**What is Automation:**

Testing the given application using tool or a program.

Automation testing is a process of automating the manual process to test the application/system under test, using a separate testing tool or a program.

**How to automate** : We automate using a tool or program, which lets you create test scripts which can be executed repeatedly and doesn’t require any manual intervention and we add assertions to verify the results and generate the reports.

Automation can be done for the following using different tools :

UI / Web Application Automation : using Selenium(IDE, IDE(tool),RC, Grid, Web Driver), Casper.js,Protractor-Angular.JS apps, Nightwatch.js ,Silk Test, QTP

Web Services Automation: Rest API’s Automation – HttpClient, RestAssured, Chakram, SoapUI tool

Mobile Automation: Appium, Perfecto, Calabash

**Advantages of Automation:**

* Saves time and resources
* Regression testing – testing the new feature and testing the existing feature: when ever new feature is added we also need to test the old features also which is not affecting the original one..so we can repeat running the running the testcases…
* Data Driven Testing- test cases will be repeated with different sets of data- valid/invalid, boundary values
* Cross Browser Testing- Mozilla, chrome, IE, safari etc
* Cross Platform- mac, windows, linux—executing the same testcases on multiple platforms is easy using tools compared to manually doing.
* Manual Testing- Error Prone sometimes—human can make mistakes sometimes but tool does the tasks automatically..

When to Automate and When not to Automate?

Tight DeadLines – we cannot automate under tight dead lines

When UI changes /going to change in future then do not automate that particular module.

When application is not going to change at all and its already in maintenance- meaning no enhancements or no new user stories – then also no need to automate

import java.util.ArrayList;  
import java.util.Collections;  
import java.util.List;  
  
public class SortingwithoutDuplicates {  
  
 public static List<Integer> checkForduplicates(List<Integer> res, int num){  
  
  for(int i=0; i<res.size(); i++){  
   System.out.println("size of res : " +res.size());  
   if(res.get(i).equals(num)){  
    System.out.println("In for Loop");  
    break;  
   }else{  
    System.out.println("In Else Loop");  
    res.add(num);  
    System.out.println(res);  
   }  
  }  
  
  return res;  
  
 }  
  
  
 public void mergeArrays(){  
  
  int[] arr1 = {2,5,1,10};  
  int[] arr2 = {4,3,5,10};  
  
  List<Integer> result = new ArrayList<Integer>();  
  for(int i=0;i<arr1.length;i++){  
   result = checkForduplicates(result, arr1[i]);  
  }  
  
  for(int i=0;i<arr2.length;i++){  
   result = checkForduplicates(result, arr2[i]);  
  }  
  //2,5, 1, 10, 4, 3  -result  
  
  System.out.println(result);  
  Collections.sort(result);  
  System.out.println("Final Sorted Array : " + result.toString());  
  
 }  
  
 public st

**Selenium** developed in 2014 –by Jason Huggins from thoughtworks

Selenium: is a set of Tools: has four different parts :

* Selenium IDE
* RC
* Selenium Grid
* Selenium 2

**Selenium IDE** – It is a record and playback tool for UI automation, it is plugin in firefox and it is only available with firefox browser.

It has mainly three parts:

* **Command**- what action we are doing is the command (action which we do in browser will be recorded)..Ex. open, click, type, clickAndWait etc.
* **Target**- html element (textbox, link, plaintext, dropdown, checkbox, radio,)which we are using to automate
* **Value**- text you are giving as input in text box Ex. laptop etc.

We can record or we can also type our own commands, target, value and add assertions

We can also save the script in a file. we can add assertions for testing.

We can modify the commands and insert the commands too.

We just play and stop the recordings. We have options to control the speed of recording.we can also set some debug points which will pause the execution for some time and later we can start the execution.

We can set the Start point, so that we can decide our own starting point for exection.

Log tab can be used to see the complete Log and in Reference tab it explaind about the element on which actions are performed.

It supports different languages..we can export the testcases directly with selection of lang in which we want the script to be generated. Auto generated (internally generated) code can be saved.

**Diffrence between Assert and Verify?**

Verify is used to verification of the testcase, even it the test fails it doesn’t stops the execution where as when assertion fails it pause the execution and the statements after the assert wont be executed.

**Drawbacks:**

There is no continuous integration or build can be done using IDE tool.

We can not do multiple actions at a time

We can not do data driven using multiple sets of data

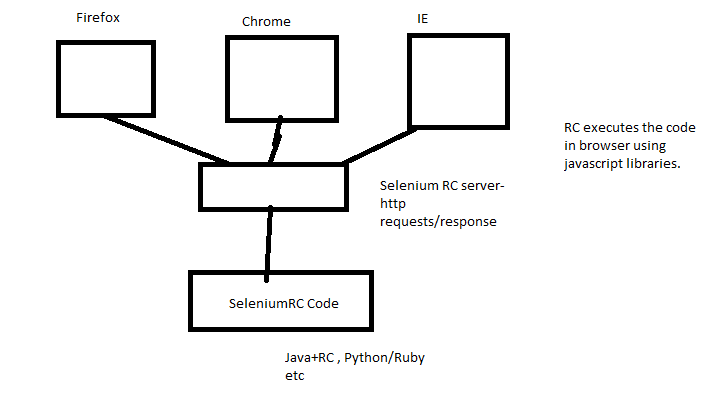
Cross platform testing is not possible because ide plugin is only available in firefox browser

It used for basic testing for practice kind of.

Logical combinations which can be done in programming is not possible

**Selenium 1: which mainly refers to RC :**  to overcome the drawbacks of the selenium IDE, RC was developed. Nowadays it is outdated only using as RC server in someplaces. RC also supports multiple languages (Java, Ruby, Python etc.)

RC was developed as a server which executes the script on respective browser.



Selenium RC server acts as mediator between Selenium RC Code and the browser. RC executes the code in diff browsers using java script libraries, and it converts the RC Code into browser understandable language code hence it is **Slow** compared to Wed Driver.

To overcome this problem Selenium 2(WebDriver) is developed.

**Selenium Web Driver:** The primary new feature in Selenium 2.0 is the integration of the WebDriver API(Library Based API).

WebDriver – its an interface with some common methods and it has Implementation Classes- Given by specific browsers.eg- FirefoxDriver, ChromeDriver,IEDriver,SafariDriver,OperaDriver

## How Does WebDriver ‘Drive’ the Browser Compared to Selenium-RC?

Selenium-WebDriver makes direct calls to the browser using each browser’s native support for automation. How these direct calls are made, and the features they support depends on the browser you are using. Information on each ‘browser driver’ is provided later in this chapter.

For those familiar with Selenium-RC, this is quite different from what you are used to. Selenium-RC worked the same way for each supported browser. It ‘injected’ javascript functions into the browser when the browser was loaded and then used its javascript to drive the AUT within the browser. WebDriver does not use this technique. Again, it drives the browser directly using the browser’s built in support for automation.

## What is 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

Generally speaking, there’s 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.

Selenium-Grid is used to speed up the execution of a test pass by using multiple machines to run tests in parallel. For example, i f you have a suite of 100 tests, but you set up Selenium-Grid to support 4 different machines.

Selenium-Grid is also used to support running tests against multiple runtime environments, specifically, against different browsers at the same time. For example, a ‘grid’ of virtual machines can be setup with each supporting a different browser that the application to be tested must support. So, machine 1 has Internet Explorer 8, machine 2, Internet Explorer 9, machine 3 the latest Chrome, and machine 4 the latest Firefox. When the test suite is run, Selenium-Grid receives each test-browser combination and assigns each test to run against its required browser.

A grid consists of a single hub, and one or more nodes. Both are started using the selenium-server.jar executable.

A grid consists of a single hub, and one or more nodes. Both are started using the selenium-server.jar executable.

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.

Automation mainly consists of four main steps:

* Open browser
* Find the element
* Perform action on element
* Get the result- actual- assert it with expected

Selenium2-WebDriver:

Download web driver and extract the folder and you can copy the jars into local project in Eclipse projectand set the buildpath..or you can add jars from exteral file also.

Webdriver is called as Web Driver API, initially it was by thoughtworks, later it was owned by google and made it public as openqa. There are set of classes which are used for scripts.

There are different driver implementations for that we need to download the executable files to use particular drivers accordingly. And we need to specify the complete path of the executable files by setting the property.

**WebDriver- interface methods**

1. get
2. findElement(By class object)

we need to find the elements for UI testing. Using **“By”** - class which provides static methods to locate element.Every html element has attributes: Ex.div, a, input, select, header, strong- html elements id, class, name, value etc....Using these elements, attributes we create locators for the specific element.

By Class takes 8 diff methods which take string input and return By object:

id - id is most efficient and fastest way as it internally uses JS doc.getElementByid method directly.

name

class

tag

linkText

partialLinkText

xpath- Try to locate the element by traversing across the html document

CSS and Xpath Selectors :

**XPATH: tries**  to locate the element by traversing across the html document.Xpath is of two types

1. Absolute Xpath

2. Relative Xpath

**Absoulte xpath**: - starts from the root and traverses for the given element. It starts with single slash.normally it is lengthy.

Ex.html/body/div[1]/div/div/div/header/div/div[2]/div/div/div/div/div/div[5]/form/div/div[2]/div/label/input

**Relative xpath**: - It starts with // and it directly points to specific element instead of pointing the element from the root.

**Basic xpath syntax**: - //elementNameOr\*[@attrName='valueOfAttribute']

Ex. .//\*[@id='global-search-input']

.//\*[@id='c5H2m-location']

.//\*[@id='c0DpC-location']

**Xpath using Multiple Attributes**, which does "and" condition-Ex .//input[@name='email'][@id='email']

When Id/ any other attribute is dynamically changing we use certain methods in xpath like contains, starts-with, text(),following/,preceding/ methods.

contains(@attrName,'value') --- Ex. .//input[contains(@id,'location')]

starts-with : Ex .//\*[starts-with(@class,'locationField')]

text() --- Ex .//div[contains(text(),'My Account')]

preceding – Ex. .//\*[@id='u\_0\_3']/preceding::div[2]

Following – Ex. .//\*[@id='u\_0\_3']/following::div[2]

css selectors:

tagname

#idValue

.classValue

Basic css - //elementName(optional)[attrName='valueOfAttribute']

multiple attributes- input[name='email'][class='inputtext']

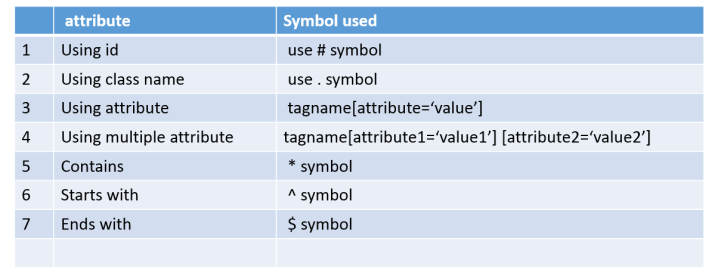
input[id\*='location']- this gives all elements which have location in their id

div[class^='location'] - startswith

div[class$='location'] – endswith

div.airportBlock input[id$='-destination’]

## Symbol used while writing CSS selector in Selenium Webdriver



1. **What is the Difference between Css and XPATH ?**

**When ever the elements are dynamic (constatly changing), in such scenarios we use Css Selector or xpath. Both are equally preferable based on the application. In some scenarios CSS is preferable over the Xpath because, CSS is Faster than Xpath and compatible with most of the browsers unlike XPATH.**

**But in Scenarios to deal with traversal with preceeding nodes Xpath is preferred because we can not traverse back using CSS.**

**Xpth--- XML based, CSS---- HTML based**

In terms of performance, CSS perform well as compared to [XPATH](http://learn-automation.com/how-to-write-dynamic-xpath-in-selenium/)and CSS will not change based on browsers, that is it will behave same in all browsers but xpath will behave differently in IE browser.

