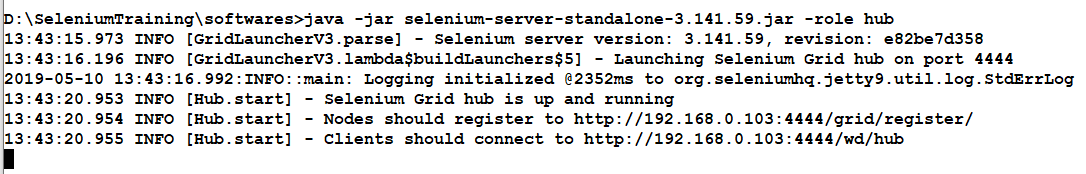


Hub Configuration

To run the hub using the default options simply specify -role hub to the Selenim-Server

**java -jar selenium-server-standalone-3.141.59.jar -role hub**

You should see the following logging output.



If you see error like

Exception in thread "main" java.net.BindException: Selenium is already running on port 4444. Or some other service

If this occurs you can either shutdown the other process that is using port 4444, or you can tell Selenium-Grid to use a different port for its hub. Use the -port option for changing the port used by the hub.

**java -jar selenium-server-standalone-3.141.59.jar -role hub –port 4441**

This will work even if another hub is already running on the same machine, that is, as long as they’re both not using port 4441.

Designing Test Scripts That Can Run on the Grid

To design test scripts that will run on the grid, we need to use the RemoteWebDriver objects.

* RemoteWebDriver is used to set which node (or machine) that our test will run against.

Using the FirefoxOptions Object

The way we used to construct a FirefoxDriver is by passing desired capabilities object like FirefoxDriver(Capabilities desiredCapabilities) which is now Deprecated. But from Selenium v3.6.0 it is recommended to construct a FirefoxDriver with FirefoxOptions

FirefoxOptions opt = new FirefoxOptions();

Using the ChromeOptions Object

Like FirefoxOptions other browsers too have Browser options class. Chrome Browser options can be added by creating ChromeOptions class like below.

ChromeOptions opt = new ChromeOptions();

Note: Please refer the latest documentation on Firefox and Chrome options to update as they may change Build to Build.

Using the RemoteWebDriver Object

To use the RemoteWebDriverobject, you must import these packages.

import java.net.URL;

import org.openqa.selenium.remote.RemoteWebDriver;

Import the necessary packages for RemoteWebDriver and then pass the Broswer Options that we created above as a parameter for the RemoteWebDriver object.

WebDriver driver = new RemoteWebDriver(new URL("http://192.168.0.103:4444/wd/hub"), opt);

A Sample Test code for Grid

@Test

public void launch() throws Exception {

System.setProperty("webdriver.gecko.driver","D:\\SeleniumTraining\\softwares\\driver

exes\\geckodriver.exe");

System.setProperty("webdriver.chrome.driver","D:\\SeleniumTraining\\softwares\\drive

rexes\\chromedriver.exe");

String nodeUrl = **"http://localhost:4444/wd/hub";**

if(browser.equalsIgnoreCase("firefox")) {

FirefoxOptions opt = new FirefoxOptions();

driver = new **RemoteWebDriver**(new URL(nodeUrl), opt);

} else if (browser.equalsIgnoreCase("chrome")) {

ChromeOptions opt = new ChromeOptions();

driver = new **RemoteWebDriver**(new URL(nodeUrl), opt);

}

driver.get(baseURL);

System.out.println("Maximizing window");

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

Thread.sleep(2000);

System.out.println("Setting temeout Settings..");

driver.manage().timeouts().pageLoadTimeout(300, TimeUnit.SECONDS);

driver.manage().timeouts().implicitlyWait(120, TimeUnit.SECONDS);

driver.get(baseURL);

System.out.println(" Broswer launched for URI:"+ baseURL);

String getURLStr = driver.getCurrentUrl();

boolean rFlag = getURLStr.contains("newtours.demoaut.com");

Assert.assertTrue(rFlag, "Failed to Launch");

}

Sample testng.xml to run

<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >

<suite name="Automation Framework" parallel="**tests**" thread-count="**2**">

<test name="Mercury Tours" >

<classes >

<class name="com.h2ki.grid.Grid\_Setup" />

</classes>

</test>

<test name="Mercury Tours2" >

<classes >

<class name="com.h2ki.grid.GridDemo" />

</classes>

</test>

</suite>

**Maven**

What is Maven?

Maven ([**http://maven.apache.org**](http://maven.apache.org/) ) is a powerful build tool for Java software projects. The word [maven](https://en.wikipedia.org/wiki/Maven) means “accumulator of knowledge”. Actually, you can build software projects using other languages too, but Maven is developed in Java, and is thus historically used more for Java projects.

What is a Build Tool?

A build tool is a tool that automates everything related to building the software project. Building a software project typically includes one or more of these activities:

* Generating source code (if auto-generated code is used in the project).
* Generating documentation from the source code.
* Compiling source code.
* Packaging compiled code into JAR files or ZIP files.
* Installing the packaged code on a server, in a repository or somewhere else.

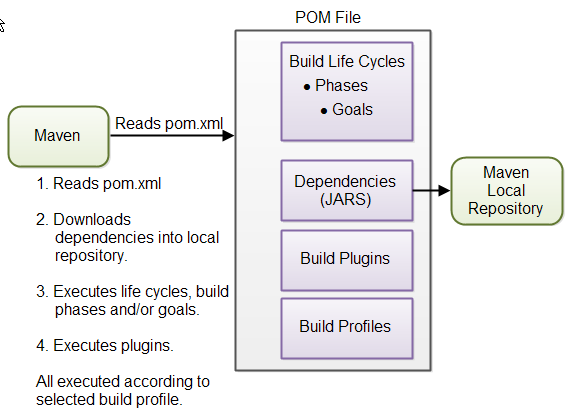
Any given software project may have more activities than these needed to build the finished software. Such activities can normally be plugged into a build tool, so these activities can be automated too.

The advantage of automating the build process is that you minimize the risk of humans making errors while building the software manually. Additionally, an automated build tool is typically faster than a human performing the same steps manually.

Maven Overview

Maven is centered around the concept of POM files (Project Object Model). A POM file is an XML representation of project resources like source code, test code, dependencies (external JARs used) etc. The POM contains references to all of these resources. The POM file should be located in the root directory of the project it belongs to.

Here is a diagram illustrating how Maven uses the POM file, and what the POM file primarily contains:



|  |
| --- |
|  |

**POM Files** - When you execute a Maven command you give Maven a POM file to execute the commands on. Maven will then execute the command on the resources described in the POM.

**Build Life Cycles, Phases and Goals -** The build process in Maven is split up into build life cycles, phases and goals. A build life cycle consists of a sequence of build phases, and each build phase consists of a sequence of goals. When you run Maven you pass a command to Maven. This command is the name of a build life cycle, phase or goal. If a life cycle is requested executed, all build phases in that life cycle are executed. If a build phase is requested executed, all build phases before it in the pre-defined sequence of build phases are executed too.

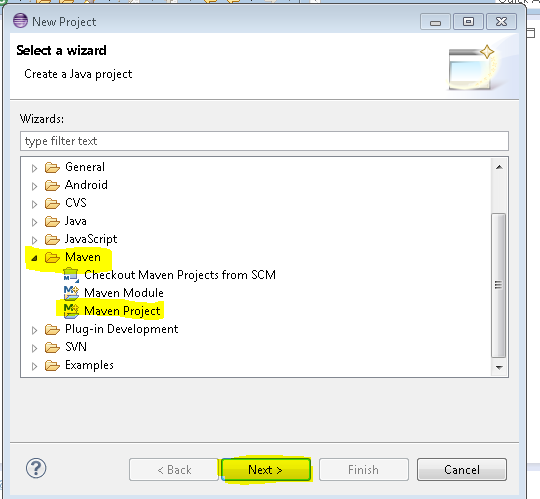
**Dependencies and Repositories -** One of the first goals Maven executes is to check the dependencies needed by your project. Dependencies are external JAR files (Java libraries) that your project uses. If the dependencies are not found in the local Maven repository, Maven downloads them from a central Maven repository and puts them in your local repository. The local repository is just a directory on your computer's hard disk. You can specify where the local repository should be located if you want to (I do). You can also specify which remote repository to use for downloading dependencies.

**Build Plugins -** Build plugins are used to insert extra goals into a build phase. If you need to perform a set of actions for your project which are not covered by the standard Maven build phases and goals, you can add a plugin to the POM file. Maven has some standard plugins you can use, and you can also implement your own in Java if you need to.

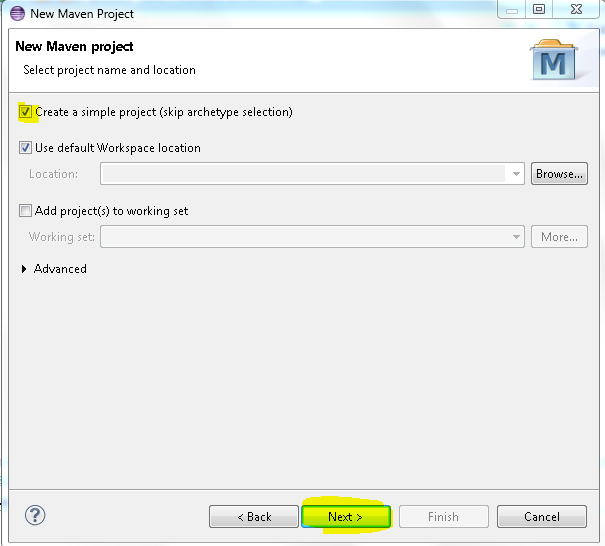
**Build Profiles -** Build profiles are used if you need to build your project in different ways. For instance, you may need to build your project for your local computer, for development and test. And you may need to build it for deployment on your production environment. These two builds may be different. To enable different builds you can add different build profiles to your POM files. When executing Maven you can tell which build profile to use.

Setting up Maven Project in Eclipse

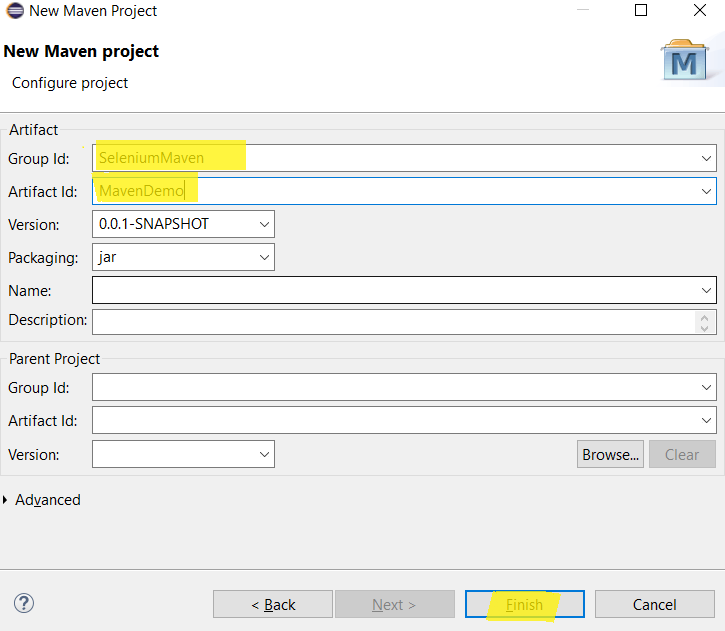
**Step1-** Open Eclipse and Click on New > Project. If you are seeing Maven Project like below means Maven already added to Eclipse. If not, install from Help > Eclipse Market place. Select below as highlighed and click on Next button.



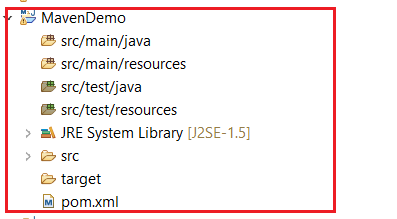
**Step2-** Select checkbox as shown below and click on Next button.



**Step3-** Enter a GroupId and ArtifactId of your choice and click on Finish button



**Step4:** A Project is created with ArtifactId and you would see pom.xml is created for the Project.



And you would see the content of the pom.xml like below

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>SeleniumMaven</groupId>

<artifactId>MavenDemo</artifactId>

<version>0.0.1-SNAPSHOT</version>

</project>

Maven POM Files

A Maven POM file (Project Object Model) is an XML file that describe the resources of the project. This includes the directories where the source code, test source etc. is located in, what external dependencies (JAR files) your projects has etc.

