Table of Contents

Introduction

This book is a tutorial on how to use selenium API for web application automation testing. It's not an in-depth book on java, testing or selenium, rather it focusses on how get up and running with Selenium and testNG. We will also briefly cover JUnit. By the end of the book we will have a fair understanding of below technologies.

1. HTML
2. CSS
3. Javascript
4. Eclipse
5. JAVA, Selenium API, testNG
6. ANT

The basics

Bare minimum necessities to get us started with programming.

Tools required

Mandatory tools

- Firefox Browser
- Java Development Kit (JDK)
- Eclipse IDE
- TestNG Jar File
- Below Listed .jar files from Selenium Official Website
    1. Selenium Server
    2. Selenium Client Drivers (JAVA)

Required Later

- Firebug firefox plugin
- Firepath firefox plugin
- Below Listed drivers and plugins from Selenium Official Website
    1. Selenium IDE (Firefox plugin) Download in firefox and it will automatically get installed. Click on the version number to download
    2. The Internet Explorer Driver Server
    3. Third party browser drivers (Chrome)

- Apache ANT

Eclipse project setup

We assume in this tutorial that all the required tools are downloaded to your system. In order to setup your automation project in eclipse please follow the below steps.

Step 1:

Open your eclipse IDE and create a JAVA Project (File > New > Java Project) if the java Project option doesn't appear click on other.. and search for java project. Give a name to the project and click Finish.

Step 2:

Create a folder with name "lib" (Right click project Name > New > Folder).

Step 3:

Copy the below mentioned jar files in the lib folder. The version numbers may change for the mentioned JAR files. You can copy the files directly into the eclipse folder or else copy to the actual directory and then refresh the project.

1. selenium-server-standalone-2.29.0
2. selenium-java-2.29.0
3. testng-6.8

Step 4:

Add the copied jars to the build path: [Right click Project Name > Click Build Path > Click Configure Build Path > Select libraries tab > Click Add JARs.. > Expand the project name to reveal lib folder > select all the JARs in lib folder > Click OK > Click OK once more to close the window.

Step 5:

Create a folder named tools at the project root directory.

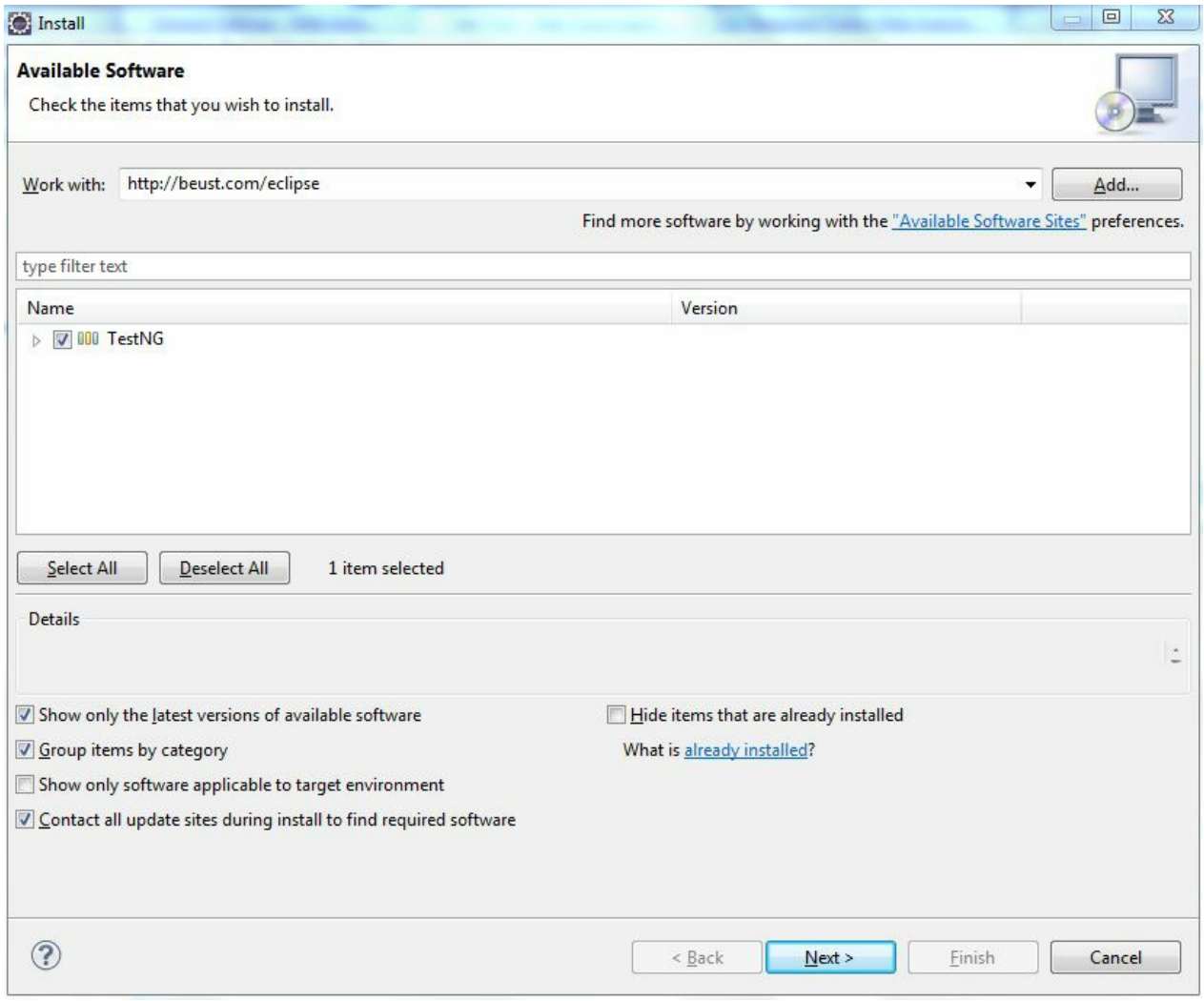
Step 6:

If downloaded copy the below mentioned executable files into the tools directory.

1. IEDriverServer
2. chromedriver

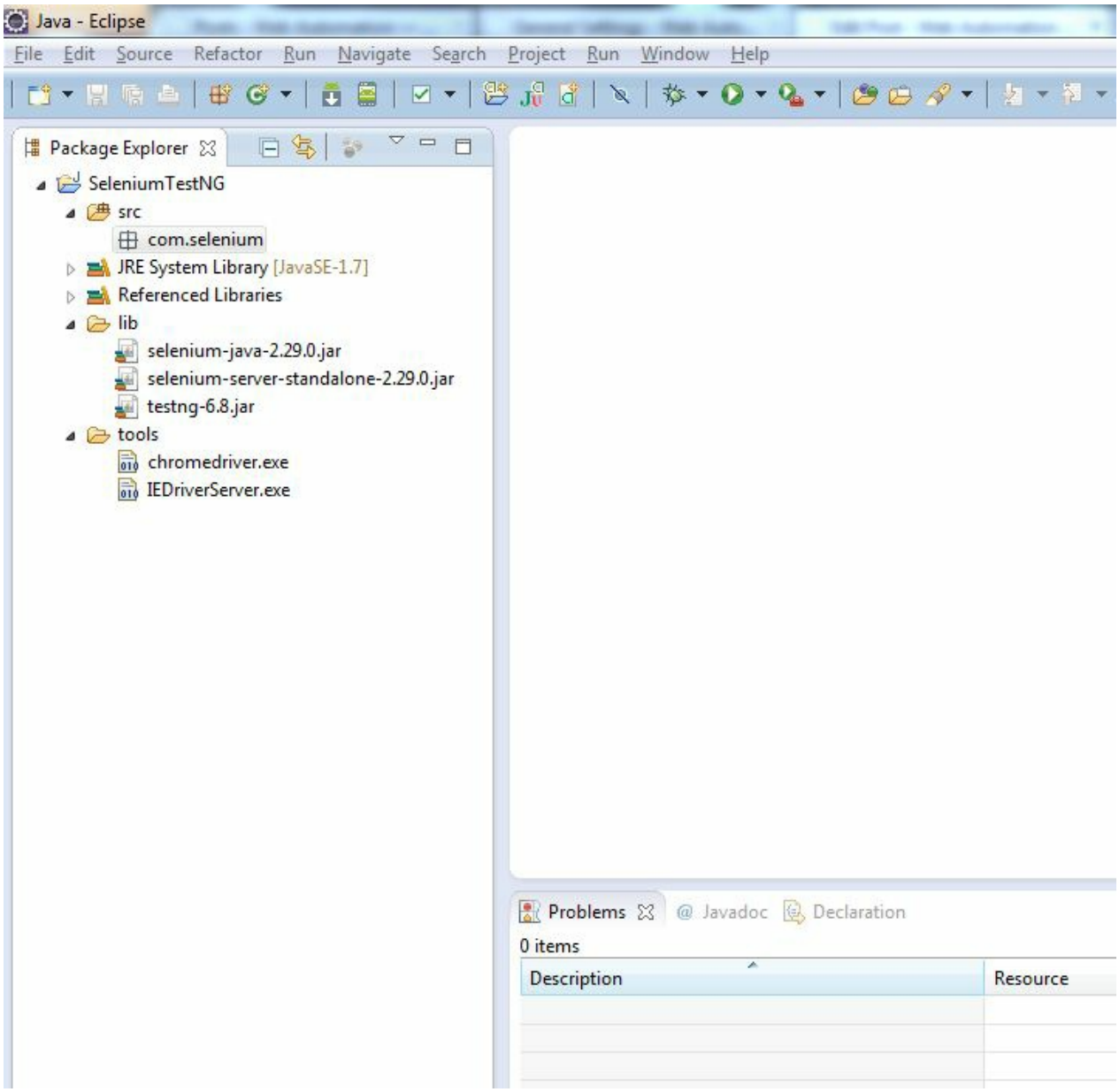
Step 7:

Download testNG plugins into eclipse: Click Help on menu bar > Install New Software > Paste this URL (http://beust.com/eclipse) in the provided textbox. > testNG should appear for download > Click Next and finish the process of Installation. Please note internet should be accessible via Eclipse to do this. In case of issues check internet settings under > Window > preferences > General > Network Connections.. Refer the below screenshot to get an idea.



Step 8:

Create a package with name “com.selenium” or any suitable name: Right click src > New > package. We will store our scripts in this package. Final View of our project must look as below.



Step 9: Once the project is setup as per the steps above, you are ready to start working with selenium scripts. Follow Simple testNG Class tutorial to get started.

Selenium IDE

Selenium IDE gives an idea of how record and playback works in an automation tool. However it's not a tool to completely rely on. Also you cannot make conditional decisions in the selenium IDE, for that purpose we need to get handson with some java programming. Selenium IDE however is very handy for beginners as it can automatically generate boilerplate code for you.

Selenium IDE Documentation

The link above redirects to Selenium IDE documentation on seleniumhq.org. The documentation is detailed and precise. Hope you enjoy it. You can enable the experimental features in Selenium IDE : Open IDE > Click Options > Click Options > Enable the checkbox with text - "Enable Experimental Features", also the textbox with text - "Disable format change warning messages".

Once experimental feature you can record a test while default html format is enabled then convert code to appropriate language by clicking Options > Format and choose the code format of your choice. You can switch back to original table view by choosing HTML as format.

Simple TestNG Class

Create a New Class Named SimpleTestNGClassExample in Eclipse. and copy paste the below code to the class. Run the Script by right Clicking anywhere in the code editor and click "Run As" > TestNG Test. Please note the option "Run As > TestNG" won't appear if you don't have testNG plugins. Follow this tutorial to install testng plugins.

```
package com.selenium;

import org.testng.annotations.Test;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.AfterTest;

public class SimpleTestNGClassExample {

   @BeforeTest
   public void initialization() {
      System.out.println("This method will be executed once before the"
            + "testMethod gets called..");
      System.out.println("All the statements like browser setup, "
            + "data-variable Initilization etc. should be done here..");
   }

   @Test
   public void testMethod() {
      System.out.println("This method will be called after the setup method"
            + "is called..");
      System.out.println("All the statements related to test execution"
            + "Should appear here..");
   }

   @AfterTest
   public void cleanup() {
      System.out
            .println("This method will be called after all previous methods"
                  + "are executed..");
      System.out.println("All the statements related to clean up activity"
            + "or closure activity should be listed here..");
   }

}
```

Testng works by looking for Annotation in the code, name @BeforeTest, @Test, @AfterTest and so on. The code in these annotations is run sequentially.

Selenium Beginner

We will discuss simple scripting techniques in this chapter.

Setting up browsers

Selenium supports many browsers, here we provide you with the code to run a few. It is assumed that you have all the necessary jars and tools. Please refer Eclipse Project Setup for information on how to setup eclipse project.

