**Selenium Concepts**

Installing/configure selenium

* Install Java
* install Eclipse IDE
* download Selenium Java client driver
* download jar files
* Import Selenium jar files into eclipse
* Now selenium WebDriver's into Java Build Path

what is absolute path and relative path in xpath

import org.junit.Test;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

public class Paths

{

@Test

public void absolutePath() throws InterruptedException

{

System.setProperty("webdriver.chrome.driver", "E:\\Selenium\\chromedriver\_win32\\chromedriver.exe");

WebDriver driver = new ChromeDriver();

driver.get("E:\\Selenium\\absolute.html");

// Absolute Path starts from root path

WebElement link1 = driver.findElement(By.xpath("/html/body/li[@id='test']/a"));

// Relative Path starts from current path

WebElement link2 = driver.findElement(By.xpath(".//\*[@id='test']/a"));

driver.quit();

}

}

Limitations of Selenium:

1) selenium was limited to Firefox program just it implies whatever the activities we were performing in the Firefox program for those activities just our selenium IDE will set up the script on the off chance that we were playing out a few activities on IE or chrome or safari my selenium IDE wont record any script

2) we can't use conditional statements like if ,while, if else and for in selenium IDE.

Limitations of Web driver:

* Web driver supports Firefox and html (running the scripts in the invisible mode) unit driver only,
* if we want to run the script on any other browser in that cases we need to add some executable files related those browsers then only our script will execute
* it does not support and non web-based applications, it only supports web based applications.
* Its an open source tool so in case of any technical issues you need to rely on the selenium community forums to get your issue resolved.

PROS

* Selenium has no upfront, out-of-pocket costs. It is a free download and support is free too, although it is community-based.
* Selenium tests are, in principle, able to run under multiple browsers.
* Although Selenium has its own script language, you are not limited to writing in that language since it can work with language bindings to support whatever your developers/testers are comfortable with including C#, Java, JavaScript, PHP, Python and others.
* Selenium scripts are created by recording actions using the web application under test running in a browser. These scripts can be saved and re-run at any time. Selenium tests can be created manually through the use of web development tools such as Firebug also.
* Selenium does not restrict QA’s choice of reporting tools, build systems or any other aspect of their testing framework. It integrates well with popular tools such as Hudson, SauceLabs, Selenium-Grid, QMetry and others.
* It also supports web applications that implement part of their functionality within the browser using JavaScript and AJAX technologies.

Because of its many advantages, Selenium finds wide usage for UI, regression, unit and acceptance testing. Because of the rapid test development it enables, it is quite popular for quick-cycle development methodologies such as Agile or Extreme Programming. Selenium is also popular with IT staff who automate repetitive, web-based administrative tasks.

**Selenium Cons**

* Selenium is not a complete, comprehensive solution to fully automating the testing of web applications. It requires third-party frameworks, language bindings and so on to be truly effective.
* Despite its acceptance of other test script languages, it demands higher-level technical skills, such as programming, from QA team members.
* It has no test management facilities. Test scripts are saved as simple files without attributes. Organizing individual scripts in any fashion via a user interface requires a third-party tool or a custom application.
* Because native “Selenese” test scripts are not user-friendly in terms of readability, they are difficult to modify. Many testers simply resort to discarding the original scripts and recording them again, which can be time consuming.
* Selenium does not support test and result sharing in anything but a manual way.
* There is no support for running tests in parallel on a single computer.
* Selenium has technical issues with browsers other than Firefox. Furthermore, it does not support conditionals, loops and has trouble finding locators without the help of additional tools such as Firebug.

What are different ways of locating elements in selenium?

There many ways of locating elements are

* ID
* Name
* Xpath
* ClassName
* CSSselector
* LinkText
* Partial LinkText
* tagname

By ID:

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.firefox.FirefoxDriver;

public class Id

{

public static void main (String args [])

{

WebDriver driver =new FirefoxDriver();

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

driver.findElement(By.id("sb\_form\_q")).sendkeys(“sam”);

By ClassName:

driver.findElement(By.ClassName("b\_searchbox")).sendkeys(“laxmi”);

By CSS:

driver.findElement(By.CssSelector(“#sb\_from\_q”)).sendkeys(“ram”);

By Name:

driver.findElement(By.Name("q")).sendkeys(“kalyan”);

By Xpath:

driver.findElement(By.xpath(“input[@id='next']")).click();

Driver.quit();

}

}

which is fastest way to identify elements in web page?

|  |  |  |
| --- | --- | --- |
| ID | Xpath | CSS and Name |
| IDs are the fastest locator in all of the remaining locators and I should say that it can be always be your first choice | most flexible in order to build reliable web element locators | CSS is faster than XPath |
| IDs should be unique in every page | very slow locator (particularly in IE) since in order to locate the element it needs | whenever IDs are not available/ usable, usually @name can be used to identify elements |
|  |  |  |

***IDs are king!***

IDs are the safest locator option and should always be your first choice. By W3C standards, it should be unique in the page meaning you will never have a problem with finding more than one element matching the locator.

