**Author: Vaibhav Mahajan**

**-------------------------------------------------------------------------------------------------------------------------------**

**Selenium WebDriver:**

WebDriver is a tool for automating testing web applications, to verify that they work as expected. It provides API which helps to make tests easier to read and maintain. It supports various types of drivers like Firefox, Internet Explorer, Chrome, HTML Unit etc.

**Steps to install:**

1. Add jar "selenium-server-standalone-2.42.2" to the project library.

**Packages and API Used:**

Import following package:

org.openqa.selenium.WebDriver

Create WebDriver's object as:

WebDriver driver = new <WebDriver Name>();

Get connected to server by providing url in get method as:

driver.get(<url>);

**Examples:**

1. Open firefox driver:

Controls the Firefox browser using a Firefox plugin.

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.firefox.FirefoxDriver;

public class MyFirefox {

public static void main(String[] args) {

WebDriver driver = new FirefoxDriver();

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

}

}

2. Open Internet Explorer driver:

This driver is controlled by a .dll and is thus only available on Windows OS.

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.ie.InternetExplorerDriver;

public class IEDriver {

public static void main(String[] args) {

System.setProperty("webdriver.ie.driver", <path to IE Driver>);

WebDriver driver = new InternetExplorerDriver();

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

}

}

3. Open Chrome driver:

WebDriver works with Chrome through the chromedriver binary. You need to have both

chromedriver and a version of chrome browser installed. chromedriver needs to be placed

somewhere on your system’s path in order for WebDriver to automatically discover it.

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.chrome.ChromeDriver;

public class ChromeDriverDemo {

public static void main(String[] args) {

System.setProperty("webdriver.chrome.driver", <path to chrome Driver>");

WebDriver driver = new ChromeDriver();

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

}

}

4. Open HTML Unit driver:

This is currently the fastest and most lightweight implementation of WebDriver. As the name

suggests, this is based on HtmlUnit. HtmlUnit is a java based implementation of a WebBrowser

without a GUI.

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.htmlunit.HtmlUnitDriver;

public class HtmlUnitDriverDemo {

public static void main(String[] args) {

WebDriver driver = new HtmlUnitDriver();

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

}

**Finding Elements:**

**1. XPath:**

XPath is the language used for locating nodes in an XML document such as HTML. It uses path

expressions to navigate in XML document. We can use XPath to either locate the element in absolute

terms, or relative to an element that does have an id or name attribute. XPath locators can also be

used to specify elements via attributes other than id and name.

**Packages and API Used:**

Import following package:

import org.openqa.selenium.WebElement;

Get element by calling function on WebDriver object:

WebElement element = driver.find\_element\_by\_xpath(<XPath to the element>)

**Absolute xpath:**

It provides extact path to the element. This will change if page item position get change and these are

too long in many cases. So it is always better to choose Relative xpath, as it helps us to reduce the

chance of element not found exception.

**Example:**

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.firefox.FirefoxDriver;

public class MyFirefox {

public static void main(String[] args) {

WebDriver driver = new FirefoxDriver();

driver.get("http://destinationqa.com/aut/

SamplePersonalInformationForm.html");

WebElement fname = driver.findElement(By.xpath("html/body/form/

table/tbody/tr[2]/td[1]/div/input[1]"));

WebElement lname = driver.findElement(By.xpath("html/body/form/

table/tbody/tr[2]/td[1]/div/input[2]"));

lname.sendKeys("Mahajan");

}

}

**Relative Path:**

Relative xpath uses specific syntax to locate an element in HTML using some unique ids. These will

be short in nature and may not change when position of element get change.

**Example:**

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.firefox.FirefoxDriver;

public class MyFirefox {

public static void main(String[] args) {

WebDriver driver = new FirefoxDriver();

driver.get("http://destinationqa.com/aut/SamplePersonal

InformationForm\_NextIteration.html");

WebElement fname = driver.findElement(By.xpath(".//\*[@name='firstname']"));

fname.sendKeys("Vaibhav");

WebElement fname = driver.findElement(By.xpath(".//\*[@name='lastname']"));

lname.sendKeys("Mahajan");

}

}

**2. CSS Selector:**

CSS Uses expressionsions like XPath to identify elements in HTML. CSS has more advantages than

Xpath. CSS is much more faster and simpler than the Xpath.