1. Firefox

```
package com.selenium;

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.testng.annotations.Test;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.AfterTest;

public class BrowserSetup {

   WebDriver driver = null;
   String baseUrl = null;

   @BeforeTest
   public void initialization() {
      // Create Firfox browser Instance..
      driver = new FirefoxDriver();
      // Define URL to be opened..
      baseUrl = "http://www.google.co.in/";
      // Define timeout, selenium will wait maximum of 30 seconds before
      // it quits while searching for an element..
      driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);
            driver.manage().deleteAllCookies(); // Delete cookies
            driver.manage().window().maximize(); // Maximize window
   }

   @Test
   public void testMethod() {
      // Open the URL in browser..
      driver.get(baseUrl);
   }

   @AfterTest
   public void cleanup() {
      // Close the browser..
      driver.quit();
   }

}
```

2. InternetExplorer (Assuming you have IEDriverServer in your tools directory)

Before Running Internet Explorer from your script please ensure that all the security settings (Tools > Internet Options > Security) are either checked or unchecked.

```
package com.selenium;

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.ie.InternetExplorerDriver;
import org.testng.annotations.Test;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.AfterTest;

public class SetupIE {

    WebDriver driver = null;
    String baseUrl = null;

    @BeforeTest
    public void initialization() {
            baseUrl = "http://www.google.co.in/";
        // The path is defined considering you have IEDriverServer_32Bit.exe
        // copied to the path "toolsIE" in your project root directory
        String path = System.getProperty("user.dir")
                + "toolsIEIEDriverServer_32Bit.exe";
        System.setProperty("webdriver.ie.driver", path);
        driver = new InternetExplorerDriver();
        driver.manage().timeouts().implicitlyWait(50, TimeUnit.SECONDS);
        driver.manage().deleteAllCookies();
        driver.manage().window().maximize();
    }

    @Test
    public void testMethod() {
        // Open the URL in browser..
        driver.get(baseUrl);
    }

    @AfterTest
    public void cleanup() {
        // Close the browser..
        driver.quit();
    }

}
```

3. Google Chrome (We assume you have chromedriver in your tools directory)

```
package com.selenium;

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.annotations.Test;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.AfterTest;

public class SetupChrome {

    WebDriver driver = null;
    String baseUrl = null;

    @BeforeTest
    public void initialization() {
            baseUrl = "http://www.google.co.in/";
        // The path is defined considering you have chromedriver.exe
        // copied to the path "toolschrome" in your project root directory
        String path = System.getProperty("user.dir")
                + "tools/chromechromedriver.exe";
        System.setProperty("webdriver.chrome.driver", path);
        driver = new ChromeDriver();
        driver.manage().timeouts().implicitlyWait(50, TimeUnit.SECONDS);
        driver.manage().deleteAllCookies();
    }

    @Test
    public void testMethod() {
        // Open the URL in browser..
        driver.get(baseUrl);
    }

    @AfterTest
    public void cleanup() {
        // Close the browser..
        driver.quit();
    }

}
```

Webdriver locators

Webdriver reads the html source code to locate the elements we need to interact with while automating. A typical HTML source code consists of several tags (div, h1, form, span etc.), the tags have several attributes (class, id, name, type etc.). Webdriver makes use of this tag-attribute structure to find the desired elements and perform actions on them. HTML code looks as below.

```
<div class="wrapper">
   <h2>Sample Form</h2>
   <form action="#">
      <input name="text" type="text" placeholder="Enter name" />
      <input name="password" type="password" placeholder="Enter password" />
      <button id="submitBtn">Submit</button>
   </form>
</div>
```

We can locate elements in following ways

```
driver.findElement(By.id("submitBtn"));
driver.findElement(By.name("username"));
driver.findElement(By.linkText("Link to google"));
driver.findElement(By.className("form"));
driver.findElement(By.cssSelector(".form input"));
driver.findElement(By.xpath("//input[@name='password']"));
```

Follow the below sequence while choosing the method of choosing element. Choose the one that appears higher in the sequence. Choose xpath only if there is a lot of decision making to be done for complex user interface.

1. id
2. name
3. class
4. linkText (if link)
5. cssSelector
6. xpath

Create a new package called code and add a class named SampleScript in it. Copy the below code into editor. Run the script and verify it works.

```
package code;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.testng.annotations.AfterTest;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.Test;

public class SampleScript {

    WebDriver driver;
    String baseUrl = "http://ajaymore.net/selenium/demo";

    @BeforeTest
    public void initialize() {
        driver = new FirefoxDriver();
        driver.get(baseUrl);
        driver.manage().window().maximize();
    }

    @Test
    public void testLocators() throws InterruptedException {
        WebElement submitButton = driver.findElement(By.id("submitBtn"));
        driver.findElement(By.name("username")).sendKeys("johnDoe");
        driver.findElement(By.cssSelector(".form .pswd")).sendKeys("ABC@xyz");
        submitButton.click();
        Thread.sleep(3000);
        driver.findElement(By.linkText("Link to google")).click();
        Thread.sleep(3000);
    }

    @AfterTest
    public void tearDown() {
        driver.quit();
    }

}
```

Multiple Elements

When we have multiple elements to work with, and wish to use few of them based on business requirement, then we need a logical loop to take those decisions. In this demo we will be demonstrating how we can make a choice based action.