Xpath---- can traverse back and forth, cant locate dynamically changing web element, doesnt support all browser

CSS---- cleaner, faster, supports multiple browser,can identify dynamically changing webelement,Easy to construct,

1. **Which the the most preferred attribute?** Id is most preferred and it is always faster.
2. **Diff between navigate and get?** To() method is similar to get(). Get is to open the website and navigate().To also does the same with thing but it has extra methods like back ,forward,refresh etc.
3. **Diff between close and quit?** “Quit” closes all the opened windows and “close” closes the current window. driver.cose() : it just closes the current window and driver.quit() : closes all the opened windows of the browser.
4. **How do you handle the windows in webdriver and how do you switch between parent and child windows?** In handle the multiple windows and to switch the curser from parent window to child window we use WindowHandles and switchTo() methods .

**Driver.getWindowHandle() : -- gives the curr window name**

**Driver.getWindowHandles(): -- gives the Set of all windows that are opened.**

Set<String>windows = driver.getWindowHandles(); --gives the set of all the windows that are open : we are taking them into Set because windows are unique it wont allow duplicates

To switch the control between the windows : **switchTo** method is used. **driver.switchTo.window(nameof thewindow);**

1. **Can you write a code to get the all the options of a dropdown and get their text or display the text?** This can be done using **Select** class element method - getText().

Select select = new Select(element);

List<WebElements> options = select.getOptions();

For(WebElement element:options){

System.out.println(“text : “ + element.getText());

1. **Explain about isSelected,isdisplayes,isenabled(),isDisabled() ?**

**element.isSelected()** –can be only used with select dropdowns, checkboxes, and radio buttons

**isEnabled(), isDisplayed(), isDisabled():** can be applied to any element.

1. **Diff between thread.sleep and waitExplicitly ?** WebDriver waits are intellectual becoz they only waits if the element is not found due to delay in loading the page or delay in aoding specific element, Where as thread.sleep is the java way to pause the execution of method for given time.
2. **Explain waits? There are two types of waits**
   1. **Implicit Waits**
   2. **Explicit waits**

**FluentWait()**:

Webdriver allows programmer to wait for the element/elements to be located , before throwing the Exception like “No such Element/ Element not found” exception.

Two kinds of wait :

1.**implicitly** : Waits for entire time the browser is open. It internally waits for all the elements, Whenever there is any delay in finding element then only it waits not like Thread.sleep().It waits for the element which you are trying to access. It wait all the time before driver.close() executes.

It checks initially

Syntax. **driver.manage().timeOuts().implicitlyWaits()**

2.**Explicitly** : waits for specific condition or specific elements for given time or it throws the exception. It checks in between the wait time for presence of the wait time.by default wait time is 500millisec. If it finds the element after

Syntax : for explicit wait have a class called “WebDriverWait” and it has a default constructor which takes driver object and time…and time it takes in seconds by default.

After creating objects we use methods called “until” and it takes overloaded parameters : one is function and the other is class “ExpectedConditions” which again has diff methods like …visisbilityOfElementLocator(locator).

**WebDriverWait wait = new WebDriverWait(webdriver Object,timeOutinSeconds);**

Explicit wait until the timeout time, still if there is any mistake in locator also…

**FluentWait** :

Function is a Interface which is also like a method but it has specific definition and anonymous class declaration expecting two parameters one is input parameter and the other is return type parameter. we need to add unimplemented method in it.

**We should not use explicitWait and implicitWait together.**

**Predicate** is also same as function ,but it won’t allow our own return type….it’s only returns “Boolean”

1. **How you write ImplicitWait, when do you use ImplicitlyWait explain?**

driver.manage().timeOuts().implicitlyWait(20,TimeUnit.Seconds)

1. **Explain about explictWait, when you use it and how you write it?**

For explicitwait we need to create the object of WebDriverWait class,which takes two parameters driver reference and time in seconds, and it wait for the particular element until expected condition method satisfies in given time. ExpectedConditions has diffmethods.

**Syntax:**

WebDriverWait wait = new WebDriverWait(driver,Time.Unit.Seconds);

Wait.until(ExpectedConditions.visibilityOfElementLocated(By.id(“eww”)));

If we want to write the logic inside the function, so that complete code will be executed from wait.until method parameter..then we give the function as the parameter.

Function

Predicate

FluientWait

1. **Diffrence between Find Element and FindElements ?** FindElement : it returns Single WebElement and returns NoSuchElementException when element is not present in html. FindElement can also be used to check the element presence. FindElements : it returns the List of WebElements. Returns empty list.

when we are not able to find the element then the list takes as emply and it returns the size as ZERO, where as findElement if it won’t find the element then it throws the exception.

1. **When are submit and Click used ?** generally used for html form submission buttons, button inside a html form. This can’t be used for the button outside from tag.

Whenever <input type=’submit’> and element is inside form we use submit. Submit is only for buttons not for inks.

**Click** works in all the situations – inside or outside forms. but for forms submit is preferable.

1. **Explain Actions?** ACTIONS: Actions is a class in webdriver is to deal with mouse actions, to automate mouse over ,click ,double click ,key press ,key Release ,drag and drop using an Interface Builder Patterns. It builds composite actions containing all the actions to be performed.

**1.**Mouse Hover action

**2.** Drag And Drop

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

**Example:** WebElement **From** = driver.findElement(By.xpath(" [1]/td[4]/span"));

WebElement **To** = driver.findElement(By.xpath("hhhh"));

 Actions builder = new Actions(driver);

Action **dragAndDrop** = builder.clickAndHold(From).moveToElement(To)

  .release(To).build();

 dragAndDrop.perform();

**Example**: WebElement element = driver.findElement(By.linkText("Product Category"));

Actions action = new Actions(driver);

 action.moveToElement(element).moveToElement(driver.findElement(By.linkText("iPads"))).click().build().perform();

1. **How do you handle popup windows and browser alerts using webdriver ?** Selenium provides us with an interface called ***Alert***. It is present in the ***org.openqa.selenium.Alert***package. Alert interface gives us following methods to deal with the alert:

* ***accept()***To accept the alert
* ***dismiss()***To dismiss the
* ***getText()***To get the text of the alert
* ***sendKeys()***To write some text to the alert

***Syntax:*** Alert simpleAlert = driver.switchTo().alert();

simpleAlert.accept();

simpleAlert.dismiss();

promptAlert .sendKeys("Accepting the alert");

promptAlert .accept();

1. **How do you handle Frames?** Using switchTo method we move from onr frame to other..frames can be identified using overloaded methods id,name or by index. If there are multiple frames in page then we can switch to frames using index value.

To Switch between iFrames we have to use the driver’s ***switchTo().frame*** command. We can use the *switchTo().frame()* in three ways:

* ***switchTo.frame(int frameNumber)*:**Pass the frame index and driver will switch to that frame.
* ***switchTo.frame(string frameNameOrId)*:**Pass the frame element Name or ID and driver will switch to that frame.
* ***switchTo.frame(WebElement frameElement)*:**Pass the frame web element and driver will switch to that frame.

Ex. driver.switchTo().frame(0);

driver.switchTo().frame("iframe1");

driver.switchTo().frame("IF1");

WebElement iframeElement = driver.findElement(By.id("IF1"));

//now use the switch command

driver.switchTo().frame(iframeElement);

//Switch back to the main window

driver.switchTo().defaultContent();

1. **DisiredCapabilities :**
2. **ScreenShot**
3. **Listeners**
4. **How tohandle window alerts ?** window based alerts can be handled using AutoIt ,RobotClass or SilkTest.

### What is AutoIt?

[AutoIt V3](https://www.autoitscript.com/site/autoit/) is a freeware tool which is used for automating anything in Windows environment. AutoIt script is written in BASIC language. It can simulate any combination of keystrokes, mouse movement and window/control manipulation.

While [doing automation through Selenium](http://www.softwaretestinghelp.com/selenium-tutorial-1/) or through any other tool for that matter, we all encounter a common problem, windows pop-ups. As Selenium is confined to automating browsers, desktop window is out of scope. Web applications sometimes need to interact with the desktops to perform things like file downloads and uploads. There are tools available for automating these sorts of workflow such as AutoIt, Robot Framework, Silk Test etc.

We can upload or download the files or images by transferring our [control from Selenium WebDriver](http://www.softwaretestinghelp.com/selenium-webdriver-selenium-tutorial-8/) to AutoIt. We need to explicitly call the AutoIt script from our program.

### Finding element through element Identifier and writing script on AutoIT editor.

Under this, we will see how to find element on file uploader window through AutoIT Element Identifier (Element identifier is a tool like selenium IDE, identifier find the element of window GUI or non HTML popups and provide the attribute of element like**title**, **class**, **instance** ) and how to write script on AutoIT editor using 3 methods.

**Step 1)**: Now open element Identifier- Go to 'C:\Program Files (x86)\AutoIt3' and click on 'Au3Info.exe' file, the element identifier window opens as shown in below screen.

**Step 2)**: Now open file uploader window by clicking on 'Choose File' which is windows activity.

**Step 3)**: Drag the finder tool on the " File Name" box element of file uploader window to find the basic attributes info as shown in the below screen with the arrow.

**Syntaxes : writing style:**

**WebDriver driver; --- initialization of driver**

**System.SetProperty(“webdriver.chrome.driver”,”path”);**

**driver.get(“url”);**

**NAVIGATE() method :**

**driver.navigate().to(“url”);**

**driver.navigate().forward();**

**driver.navigate().refresh();**

**driver.navigate().back();**

**WINDOW HANDLING :**

In handle the multiple windows and to switch the curser from parent window to child window we use WindowHandles and switchTo() methods .

**Driver.getWindowHandle() : -- gives the curr window name**

**Driver.getWindowHandles(): -- gives the Set of all windows that are opened.**

To switch the control between the windows : **switchTo** method is used. **driver.switchTo.window(nameof thewindow);**

**Code:**

// using window handle it gives the curr window name

String currWindow = driver.getWindowHandle();

System.***out***.println("curr window name : "+ currWindow);

// gives the set of all the windows that are open : we are taking them into Set because windows are unique it wont allow duplicates

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

// we dont know the ordering for the particular windiw hence we need to iterate and check for the curr child window

**for**(String win:windows){

System.***out***.println("Child Window : " + win);

**if**(win!=currWindow){

driver.switchTo().window(win);

}

}

**WebElement Methods :**

1. **List<WebElement> elements = driver.findElements(By.id(“33dd”)); --** findElements always gives the List of elements.

findElement methods can also be used with element apart from driver…

**element = elements.get(1); element.findElement(By.name(“lastName”));**

1. **element.click();**
2. **element.clear(); -- clears the text in textbox**
3. **element.getAttribute();-- gives the specific attribute name of element**
4. **element.getTagName(); -- gives the elements tag name**
5. **element.getCssValue(“line-height”);-- give the css value styling related**
6. isDiplayed() ---for hidden elements (with input type=hidden)
7. isEnabled()
8. isDisabled()
9. isSelected

**Select Class** : in order to select dropdowns and radio buttons we use Select Api. There arethree different options to use :

1. SelectByIndex() : index starts from 1
2. SelectByVisibileText():
3. selectByValue()

**Code**:

WenElement element = driver.findElement(By.id(“aa”);

Select select = new Select(element) : -- Select class takes the constructor parameter

select.selectByIndex(3);

select.selectByVisibleText();

select.selectByValue();

Dropdown Select can be **Single Select** or **multi Select** ..Select works for both Single and multi where as , there are some methods deselect(), deSelectAll(), deSelectByIndex(), deSelectByValue(),deSelectByVisibleText()—these can be only used with Multi Select dropdown options.

**Radio Buttons:**  radio buttons can be clicked directly by finding the element and clicking.

**Following element methods are used to confirm whether the element is displayed or not/ is enabled or is disabled…**returns the Boolean results.sometime in applications some options would be grayed out and they can be only enabled after some actions.

1. element.isDiplayed() ---for hidden elements (with input type=hidden)
2. element.isEnabled()
3. element.isDisabled()
4. element.isSelected() –can be used with select dropdowns, checkboxes, and radio buttons

**Point Class(API):**

### Why Do We Need Waits In Selenium?

Most of the web applications are developed using Ajax and Javascript. When a page is loaded by the browser the elements which we want to interact with may load at different time intervals.