**Packages and API Used:**

Import following package:

import org.openqa.selenium.WebElement;

Get element by calling function on WebDriver object:

WebElement element = driver.find\_element\_by\_css\_selector(<CSS Selector>)

**Example:**

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.firefox.FirefoxDriver;

public class CSSDemo {

public static void main(String[] args) {

WebDriver driver = new FirefoxDriver();

driver.get("http://destinationqa.com/aut/RadioButtons.html");

//Radio buttons

WebElement days = driver.findElement(

By.cssSelector("input[checked='checked'][type='radio']"));

System.out.println(days.getAttribute("value"));

//Check box

List<WebElement> colors = driver.findElements

(By.cssSelector("input[checked='checked'][type='checkbox']"));

for (WebElement webElement : colors) {

System.out.println(webElement.getAttribute("name"));

}

}

}

**3. Other Methods:**

WebDriver supports some methods which are useful to locate element in HTML

**A) By ID:**

This function uses input as "id" of an HTML tag. This is the most efficient and preferred way to

locate an element as ids are unique and have less chances to change.

**Syntax:**

WebElement element = driver.findElement(By.id(<id name>));

**B) By Class Name:**

This function uses input as "class" attribute of an HTML tag. “Class” in this case refers to the

attribute on the DOM element. It may return multiple elements.

**Syntax:**

List<WebElement> cheeses = driver.findElements(By.className(<class name>));

**C) By Name:**

This will find the input element with matching name attribute.

**Syntax:**

WebElement cheese = driver.findElement(By.name(<name>));

**D) By Link Text:**

This will find the link element with matching visible text.

**Syntax:**

WebElement cheese = driver.findElement(By.linkText(<link name>));

**Popup Handling:**

Any application can generate following type of popups:

1. JavaScript PopUps

2. Browser PopUps

3. Native OS PopUps

**1. JavaScript PopUps:**

Generally JavaScript popups are generated by web application and hence they can be easily controlled

by the browser. Webdriver provides Alerts API to deal with JavaScript PopUps.

**Packages and API Used:**

Import following package:

import org.openqa.selenium.Alert;

**Sysntax:**

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

Alert class provides various methods to work with popup:

A. accept()

This method will Click on "OK" or "Yes" button.

B. dismiss()

This method will Click on "Cancel" or "No" button.

C. getText()

This method will get the text which is present on the Alert.

D. sendkeys()

This method will pass the text to the prompt popup

**Example:**

import org.openqa.selenium.Alert;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.firefox.FirefoxDriver;

public class popup {

public static void main(String[] args) {

WebDriver driver = new FirefoxDriver();

driver.get("http://destinationqa.com/aut/AlertsPopups.html");

WebElement confirm = driver.findElement(By.id("btnConfirm"));

confirm.click();

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

System.out.println(alert.getText());

alert.accept();

WebElement output = driver.findElement(By.id("output"));

System.out.println(output.getText());

confirm.click();

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

System.out.println(alert.getText());

alert.dismiss();

output = driver.findElement(By.id("output"));

System.out.println(output.getText());

WebElement showAlert = driver.findElement(By.id("btnAlert"));

showAlert.click();

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

System.out.println(alert.getText());

alert.accept();

output = driver.findElement(By.id("output"));

System.out.println(output.getText());

}

}

**2. Browser PopUps:**

Selenium does not provide any kind of APIs to handle this type of browsers.There is many

alternative methods to solve this problem. One is to switch the driver to the popup window and then

running the corresponding actions. Another way is to setup browser defaults to avoid such popups

**Setting Browser defaults:**

This is mostly used where we need to download some files. For this we will create browser profile to

specific file to download without prompting to download window using firefox driver.

**Packages and API Used:**

Import following package:

import org.openqa.selenium.firefox.FirefoxProfile

**Example:**

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.firefox.FirefoxProfile;

public class download {

public static void main(String[] args) {

FirefoxProfile profile = new FirefoxProfile();

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

profile.setPreference("browser.downloard.dir",<Path to downlad file>);

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

"application/x-zip;application/x-zip-compressed;application/zip");

WebDriver driver = new FirefoxDriver(profile);

driver.get("http://docs.seleniumhq.org/download/");

WebElement download =

driver.findElement(By.xpath(".//\*[@id='mainContent']/p[5]/a"));

download.click();

}

}

**3. Native OS PopUps:**

This Popups works in same way as Browser popups. We can use Java awt Robot class to deal with

these popups. These are the special APIs in Java to handle OS events. But these APIs has many

limitations like they use coordinates to handle events, they can handle one character at single time

etc.