The POM file describes what to build, but most often not how to build it. How to build it is up to the Maven build phases and goals. You can insert custom actions (goals) into the Maven build phase if you need to, though.

Each project has a POM file. The POM file is named pom.xml and should be located in the root directory of your project. A project divided into subprojects will typically have one POM file for the parent project, and one POM file for each subproject. This structure allows both the total project to be built in one step, or any of the subprojects to be built separately.

Project Dependencies

Unless your project is small, your project may need external Java APIs or frameworks which are packaged in their own JAR files. These JAR files are needed on the classpath when you compile your project code.

Keeping your project up-to-date with the correct versions of these external JAR files can be a comprehensive task. Each external JAR may again also need other external JAR files etc. Downloading all these external dependencies (JAR files) recursively and making sure that the right versions are downloaded is cumbersome. Especially when your project grows big, and you get more and more external dependencies.

Luckily, Maven has built-in dependency management. You specify in the POM file what external libraries your project depends on, and which version, and then Maven downloads them for you and puts them in your local Maven repository. If any of these external libraries need other libraries, then these other libraries are also downloaded into your local Maven repository.

You specify your project dependencies inside the dependencies element in the POM file. You can find selenium maven refernce at <http://docs.seleniumhq.org/download/maven.jsp>

<dependency>

<groupId>org.seleniumhq.selenium</groupId>

<artifactId>selenium-java</artifactId>

<version>3.141.59</version>

</dependency>

External Dependencies

An external dependency in Maven is a dependency (JAR file) which is not located in a Maven repository (neiterh local, central or remote repository). It may be located somewhere on your local hard disk, for instance in the lib directory of a webapp, or somewhere else. The word "external" thus means external to the Maven repository system - not just external to the project. Most dependencies are external to the project, but few are external to the repository system (not located in a repository).

You configure an external dependency like this:

<dependency>

<groupId>mydependency</groupId>

<artifactId>mydependency</artifactId>

<scope>system</scope>

<version>1.0</version>

<systemPath>${basedir}\war\WEB-INF\lib\mydependency.jar</systemPath>

</dependency>

Adding TestNG Repository to Maven

#### You can find TestNg maven sample refernce at <http://testng.org/doc/maven.html>

<dependency>

<groupId>org.testng</groupId>

<artifactId>testng</artifactId>

<version>6.14.3</version>

<scope>test</scope>

</dependency>

Snapshot Dependencies

Snapshot dependencies are dependencies (JAR files) which are under development. Instead of constantly updating the version numbers to get the latest version, you can depend on a snapshot version of the project. Snapshot versions are always downloaded into your local repository for every build, even if a matching snapshot version is already located in your local repository. Always downloading the snapshot dependencies assures that you always have the latest version in your local repository, for every build.

You can tell Maven that your project is a snapshot version simply by appending -SNAPSHOT to the version number in the beginning of the POM (where you also set the groupId and artifactId). Here is a version element example:

<version>1.0-SNAPSHOT</version>

Notice the -SNAPSHOT appended to the version number.

Depending on a snapshot version is also done by appending the -SNAPSHOT after the version number when configuring dependencies. Here is an example:

<dependency>

<groupId>com.h2k</groupId>

<artifactId>selenium\_project</artifactId>

<version>1.0-SNAPSHOT</version>

</dependency>

Maven Repositories

Maven repositories are directories of packaged JAR files with extra meta data. The meta data are POM files describing the projects each packaged JAR file belongs to, including what external dependencies each packaged JAR has. It is this meta data that enables Maven to download dependencies of your dependencies recursively, until the whole tree of dependencies is download and put into your local repository.

Maven has three types of repository:

* Local repository
* Central repository
* Remote repository

Maven searches these repositories for dependencies in the above sequence. First in the local repository, then in the central repository, and third in remote repositories if specified in the POM.

Here is a diagram illustrating the three repository types and their location:

|  |
| --- |
| Maven Repository Types and Location. |

Maven Cental Repository can be found at <https://mvnrepository.com/>

Maven Build Life Cycles, Phases and Goals

When Maven builds a software project it follows a build life cycle. The build life cycle is divided into build phases, and the build phases are divided into build goals.

**Build Life Cycles**  
Maven has 3 built-in build life cycles. These are:

* default
* clean
* site

Each of these build life cycles takes care of a different aspect of building a software project. Thus, each of these build life cycles are executed independently of each other. You can get Maven to execute more than one build life cycle, but they will be executed in sequence, separately from each other, as if you had executed two separate Maven commands.

The default life cycle handles everything related to compiling and packaging your project. The clean life cycle handles everything related to removing temporary files from the output directory, including generated source files, compiled classes, previous JAR files etc. The site life cycle handles everything related to generating documentation for your project. In fact, site can generate a complete website with documentation for your project.

**Build Phases**

Each build life cycle is divided into a sequence of build phases, and the build phases are again subdivided into goals. Thus, the total build process is a sequence of build life cycle(s), build phases and goals.

You can execute either a whole build life cycle like clean or site, a build phase like install which is part of the default build life cycle, or a build goal like dependency:copy-dependencies.

Note: You cannot execute the default life cycle directly. You have to specify a build phase or goal inside the default life cycle.

When you execute a build phase, all build phases before that build phase in this standard phase sequence are executed. Thus, executing the install build phase really means executing all build phases before the install phase, and then execute the install phase after that.

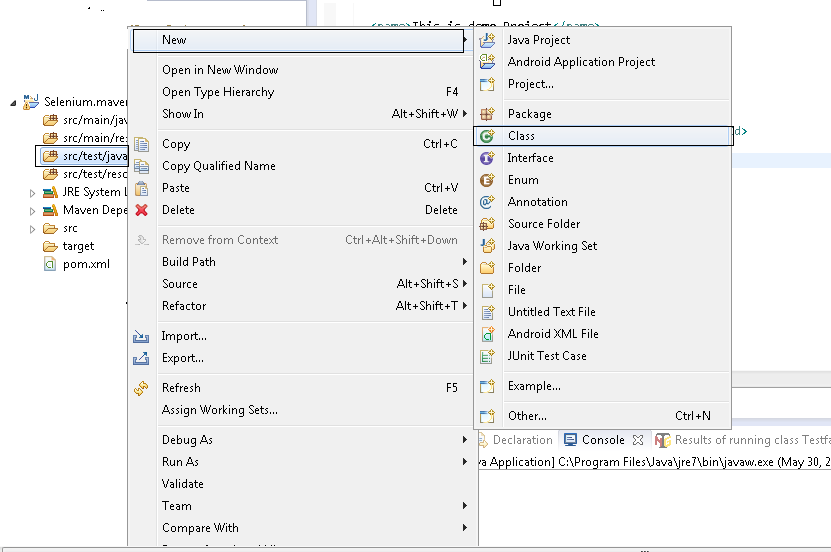
The default life cycle is of most interest since that is what builds the code. Since you cannot execute the default life cycle directly, you need to execute a build phase or goal from the default life cycle. The default life cycle has an extensive sequence of build phases and goals, ,so I will not describe them all here. The most commonly used build phases are:

|  |  |
| --- | --- |
| **Build Phase** | **Description** |
| validate | Validates that the project is correct and all necessary information is available. This also makes sure the dependencies are downloaded. |
| compile | Compiles the source code of the project. |
| test | Runs the tests against the compiled source code using a suitable unit testing framework. These tests should not require the code be packaged or deployed. |
| package | Packs the compiled code in its distributable format, such as a JAR. |
| install | Install the package into the local repository, for use as a dependency in other projects locally. |
| deploy | Copies the final package to the remote repository for sharing with other developers and projects. |

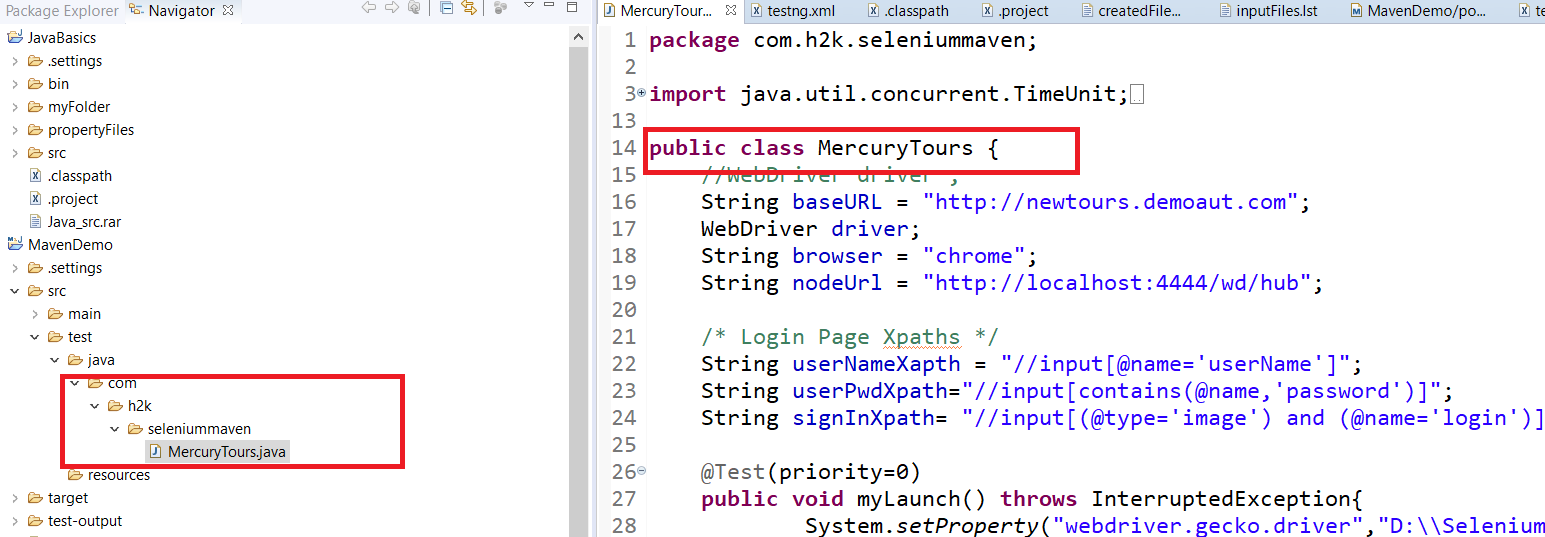
**Build Goals**  
Build goals are the finest steps in the Maven build process. A goal can be bound to one or more build phases, or to none at all. If a goal is not bound to any build phase, you can only execute it by passing the goals name to the mvn command. If a goal is bound to multiple build phases, that goal will get executed during each of the build phases it is bound to.

Writing Test cases in Maven

Inside project select src>test>java and right click then create a class



And Add Selenium code as shown below.