Not only it makes this difficult to identify the element but also if the element is not located it will throw an "**ElementNotVisibleException**" exception. Using Waits, we can resolve this problem.

**Selenium Web Driver Waits**

1. Implicit Wait
2. Explicit Wait

**Implicit Wait**

Selenium Web Driver has borrowed the idea of implicit waits from Watir.

The implicit wait will tell to the web driver to wait for certain amount of time before it throws a "No Such Element Exception". The default setting is 0. Once we set the time, web driver will wait for that time before throwing an exception.

In the below example we have declared an implicit wait with the time frame of 10 seconds. It means that if the element is not located on the web page within that time frame, it will throw an exception.

To declare implicit wait:

**Syntax**:

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

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

Implicit wait will accept 2 parameters, the first parameter will accept the time as an integer value and the second parameter will accept the time measurement in terms of SECONDS, MINUTES, MILISECOND, MICROSECONDS, NANOSECONDS, DAYS, HOURS, etc.

### Explicit Wait

The explicit wait is used to tell the Web Driver to wait for certain conditions (**Expected Conditions**) or the maximum time exceeded before throwing an "**ElementNotVisibleException**" exception.

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**. These days while implementing we are using **Thread.Sleep()**generally it is not recommended to use

In the below example, we are creating reference wait for "**WebDriverWait**" class and instantiating using "**WebDriver**" reference, and we are giving a maximum time frame of 20 seconds.

**Syntax:**

WebDriverWait wait = new WebDriverWait(WebDriverRefrence,TimeOut);

**Consider Following Code:**

WebElement guru99seleniumlink;

guru99seleniumlink= wait.until(ExpectedConditions.visibilityOfElementLocated(By.xpath( "/html/body/div[1]/section/div[2]/div/div[1]/div/div[1]/div/div/div/div[2]/div[2]/div/div/div/div/div[1]/div/div/a/i")));

guru99seleniumlink.click();

In the above example, wait for the amount of time defined in the "**WebDriverWait**" class or the "**ExpectedConditions**" to occur whichever occurs first.

The above[Java](http://www.guru99.com/java-tutorial.html)code states that we are waiting for an element for the time frame of 20 seconds as defined in the "**WebDriverWait**" class on the webpage until the "**ExpectedConditions**" are met and the condition is "**visibilityofElementLocated**".

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

1. alertIsPresent()
2. elementSelectionStateToBe()
3. elementToBeClickable()
4. elementToBeSelected()
5. frameToBeAvaliableAndSwitchToIt()
6. invisibilityOfTheElementLocated()
7. invisibilityOfElementWithText()
8. presenceOfAllElementsLocatedBy()
9. presenceOfElementLocated()
10. textToBePresentInElement()
11. textToBePresentInElementLocated()
12. textToBePresentInElementValue()
13. titleIs()
14. titleContains()
15. visibilityOf()
16. visibilityOfAllElements()
17. visibilityOfAllElementsLocatedBy()
18. 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.

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

Let's consider a scenario where an element is loaded at different intervals of time. The element might load within 10 seconds, 20 seconds or even more then that if we declare an explicit wait of 20 seconds. It will wait till the specified time before throwing an exception. In such scenarios, the fluent wait is the ideal wait to use as this will try to find the element at different frequency until it finds it or the final timer runs out.

**Syntax:**

Wait wait = new FluentWait(WebDriver reference)

.withTimeout(timeout, SECONDS)

.pollingEvery(timeout, SECONDS)

.ignoring(Exception.class);

**Consider Following Code:**

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

.withTimeout(30, TimeUnit.SECONDS)

.pollingEvery(5, TimeUnit.SECONDS)

.ignoring(NoSuchElementException.class);

In the above example, we are declaring a fluent wait with the timeout of 30 seconds and the frequency is set to 5 seconds by ignoring "**NoSuchElementException**"

## Difference between Implicit Wait Vs Explicit Wait

|  |  |
| --- | --- |
| **Implicit Wait** | **Explicit Wait** |
| * Implicit Wait time is applied to all the elements in the script | * Explicit Wait time is applied only to those elements which are intended by us |
| * In Implicit Wait, we need **not** specify "ExpectedConditions" on the element to be located | * In Explicit Wait, we need to specify "ExpectedConditions" on the element to be located |
| * It is recommended to use when the elements are located with the time frame specified in implicit wait | * It is recommended to use when the elements are taking long time to load and also for verifying the property of the element like(visibilityOfElementLocated, elementToBeClickable,elementToBeSelected) |

**Conclusion:**

**Implicit, Explicit** and **Fluent Wait** are the different waits used in selenium. Usage of these waits are totally based on the elements which are loaded at different intervals of time. It is always **not recommended** to use **Thread.Sleep()**while[Testing](http://www.guru99.com/software-testing.html)our application or building our framework.

**Wait notes by toolsQA:**

## Implicit Wait

Selenium WebDriver has borrowed the idea of **implicit waits** from **Watir**. This means that we can tell Selenium that we would like it to wait for a certain amount of time before throwing an **exception** that it cannot find the element on the page. We should note that implicit waits will be in place for the entire time the browser is open. This means that any search for elements on the page could take the time the implicit wait is set for.

|  |  |
| --- | --- |
|  | WebDriver driver = new FirefoxDriver();    driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);    driver.get("http://url\_that\_delays\_loading");   WebElement myDynamicElement = driver.findElement(By.id("myDynamicElement")); |

## Fluent Wait

Each **FluentWait** instance defines the maximum amount of time to wait for a condition, as well as the frequency with which to check the condition. Furthermore, the user may configure the wait to ignore specific types of exceptions whilst waiting, such as **NoSuchElementExceptions** when searching for an element on the page.

|  |  |
| --- | --- |
|  | // Waiting 30 seconds for an element to be present on the page, checking   // for its presence once every 5 seconds.    Wait wait = new FluentWait(driver).withTimeout(30, SECONDS).pollingEvery(5, SECONDS).ignoring(NoSuchElementException.class);   WebElement foo = wait.until(new Function() {  public WebElement apply(WebDriver driver) {   return driver.findElement(By.id("foo"));   }  }); |

## Explicit Wait

It is more extendible in the means that you can set it up to wait for any condition you might like. Usually, you can use some of the prebuilt **ExpectedConditions** to wait for elements to become clickable, visible, invisible, etc.

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

# Difference Between Implicit, Explicit and Fluent Wait

**Implicit Wait:** During Implicit wait if the Web Driver cannot find it immediately because of its availability, it will keep polling (around 250 milli seconds) the DOM to get the element. If the element is not available within the specified Time an NoSuchElementException will be raised. The default setting is zero. Once we set a time, the Web Driver waits for the period of the WebDriver object instance.

**Explicit Wait:** There can be instance when a particular element takes more than a minute to load. In that case you definitely not like to set a huge time to Implicit wait, as if you do this your browser will going to wait for the same time for every element.

To avoid that situation you can simply put a separate time on the required element only. By following this your browser implicit wait time would be short for every element and it would be large for specific element.

**Fluent Wait:**Let’s say you have an element which sometime appears in just 1 second and some time it takes minutes to appear. In that case it is better to use fluent wait, as this will try to find element again and again until it find it or until the final timer runs out.

|  |
| --- |
| public WebElement getWhenVisible(By locator, int timeout) {  WebElement element = null;  WebDriverWait wait = new WebDriverWait(driver, timeout);   element = wait.until(ExpectedConditions.visibilityOfElementLocated(locator));  return element;  } |
|  |

public void clickWhenReady(By locator, int timeout) {

 WebElement element = null;

 WebDriverWait wait = new WebDriverWait(driver, timeout);

 element = wait.until(ExpectedConditions.elementToBeClickable(locator));

 element.click();

**ACTIONS:** Actions is a class in webdriver is to deal with mouse actions, to automate mouse over ,click ,double click ,key press ,key Release ,drag and drop using an Interface Builder Patterns. It builds composite actions containing all the actions to be performed.

Example : load theURL : driver.get(“yahoo.com”);

find the Elements for the actions: WebElement element = driver.findElement(By.xpath(“dd”));

create the Actions class object which take the driver as the constructor parameter.

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

**Diff methods of Actions Class :**

1. click()

2. contextClick: for right click on the specific element

3. doubleClick

4. keyDown() : press thekeydown

5. kepUp() : releasing the key

6. moveToElement() : it will move to some element

7. sendKeys():

8. clickAndHold() etc.

perform(): it performs the actions : to release any action click, we need perform().

build().perform() : we need this only when we need to perform more than one actions. we can use it for single action too, it won’t through any exception but it’s not required.

ONLINE NOTES :

### Handling Keyboard & Mouse Events

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

|  |  |
| --- | --- |
| **Method** | **Description** |
| **clickAndHold()** | Clicks (without releasing) at the current mouse location. |
| **contextClick()** | Performs a context-click at the current mouse location. |
| **doubleClick()** | Performs a double-click at the current mouse location. |
| **dragAndDrop(source, target)** | Performs click-and-hold at the location of the source element, moves to the location of the target element, then releases the mouse.  **Parameters:**  source- element to emulate button down at.  target- element to move to and release the mouse at. |
| **dragAndDropBy(source, x-offset, y-offset)** | Performs click-and-hold at the location of the source element, moves by a given offset, then releases the mouse.  **Parameters**:  source- element to emulate button down at.  xOffset- horizontal move offset.  yOffset- vertical move offset. |
| **keyDown(modifier\_key)** | Performs a modifier key press. Does not release the modifier key - subsequent interactions may assume it's kept pressed.  **Parameters**:  modifier\_key - any of the modifier keys (Keys.ALT, Keys.SHIFT, or Keys.CONTROL) |
| **keyUp(modifier \_key)** | Performs a key release.  **Parameters**:  modifier\_key - any of the modifier keys (Keys.ALT, Keys.SHIFT, or Keys.CONTROL) |
| **moveByOffset(x-offset, y-offset)** | Moves the mouse from its current position (or 0,0) by the given offset.  **Parameters**:  x-offset- horizontal offset. A negative value means moving the mouse left.  y-offset- vertical offset. A negative value means moving the mouse up. |
| **moveToElement(toElement)** | Moves the mouse to the middle of the element.   **Parameters**:  toElement- element to move to. |
| **release()** | Releases the depressed left mouse button at the current mouse location |
| **sendKeys(onElement, charsequence)** | Sends a series of keystrokes onto the element.   **Parameters**:  onElement - element that will receive the keystrokes, usually a text field  charsequence - any string value representing the sequence of keystrokes to be sent |

**Step 1:**Import the **Actions** and **Action** classes.

[Keyboard & Mouse Event using Action Class in Selenium Webdriver](http://cdn.guru99.com/images/image047.png)

**Step 2:**Instantiate a new Actions object.

[Keyboard & Mouse Event using Action Class in Selenium Webdriver](http://cdn.guru99.com/images/image048.png)

**Step 3:**Instantiate an Action using the Actions object in step 2.

[Keyboard & Mouse Event using Action Class in Selenium Webdriver](http://cdn.guru99.com/images/image049.png)

In this case, we are going to use the moveToElement() method because we are simply going to mouse-over the "Home" link. The build() method is always the final method used so that all the listed actions will be compiled into a single step.

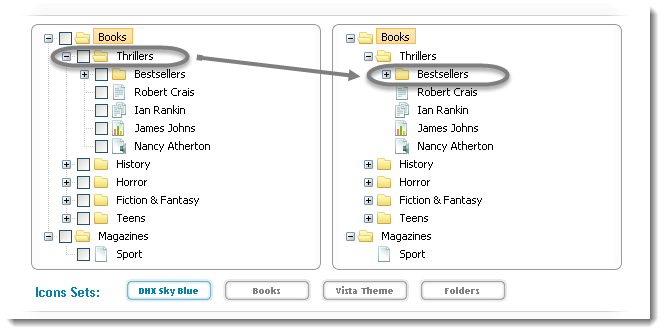
**Step 4:**Use the perform() method when executing the Action object we designed in Step 3.