Below is the snapshot of seleniumhq.org website. Using firebug and firepath I have calculated a xpath : ".//*[@id='header']//li/a" which highlights all the seleniumhq links on the header. We use this xpath to fetch all these links in our code and then click the link that matches the link text of our choice. In this demo it will be "Documentation". More on xpath in our tutorial Robust Xpath.

In the code below we are using cssSelector instead of xpath, to use xpath uncomment the xpath code and comment cssSelector code block.

```
package code;

import java.util.List;
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.annotations.AfterTest;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.Test;

public class MultipleElements {

    private WebDriver driver;
    private String baseUrl;

    @BeforeTest
    public void initialization() {
        driver = new FirefoxDriver();
        baseUrl = "http://seleniumhq.org/";
        driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);
        driver.manage().window().maximize();
```

```
    }

    @Test
    public void testMethod() throws InterruptedException {
        // Open seleniumhq.org website..
        driver.get(baseUrl);

        // Print title of the page..
        System.out.println("Title : " + driver.getTitle());

        // Locate all the Menulinks present in header..
        List<WebElement> menuLinks = driver.findElements(By
                .cssSelector("#header ul a"));

        /* Elements can also be selected by xpath */
        /*
         * List<WebElement> menuLinks = driver.findElements(By
         * .xpath(".//*[@id='header']//li/a"));
         */

        // Print the number of links present..
        System.out.println(menuLinks.size());

        /*
         * Iterate over the list and click the link that matches text
         * Documentation
         */
        for (WebElement option : menuLinks) {
            String linkName = option.getText();
            if (linkName.equals("Documentation")) {
                option.click();
                break; // Exit from the loop, this is important
            }
        }
        Thread.sleep(3000);
    }

    @AfterTest
    public void cleanup() {
        driver.quit();
    }

}
```

Assertions

Assertions are statements of the code that check whether the pages loading are as per the requirements. A failed assertion will lead to a failed script. testNG provides quite a many assertion methods of which we will use a few.

Verification are statements of the code that do the exact same things that assertions do, however a failed verification won't lead to a failed script. testNG does not provide verification methods as such but we can design our own verification methods and add them to our reusable code.

In the code example below we will visit the seleniumhq.org website and assert Title of the page and assert if the selenium logo is present, also we will verify if the text "Browser Automation" is visible on the page.

```
package code;

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.By;
import org.openqa.selenium.NoSuchElementException;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.testng.Assert;
import org.testng.annotations.Test;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.AfterTest;

public class AssertionsNVerifications {

    private WebDriver driver;
    private String baseUrl;

    @BeforeTest
    public void initialization() {
        driver = new FirefoxDriver();
        baseUrl = "http://docs.seleniumhq.org/";
        driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);
        driver.manage().window().maximize();
    }

    @Test
    public void testMethod() {
        driver.get(baseUrl + "/");

        // Assert presence of an element
        Assert.assertTrue(isElementPresent(By
                .cssSelector("img[alt=\"Selenium Logo\"]")));

        // Assert presence of page title
        Assert.assertTrue("Selenium - Web Browser Automation".equals(driver
                .getTitle()));

        // Assert absence of wrong title
        Assert.assertFalse("Selenium False Title".equals(driver.getTitle()));

        /*
         * Below we verify presence of a text.Below we are just printing out
         * true or false, based on whether text is present on the page
```

```
         * true or false, based on whether text is present on the page.
         */
        WebElement header = driver.findElement(By.id("header"));
        System.out.println("Browser Automation text presence = "
                + isTextPresent(header, "Browser Automation"));
    }

    @AfterTest
    public void cleanup() {
        driver.quit();
    }

    private boolean isElementPresent(By by) {
        try {
            driver.findElement(by);
            return true;
        } catch (NoSuchElementException e) {
            return false;
        }
    }

    private boolean isTextPresent(WebElement element, String searchText) {
        try {
            return element.getText().contains(searchText);
        } catch (NoSuchElementException e) {
            return false;
        }
    }
}
```