**Packages and API Used:**

Import following package:

import java.awt.Robot;

import java.awt.event.KeyEvent;

**Example:**

import java.awt.Robot;

import java.awt.event.KeyEvent;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.firefox.FirefoxDriver;

public class uploadfile {

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

WebDriver driver = new FirefoxDriver();

driver.get("http://www.2shared.com/");

WebElement upload = driver.findElement(By.id("upField"));

upload.click();

int x = upload.getLocation().getX();

int y = upload.getLocation().getY();

Robot myrobot = new Robot();

//myrobot.keyPress(KeyEvent.VK\_ESCAPE);

myrobot.mouseMove(x, y);

}

}

**Synchronization:**

At the time of execution of the tests, there should be fair communication between tool and application. Synchronization is handling the co-ordination of your script with the application under test.

Synchronization can be done in two ways.

1. Explicit Waits

2. Implicit Waits

**1. Explicit Waits**

This can be achieved in 2 ways.

**A) sleep method call:**

sleep method waits the specified time irrespective of the object state. It uses the default java API.

**Example:**

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;

public class autosuggesion {

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

WebDriver driver = new FirefoxDriver();

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

//CSS Path

WebElement auto = driver.findElement(By.name("q"));

auto.sendKeys("adidas");

Thread.sleep(3000); // Explicitly wait for 3 seconds

WebElement items = auto.findElement(By.name("list\_?"));

System.out.println(items);

}

}

**B) WebDriverWait:**

We can tell the tool to wait only till the condition met. Once the condition is met, the tool proceed

with the next step.

**Packages and API Used:**

Import following package:

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

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

**Example:**

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.Actions;

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

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

public class myExplicitWait {

static By myElement;

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

WebDriver driver = new FirefoxDriver();

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

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

WebElement auto = driver.findElement(By.name("\_nkw"));

Actions actions = new Actions(driver);

actions.sendKeys(auto, "adidas").build().perform();

WebElement suggessions = explicitelyWait(driver, By.id("gAC"), 10);

System.out.println(suggessions.getText());

}

public static WebElement explicitelyWait(WebDriver driver, By by, long timeOut)

throws InterruptedException {

myElement = by;

WebDriverWait myWait = new WebDriverWait(driver, timeOut);

ExpectedCondition<WebElement> condition = new

ExpectedCondition<WebElement>() {

@Override

public WebElement apply(WebDriver d) {

System.out.println("Inside Explicit Wait ..");

if(! d.findElement(myElement).getText().isEmpty())

return d.findElement(myElement);

return null;

}

};

return myWait.until(condition);

}

}

**2. Implicit Waits:**

An implicit wait is to tell WebDriver to poll the DOM for a certain amount of time when trying to find

an element or elements if they are not immediately available. The default setting is 0. Once set, the

Implicit wait is set for the life of the WebDriver object instance.

**Example:**

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;

public class autosuggesion {

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

WebDriver driver = new FirefoxDriver();

driver.manage().timeouts().implicitlyWait(5, TimeUnit.SECONDS);

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

//CSS Path

WebElement auto = driver.findElement(By.name("q"));

auto.sendKeys("adidas");

WebElement items = auto.findElement(By.name("list\_?"));

System.out.println(items);

}

}

**TestNG**

TestNG is similar to JUnit, but it’s not a JUnit extension. It’s inspired by JUnit. It is designed to be better than JUnit, especially when testing integrated classes. The creator of TestNG is Cedric Beust. TestNG eliminates most of the limitations of the older framework and gives the developer the ability to write more flexible and powerful tests. As it heavily borrows from Java Annotations to define tests, it can also show you how to use this new feature of the Java language in a real production environment.

**Features:**

1. Annotations.

2. TestNG uses more Java and OO features.

3. Supports testing integrated classes (e.g., by default, no need to create a new test class instance for

every test method).

4. Separate compile-time test code from run-time configuration/data info.

5. Flexible runtime configuration.

6. Introduces ‘test groups’. Once you have compiled your tests, you can just ask TestNG to run all the

"front-end" tests, or "fast", "slow", "database", etc...

7. Supports Dependent test methods, parallel testing, load testing and partial failure.

8. Flexible plug-in API.

9. Support for multi threaded testing.

**Annoatations:**

Here is the list of some annotations that TestNG supports:

1. @BeforeSuite:

The annotated method will be run only once before all tests in this suite have run.