[Keyboard & Mouse Event using Action Class in Selenium Webdriver](http://cdn.guru99.com/images/image050.png)

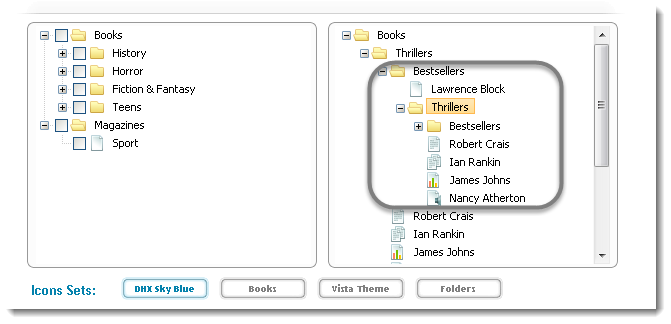
 Selenium has provided a separate “Actions” class to handle these advanced user interactions.

**How it works:**The action chain generator implements the **Builder**pattern to create a Composite Action containing a group of other actions. This should ease building actions by configuring an **Actions** chains generator instance and invoking its **build( )** method to get the complex action.

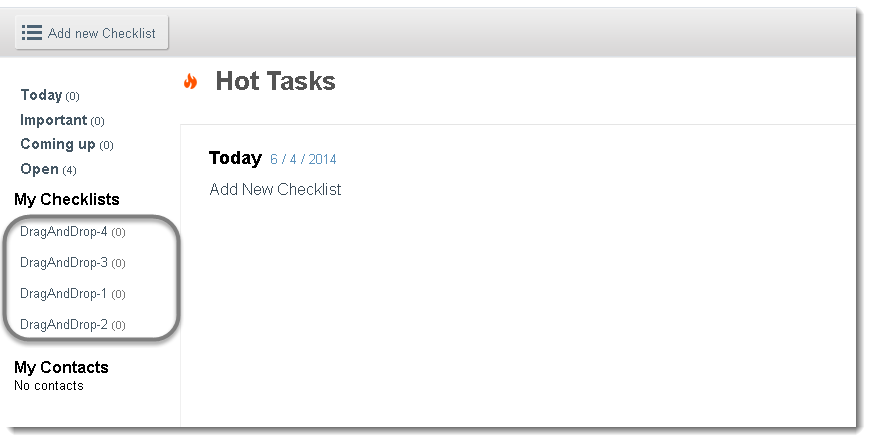
It was really very hard for me to search for a website where I can try ‘Drag n drop’ feature of WebDriver. I was lucky to find two website.

**Example 1:** In this example we will drag the **Thriller** folder from the left table on to the **Bestsellers** folder of the right side table. 

|  |  |
| --- | --- |
|  | package practiceTestCases;    import java.util.concurrent.TimeUnit;    import org.openqa.selenium.By;    import org.openqa.selenium.WebDriver;    import org.openqa.selenium.WebElement;    import org.openqa.selenium.firefox.FirefoxDriver;    import org.openqa.selenium.interactions.Action;    import org.openqa.selenium.interactions.Actions;    public class DragAndDrop {    public static void main(String[] args) throws InterruptedException {    WebDriver driver = new FirefoxDriver();    String URL = "http://www.dhtmlx.com/docs/products/dhtmlxTree/index.shtml";    driver.get(URL);    // It is always advisable to Maximize the window before performing DragNDrop action    driver.manage().window().maximize();    driver.manage().timeouts().implicitlyWait(10000, TimeUnit.MILLISECONDS);    WebElement From = driver.findElement(By.xpath(".//\*[@id='treebox1']/div/table/tbody/tr[2]/td[2]/table/tbody/tr[2]/td[2]/table/tbody/tr[1]/td[4]/span"));    WebElement To = driver.findElement(By.xpath(".//\*[@id='treebox2']/div/table/tbody/tr[2]/td[2]/table/tbody/tr[2]/td[2]/table/tbody/tr[2]/td[2]/table/tbody/tr[1]/td[4]/span"));    Actions builder = new Actions(driver);    Action dragAndDrop = builder.clickAndHold(From)    .moveToElement(To)    .release(To)    .build();    dragAndDrop.perform();    }    } |

Now if you carefully look at the folder, you will notice that the Thrillers folder is been moved to Bestsellers folder. The new folder structure will look like this:

**Example 2:**In this example there is a **Checklist** on the left side and there are four sub menus with the name of DragAndDrop-[1-3]. We will login to this website and then drag DragAndDrop-1 to DragAndDrop-4 of the left side sub menu.

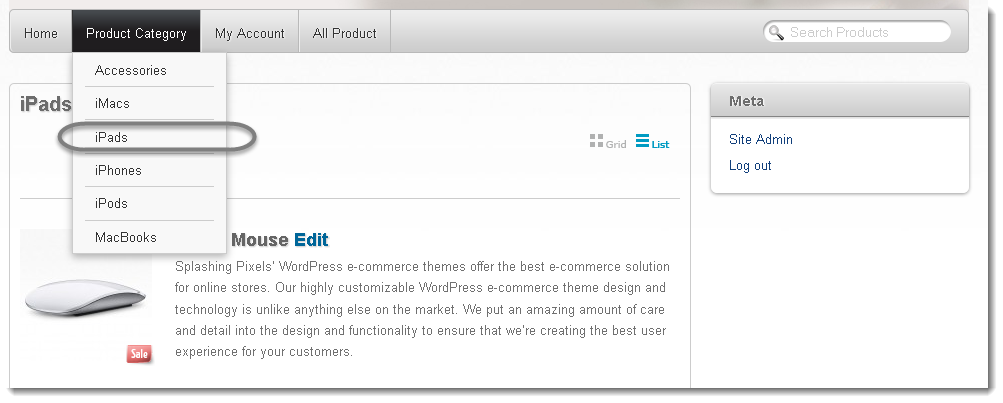


|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55 | package practiceTestCases;    import java.util.concurrent.TimeUnit;    import org.openqa.selenium.By;    import org.openqa.selenium.WebDriver;    import org.openqa.selenium.WebElement;    import org.openqa.selenium.firefox.FirefoxDriver;    import org.openqa.selenium.interactions.Action;    import org.openqa.selenium.interactions.Actions;    public class DragAndDrop {    public static void main(String[] args) throws InterruptedException {      WebDriver driver = new FirefoxDriver();      String URL = "http://sandbox.checklist.com/account/";      driver.get(URL);      driver.findElement(By.name("j\_username")).sendKeys("Username");      driver.findElement(By.name("j\_password")).sendKeys("Password);      driver.findElement(By.name("login")).click();      driver.manage().window().maximize();      driver.manage().timeouts().implicitlyWait(10000, TimeUnit.MILLISECONDS);      WebElement From = driver.findElement(By.xpath(".//\*[@id='userChecklists']/li[1]/a/span"));      WebElement To = driver.findElement(By.xpath(".//\*[@id='userChecklists']/li[4]/a/span"));      Actions builder = new Actions(driver);          Action dragAndDrop = builder.clickAndHold(From)         .moveToElement(To)        .release(To)       .build();      dragAndDrop.perform();    }    } |

# Mouse Hover action in Selenium Webdriver

There will be situations where it is required to click on the item of the drop down menu for e.g. Go to[**ToolsQA demo online store**](http://www.store.demoqa.com/) > Click on Product category link on the top menu > Then select any of the items from the drop down menu:

See the screen shot below:



One way of doing this is by using **Action** class:

|  |  |
| --- | --- |
|  | package automationFramework;  import java.util.concurrent.TimeUnit;  import org.openqa.selenium.\*;  import org.openqa.selenium.interactions.Actions;  public class mouseHover{   public static WebDriver driver;  public static void main(String[] args) {   driver = new FirefoxDriver();   driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);   driver.get("http://www.onlinestore.toolsqa.wpengine.com");   WebElement element = driver.findElement(By.linkText("Product Category"));   Actions action = new Actions(driver);   action.moveToElement(element).build().perform();   driver.findElement(By.linkText("iPads")).click();  }} |

It can be done differently like this:

|  |  |
| --- | --- |
|  | WebElement element = driver.findElement(By.linkText("Product Category"));  Actions action = new Actions(driver);   action.moveToElement(element).moveToElement(driver.findElement(By.linkText("iPads"))).click().build().perform(); |

With some of the browser it happens that once mouse hover action is performed, the menu list disappear with in the fractions of seconds before Selenium identify the next submenu item and perform click action on it. In that case it is better to use ‘perform()’ action on the main menu to hold the menu list till the time Selenium identify the sub menu item and click on it.

|  |  |  |
| --- | --- | --- |
|  | WebElement element = driver.findElement(By.linkText("Product Category"));   Actions action = new Actions(driver);    action.moveToElement(element).perform();    WebElement subElement = driver.findElement(By.linkText("iPads"));   action.moveToElement(subElement);   action.click();   action.perform(); Handling alerts using Selenium WebDriver Selenium provides us with an interface called ***Alert***. It is present in the ***org.openqa.selenium.Alert***package. Alert interface gives us following methods to deal with the alert:   * ***accept()***To accept the alert * ***dismiss()***To dismiss the alert * ***getText()***To get the text of the alert * ***sendKeys()***To write some text to the alert    Simple alert Simple alerts just have a***OK*** button on them. They are mainly used to display some information to the user. The first alert on our test page is a simple alert. Following code will read the text from the Alert and then accept the alert. Important point to note is that we can switch from main window to an alert using the***driver.switchTo().alert().*** Below is the usage of that also:  public static void main(String[] args) {  WebDriver driver = new FirefoxDriver();  driver.get("http://toolsqa.wpengine.com/handling-alerts-using-selenium-webdriver/");  driver.manage().window().maximize();  // This step will result in an alert on screen  driver.findElement(By.xpath("//\*[@id='content']/p[4]/button")).click();    Alert simpleAlert = driver.switchTo().alert();  String alertText = simpleAlert.getText();  System.out.println("Alert text is " + alertText);  simpleAlert.accept();  } Confirmation Alert This alert comes with an option to accept or dismiss the alert. To accept the alert you can use **Alert.accept()** and to dismiss you can use the***Alert.dismiss()***. Here is the code to dismiss a prompt alert.   |  | | --- | | public static void main(String[] args) {  WebDriver driver = new FirefoxDriver();  driver.get("http://toolsqa.wpengine.com/handling-alerts-using-selenium-webdriver/");  driver.manage().window().maximize();  // This step will result in an alert on screen  WebElement element = driver.findElement(By.xpath("//\*[@id='content']/p[11]/button"));  ((JavascriptExecutor) driver).executeScript("arguments[0].click()", element);    Alert confirmationAlert = driver.switchTo().alert();  String alertText = confirmationAlert.getText();  System.out.println("Alert text is " + alertText);  confirmationAlert.dismiss();  } |      Prompt Alerts In prompt alerts you get an option to add text to the alert box. This is specifically used when some input is required from the user. We will use the ***sendKeys()***method to type something in the Prompt alert box.  Code: |

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

WebDriver driver = new FirefoxDriver();

driver.get("http://toolsqa.wpengine.com/handling-alerts-using-selenium-webdriver/");

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

// This step will result in an alert on screen

WebElement element = driver.findElement(By.xpath("//\*[@id='content']/p[16]/button"));

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

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

String alertText = promptAlert .getText();

System.out.println("Alert text is " + alertText);

//Send some text to the alert

promptAlert .sendKeys("Accepting the alert");

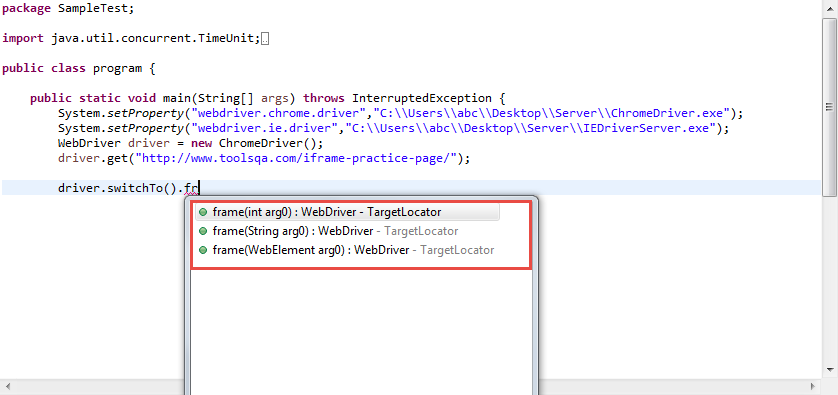
Thread.sleep(4000); //This sleep is not necessary, just for demonstration

promptAlert .accept();

}

**FRAMES:** To Switch between iFrames we have to use the driver’s ***switchTo().frame*** command. We can use the switchTo().frame() in three ways:

* **switchTo.frame(int frameNumber):**Pass the frame index and driver will switch to that frame.
* **switchTo.frame(string frameNameOrId):**Pass the frame element Name or ID and driver will switch to that frame.
* **switchTo.frame(WebElement frameElement):**Pass the frame web element and driver will switch to that frame.