Waits, timeouts

The implicit wait command in the below code makes selenium wait maximum of 30 seconds before throwing an error. If the element is found in less than 30 seconds then the wait is stopped and the element is acted upon.

```
driver = new FirefoxDriver();
        baseUrl = "http://seleniumhq.org/";
        driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);
        driver.manage().window().maximize();
```

In the code below we check every 1 second if the element is present. Once the element is present, the loop breaks and the element is acted upon.

```
for (int second = 0;; second++) {
            if (second >= 60)
                Assert.fail("timeout");
            try {
                if (isElementPresent(By.linkText("Download")))
                    break;
            } catch (Exception e) {
            }
            Thread.sleep(1000);
        }
        driver.findElement(By.linkText("Download")).click();
```

In the code below we define the wait time and wait for the element to meet the condition like being Clickable.

```
WebDriverWait wait = new WebDriverWait(driver, 10);
        WebElement element = wait.until(ExpectedConditions
                .elementToBeClickable(By.linkText("Documentation")));
        element.click();
```

We can also use explicit waits using Thread.sleep method. In the code below we halt the code execution for 5 seconds.

```
Thread.sleep(5000);
```

Test Frameworks

Testing framework is necessary to organize our tests. When the count of automated test increases it becomes very tediuos to manage them. Test frameworks provide us with below features.

1. Setting up data before tests run
2. Cleaning up data after tests are finished
3. Creating suites to run multiple tests
4. Externalize configurations from actual scripts
5. Integration with CI server

We will be using TestNG as our framework of choice, however we will also briefly cover a Junit Test.

Junit

TestNG improves upon some of the drawbacks in Junit, which we will discuss later. Below is a sample Junit test.

```
package code;

import junit.framework.TestCase;

public class Junit extends TestCase {

   protected int value1, value2;

   // assigning the values
   protected void setUp() {
      value1 = 6;
      value2 = 2;
   }

   // test method to add two values
   public void testAdd() {
      Calculator calc = new Calculator();
      assertTrue(8 == calc.add(value1, value2));
      assertTrue(4 == calc.subtract(value1, value2));
      assertTrue(12 == calc.multiply(value1, value2));
      assertFalse(20 == calc.divide(value1, value2));
   }

}

class Calculator {

   public int add(int num1, int num2) {
      return num1 + num2;
   }

   public int subtract(int num1, int num2) {
      return num1 - num2;
   }

   public int multiply(int num1, int num2) {
      return num1 * num2;
   }

   public int divide(int num1, int num2) {
      return num1 / num2;
   }
}
```

Revisiting TestNG Class

We used few of the testNG annotations in the tutorial Simple testNG Class. In this tutorial we introduce you to the other set of annotations that can be utilized in your Test Scripts. Even if you have a single annotation like @Test even then testNG will work. So it's up to you to decide which ones serve your testing purpose and choose accordingly.

Here is sample class with more annotations.

```
package testng;

import org.testng.annotations.AfterClass;
import org.testng.annotations.AfterMethod;
import org.testng.annotations.AfterSuite;
import org.testng.annotations.AfterTest;
import org.testng.annotations.BeforeClass;
import org.testng.annotations.BeforeMethod;
import org.testng.annotations.BeforeSuite;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.Test;

public class TestNG_Skeleton {

   @BeforeSuite
   public void beforeSuiteActivity() {
      System.out
            .println("1. Things to be done before even the test suite is read");
   }

   @BeforeTest
   public void setup() {
      System.out.println("2. Browser setup here..");
   }

   @BeforeClass
   public void beforeClassActivity() {
      System.out
            .println("3. Things to be done before class is instantiated..");
   }

   @BeforeMethod
   public void testDataSetup() {
      System.out
            .println("4. Read testData from property files, Excel files here..");
   }

   @Test
   public void myTestCase() {
      System.out.println("5. All the test steps here..");
   }

   @AfterMethod
   public void cleanUpPostTestExecution() {
      System.out.println("6. Cleanup actions post test execution here..");
   }

   @AfterClass
   public void afterClassActivity() {
      System.out.println("7. Things to be done after class is executed");
   }
```

```
    }

    @AfterTest
    public void teardown() {
        System.out
                .println("8. Browser close and other closing activities to finish the Test execution here..");
    }

    @AfterSuite
    public void afterSuiteActivity() {
        System.out.println("9. Things to be done after Suite is executed..");
    }
}
```