2. @AfterSuite:

The annotated method will be run only once after all tests in this suite have run.

3. @BeforeClass:

The annotated method will be run only once before the first test method in the current class is

invoked.

4. @AfterClass:

The annotated method will be run only once after all the test methods in the current class have been

run.

5. @BeforeTest

The annotated method will be run before any test method belonging to the classes inside the <test>

tag is run.

6. @AfterTest

The annotated method will be run after all the test methods belonging to the classes inside the <test>

tag have run.

7. @BeforeGroups

The list of groups that this configuration method will run before. This method is guaranteed to run

shortly before the first test method that belongs to any of these groups is invoked.

8. @AfterGroups

The list of groups that this configuration method will run after. This method is guaranteed to run

shortly after the last test method that belongs to any of these groups is invoked.

9. @BeforeMethod

The annotated method will be run before each test method.

10. @AfterMethod

The annotated method will be run after each test method.

11. @DataProvider

Marks a method as supplying data for a test method. The annotated method must return an Object[ \

][ ] where each Object[ ] can be assigned the parameter list of the test method. The @Test method

that wants to receive data from this DataProvider needs to use a dataProvider name equals to the

name of this annotation.

12. @Factory

Marks a method as a factory that returns objects that will be used by TestNG as Test classes. The

method must return Object[ ].

13. @Listeners

Defines listeners on a test class.

14. @Parameters

Describes how to pass parameters to a @Test method.

15. @Test

Marks a class or a method as part of the test.

**Benefits of using Annotations:**

Following are some of the benefits of using annotations:

1. TestNG identifies the methods it is interested in by looking up annotations. Hence, method names are

not restricted to any pattern or format.

2. We can pass additional parameters to annotations.

3. Annotations are strongly typed, so the compiler will flag any mistakes right away.

4. Test classes no longer need to extend anything (such as TestCase, for JUnit 3).

**Steps to run TestNG Code:**

1. Install following jar:

testng-6.8.jar

**Packages and API Used:**

Import following package:

import org.testng.annotations.\*

2. Create testng.xml file:

TestNG framework uses testng.xml file to run the code. Details of the testng.xml file are as below:

A) Suite is represented by one XML file. It can contain one or more tests and is defined by the

<suite> tag.

B) Tag <test> represents one test and can contain one or more TestNG classes.

C) <class> tag represents a TestNG class is a Java class that contains at least one TestNG annotation. It

can contain one or more test methods.

3. Use following cmd to run code:

java -cp "C:\TestNG\_WORKSPACE" org.testng.TestNG testng.xml

**Example:**

**testng.xml:**

<?xml version="1.0" encoding="UTF-8"?>

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

<suite name="Default suite">

<test verbose="2" name="Default test">

<classes>

<class name="mypackage.MyTestNG\_01"/>

</classes>

</test> <!-- Default test -->

</suite> <!-- Default suite -->

**Code:**

import org.testng.annotations.AfterMethod;

import org.testng.annotations.AfterTest;

import org.testng.annotations.BeforeMethod;

import org.testng.annotations.BeforeTest;

import org.testng.annotations.Test;

public class MyTestNG\_01 {

@BeforeTest

public void configureSettings(){

System.out.println("IN BEFORE TEST");

}

@BeforeMethod

public void startBrowser(){

System.out.println("IN BEFORE METHOD");

}

@Test

public void MyFirstTestCase(){

System.out.println("MY FIRST TESTNG TESTCASE");

}

@Test

public void MySecondTestCase(){

System.out.println("MY SECOND TESTCASE");

}

@AfterMethod

public void closeBrowser(){

System.out.println("IN AFTER METHOD");

}

@AfterTest

public void tesardown(){

System.out.println("IN AFTER TEST");

}

}

**Passing parameters to Test Case:**

Sometimes there is a need to send parameters (like browser name, browser version ..etc). We may need to run the same test case with different values for same attribute. We can achieve above cases by using @parameter annotation in testng. Below is the example for running same test cases in different browsers (firefox, chrome) by passing different parameters (browser name, version, profile). All these parameters are passed through the testing.xml file.