Here is the image showing all the three overloads:  
[](http://toolsqa.wpengine.com/wp-content/uploads/2015/06/IframeOverload.png)

Lets see how each of these work but before that we have to know answers to following questions  
– What is a frame index?  
– How to get total number of frames on a webpage?

## How to find total number of iFrames on a webpage

There are two ways to find total number of iFrames in a web page. First by executing a JavaScript and second is by finding total number of web elements with a tag name of iFrame. Here is the code using both these methods:

WebDriver driver = new FirefoxDriver();

driver.get("http://toolsqa.wpengine.com/iframe-practice-page/");

//By executing a java script

JavascriptExecutor exe = (JavascriptExecutor) driver;

Integer numberOfFrames = Integer.parseInt(exe.executeScript("return window.length").toString());

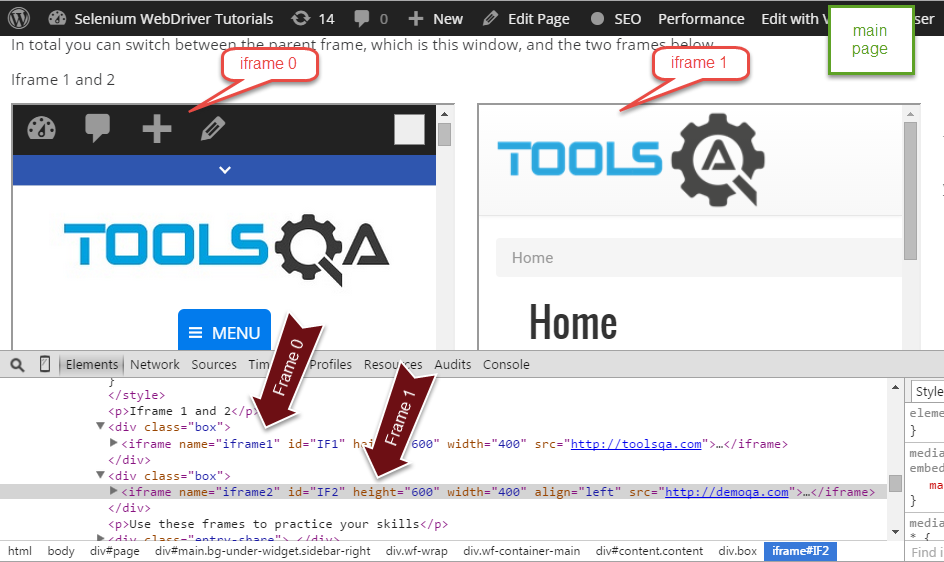
System.out.println("Number of iframes on the page are " + numberOfFrames);

//By finding all the web elements using iframe tag

List<WebElement> iframeElements = driver.findElements(By.tagName("iframe"));

System.out.println("The total number of iframes are " + iframeElements.size());

## Switch to Frames by Index

Index of an iFrame is the position at which it occurs in the HTML page. In the above example we have found total number of iFrames. In the sample page we have two IFrames, index of iFrame starts from 0. So there are two iFrames on the page with index 0 and 1. Index 0 will be the iFrame which exists earlier in the HTML page. Refer to the image below:  
[](http://toolsqa.wpengine.com/wp-content/uploads/2015/06/FramePageDesc.png)

to switch to0th iframe we can simple write ***driver.switchTo().frame(0)***. Here is the sample code:

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

WebDriver driver = new FirefoxDriver();

driver.get("http://toolsqa.wpengine.com/iframe-practice-page/");

//Switch by Index

driver.switchTo().frame(0);

driver.quit();

}

## Switch to Frames by Name

Now if you take a look at the HTMLcode of iFrame you will find that it has ***Name***attribute. Name attribute has a value iframe1

WebDriver driver = new FirefoxDriver();

driver.get("http://toolsqa.wpengine.com/iframe-practice-page/");

//Switch by frame name

driver.switchTo().frame("iframe1");

driver.quit();

## Switch to Frame by ID

Similar to the name attribute in the iFrame tag we also have the **ID** attribute. We can use that also to switch to the frame. All we have to do is pass the id to the switchTo command like this ***driver.SwitchTo().frame(“IF1”)***. Here is the sample code:

WebDriver driver = new FirefoxDriver();

driver.get("http://toolsqa.wpengine.com/iframe-practice-page/");

//Switch by frame ID

driver.switchTo().frame("IF1");

driver.quit();

## Switch to Frame by WebElement

Now we can switch to an iFrame by simply passing the iFrame WebElement to the ***driver.switchTo().frame()*** command. First find the iFrame element using any of the locator strategies and then passing it toswitchTo command. Here is the sample code:

WebDriver driver = new FirefoxDriver();

driver.get("http://toolsqa.wpengine.com/iframe-practice-page/");

//First find the element using any of locator stratedgy

WebElement iframeElement = driver.findElement(By.id("IF1"));

//now use the switch command

driver.switchTo().frame(iframeElement);

driver.quit();

## Switching back to Main page from Frame

There is one very important command that will help us to get back to the main page. Main page is the page in which two iFrames are embedded. Once you are done with all the task in a particular iFrame you can switch back to the main page using the ***switchTo().defaultContent()***. Here is the sample code which switches the driver back to main page.

WebDriver driver = new FirefoxDriver();

driver.get("http://toolsqa.wpengine.com/iframe-practice-page/");

//First find the element using any of locator stratedgy

WebElement iframeElement = driver.findElement(By.id("IF1"));

//now use the switch command

driver.switchTo().frame(0);

//Do all the required tasks in the frame 0

//Switch back to the main window

driver.switchTo().defaultContent();

driver.quit();

<http://roadtoautomation.blogspot.com/2014/05/webdriver-selenium-interview-questions.html>

<http://career.guru99.com/top-10-automation-testing-interview-questions/>

http://www.guru99.com/locators-in-selenium-ide.html

What is Automation ?

Adv. Of Automation…

Diff between Retesting and Regression Testing ?

Regression testing is part of all phases ….everycycle…functional,UAT

**Diff between Assert and Verify?**

Assert stops the execution if the test fails…

Verify still executes the next statements …

**Selenium IDE** – is a tool which geneartes automation scripts

It’s a plugin/add only in Mozilla browser.

Table consists of three a

1.Command : are the actions we do

2.Target : what we want to do ( Finding the Elements)

3.Value: input data

In IDE… we can insert the new commands, insert new comments, and we can set the start point and we can control the execution using Toggle break points.

We can save the test cases and import them from any saved location and use them for regression testing.

**Drawbacks of IDE :**

it only supports Mozilla browser

 Cannot perform few automations

its not efficient in finding the elements also some times

don’t support programming

doesn’t support data driven testing

 not suitable for complex test case design

no centralized maintenance of Elements/Objects

**What is WebDriver ?**

It’s an interface which have several implementation classes w.r.t different browsers.

When to Automate and when not to automate ?

<http://www.seleniumhq.org/> : documentation

There are two main interfaces in webdriver which we use most of the time

1.WebDriver Interface :

**All Known Implementing Classes:**

[ChromeDriver](http://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/chrome/ChromeDriver.html), [EdgeDriver](http://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/edge/EdgeDriver.html), [EventFiringWebDriver](http://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/support/events/EventFiringWebDriver.html), [FirefoxDriver](http://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/firefox/FirefoxDriver.html), [InternetExplorerDriver](http://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/ie/InternetExplorerDriver.html), [MarionetteDriver](http://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/firefox/MarionetteDriver.html), [OperaDriver](http://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/opera/OperaDriver.html), [RemoteWebDriver](http://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/remote/RemoteWebDriver.html), [SafariDriver](http://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/safari/SafariDriver.html)

Diff Methods :

FindElement and FindElements are the methods of webDriver Interface : used to to find the element location and try to retrun that particular web element. For list of elements we use webElements.Returns A list of all [WebElement](http://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/WebElement.html)s, or an empty list if nothing matches.

[WebElement](http://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/WebElement.html) findElement([By](http://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/By.html" \o "class in org.openqa.selenium) by)

List<WebElement> findElement(By by)

By is the class which we used to locate the elements .It has 8 different static locater methods which takes strings as the parameter.

We find web elements using “By” Class parameter.

Pink color are the html elements and maroon color one are Attributes of the Elements.

By - class which provides static methods to locate element

8 diff methods which take string input and return By object:

id - id is most efficient and fastest way as it internally uses JS doc.getElementByid method directly.

name

class

tag

linkText

partialLinkText

xpath- Try to locate the element by traversing across the html document

2.WebElement Interface has diff methods …ex. Clear,Click,Submit,sendKeys etc.

Get(“”) : opens the browser with to get the given url

http://www.softwaretestinghelp.com/using-selenium-xpath-and-other-locators-selenium-tutorial-5/

CSS :

<http://www.simplehtmlguide.com/whatiscss.php>

**CSS** is short for Cascading Style Sheets, and is the preferred way for setting the look and feel of a website. The style sheets define the [colour](http://www.simplehtmlguide.com/colours.php), size and position of text and other [HTML tags](http://www.simplehtmlguide.com/whatishtml.php#htmltags), while the [HTML files](http://www.simplehtmlguide.com/whatishtml.php#htmlfiles) define the content and how it is organised. Separating them allows you to change the colour scheme without having to rewrite your entire web site.

The cascading means that a style applied to a parent element will also apply to all children elements within the parent. For example, setting the colour of body text will mean all headings and paragraphs within the body will also be the same [colour](http://www.simplehtmlguide.com/colours.php).

### Specifying and Using Styles

There are three main ways of including a style sheet for a web page or site:

1. Setting the sytle="?" attribute of a tag, called [inline styles](http://www.simplehtmlguide.com/whatiscss.php#inline)
2. Using the <sytle> tag within the [HTML header](http://www.simplehtmlguide.com/whatiscss.php#header) tag
3. Creating and linking to an [external CSS file](http://www.simplehtmlguide.com/whatiscss.php#external)

Basic style sheets usually modify the appearance of html tags such as <body> and <p>. When using CSS files or style sheets within the header, we can also define classes of styles and apply them to any element using the class="?" attribute, but this is beyond the scope of this simple guide.

### Inline Styles

Styles defined inline in HTML will only apply to the tag they are added to. Note: [colours](http://www.simplehtmlguide.com/colours.php) can be specified as either a CSS colour name or hex colour code.

<p style="color:red;">Some red text</p>

### Within the HTML header

A style defined in the header will apply to the whole page. The example below will make all h1 tags in your page show the heading in red.

<head>

<style type="text/css">

h1 {

color: red;

}

</style>

</head>

### External CSS file

Like HTML files, CSS files are also plain text, and usually have a **.css** file extension. An example of a CSS file name style.css can be seen below.

body {

background-color: beige;

color: #000080;

}

h1 {

color: red;

}

The file can then be included using the <link ... > tag in the HTML header.