Create testng.xml

<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >

<suite name="Maven Project">

<test name="Maven Demo" >

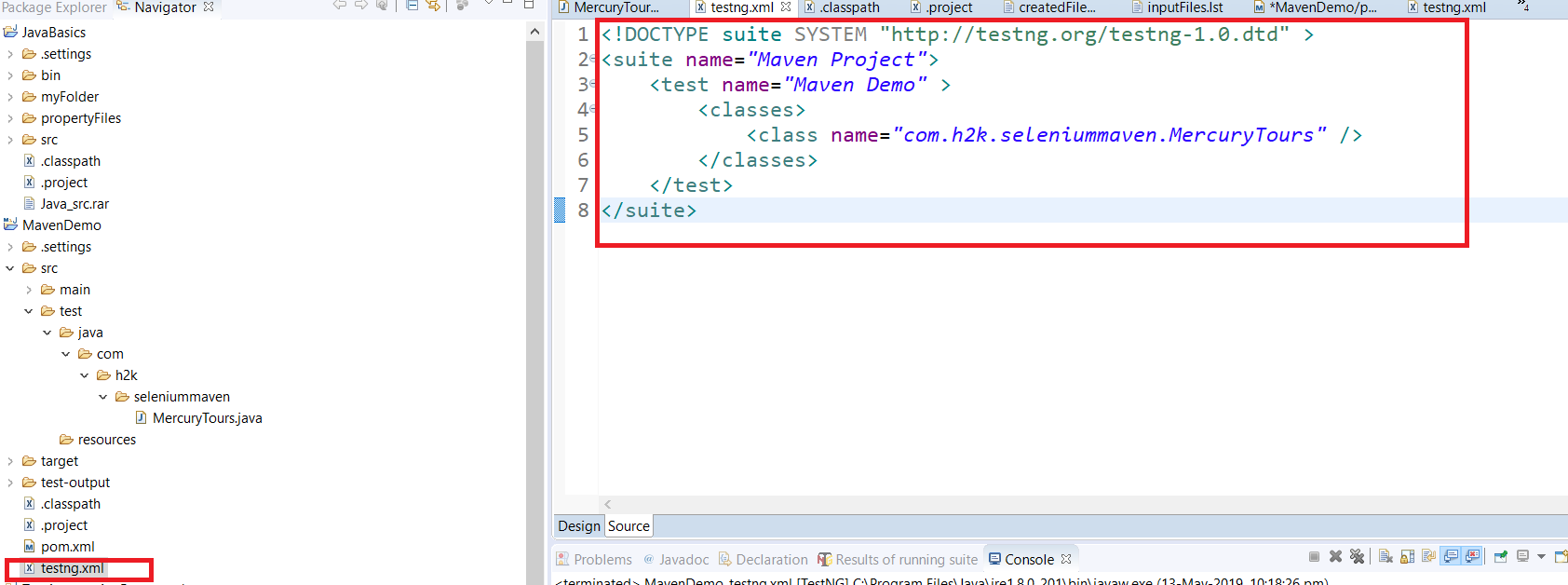
<classes>

<class name="com.h2k.seleniummaven.MercuryTours" />

</classes>

</test>

</suite>

****

Running Maven Project in Eclipse

In pom.xml , add maven compiler plug-in and maven surefire plug-in to run testng.xml. So your pom.xml looks like below

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>SeleniumMaven</groupId>

<artifactId>MavenDemo</artifactId>

<version>0.0.1-SNAPSHOT</version>

<dependencies>

<dependency>

<groupId>org.seleniumhq.selenium</groupId>

<artifactId>selenium-java</artifactId>

<version>3.141.59</version>

</dependency>

<dependency>

<groupId>org.testng</groupId>

<artifactId>testng</artifactId>

<version>6.14.3</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.8.1</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

<version>3.0.0-M3</version>

<configuration>

<suiteXmlFiles>

<suiteXmlFile>testng.xml</suiteXmlFile>

</suiteXmlFiles>

</configuration>

</plugin>

</plugins>

</build>

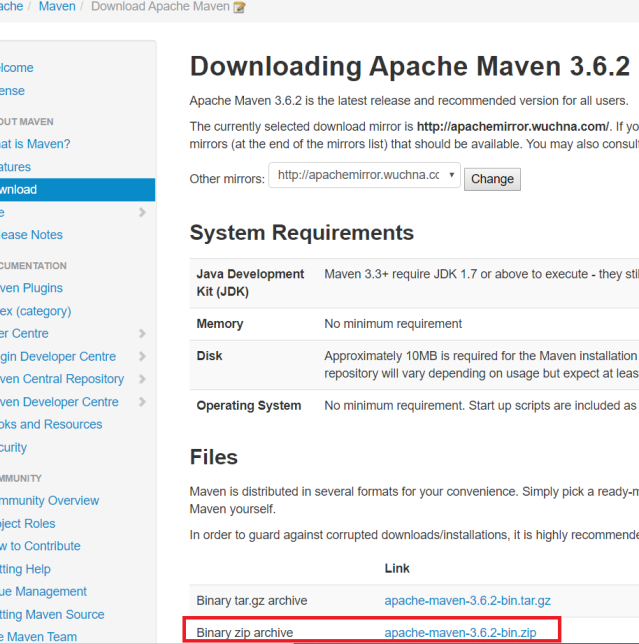
</project>

Select pom.xml and right click onit, Select Run As > Maven Test

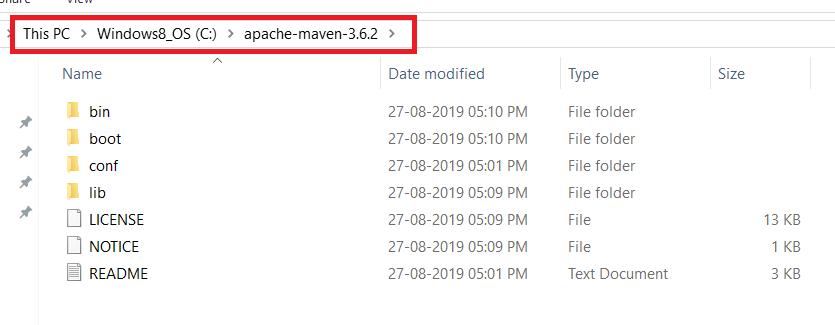


Running Maven from Command prompt

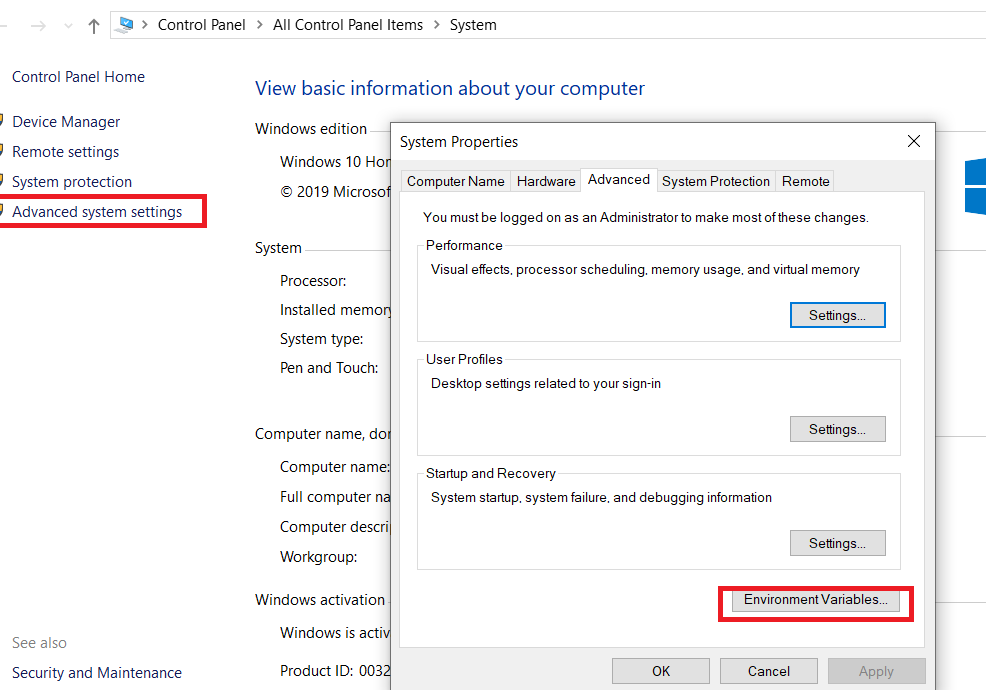
**Step1**: Downlaod Maven zip from <https://maven.apache.org/download.cgi>



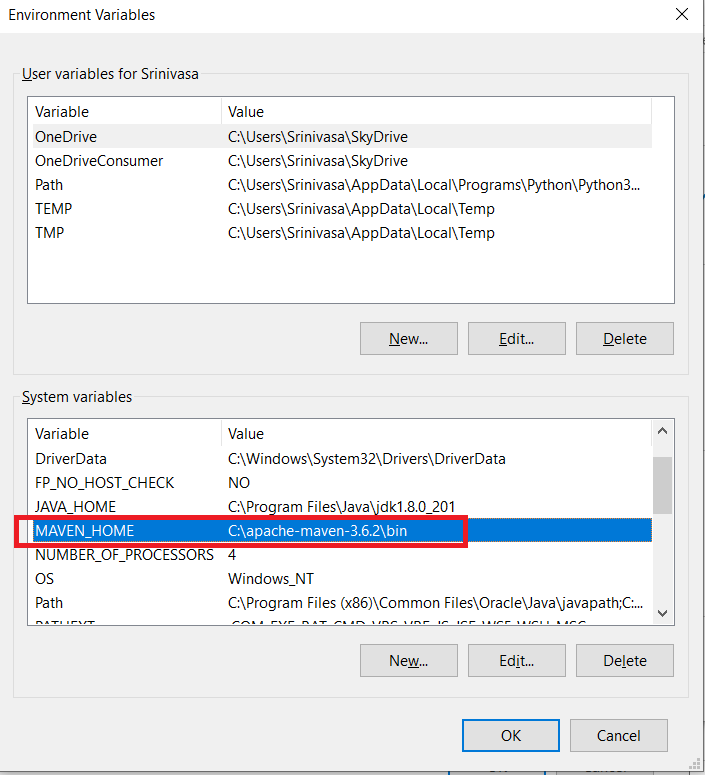
**Step2**: Unzip downloaded zip file in an location ( for eg: C:\Maven) and no further installation required.



**Step3**: Goto Control Panel->System->Adavanced Settings->Environment Variables

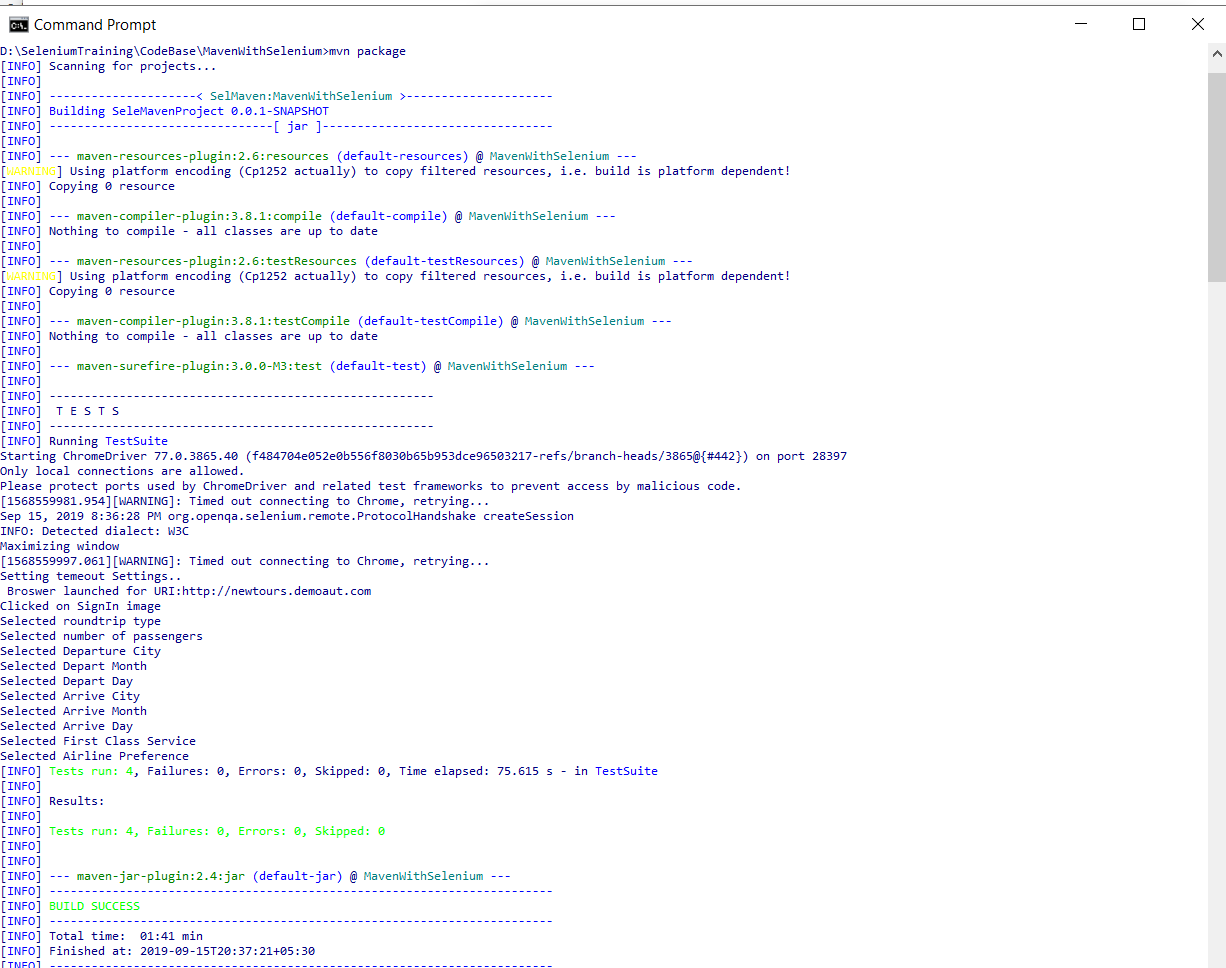


**Step4**: Add system Env Variable MAVEN\_HOME with the Maven folder path like below and clikc on OK button. Now you are set to run Maven from command prompt



**Step4**: To run Maven pom.xml , open command prompt and navigate to project folder where pom.xml available. Run command “mvn <goals>“ and you can see run

Eg: mvn test



**Step4**: Goto testng-output folder, refresh and open index.html for latest results.

**Jenkins for Selenium Automation**

What is Jenkins?