Dataprovider

Sometimes we may need to send a complex set of data to the test. Also we may wish to repeat our test for a different set of data. Dataprovider annotation comes to our rescue in such cases. In the below test we pass two sets of users and cars two the test, which in turn runs twice according to the provided data.

```
package code;

import org.testng.annotations.DataProvider;
import org.testng.annotations.Test;

public class DataProviderTest {

   @Test(dataProvider = "test1")
   public void testDataProvider(User user, Car car) {
      System.out.println(user.getName() + " " + user.getEmail());
      System.out.println(car.getName() + " " + car.getColor());
   }

   @DataProvider(name = "test1")
   public Object[][] createData1() {
      return new Object[][] {
            { new User("Cedric", "cedric@gm.co"),
                  new Car("mercedes", "gray") },
            { new User("Anne", "anne@gm.co"), new Car("porsche", "red") } };
   }
}

class User {
   private String name;
   private String email;

   public User(String name, String email) {
      this.name = name;
      this.email = email;
   }

   public String getName() {
      return name;
   }

   public String getEmail() {
      return email;
   }
}

class Car {
   private String name;
   private String color;

   public Car(String name, String color) {
      this.name = name;
      this.color = color;
   }

   public String getName() {
      return name;
   }

   public String getColor() {
      return color;
   }
}
```

TestNG over Junit

testNG has some obvious advantages over Junit. The link provided here is a detailed analysis of how testNG scores over Junit. testNG is very easy to understand and addresses some of the key Junit issues. Please click the link Why TestNG? for details.

TestSuite

A sample TestSuite is provide below. The Suite contains two Tests, The tests are defined under tag, The Class name are provided with packageName.ClassName e.g. for Class SampleTest located under package com.selenium the Class name will be com.selenium.SampleTest. Then you can declare all the methods with @Test annotation, under tag.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<suite name="Test_Suite" verbose="-1" parallel="tests"
   thread-count="1">
   <test name="First-Test" preserve-order="True">
      <classes>
         <class name="myPackage.Test1">
            <methods>
               <include name="testMethod" />
            </methods>
         </class>
      </classes>
   </test>
   <test name="Second-Test" preserve-order="True">
      <classes>
         <class name="myPackage.Test2">
            <methods>
               <include name="testMethod" />
            </methods>
         </class>
      </classes>
   </test>
</suite>
```

Selenium Advanced

This section offers various possibilities that selenium provides, having said that I recommend visiting official documentation for more update

PROXIES, PROFILES, CERTIFICATES

This tutorial will teach how to create a firefox profile. In order to use various add-ons installed on your native browser, we will be required to create a custom profile. Follow the below steps to create a profile.

1. Create a folder with name “tools” at the root of your project. Now create a folder with name “firefox” in the same folder.

2. Open the mozilla Installation path in your computer – In my case it’s “C:Program FilesMozilla Firefox”. Copy the entire contents of this folder into the folder created at step 1. – The firefox folder under tools folder.

3. Create a folder with name “DefaultProfile” in the firefox folder. This folder already has the contents you copied from your mozilla installation directory.

4. Now go to firefox profile directory on your machine. In my case it’s “C:UsersmysticAppDataRoamingMozillaFirefoxProfileswp8ks7au.default”. – copy the contents of this folder to the DefaultProfile folder created in step 3.

Now there are two ways of setting proxy, either we set the proxy using ip and port number. or we set by the URL. The code snippet shows how to achieve the same using both the methods. The code snippet for setting proxy via URL is commented. Uncomment this code if you wish to use it and comment the other part.

Finally the code to accept security certificates is also mentioned in the snippet.

```
package code;

import java.io.File;
import java.util.concurrent.TimeUnit;

import org.openqa.selenium.Proxy;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxBinary;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.firefox.FirefoxProfile;
import org.openqa.selenium.remote.CapabilityType;
import org.openqa.selenium.remote.DesiredCapabilities;
import org.testng.annotations.AfterTest;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.Test;

public class ProxiesCertificates {
    WebDriver driver;
    String baseUrl;

    @BeforeTest
    public void setup() {

        // The below proxy is a string that contains proxy ip and port
        // The port and ip is separated by ":" below you can see port is 80
        Proxy proxy = new Proxy();
        String proxyIp = "10.76.122.32:80";

        // Below lines sets the proxy defiend above
        proxy.setHttpProxy(proxyIp);
        proxy.setSslProxy(proxyIp);
```