**Example:**

**testing.xml:**

<?xml version="1.0" encoding="UTF-8"?>

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

<suite thread-count="2" name=MyTestSuite" parallel="tests">

<test name="RunInFirefox" preserve-order="false">

<parameter name="browser" value="firefox">

<parameter name="version" value="8"/>

<parameter name="profile" value="default">

<classes preserve-order="true">

<class name="com.test.TestCase1"/>

<class name="com.test.TestCase2"/>

<class name="com.test.TestCase3"/>

</classes>

</test>

<test name="RunInChrome" preserve-order="false">

<parameter name="browser" value="chrome">

<parameter name="version" value="21"/>

<parameter name="profile" value="default">

<classes preserve-order="true">

<class name="com.test.TestCase1"/>

<class name="com.test.TestCase2"/>

<class name="com.test.TestCase3"/>

</classes>

</test>

</suite>

**Code:**

import org.openqa.selenium.WebDriver;

import org.testng.annotations.Parameters;

import org.testng.annotations.Test;

import org.testng.annotations.BeforeMethod;

import org.testng.annotations.AfterMethod;

import org.testng.annotations.BeforeClass;

import org.testng.annotations.AfterClass;

import org.testng.annotations.BeforeTest;

import org.testng.annotations.AfterTest;

public class ExampleTestCase

{

private static WebDriver driver;

@Parameters({"browser,version"})

@BeforeClass

public void beforeClass(String browser,Stringversion,String profile) {

driver=getDriverInstance(browser,version,profile);

}

@BeforeTest

public void beforeTest() {

System.out.println("beforeTest");

}

@Test

public void testFun() {

System.out.println("testFun");

}

@AfterTest

public void afterTest() {

System.out.println("afterTest");

}

@AfterClass

public void afterClass() {

System.out.println("afterClass");

driver.quit();

}

}

**Data Provider:**

We can provide the data to the test method, by using the @Test annotation's dataProvider parameter.

**Steps:**

1. Create method returning Object[][]. Use business logic to return data to the test method.

2. Use @DataProvider annotaion for method.

3. Use this method as parameter for @Test annotaion.

4. Create Test method with String as parameter to use input data in method.

**Example:**

**Testing.xml**

<?xml version="1.0" encoding="UTF-8"?>

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

<suite name="Default suite">

<test verbose="2" name="Default test">

<classes>

<class name="mypackage. dataprovider "/>

</classes>

</test> <!-- Default test -->

</suite>

<!-- Default suite -->

**Code:**

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.testng.Assert;

import org.testng.Reporter;

import org.testng.annotations.AfterMethod;

import org.testng.annotations.AfterTest;

import org.testng.annotations.BeforeMethod;

import org.testng.annotations.BeforeTest;

import org.testng.annotations.DataProvider;

import org.testng.annotations.Test;

public class dataprovider {

private static WebDriver driver = null;

@BeforeTest

public void configureSettings(){

System.out.println("IN BEFORE TEST");

driver = new FirefoxDriver();

}

@BeforeMethod

public void startBrowser(){

System.out.println("IN BEFORE METHOD");

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

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

}

@Test(dataProvider="getSearchString")

public void MyFirstTestCase(String searchString){

System.out.println("MY FIRST TESTNG TESTCASE");

WebElement search = driver.findElement(By.id("gbqfq"));

search.sendKeys(searchString);

WebElement result = driver.findElement(By.className("gssb\_m"));

Reporter.log(result.getText(), true);

Assert.assertTrue(result.getText().contains(searchString));

}

@AfterMethod

public void closeBrowser() throws InterruptedException{

System.out.println("IN AFTER METHOD");

Thread.sleep(3000);

}

@AfterTest

public void tesardown(){

System.out.println("IN AFTER TEST");

driver.close();

}

@DataProvider

public Object[][] getSearchString(){

Object[][] searchStrigs = new Object[3][1];

searchStrigs[0][0] = "adidas";

searchStrigs[1][0] = "nike";

searchStrigs[2][0] = "woodland";

return searchStrigs;

}

}

**Selenium Grid:**

Selenium Grid transparently distributes our tests across multiple physical or virtual machines so that we can run them in parallel, cutting down the time required for running tests. This dramatically speeds up testing, Selenium Grid allows us to run multiple instances of WebDriver or Selenium Remote Control in parallel. The selenium-server-standalone package includes the Hub, WebDriver, and Selenium RC needed to run the grid. To set up the Selenium Grid we use Server as a hub and the slave machines, also called as node. By default Hub provides 5 sessions on the server.

**Advantages of Grid:**

1. Scale by distributing tests on several machines that is parallel execution.

2. Manage multiple environments from a central point, making it easy to run the tests against a various

combination of browsers / OS.