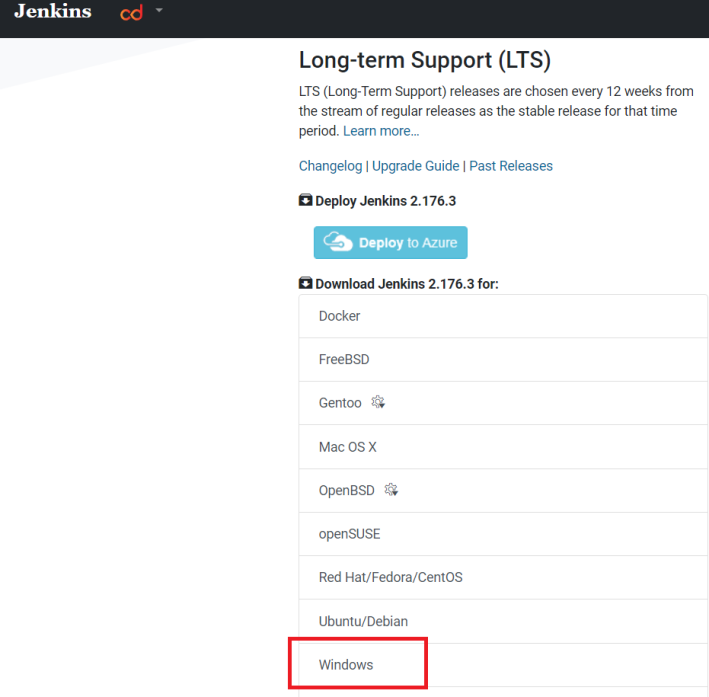
Jenkins is a self-contained, open source automation server which can be used to automate all sorts of tasks related to building, testing, and delivering or deploying software.

Jenkins can be installed through native system packages, Docker, or even run standalone by any machine with a Java Runtime Environment (JRE) installed.

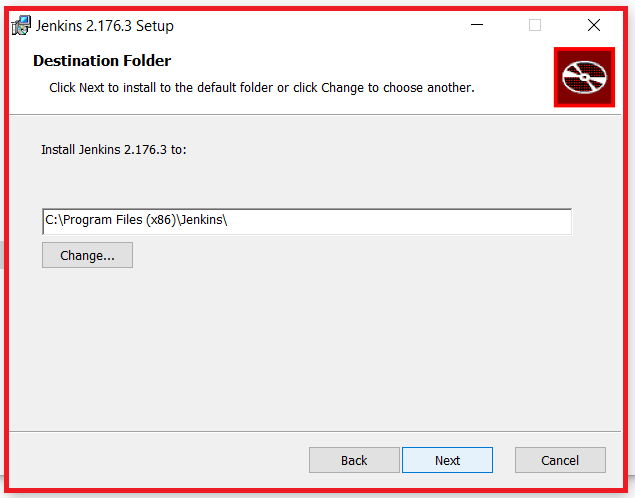
For more help and documentation visit<https://jenkins.io/doc/>

Installation Steps for Windows using Installer (*Used Jenkins version -2.176.3*)

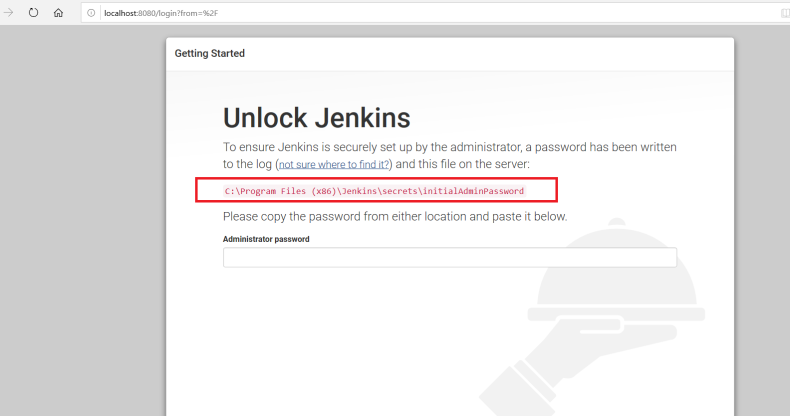
**Step1:** Download jenkins from <https://jenkins.io/download/> for Windows. An installation file will be downloaded to your downloads folder.



**Step2:**  Locate downloaded zip and unzipping zip file. You can find jenkins installer file. Click on installer file and installation process starts with following screesn and click on “Next” button

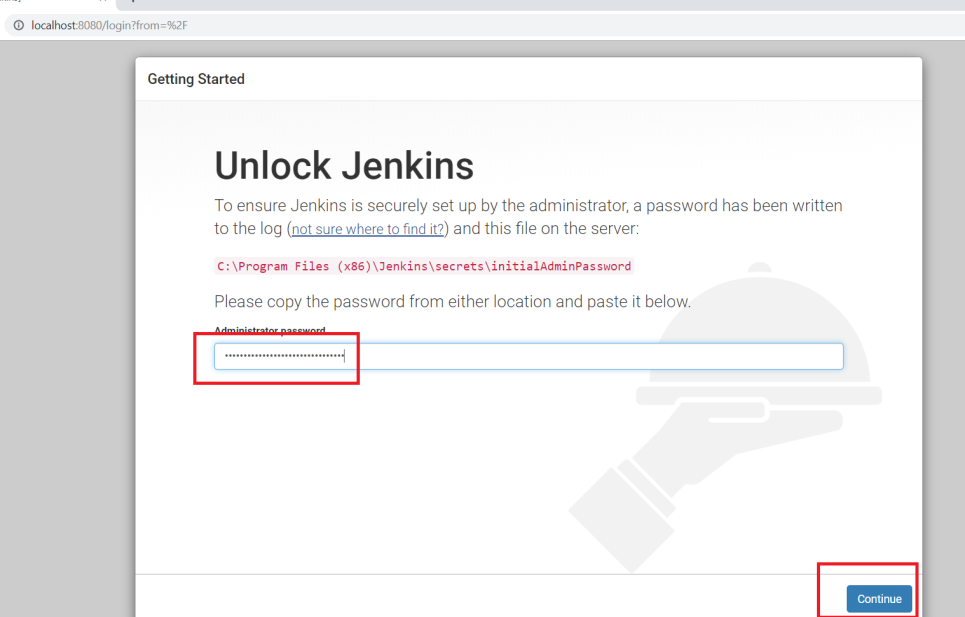


**Step3:** Once installation complete, you can see a webbrowser launched and you see the page as following (<http://localhost:8080>)

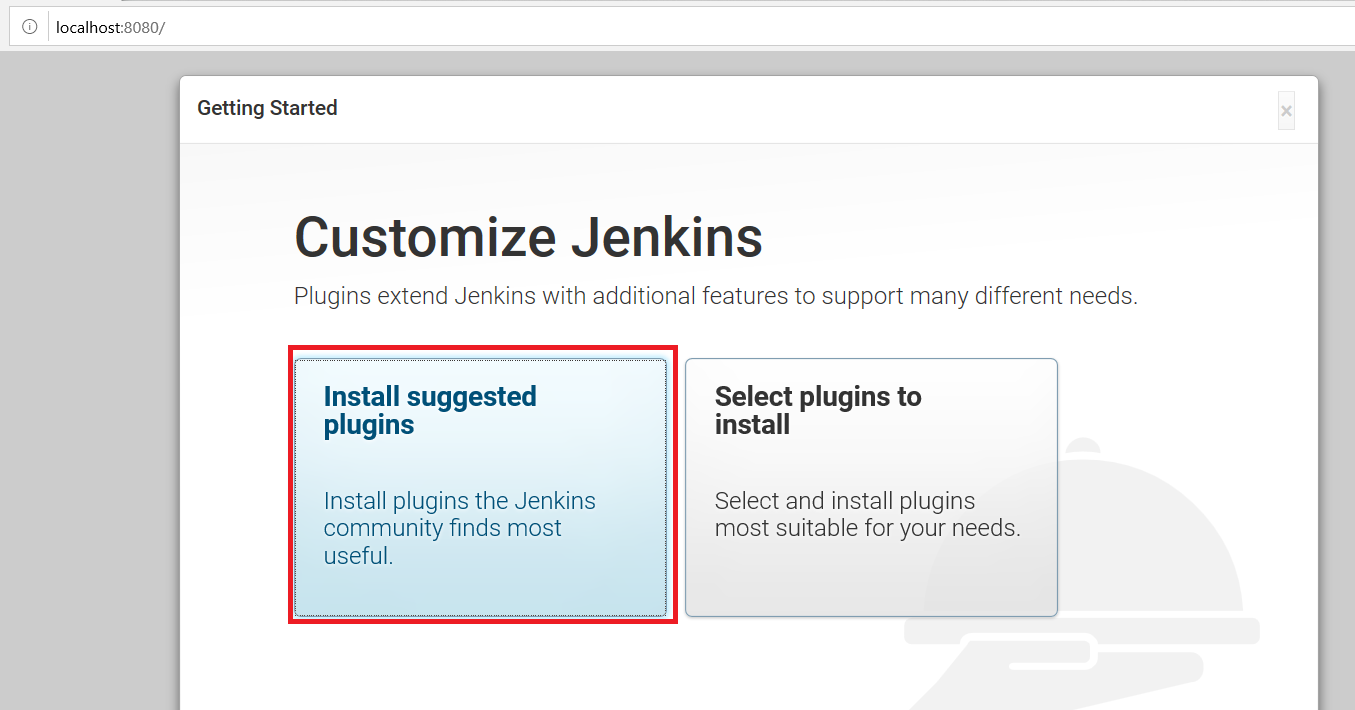


**Note:** *If you are not able to see the above page, your installation may not have done properly. As Jenkins use port 8080 by default, check if any other program is using this port.*