<head>

<link rel="stylesheet" type="text/css" href="style.css" title="style">

</head>

Frequently used methods / objects/commands in webdriver

1. driver.getCurrentUrl()
2. driver.getPageSource()
3. driver.getTitle());
4. driver.manage().window().maximize();
5. driver.getWindowHandle()
6. driver.getWindowHandles()
7. driver.getTitle()
8. driver.close()
9. driver.quit()
10. switchTo()
11. FindElements
12. Find Elements

WebElement methods :

1. element.sendKeys;
2. element.submit();
3. element.click
4. clear : used to clear text box content only.
5. Element.getText() : inner text
6. findElement() : Finding inner elements
7. fineElements() ---element.findelements
8. element.size()
9. isDiplayed() ---for hidden elements (with input type=hidden)
10. isEnabled()
11. isDisabled()
12. isSelected()
13. getsize()
14. Dimesion.getHeight()
15. getLocation()
16. point.x
17. point.y
18. element.getAttribute()
19. element.getCssValue()
20. element.getTag

Whenever we see “is” methods in java its returns always Boolean value.

for dimenstion of any elements we have a class called Dimension and we can use the webElements methods for Dimension class.

Ex.Dimension dimension = element.getsize();

For position of the elements …Class is Point

Ex. Point point = element.getLocation();

For hidden elements we see (input=hidden)

**Navigation Class methods**

Navigate() method :1.back()

2.forward()

3.refresh()

4.To(String)

5.To(URL)overloaded method

To() method is similar to get().

BreadCrums in UI : present location showing in the site

Diff between navigate().To and get methods?Get is to open the website and navigate().To also does the same with thing but it has extra methods like back ,forward,refresh etc.

For selecting the drop we have class called “Select”. Select class takes element as the constructor parameter.

Ex.Select select = new Select(element);--- then we can use the methods of the Select class using class object.

Select boxes can be single select or multiple selects.

isSelect() : is the method used to decide whether it is single select box or multiple select box.

Diff between thread.sleep and waitExplicitly ?

1.FindElements and Find Elements

2.WebDriver driver = new ChromeDriver/new FireFoxDriver/IEDriver

3.WebElement element = driver.FindElement.

(By. Id/name/class/tag/linktext/partiallinktext/xpath/css)

In Css selector for class we give “.” In the beginning and for spaces.

For id we give “#”

Diffrence between Find Element and FindElements ?

FindElement : it returns Single WebElement and returns NoSuchElementException when element is not present in html. FindElement can also be used to check the element presence.

nosuchElementException is the common exception which we come across.

staleElementException : when the element is no more connected to the document.

FindElements : it returns the List of WebElements. Returns empty list.

when we are not able to find the element then the list takes as emply and it returns the size as ZERO, where as findElement if it wont find the element then it throws the exception.

When ever <input type=’submit’> and element is inside form we use submit.

Click works in all the situations – inside or outside forms..but for forms submit is preferable.

**Diff between Quit and Close :** “Quit” closes all the opened windows and “close” closes the current window.

Two kinds of waits provided by webdrivers;

Wait : Webdriver allows programmer to wait for the element/elements to be located , before throwing the Exception like “No such Element/ Element not found” exception.

Two kinds of wait :

1.implicitly : Waits for entire time the browser is open. It internally waits for all the elements, Whenever there is any delay in finding element then only it waits not like Thread.sleep().It waits for the element which you are trying to access. It wait all the time before driver.close() executes.

It checks initially

Syntax. driver.manage().timeOuts().implicitlyWaits()

2.Explicitly : waits for specific condition or specific elements for given time or it throws the exception. It checks in between the wait time for presence of the wait time.by default wait time is 500millisec. If it finds the element after

Syntax : for explicitwe have a class called “WebDriverWait” and it has a defialu constructor which takes driver object and time…and time it takes in seconds by default.

After creating objects we use methods called “until” and ittakes overloaded parameters : one is function and the other is class “ExpectedConditions” which again has diff methods like …visisbilityOfElementLocator(locator).

WebDriverWait wait = new WebDriverWait(webdriver Object,timeOutinSeconds);

Explicit wait until the timeout time still if there is any mistake in locator also…

FluentWait

Function is a Interface which is also like a method but it has specific definition and anonymous class declaration expecting two parameters one is input parameter and the otheris return type parameter.we needto add unimplemented method in it.

We should not use explicitWait and implicitWait together.

Predicate is also same as function ,but it wont allow our own retun type….its only retorns “Boolean”

Q:anonymous class declaration in java:

Q:Generic declarations :

ACTIONS: Actions is a class in webdriver is to deal with mouse actions, to automate mouse over ,click ,double click ,key press ,key Release ,drag and drop using an Interface Builder Patterns. It builds composite actions containing all the actions to be performed.

**1st way:**

Hide   Copy Code

act.dragAndDrop(Source, Target).build().perform();

**2nd way:**

Hide   Copy Code

act.clickAndHold(From).build().perform();

act.moveToElement(To).build().perform();

act.release(To).build().perform();

**3rd way:**

Hide   Copy Code

act.dragAndDropBy(Source, xoffset, yoffset).perform();

**Diff methods of Actions Class :**

1.click()

2.ContextClick

doubleClick

keyDown() : press thekeydown

kepUp() : releasing the key

moveToElement() : it will movwto some elemnt

perform(): and it performs the actions : to release anyaction click we need perform().

sendKeys():

clickAndHold()

build().perform() : we need this only when we need to perform more than one actions. we can use it for single action it won’t through any exception but it’s not required.

contextClick(): for right click on the specific element

Example : load theURL

find the Elements for the actions

create the Actions class object which take the driver as the constructor parameter.

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

**takeScreenShot()** : we need to typecast the driver api ,because takeScreenShot() method is only present in the TakeScreenShot Class but not under the driver. And it takes the screenshot and it will give you the output type and we can store into a file. You cann take the screenshots any where.

**File fie = TakesScreenshot(driver).getScreensotAs(OutputType.FILE);**

Catch{

FileUtils.copyFile(file, new File(“c:/users/wbl/screentshot.png));

}

Double backward slash or single forward slash.

**ALERTS** : alerts ate the popup windows comes up in the browsers with them we can either accept them or cancel them.

In order to deal with alerts in WebDriver we have a class called “Alerts”--- and these alerts can be handled in two ways either accept or to dismiss.

Alerets are of two types : Window Based alerts –To identify we cannot right click or inspect these alerts.

Browser based alerts : these can be closed or accepted and we can inspect for close button and handle.

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

alert.accept();

alert.dismiss();

switchTo(): is used to handle the multiple windows actions and also for the alerts

Window.alert()

Prompt alerts: alert.sendKeys(“abcd”).click;

When you are trying to handle the exception with catch block then no need or throws, you can try to catch on your own.

JavaScriptExecuter : allows to execute javascript in webdriver.

----------------------------------------------------------------------------------------------------

Whu we need automation framework

For automation :

1.we need a lang

2.for executing we use testing framework(no need for main method of java)

3.verification and validation use assertions from testing

4.to generate reports we use testing

5.need the logs/screen shots to debug we use (log4j)

FRAMEWORK AUTOMATION

Design Patterns to build hybrid frameqork

Data Driven

PageObject Patterns

Keyword Driven Pattern

Modular Driven Framework

Actions Driven

UIMapping

MEDHA AUGUST BATCH NOTES --- WEBDRIVER

Selenium

**Automation**: Testing the given application(web or mobile) using the external tools or the automation framework code.

Selenium is used to automate the web applications.

**Advantages of Automation:**

Repeatability of the tests using tools or code rather than manual testing leads to

increase the speed of execution of test cases.

We can vary the tests for different user inputs easily by implementing data driven testing.

It easily supports frequent regression testing do be done in minimal time and generates good test reports to analyse.

It also helps to find the defects which you may miss while doing manual testing.

**When to automate and when not to automation?**

Automation depends on timeframe available, if we have a very tight dead line then we cannot go for automation.

Automation should be avoided if there is any UI change expected in near future as we need to rewrite the automations scripts again.

**Also application scenarios wise – pages which need manual interactions cannot be automated like**

1. If there is some captcha code being generated instantly and u need to enter that code or payment page where OTP is bing sent to mobile.

2.Image / Video Previews - Between transitioning between thumbnail and playback of videos I have observed not well automated

3.BarCodeReader cannot be automated

**Different Automation tools & frameworks**: Selenium, QTP-vbscript,SilkTest etc

4- Few look and feel related testcases need manual testing

Selenium is set of different tools to support different test approaches…

In 2004- Jason Huggins started Selenium at Thoughtworks.

Initially he developed a javascript library

**Selenium IDE** – is a tool which generates automation scripts

It’s a plugin/add only in Mozilla browser.

Create automation scripts quickly , to learn automation scripts with the help of a tool aided mechanism.

Also as a begineer who move from manual testing to automation- he can have a glance of automation scripts using the selenium IDE.

He can also get an idea about locators.

**Selenium 1** – programmatic way of automation which had been now developed as webdriver..

Selenium RC – Selenium Remote Control - client server architecture.

Its currently deprecated.

**Limitations of Selenium RC:**

Security became an issue due to javascript injection

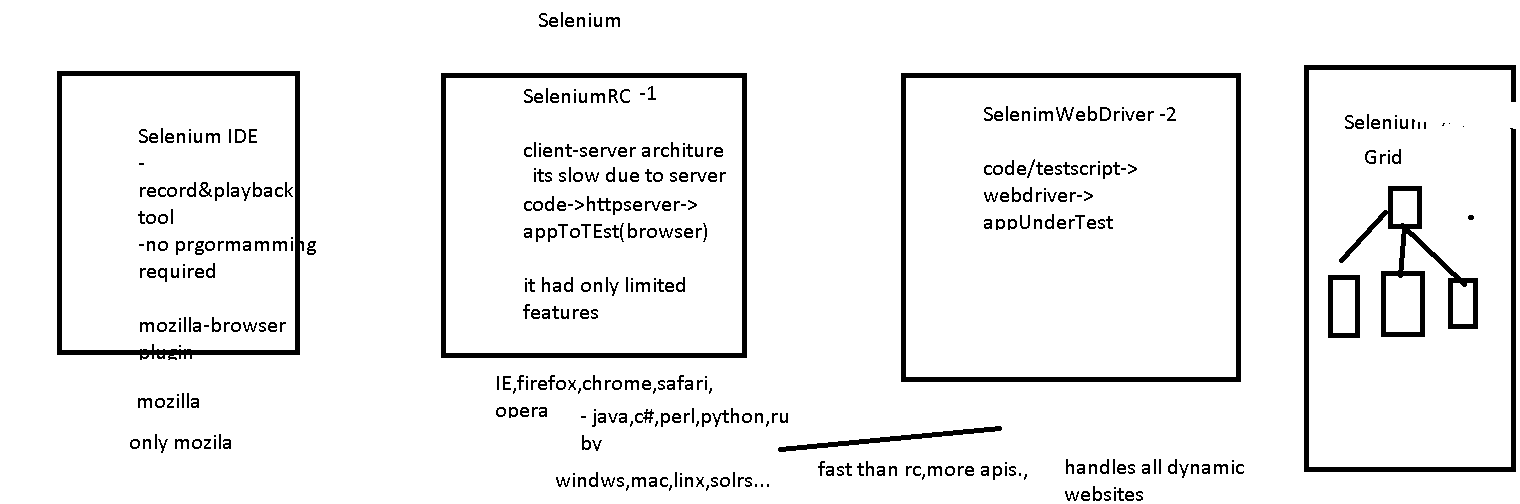
Its slower because of client –server architecture i.e., there was RC server in between TestAutomationCode and application to Test unlike webdriver

**Selenium 2 - Selenium Webdriver-** programmatic way of automation

It has number of API’s(set of libraries) to achieve automation programtically.

It helps to create very robust, browser-base regression testcases and suites.

It helps to manage things more efficiently than a tool for mutiple environments.