3. Minimizes the maintenance time for the grid by allowing you to implement custom hooks to leverage

virtual infrastructure for instance.

**How Grid works:**

A grid consists of a single hub, and one or more nodes. Both are started using the selenium-server.jar executable. The hub receives a test to be executed along with information on which browser and ‘platform’ (i.e. WINDOWS, LINUX, etc) where the test should be run. It ‘knows’ the configuration of each node that has been ‘registered’ to the hub. Using this information it selects an available node that has the requested browser-platform combination. Once a node has been selected, Selenium commands initiated by the test are sent to the hub, which passes them to the node assigned to that test. The node runs the browser, and executes the Selenium commands within that browser against the application under test.

**Packages and API Used:**

Import following package:

import java.net.URL

import org.openqa.selenium.remote.DesiredCapabilities

**Steps to selenium grid**

1. Install following jar:

selenium-server-standalone-2.38.0.jar

2. Setting up a Hub:

java -jar selenium-server-standalone-2.25.0.jar -port 4444 -role hub -nodeTimeout 600

3. Start and register a node to the Hub:

java -jar selenium-server-standalone-2.25.0.jar -role webdriver -browser browserName=internet

explorer,version=8,maxinstance=1,platform=WINDOWS" -hubHost localhost – port 8989

**Example:**

**Testing.xml**

<suite name="SeleniumGridExample" parallel="tests" thread-count="10" verbose="10">

<test name="myTest1" parallel="methods">

<parameter name="browser" value="internet explorer"/>

<classes>

<class name=

"com.destinationqa.grid.Grid\_differentTests\_differentBrowsers"></class>

</classes>

</test> <!-- Test -->

<test name="myTest2" parallel="methods">

<parameter name="browser" value="chrome"/>

<classes>

<class name=

"com.destinationqa.grid.Grid\_differentTests\_differentBrowsers"></class>

</classes>

</test> <!-- Test -->

<test name="myTest3" parallel="methods">

<parameter name="browser" value="firefox"/>

<classes>

<class name="com.destinationqa.grid.Grid\_differentTests\_differentBrowsers"></class>

</classes>

</test> <!-- Test -->

</suite> <!-- Suite -->

**Code:**

import java.net.MalformedURLException;

import java.net.URL;

import org.openqa.selenium.Capabilities;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.remote.DesiredCapabilities;

import org.openqa.selenium.remote.RemoteWebDriver;

import org.testng.annotations.Parameters;

import org.testng.annotations.Test;

public class Grid\_differentTests\_differentBrowsers {

@Parameters({ "browser" })

@Test

public void test1(String browsername) throws

MalformedURLException, InterruptedException {

WebDriver driver;

DesiredCapabilities capabilities = new DesiredCapabilities();

capabilities.setBrowserName(browsername);

System.setProperty("webdriver.ie.driver",<Path to IE Driver>);

System.setProperty("webdriver.chrome.driver",<Path to Chrome Driver>);

driver = new RemoteWebDriver(

new URL("http://localhost:4444/wd/hub"),

capabilities);

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

Thread.sleep(10000);

Capabilities actualCapabilities = ((RemoteWebDriver) driver).getCapabilities();

System.out.println("test1: "+driver.getTitle()+

" :" + actualCapabilities.getBrowserName());

driver.quit();

}

@Parameters({ "browser" })

@Test

public void test2(String browsername) throws MalformedURLException {

WebDriver driver;

DesiredCapabilities capabilities = new DesiredCapabilities();

capabilities.setBrowserName(browsername);

driver = new RemoteWebDriver(

new URL("http://localhost:4444/wd/hub"),

capabilities);

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

Capabilities actualCapabilities = ((RemoteWebDriver) driver).getCapabilities();

System.out.println("test2: "+driver.getTitle()+

" :" + actualCapabilities.getBrowserName());

driver.quit();

}

@Parameters({ "browser" })

@Test

public void test3(String browsername) throws MalformedURLException {

WebDriver driver;

DesiredCapabilities capabilities = new DesiredCapabilities();

capabilities.setBrowserName(browsername);

driver = new RemoteWebDriver(

new URL("http://localhost:4444/wd/hub"),

capabilities);

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

Capabilities actualCapabilities = ((RemoteWebDriver) driver).getCapabilities();

System.out.println("test3: "+driver.getTitle()+

" :" + actualCapabilities.getBrowserName());

driver.quit();

}

}