**Step4:** Start setting up your jenkins by entering password mentioned in the above path and click on continue



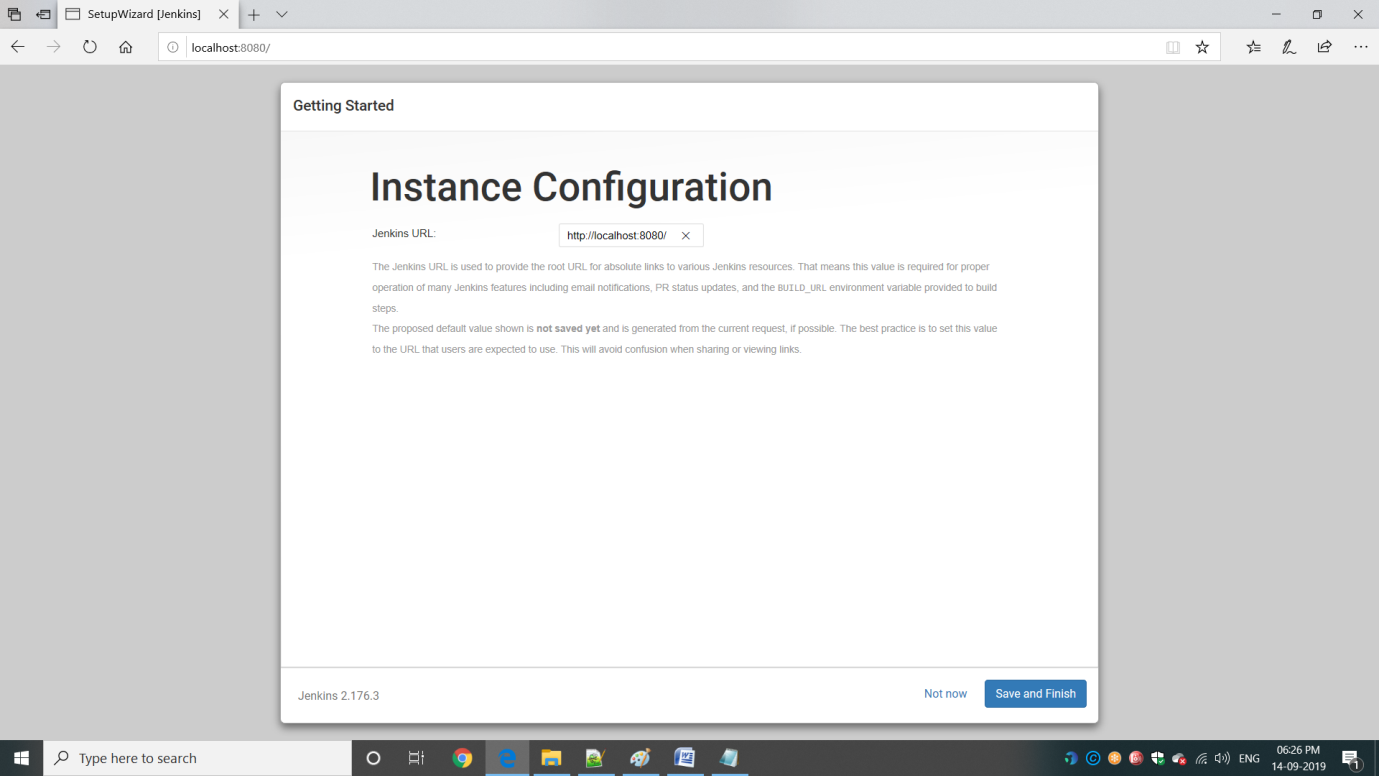
**Step5:** Install suggested plug-ins as shown below ( This may take few mins time to complete)



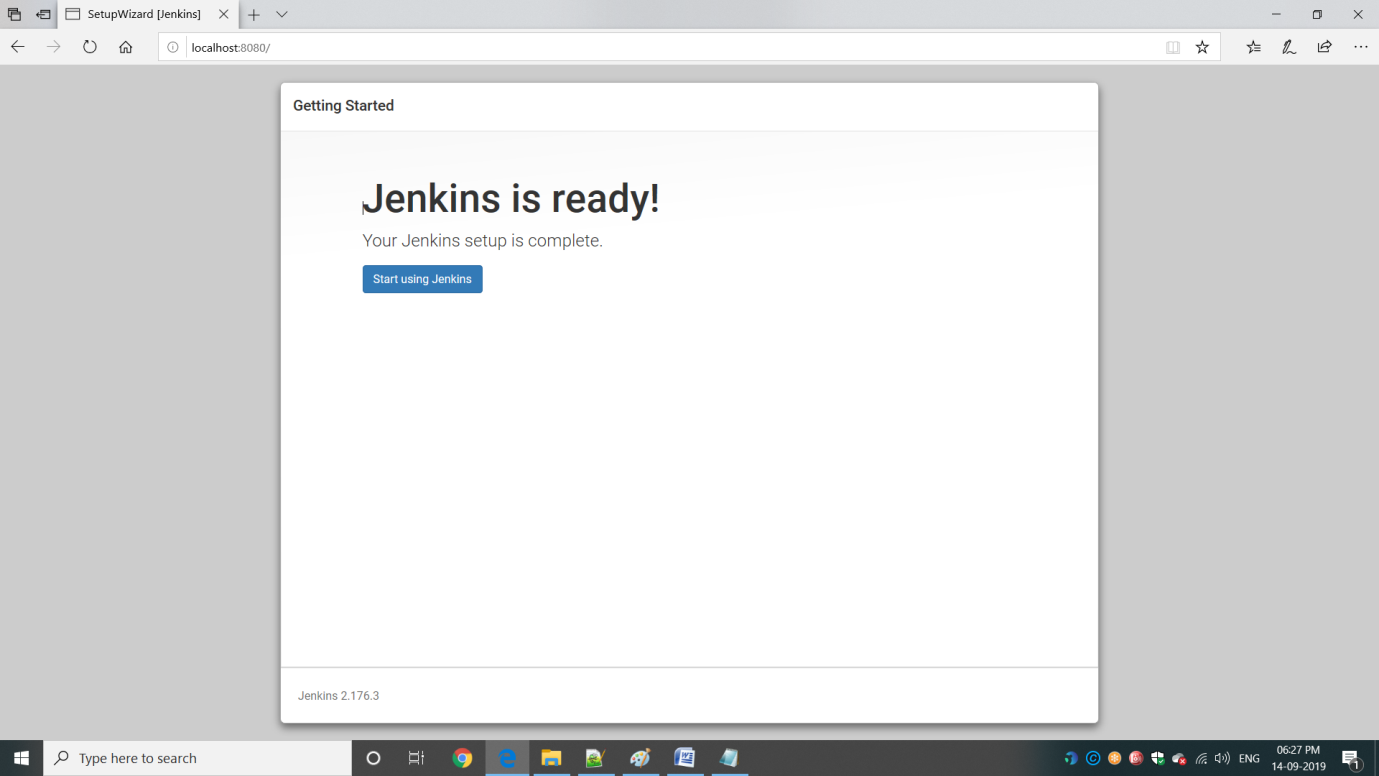
**Step6:** Create Admin username and password to login into Jenkins. Keep the user credentials safe for future use.



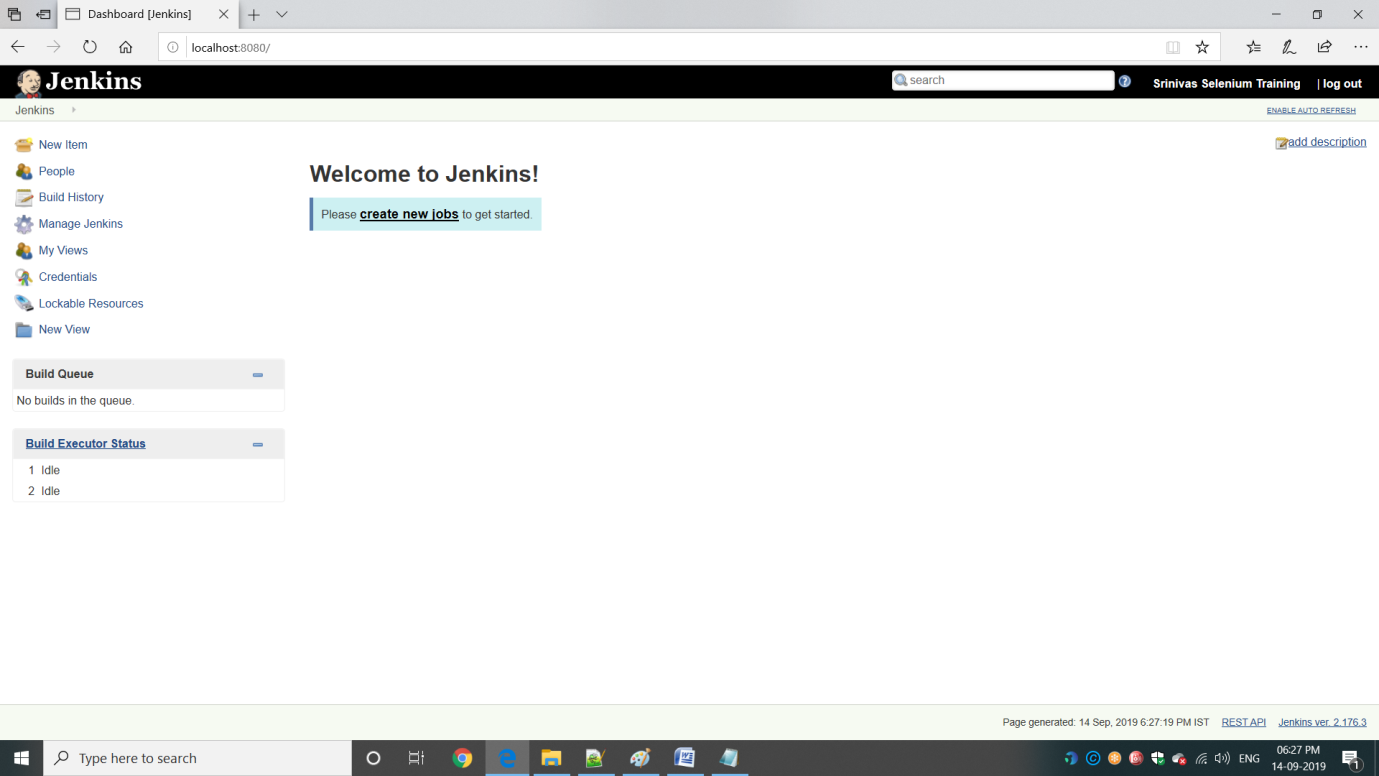
**Step7:** Click on Save and continue on Getting started page