****

**SeleniumWebDriver:**

**What we do in automation:**

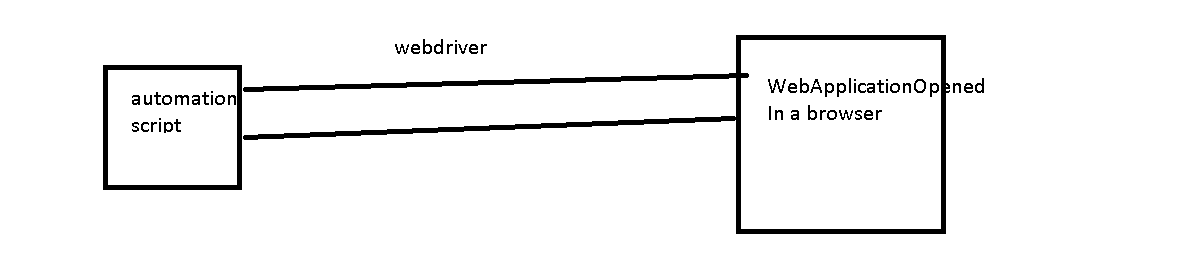
**Write scripts – selenium webdriver + java**

**Assert the testcases**

**Log the errors**

**Generate report to analyse the tests**

**Unittesting f/w – testng /junit**

****

**Automating steps:**

Open the browser and loading our website/any page we want to automate

Find the elements in a given page to automate and do the actions

Add assertions

Close the browser

**Webpage will be composed of html elements in window, html elements can be**

**Buttons,text boxes, links,dropdowns,statictext, titles, info, divs, image……**

**WebDriver-(I) deals mainly with browser/window related actions**

**Automation means we do some actions on this elemnts using our programs**

**WebElement(I) interface deals everything related to specfic html element you find—**

**How u find element – we use By class**

**WebDriver methods:**

Void **get**(String url) – pass what url we want to test as a string

WebElement **findElement**(By by)-we use this method to find a web element using By class(helps to pass the locator).It returns WebElement.

This will simply throw NoSuchElementException when it is not able to find the element.

**List<WebElement> findElements**(By by) – will return List<WebElement> eg: breadcumbs,nav links etc

to locate presence of element in a web page: if the element we are trying to get is not found then those findElements will return an emptylist- so we can also use this method to check for list size is not empty and perform logic accordingly in situations where we are not sure that element may be present or may not be present.

String **getTitle**() – title of the current page

Options **manage**() – 1. To maximize the browser window

Navigation **navigate**():

Naviagation: to – go to the paricular page url – its same like the get method

back()

forward()

refresh()

Void **close**()- close the curent window opened by the driver.

Void **quit**()- close all the windows opened by driver

String **getCurrentUrl**() – returns current url of the page which is loaded

String **getPageSource**()- gives the source code for the current page

**Takescreenshot** using TakeScreenShot Interface – used to record the error scenarios or log the error pages:

File screenshot = ((TakesScreenshot)driver).getScreenshotAs(OutputType.FILE);

FileUtils.copyFile(screenshot, new File("C:\\Users\\Training room\\MedhaJavaWS\\SeleniumWebDriver\\test.png"));

**WebElement methods:**

Void **Click**() – performs click action on given element- **links,buttons(both in and out of forms),checkboxes,radiobuttons**

Void **Clear**()- it will clear the text in the **input textbox or textarea**.

Void sendKeys(String text)- send some text to the textbox field

Void **submit**()- generally use for html form submissions on buttons i.e., **button inside a html form.**

WebElement **findElement**(By arg) – help to find element inside the element.

List< WebElement> **findElements**(By arg)- help to find elements inside the element

String **getText**() – returns inner text of the given html element

<a href=’testjk.com’..>CIiCKME</a>

String **getTagName**()- return the particular element tagName

<a href=’testjk.com’..>CIiCKME</a>

String **getAttribute** – returns html elemnt’s attribute value

String **getCSSValue** – returns css value for element

Point **getLocation** – gives where html element is located in webpage using x and y axis coordinates – using Point class a sreturn type

Dimension **getSize** – it will size of html element – width and height using Dimension class as return type

getScreenShotAs – to take the screenshot of the page ..

boolean isSelected – used for checkbox,radiobuttons,select dropdowns to see if option is selected or not

boolean isEnabled – to see if element is enabled or disabled

boolean isDisplayed – to see if element is present or hidden in page.

By class in webdriver has all static methods which we can simply call using the syntax By.methodName…these methods in By class help to locate a particular html element in our page.

By class methods:

1. id(String value) – this is most efficient and prferred way to locate any element.. it will take id value of html element as input and locate the html element.

Eg: <input id=”uname”…..>

<div id=”currentDiv”…>

2. name(String value)- this wil locate a html element with attribute name, this is also the most prfered way but we need to make sure that the same name is not used for more than one html element.

If we have more than matching name elements then the first macth will be returned .

<button id=”test1” name=”submit”….>

<input id=”test1” name=”username”….>

3. linkText(String value)- this wil locate the element using the link text for any of the given links.

<a href=”link.com”>Logout</a>

4. partialLinkText(String value) – this will locate the element with partial link name.

<a href=”link.com”>Logout</a>

5. className(String value) – this will locate the element using the class(css class) attribute of the html element

<button class=”submit-btn-class”…..>

6. tagName(String name) – help to locate the html element by its own tag name.

<input id=”test” name=”tt”…>

<button id=”test” name=”tt”…>

<font id=”test” name=”tt”…>

7. cssSelector(String value) – helps to locate elements based on css selector path

8. xpath(String value) – its most widely used , helps to locate the lelement based on its position in the Dom(html document object model).

Chrome browser ->inspect->copy xpath-//\*[@id="footerbot"]/div[1]/span/span/a/span[1]

Inspect using Mozilla-firebug - /html/body/div[1]/div/div/div/div/article/div[3]/div[1]/font – absolute xpath

Mozilla-Inspect element using firepath – relative xpath

.//\*[@id='result']/font

#footerbot > div.pull-left > span > span > a > span.region

Select dropdowns/Select API : First create webelement using driver.fineElement(By….)

Create Select class object by passing above created WebElement objcect as input.

On select object we get different methods to select the dropdown values using either selectByIndex or selectByVisbleText or selectByValue..

We can use getOptions method to see all the available dropdown options.

We can also use Select class to select multiple select dropdowns and here we can use methods like isMultiple,deselectAll, deselectByIndex or deselectByVisbleText or deselectByValue..

We can also use getAllSelectedOptions to see what all we selected.

Select select = new Select(element);

RadioButtons: You can simply findelement using By class and click on element.

Or we can findElements which return list of 2 or more elements() and we can select from that list whatever we want by using index.

Eg: radioList.get(1).click();

radioList.get(3).click();

Waits:

**Implicit wait:**

Whenever we are not sure which specific element we want to wait upon in a given html page and this wait will work for entire time the browser is open.

Initially it checks for element to be present and if its not present then it starts waiting for a given time and then finally at the end of wait time it again checks for element. If it could not find elemet – then it throws Exception..

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

It is generally prefered for overall small waiting time

**ExplicitWait:**

Gives option to specify which element we are trying to wait for, it accepts some condtions as input and it helps to handle waiting with specific elements/conditions.

Whenever we know that specific element takes more time to load – we prefer explicit time as it saves time not waiting for all the elements for same time.

WebDriverWait wait = new WebDriverWait(driver,20);

WebElement element = wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("close-button")));

wait.until(ExpectedConditions….anyConditionasperrequirement);

wait.until(function)

wait.until(predicate)

**FluentWait :** Same as explicit wait where in it waits for a given condition but then it takes frequency also as input.Also takes input to ignore a specific exception.

Whenever we know that a particular element is loading for 5 seconds sometimes and sometimes it takes 10seconds…wherein we are not sure about wait time…

Wait fluentwait = new FluentWait(driver)

.withTimeout(10, TimeUnit.*SECONDS*)

.pollingEvery(5, TimeUnit.*SECONDS*)//frequency

.ignoring(StaleElementReferenceException.class);

Function<applyMethodInputDataType, applyMethodReturnDataType>function = new Function<WebDriver,List<WebElement>>(){

public List<WebElement> apply(WebDriver driver){

return driver.findElements(By.*xpath*(".//\*[@class='suggestions-list']/li"));

}

};

totalWaitTime-20

Frequency - 5

5th, 10th,15th, 20th

Actions: helps to deal with mousehover interactions, drag and drop,keyboard handling…

build will compile list of actions to a single step and perform will do all actions

together

eg:actions.clickAndHold(menuElement).clickAndHold().click().build().perform();

but in single action scenaios ..even if u perform without build it will work—u need build only when u want multiple actions together

eg:

menuElement = driver.findElement(By.id("navigation-top-cat-label-1308"));

actions.moveToElement(menuElement).perform();

menuElement = driver.findElement(By.id("navigation-top-cat-label-1764"));

actions.moveToElement(menuElement).click().perform();

Deal with drapAndDrop:

actions.dragAndDrop(from,to).perform()

or

actions.clickAndHold(fromElement).moveToElement(toElement).release(toElement).build().perfom();

To handle differnet windows:

We use driver.getWindowHandle()- gives current window and driver.getWindowHandles()- all the list of windows open.

And we use driver.switchTo.window(String arg) to switch the control from one window to other window.

For all popup boxes like alert,prompt,confirm boxes we use Alert class or AlertAPI to accept/dismiss/sendkeys..

Alert alert = driver.switchTo.alert()

alert.dismiss/accept..

Switch to iframe – driver.switchto.frame

Inspectors –

Firepath- enhanced version of firebug.we can also write our own xpath and evaluate/verify them with eval button using firepath.

Firepath - Relative xpath - .//\*[@id='headerfblogin']

We can still absolute xpath also from firepath by changing the settings

Firebug- Absolute xpath - /html/body/header/div[2]/div/ul/li[1]/a/button

Chrome-xpath helper

To be discussed:

Parallel testing – testing..

Selenium grid

AutoIt- interacting with desktop tasks….

pagefactory

**HtmlUnitDriver** : Headless browser is a browser which do not have any GUI.

After 2.53 HtmlUnitDriver is not part of selenium jar ..so we need to download external jar and add it to project or add as maven dependency.

https://github.com/SeleniumHQ/htmlunit-driver/releases

When do we use it:

To do any quick sanity testing we prefer this browser as it is very fast in exceution when compared to other browser.

Also we can use it to simulate multiple browser environment in same machine.

This uses RhinoJavaScript engine.

By default javascript exceution is disabled in this browser…so we need to enable it using driver.setJavascriptEnabled(true)..

**DesiredCapabilities**: Information about browser, platform,device,etc..

These are define as JSON objects(key-value pairs) to define the features that a session will support.

Like DesiredCapabilties

We can also set any browser specific settings using

**chromeOptions** for chrome browser

**FirefoxProfile** for firefox browser

And then pass it to WebDriver objct.

1. It is a class in org.openqa.selenium.remote.DesiredCapabilities package.
2. It gives facility to set the properties of browser. Such as to set BrowserName, Platform, Version of Browser.
3. Mostly DesiredCapabilities class used when do we used Selenium Grid.
4. We have to execute mutiple TestCases on multiple Systems with different browser with Different version and Different Operating System.
5. Mobile automation – to specify platform, device, what file to test…

To mange **cookies**: We can use Cookie class in the Webdriver..we can add,delete and get existing cookies using this class methods.

**JavaScriptExecutor**: Its an interface in WebDriver which provides a way to exceute javascript code through selnium driver.

((JavascriptExecutor)driver).executeScript(…..)

When to use it: To enhance the capabilties of existing scripts – we can use this class.

AutomationFramework : To create reusable,modular and maintainable application

1. To do automation – we need to choose a language –Java,Javascript,C#
2. Don’t want to execute using main method – (Testing F/W – testng)
3. Test definitions- test cases – verify&Validate the testcases – assertions & logs -(Testing F/W – testng)
4. Test reports – to analyse -(Testing F/W – testng)
5. Also we need logs /screenshots to debug-(log4j)
6. We want a mechanism to run all the tests together –(Testing F/W – testing.xml)
7. Code reusability – Java DesignPatterns(dependencyInjection,singleton,PageObject,UIMapping,DataDriven…)
8. Avoid hardcoded data – properties file (java.uti.Properties)
9. Data driven using excel,csv,text… - apache poi
10. Build and continuously integrate the changes(Maven&Jenkins)
11. Automation F/w- web/mobile/RestAPis – SeleniumWebdriver,WebDriver+Appium,selendroid, http-client

Automation Framework:

Selenium + Testing F/W +DesignPatterns/Structure+ Utilities - external utitilities like apache poi for reading excel data,Log4j for logging etc …..