The ID is also independent of the element type and location in the tree and thus if the developer moves the element or changes its type WebDriver can still locate it.

IDs are often also used in the web page’s JavaScript so a developer will avoid changing an element’s ID to avoid having to change his JavaScript. That’s great for us testers!

If you have flexible developers or even an eye for the app source code you can always try and get extra IDs added into the code by buying them a beer on Friday evening, taking their sister on a date or just plain begging. However, sometimes adding IDs everywhere is impractical or not viable so we need to use CSS or Xpath locators.

***CSS and Xpath locators***

CSS and Xpath locators are conceptually very similar so I’ve put them together for this discussion.

These types of locators with combinations of tag name, descendant elements, CSS class or element attribute makes the matching pattern strict or loose, strict meaning that small HTML changes will invalidate it and lose meaning that it might match more than one HTML element.

When writing a CSS or Xpath locator it’s all about finding the balance between strict and loose; durable enough to work with HTML changes and strict enough to fail when the app fails.

what is desired capabilities in selenium web driver?

public java.lang.String getBrowserName()

* setBrowserName()

public void setBrowserName(java.lang.String browserName)

* getCapability Method

public java.lang.Object getCapability(java.lang.String capabilityName)

* getPlatform()

public Platform getPlatform()

* getVersion()

public java.lang.String getVersion()

* setVersion()

public void setVersion(java.lang.String version)

* getBrowserName()

how to save screen shots using selenium web driver?

public class Screenshoot

{

 @Test

public void TextseleniumS1()

{

   WebDriver driver=new FirefoxDriver();

        Screenshoot.captureScreenShot(driver);

 driver.manage().window().maximize();

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

      Screenshoot.captureScreenShot(driver);

  }

 public static void captureScreenShot(WebDriver ldriver)

{

 File src=((TakesScreenshot)ldriver).getScreenshotAs(OutputType.FILE);

 try

{

 FileUtils.copyFile(src,new File("C:/selenium/"+System.currentTimeMillis()+".png"));

} catch (IOException e)

{

  System.out.println(e.getMessage())

 }

  }

}

how to launch webpage using chrome driver?

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

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

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

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

**public** **class** ChromeDriver

{

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

{

WebDriver driver;

System.setProperty("webdriver.chrome.driver", "C:/lib/chromedriver.exe");

driver = new ChromeDriver();

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

WebElement searchText = driver.findElement(By.name("kalyan"));

searchText.sendKeys("s");

driver.quit();

}

}

How to assign the value to textbox other than sendkeys method?

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

WebElement cssValue= driver.findElement(By.xpath(".//\*[@id='s']"));

JavascriptExecutor jse = (JavascriptExecutor) driver;

jse.executeScript("document.getElementById('Sending').value=”names");

Difference between assert and verify?

Assert:

* When an "assert" command fails then test execution will be aborted.
* Assert is best used when the check value has to pass for the test to be able to continue to run. Like a log in.

Verify:

* When a "verify" command fails then test will continue executing and logging the failure.
* Verify is best used to check non critical things. Like the presence of a headline element.

Difference between driver.close() and driver.quit?

Close()- It is used to close the browser which is in current page

Quit()- It is used to shut down the web driver

public IWebDriver Driver;

[SetUp]

public void SetupTest()

{

Driver = WebDriverFactory.GetDriver();

}

[TearDown]

public void TearDown()

{

if (Driver != null)

Driver.Quit();

}

write code on how to use javascriptexecutor?

JavascriptExecutor js = (JavascriptExecutor)driver;

Js.executeScript("alert('happy')");

**How to handle Ajax call Using Selenium Webdriver**

The **biggest challenge in handling Ajax call is knowing the loading time for the web page.**Since the loading of the web page will last only for a fraction of seconds, it is difficult for the tester to test such application through automation tool. For that, Selenium Webdriver has to use the wait method on this Ajax Call.

So by executing this wait command, selenium will suspend the execution of current test case and wait for the expected or new value. When the new value or field appears, the suspended test cases will get executed by Selenium Webdriver.

Following are the wait methods that Selenium Webdriver can use

1. **Thread.Sleep()**

* Thread.Sleep () is not a wise choice as it suspends the current thread for the specified amount of time.
* In AJAX, you can never be sure about the exact wait time. So, your test will fail if the element won't show up within the wait time. Moreover, it increases the overhead because calling Thread.sleep(t) makes the current thread to be moved from the running queue to the waiting queue.
* After the time 't' reached, the current thread will move from the waiting queue to the ready queue, and then it takes some time to be picked by the CPU and be running.

1. **Implicit Wait()**

* This method tells webdriver to wait if the element is not available immediately, but this wait will be in place for the entire time the browser is open. So any search for the elements on the page could take the time the implicit wait is set for.

1. **Explicit Wait()**

* [Explicit wait](http://www.guru99.com/implicit-explicit-waits-selenium.html) is used to freeze the test execution till the time a particular condition is met or maximum time lapses.