**Step8:** Once Jenkins is displaying below page, click on “Start using Jenkins” button

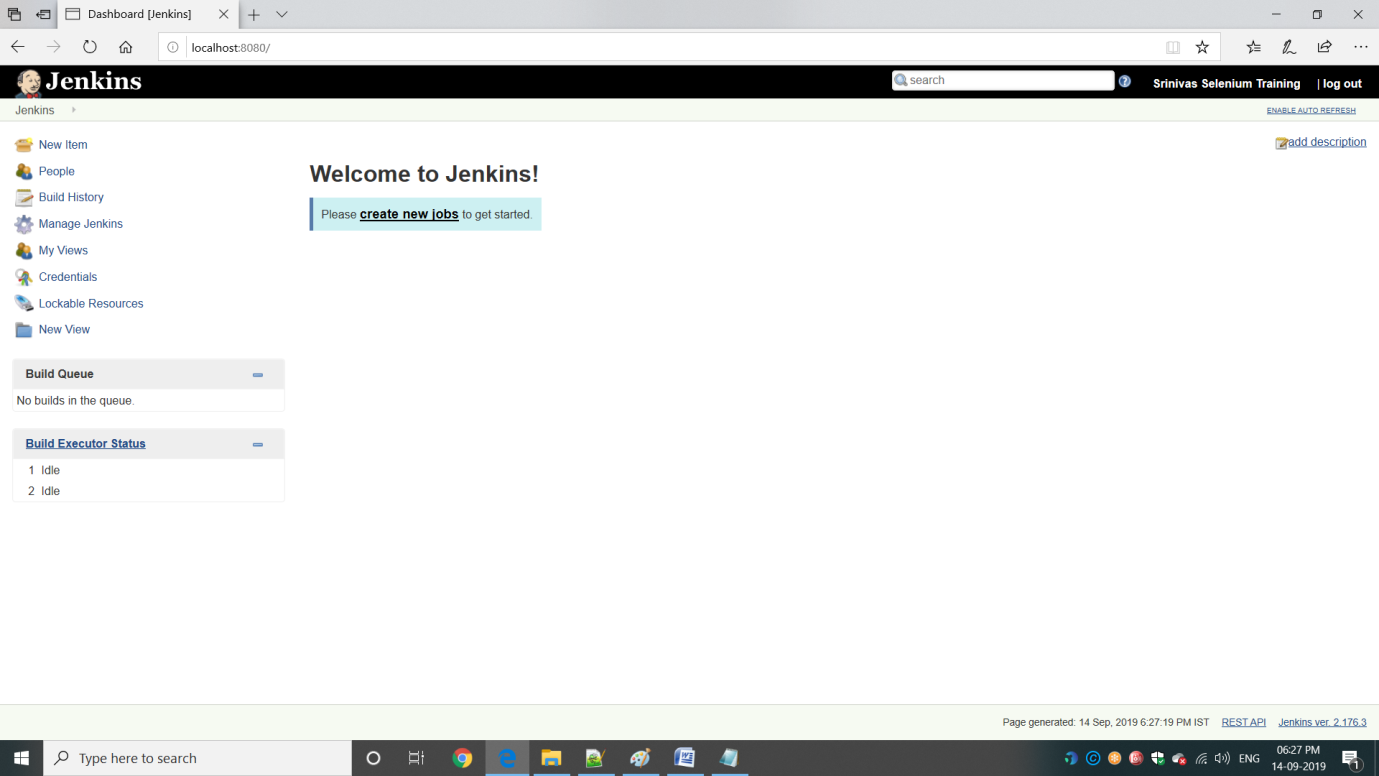


**Step9:** Jenkins home page displayed like below. You are set to use Jenkins

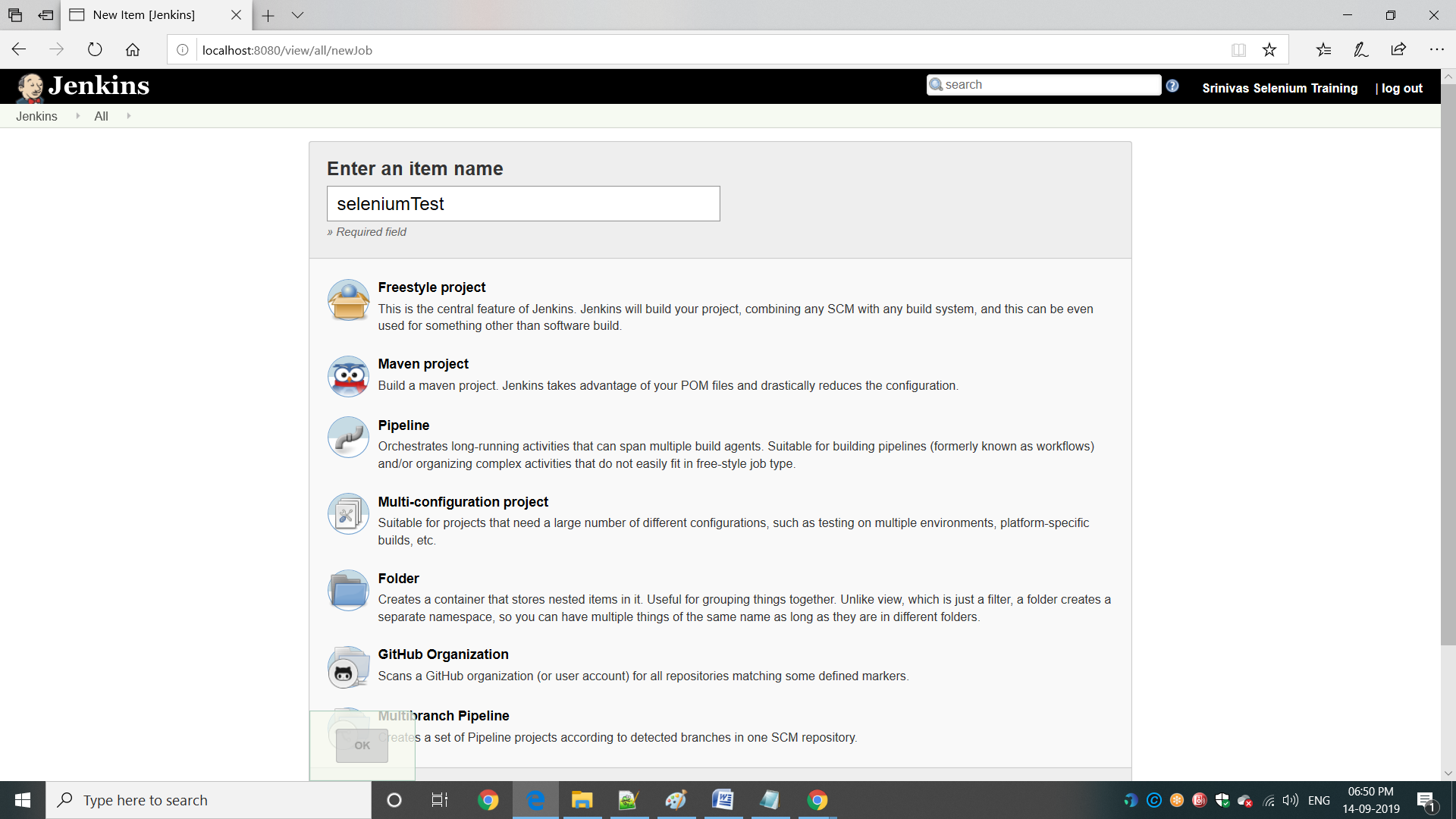


Adding MavenSelenium Project

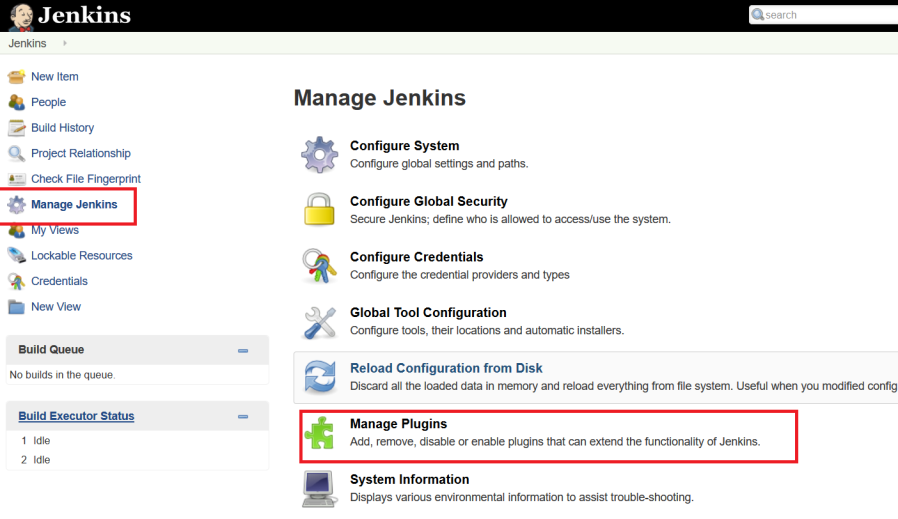
**Step1:** Click on New Item or Create new jobs link



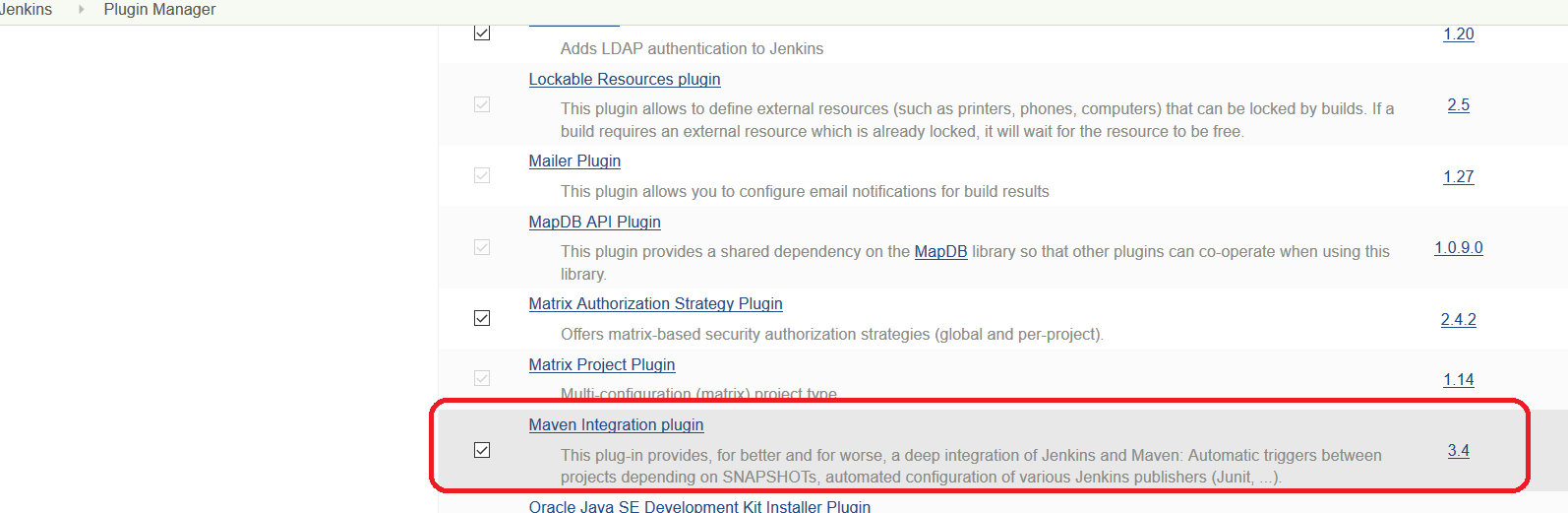
**Step2:** Give Itemname and select “Maven Porject” and click on “OK” button at the bottom



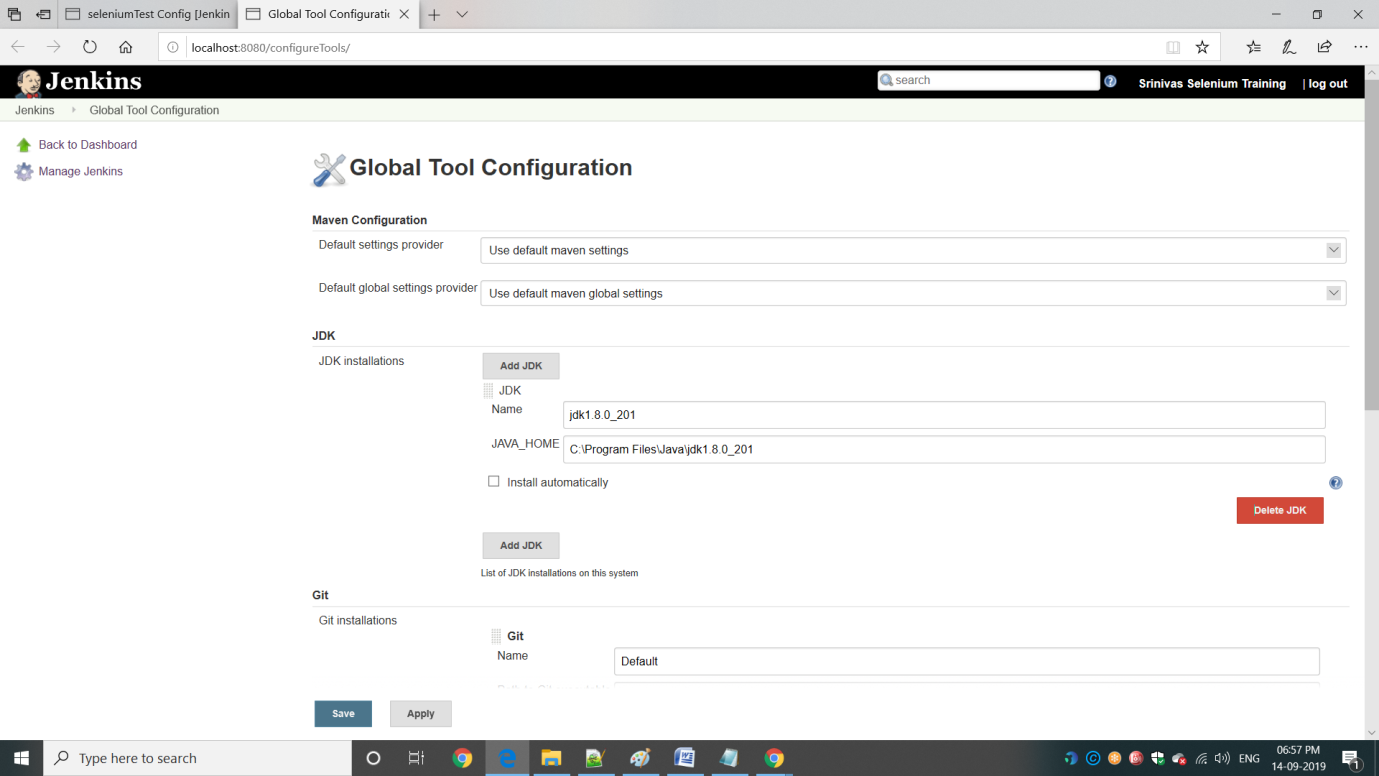
**Step3:** If you are unable to see “Maven Project” at step2, click on Manage Jenkins -> Manage Plugins