DataDriven- drive the test data from external file into the code

PageObject pattern – Every page in website has corresponding Object(java) to deal with all the functionalities of that particular page.

KeyWordDriven – this pattern follows like every keyword for certain functionalty

Modular

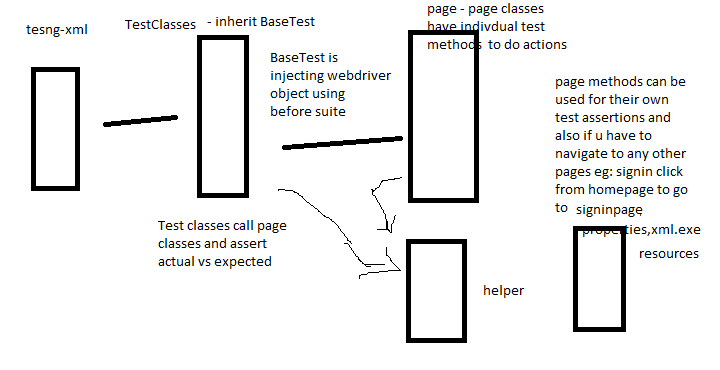
ActionsDriven

UIMapping – mapping the uilocators with certain key value pairs at one place in the project rather than hardcoding across the project.

So that we can avoid changing java code everytime when the locator is being changed. It also helps foe easy maintenance and readable.

Hybrid Framework – combination of one or more above mentioned f/w’s

Eg for hybrid : PageObject+data driven



-----------------------------------------------------------------------------------------------

August-batch

Selenium

Automation – Testing or developing automation framework with the help of tools or programs.

Cross browser Testing – chrome, mozilla, chrome,safari,IE

OS’s – mac, windws, linux

Functional testing

Integration testing

System testing

Uat Testing

Prod

Dev sit uat prod

Dev qa stage prod

Dev qa(qa1, qa2) prod

RegressionTesting – retest a defect or when you are testing a new enhancement

Regression is part of all phases like functional, sit, uat…

Datadriven testing- test a functionality with different sets of data

Advantages of Automation:

Saves time and resources

You can perform data driven testing easily

Regression testing is easy

Repetative manual tasks are avoided

More efficient and can figure out defects which might have in manual testing

It increases productivity and less eroor prone

It reduces investment cost

Tools- QTP, UFT, Selenium IDE, …

Selenium

2004- Jason huggins – Thoughtworks – Developed SeleniumCore(javascript based)

Using which people started automation with browsers

Selenium core is base for Selnium Ide and Selenium RC

Around 2006, to overcome the drawbacks of IDE and RC stewart from google

Came up with WebDriver concept

Limitations of Selenium Core:

Javascript based and it was less secure

Not all functionlaities could be automated

And could not run efficiently in all browsers.

Selenium IDE:

Record and playback tool for automation

It is a plugin that works only in mozilla

It can automate applications to some extent but not complete automation.

Because we don’t have any flow control and data driven testing not possible.

Selenium IDE can be used for prototyping the test cases.

Continous build and Continous Integration( – maven and jenkins) is not possible.

Selenium RC:

Using selnium rc jars we write the code – to execute the script you need to start and stop selenium server that interacts between your code and AUT(application under test)

Selnium 1 – RC and IDE

Selenium2- WebDriver

To automate an application with webdriver:

1. WebDriver API jars
2. Java/ruby/python
3. Testing framework- TestNG

Selenium grid – multiple platforms and browsers testing

SeleniumServer

Browsers: chrome,moziulla,IE,safari,htmlunit,phantom js, android,ios

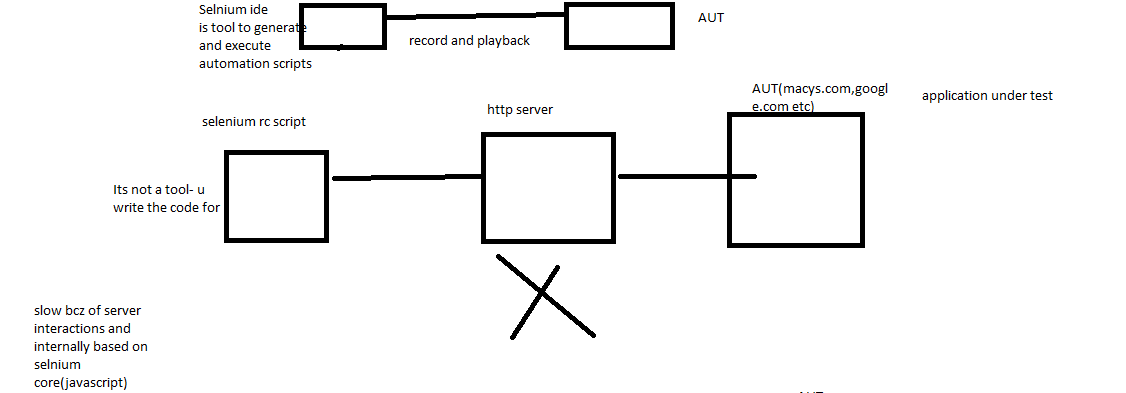
OS- mac,windows,linux

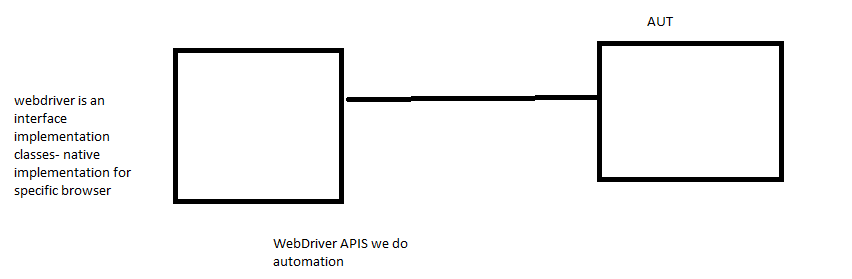
When to automate and when not to automate:

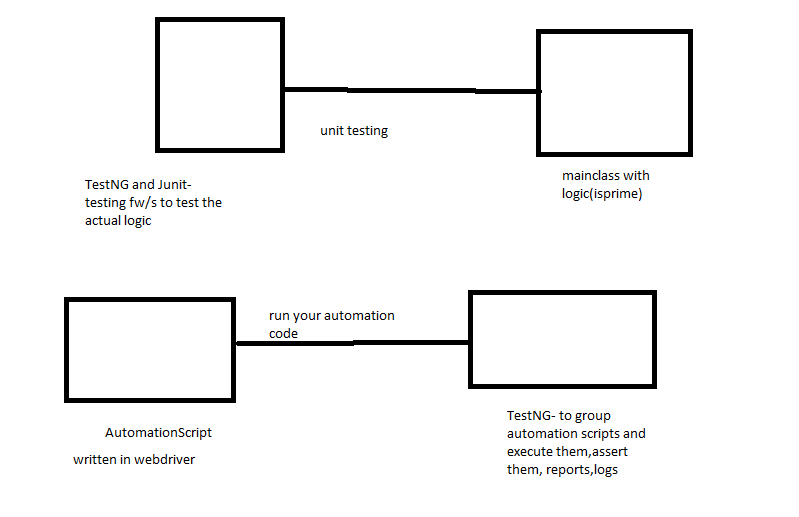
When you do not have time to automate or very tight deadlines to setu a automation framework.

When application look and feel is not stable or it is going to change in mere future.

Tasks which require human intervention cannot be automated.







WebDriver – interface- which has browser specific implementation classes

Eg: FirefoDriver,ChromeDriver,OperaDriver…..

WebDriver declares some important methods which are implemented by its subclasses.

Get

findElement – it return single webelement

this will return NoSuchElementException when element is not present in html page

findElements- list of webelements

this will return empty list when element is not present in html page

getTitle

getCurrentUrl

getPageSource

getWindowHandle

getWindowHandles

switchto()

close

quit

driver.manage.window.maximize

WebElement findElement(By by):

Click

Submit

sendKeys

clear – its used to clear textbox content

List<WebElements> findElements(By by)

By – is a class which is used to locate the elements. It has 8 different locator static methods.

By.id(“stringparam”)

By.name(“stringparam”)

By.tagName(“stringparam”)

By.class(“stringparam”)

By.linkText(“stringparam”)

By.partialLinkText(“stringparam”)

By.xpath(“stringparam”)

By.cssSelector(“stringparam”)

<input type=”text” id=”wblTextBox” class=”txt.box” >

<a id=”lid” href=”login.html” name=”linkname”>

<input type=’submit’

And element is inside form

Click will in all scenarios—inside or outside form

Select class – to select dropdowns and multiple select

Radio button click

WebElement methods…

Waits : Webdriver allows the programmer to wait for the element/elements to be located before throwing the exception like NoSuchElement/ElementNotFound exceptions…

1. Implicit wait – waits for entire time the browser is open by the driver…

It checks initially before the wait time for presence of element- if it do not find the element it will wait for given time and at the end of wait time it again checks for element (it do not check in between), if it finds the elemnt it return the element else it throws exception.

driver.manage().timeouts().implicitlyWait(20, TimeUnit.seconds);

1. Explicit Wait – waits for a specific condition or specific element for a given a amount of time before throwing the exception.

It checks in between the wait time for presence of element by default for every 500 ms and if it finds the element after this specific time it returns the element and do not wait further.

We can use until method with function/predicate or ExpectionConditions.

WebDriverWait wait = **new** WebDriverWait(driver,20);

WebElement element =wait.until(ExpectedConditions.*visibilityOfElementLocated*(By.*xpath*("//\*[text()='ASP.NET Web Forms']")));

a.FluentWait – it is similar to explicit wait but it allows to mention the frequency for which the element presence should be tested instead of going with WebDriver default frequency(500ms), along with wait time.

Also you can declare any specific exception to be ignored

20 secoonds – wait time

pollinEvery/frequency – 5 seconds

every 5 seconds it checks for the wait condition and do not for complete wait time if you find the element in between.

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

.pollingEvery(10, TimeUnit.***SECONDS***)

.withTimeout(30, TimeUnit.***SECONDS***)

.ignoring(NoSuchElementException.**class**);

WebElement elem = wait.until(function);

Actions

Mousehover

drapAndDrop

Alerts

Actions – to automate mouse actions like hover, click, contextclick, doubleclick,

Keypress,keyrelease, dragAndDrop…..

JavaScriptExecutor – allows to execute javascript code in webdriver

Eg:

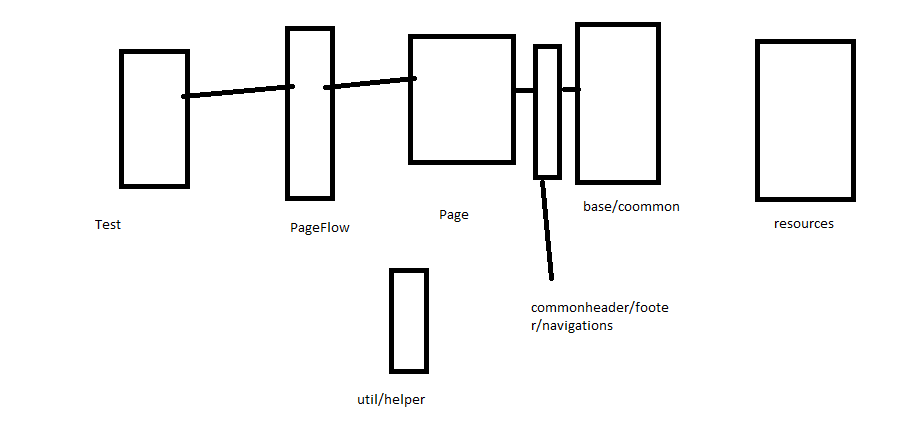
Scrolling down or up using webdriver

Click buttons, pop up some alerts while debugging, refresh the page

FileUpload in selenium:

1. We wil use AutoIt integration with selenium
2. Using Robot class(java.awt package) and selenium

Switchtoframe



Threshold

debug-> info ->warn-> error ->fatal