1. **WebdriverWait**

* It can be used for any conditions. This can be achieved with WebDriverWait in combination with ExpectedCondition
* The best way to wait for an element dynamically is checking for the condition every second and continuing to the next command in the script as soon as the condition is met.

But the problem with all these waits is, you have to mention the time out unit. What if the element is still not present within the time? So there is one more wait called Fluent wait.

1. **Fluent Wait**

* This is an implementation of the Wait interface having its timeout and polling interval. Each FluentWait instance determines the maximum amount of time to wait for a condition, as well as the frequency with which to check the condition.

**Challenges in Handling Ajax Call in Selenium Webdriver**

* Using "pause" command for handling Ajax call is not completely reliable. Long pause time makes the test unacceptably slow and increases the testing time. Instead, "waitforcondition" will be more helpful in testing Ajax applications.
* It is difficult to assess the risk associated with particular Ajax applications
* Given full freedom to developers to modify Ajax application makes the testing process challenging
* Creating automated test request may be difficult for testing tools as such AJAX application often use different encoding or serialization technique to submit POST data.

**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.

Let's consider a scenario where we have to use both implicit and explicit waits in our test. Assume that implicit wait time is set to 20 seconds and explicit wait time is set to 10 seconds.

Suppose we are trying to find an element which has some **"ExpectedConditions** "(Explicit Wait), If the element is not located within the time frame defined by the Explicit wait(10 Seconds), It will use the time frame defined by implicit wait(20 seconds) before throwing an "**ElementNotVisibleException**".

**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);

### 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);

10) Common exception in selenium

<https://seleniumhq.github.io/selenium/docs/api/py/common/selenium.common.exceptions.html>

11)How to assign the value to textbox other than sendkeys method?

**Why the hell do we need this?**

There are below possible reason that I could find, When we need any alternative to sendKeys.

1. When the Text element is disabled or locked, sendKeys can not set the value in text field.  
   **(in my opinion, this is not the correct way of Automation, because the element is locked or disabled by intend to not allow any text insertion)**
2. When we want to write huge text as input… that time the way WebDriver work, by sending Series of characters from String one by one, and Which is Very time consuming. so to minimize that time we can use this alternate method

So now we have above two problems with sendKeys ……

**Solution!!!**

The best alternative I found is JavaScript

**Way 1 to Execute Script**

|  |  |
| --- | --- |
| 1  2  3  4  5 | WebDriver driver = new FirefoxDriver();  driver.get("[http://www.google.com](http://www.google.com/)");  JavascriptExecutor myExecutor = ((JavascriptExecutor) driver);  myExecutor.executeScript("document.getElementsByName('q')[0].value='Kirtesh'", searchbox);  driver.quit(); |

in above example we have used javaScript to locate the Search Textbox and set the “Kirtesh” value in it.

lets look at another approach.

**Way 2**

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | WebDriver driver = new FirefoxDriver();  driver.get("[http://www.google.com](http://www.google.com/)");  WebElement searchbox = driver.findElement(By.xpath("//input[@name='q']"));  JavascriptExecutor myExecutor = ((JavascriptExecutor) driver);  myExecutor.executeScript("arguments[0].value='Kirtesh';", searchbox);  driver.quit(); |

second example is very controlled approach, Here we are passing WebElement as a argument to the javaScript

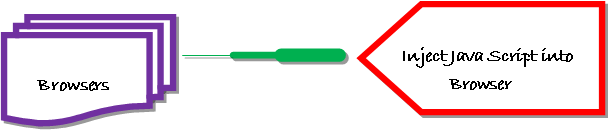
There is still many more remained from **JavascriptExecutor** ……

**What is JavaScriptExecutor**

JavaScriptExecutor is an Interface that helps to execute JavaScript through Selenium Webdriver.

JavaScriptExecutor provides two methods "executescript" & "executeAsyncScript"

to run javascript on the selected window or current page.

[](http://cdn.guru99.com/images/ccna/061516_1127_ExecuteJava1.png)

1. **executeAsyncScript**

With Asynchronous script, your page renders more quickly. Instead of forcing users to wait for a script to download before the page renders. This function will execute an asynchronous piece of JavaScript in the context of the currently selected frame or window in Selenium. The JS so executed is single-threaded with a various callback function which runs synchronously.

1. **executeScript**

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

The script can return values. Data types returned are

* Boolean
* Long
* String
* List
* WebElement.

The basic syntax for JavascriptExecutor is given below:

**Syntax:**

JavascriptExecutor js = (JavascriptExecutor) driver;

js.executeScript(Script,Arguments);

* Script – This is the JavaScript that needs to execute.
* Arguments – It is the arguments to the script. It's optional.

**write code to find out if all links are working or not**

<http://www.software-testing-tutorials-automation.com/2015/08/how-to-find-broken-linksimages-from.html>

http://toolsqa.com/selenium-webdriver/finding-broken-links-selenium-automation/

how to know if checkbox is checked or not in webpage

isChecked = e.findElement(By.tagName("input")).getAttribute("checked").equals("true");