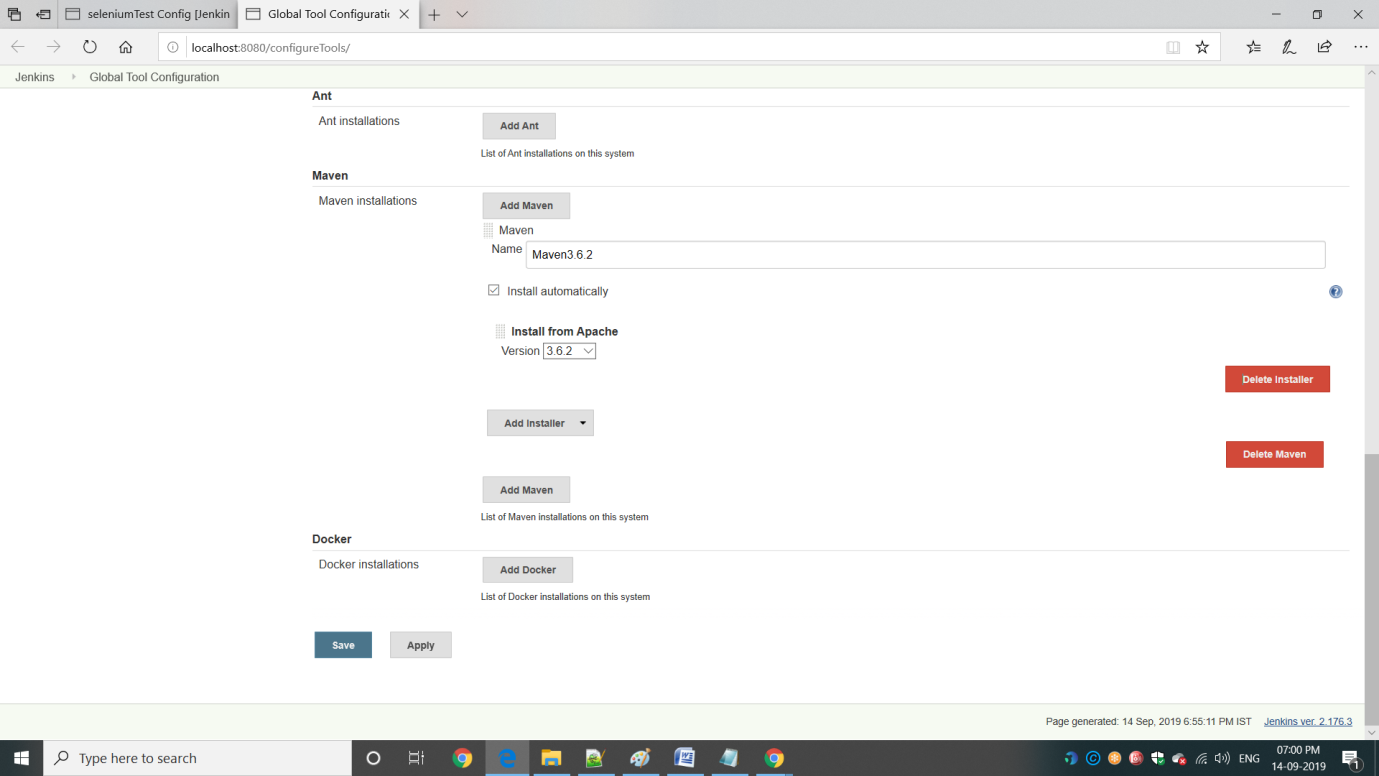
**Step4:** Manage Plug-ins page , search “Maven” and select Maven Integration Plug-in and install it and restart the services and repeat Step1 and Step2



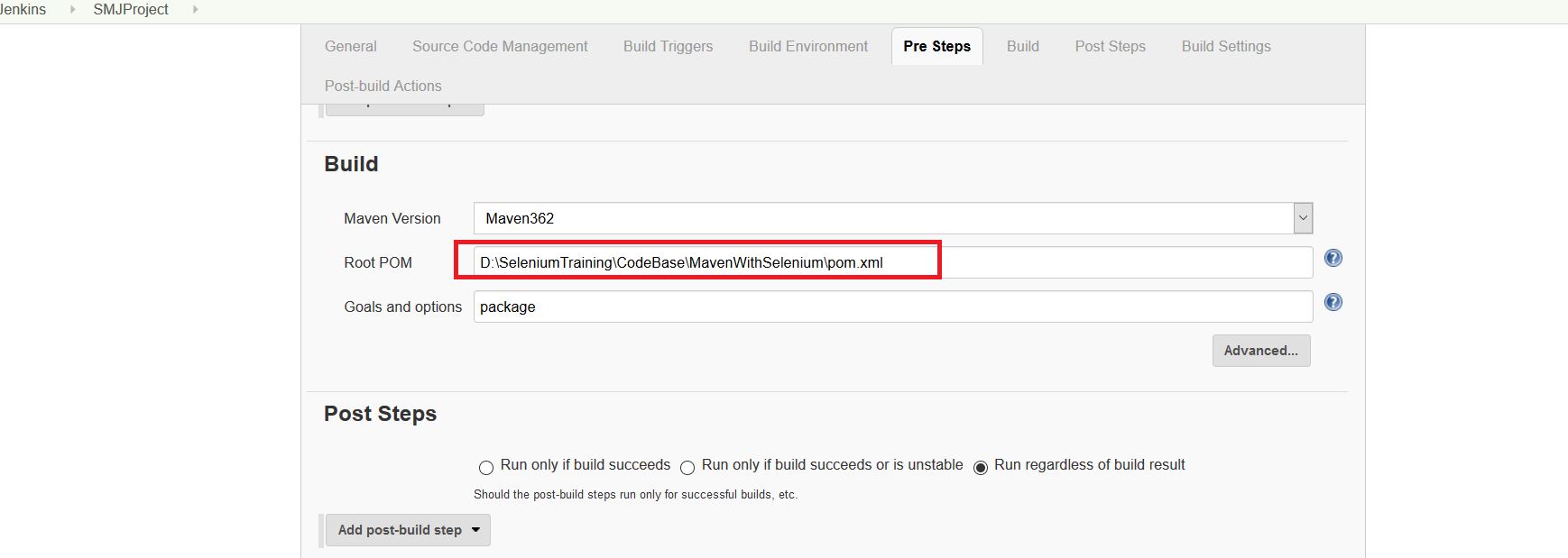
**Step5:** Set the Maven configuration details by adding JAVA\_HOME details and select Install automatically checkbox



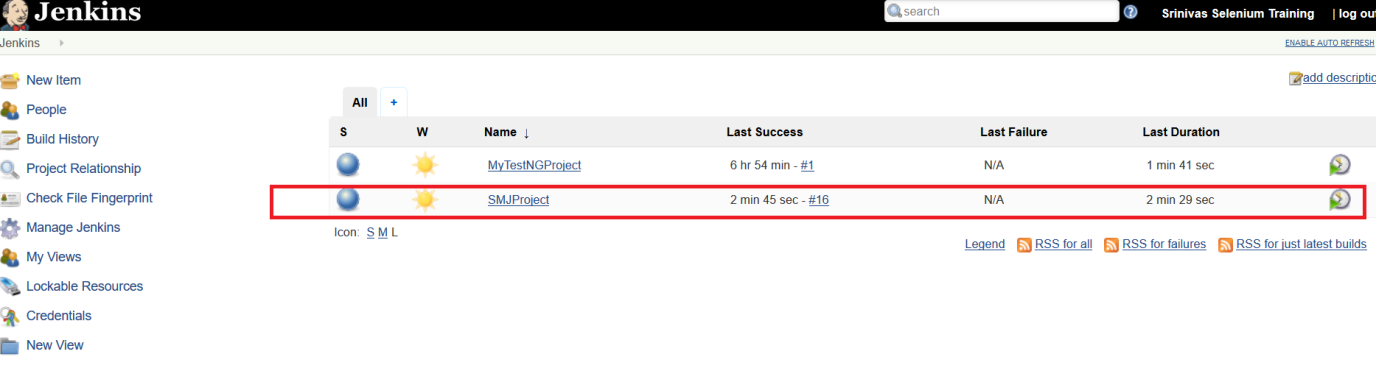
**Step6:** Scroll down Maven details for Selecting Maven Version and selecting install automatically check box from Apache



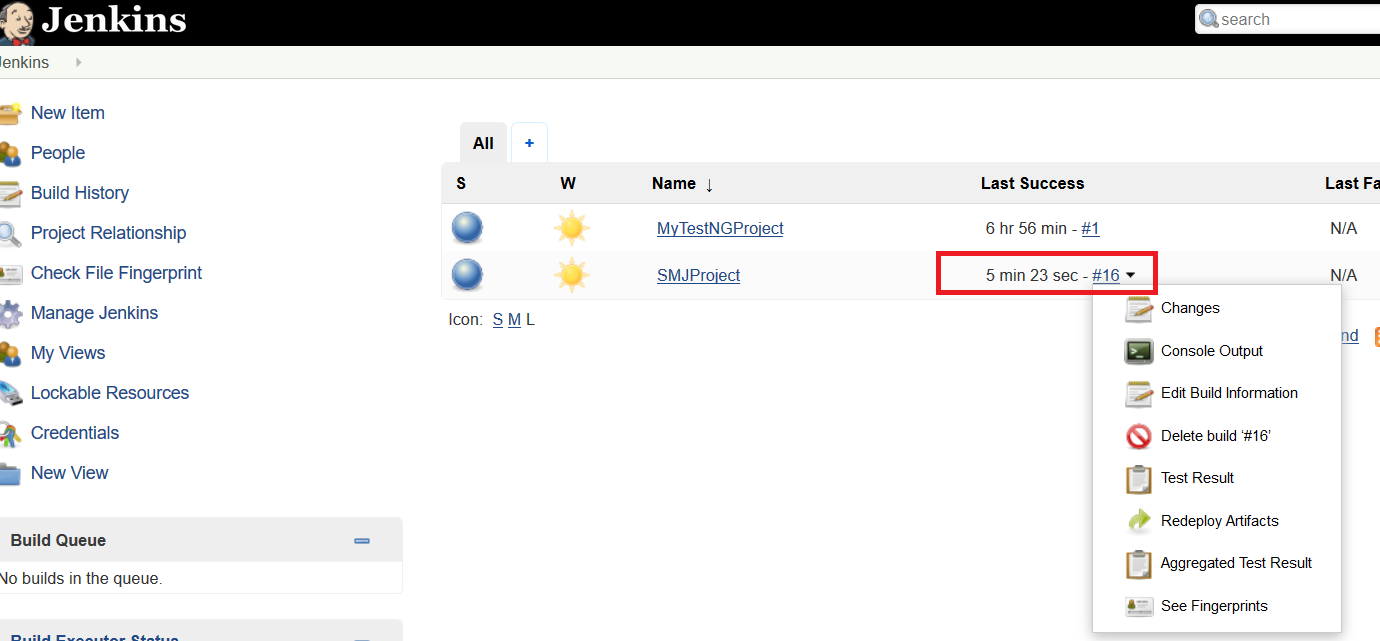
**Step7:** Scroll down Maven details for Selecting Build details and Goals and options like below and Click on Save button



**Step8:** Goto Dashboard and you can see the Project Created like below. Click on “Build Now” icon

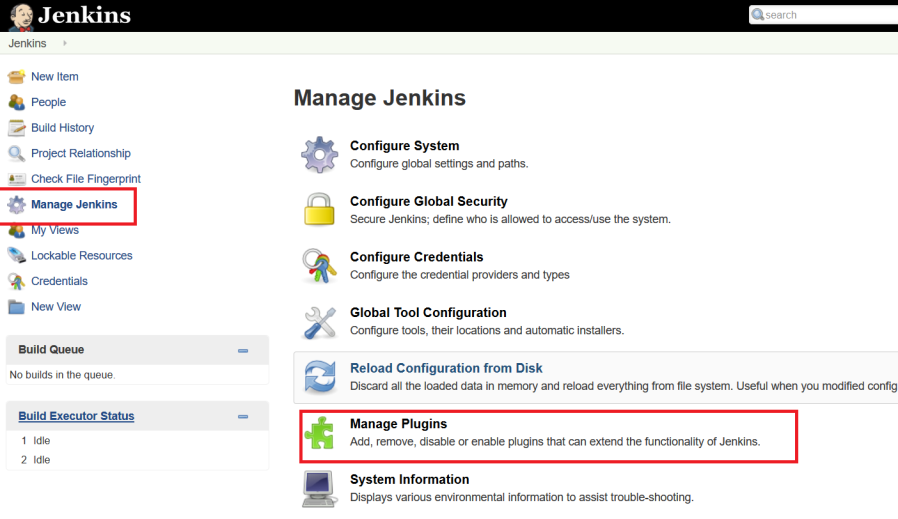


**Step9:** You can find reults generated once run complete.

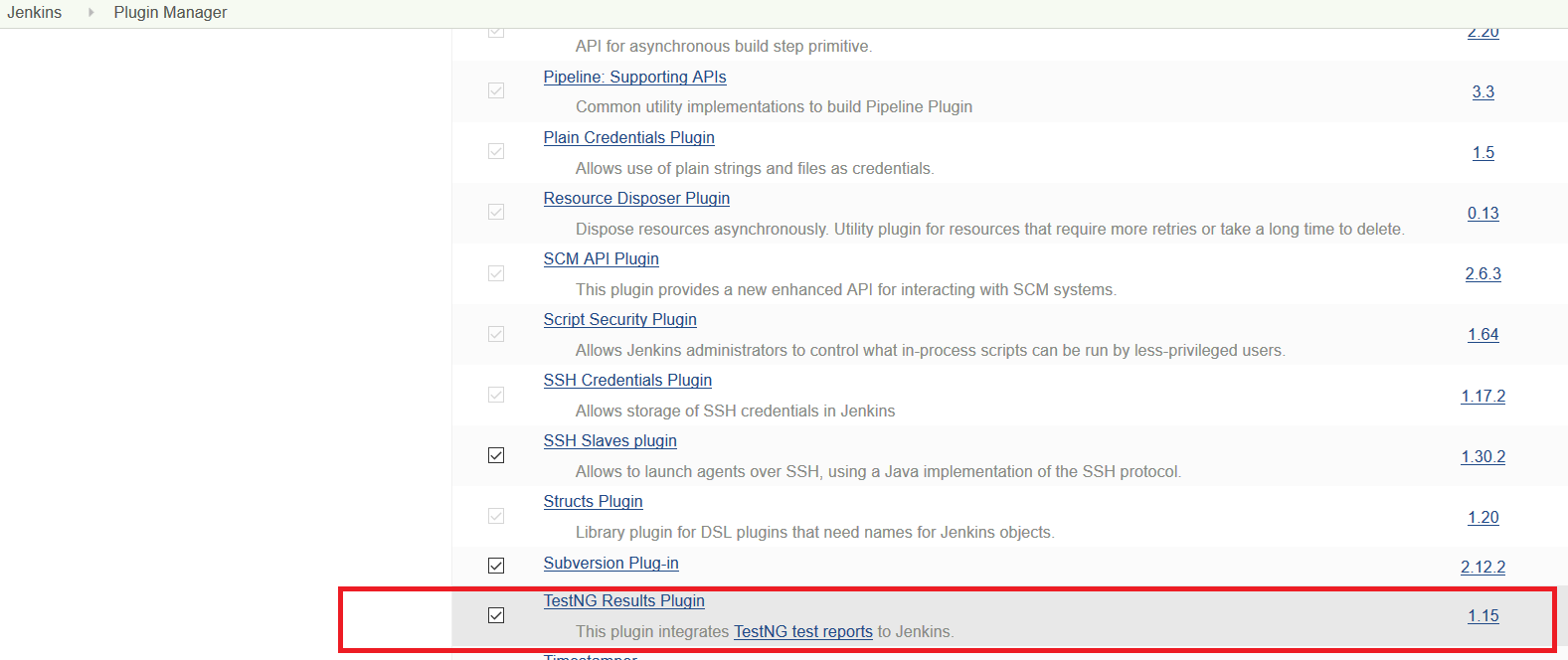


Adding TestNG Selenium Project

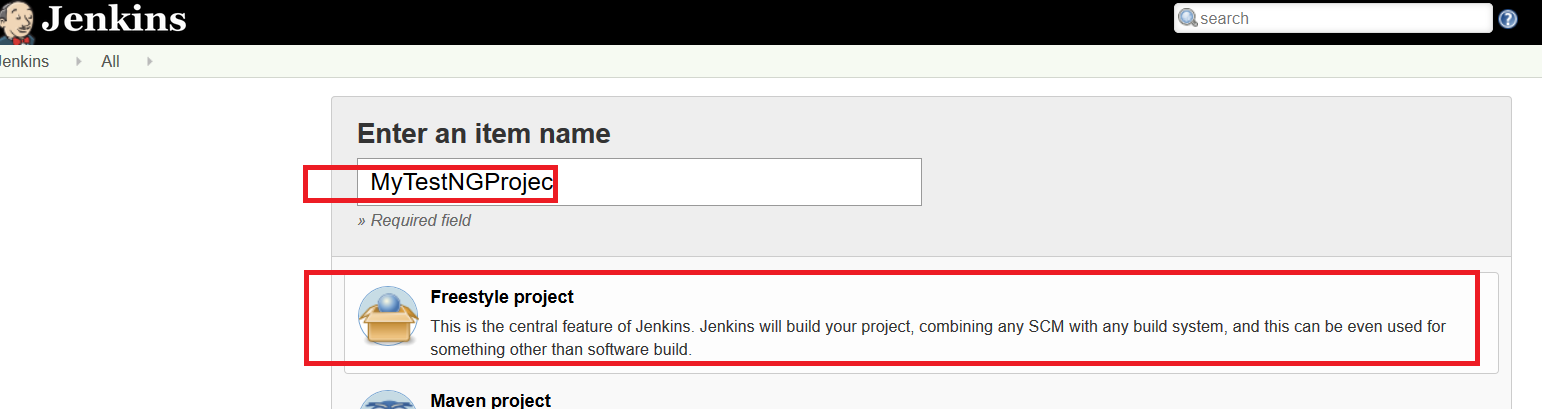
**Step1:** Click on Manage Jenkins-> Manage Plugins



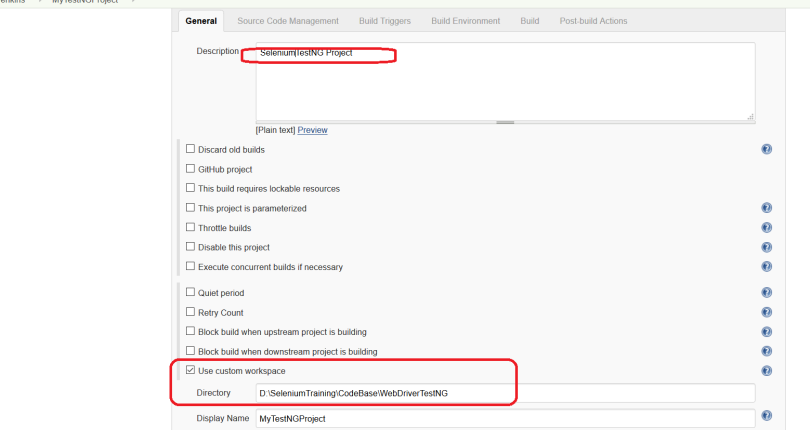
**Step2:** Select TestNG Results PlugIn and install it.



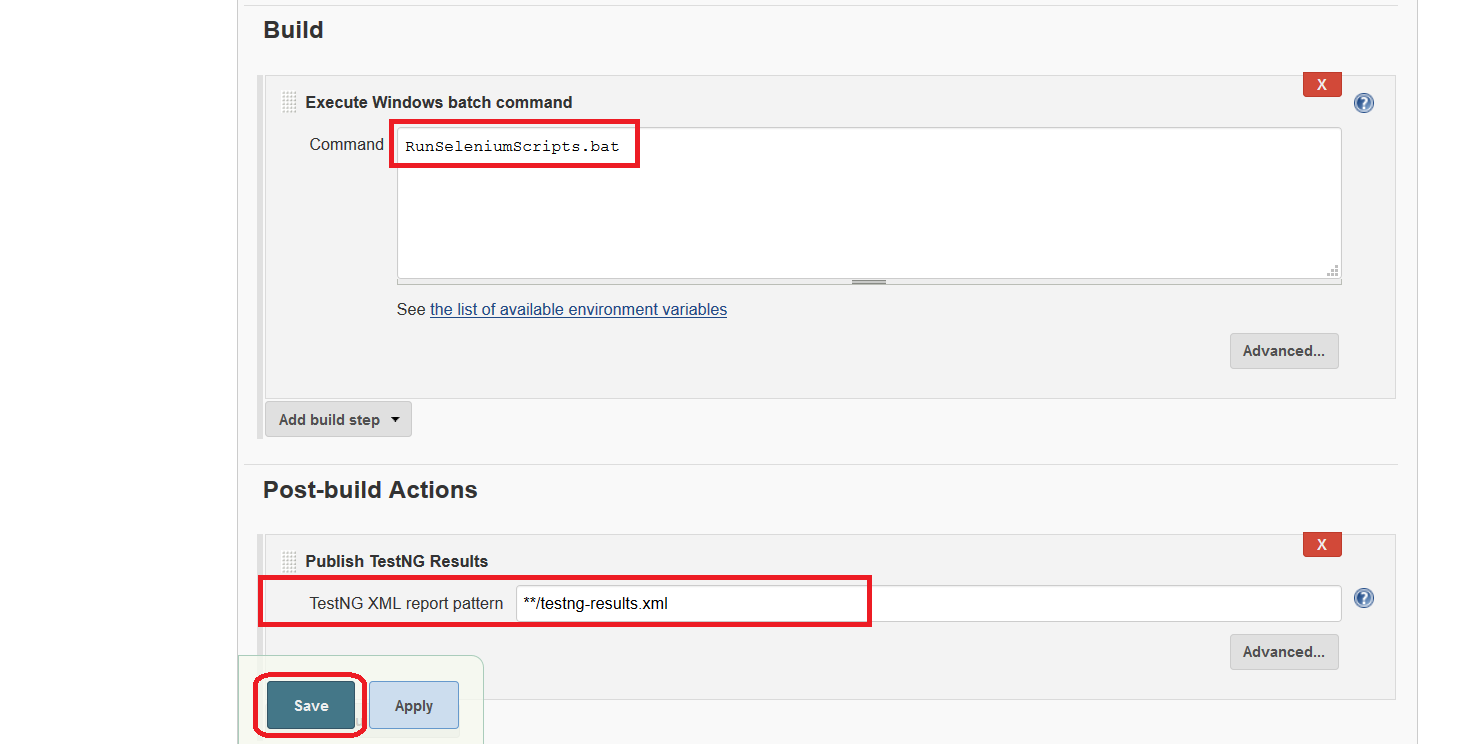
**Step3:** Now goto Dashboard page and click on on New Item. In Item page, Enter Item name and Project type as below



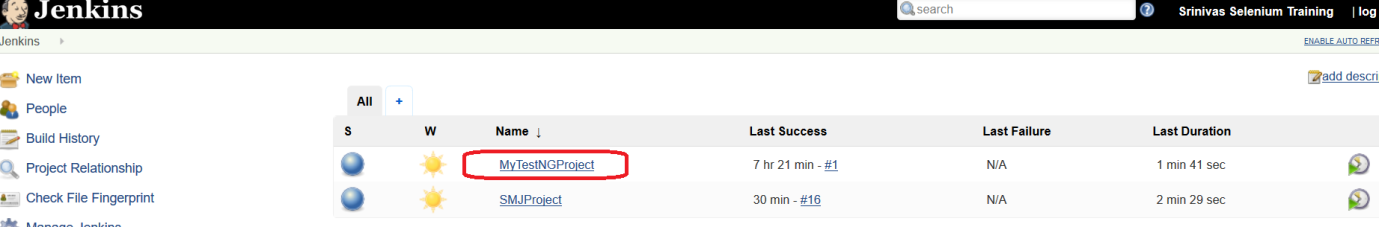
**Step4:** Enter Project Description and click on “Advanced” button, Select Use Custom workspace checkbox and provide Drectory details of Selenium TestNG project.



**Step4:** In Build option, select Windows batch command and provide the bat file name of the project. In Post-build Actions select Publish TestNG results and Click on Save button.



**Step5:** Goto Dshaboards page, you can find your project is created. Click Build now to run the build



**Step5:** You can find results of TestNG results like below.