```
        proxy.setSslProxy(proxyIp);
        proxy.setFtpProxy(proxyIp);

        // In case you wish to set proxy with URL uncomment below code. and
        // comment the above 3 lines

        /*
         * proxy.setProxyType(Proxy.ProxyType.PAC);
         * proxy.setProxyAutoconfigUrl("http://someServer/server.pac");
         */

        // Below code create capability with mentioned proxy
        DesiredCapabilities capabilities = new DesiredCapabilities();
        capabilities.setCapability(CapabilityType.PROXY, proxy);

        // Path to the copy of firefox installation. Here you can provide direct
        // path to your firefox installation also.
        System.setProperty("webdriver.firefox.bin",
                System.getProperty("user.dir")
                        + "toolsfirefoxfirefox.exe");

        // This is the path to the firefox profile you copied in your project.
        FirefoxProfile fp = new FirefoxProfile(new File(
                System.getProperty("user.dir")
                        + "toolsfirefoxDefaultProfile"));

        // Below code accepts security certificates..
        fp.setPreference("setAcceptUntrustedCertificates", "true");
        fp.setAcceptUntrustedCertificates(true);
        fp.setAssumeUntrustedCertificateIssuer(true);

        // Create browser instance with above mentioned requsites
        driver = new FirefoxDriver(new FirefoxBinary(), fp, capabilities);
        driver.manage().timeouts().implicitlyWait(50, TimeUnit.SECONDS);
        driver.manage().deleteAllCookies();
        driver.manage().window().maximize();
    }

    @Test
    public void testMethod() {
        baseUrl = "http://docs.seleniumhq.org/";
        driver.get(baseUrl);
    }

    @AfterTest
    public void teardown() {
        driver.quit();
    }

}
```

Handling multiple windows

Below is the code snippet to interact with multiple windows using selenium webdriver.

```
@Test
   public void testMethod() {

      WebDriver driver = new FirefoxDriver();
      String newwindowtitle = null;
      // URLs to be verified..
      String expectedresult[] = { "W3Schools Online Web Tutorials", "Google",
            "W3Schools Online Web Tutorials",
            "Sparsh – The Infosys Intranet", "Delicious" };

      int count = 0;
      // We are fetching all the values in dropdown below and clicking one by
      // one
      List<WebElement> elements = driver.findElements(By
            .xpath("/someXpath/to/dropdownLinks"));

      Iterator<WebElement> it = elements.iterator();
      while (it.hasNext()) {
         // Get the title of current window..
         String parentWindowHandle = driver.getWindowHandle();
         System.out.println(parentWindowHandle);
         // Now we are clicking the next link available in dropdown
         WebElement dropdownlinks = it.next();
         if (dropdownlinks.isDisplayed()) {
            dropdownlinks.click();
         }

         // Here we switch to newly opened window and check if it is correct
         // window
         Set<String> s = driver.getWindowHandles();
         System.out.println(s.size());
         Iterator<String> ite = s.iterator();
         while (ite.hasNext()) {
            String popuphandle = ite.next().toString();
            if (!popuphandle.equals(parentWindowHandle)) {
               driver.switchTo().window(popuphandle);
               newwindowtitle = driver.getTitle();
               System.out.println(newwindowtitle);
               Assert.assertEquals(newwindowtitle, expectedresult[count]);
               count++;
               // Close the new window.
               driver.close();
            }
         }
         // Switch back to the window from where we arrived at new window
         driver.switchTo().window(parentWindowHandle);
      }
   }
```

Webdriver actions

Various webpage actions like mouseHover, sliding element etc. can be performed using Webdriver.

Mouse Hover first approach

```
@Test
    public void webDriverActions() {
        driver.get("http://someURl.com");
        // Get element to be hovered upon
        WebElement hoverElement = driver.findElement(By
                .xpath("//div[@class='hoverElement']"));

        // Perform hover action
        Actions builder = new Actions(driver);
        builder.moveToElement(hoverElement).build().perform();

        // Click the element that is now visible after hover
        driver.findElement(By.xpath("//div[@class='visibleElementOnHover']"))
                .click();
    }
```

Mouse Hover second approach

```
@Test
    public void webDriverMouseActions() {
        driver.get("http://someURl.com");
        // Get element to be hovered upon
        WebElement hoverElement = driver.findElement(By
                .xpath("//div[@class='hoverElement']"));

        // Perform hover action using Mouse object
        Locatable hoverItem = (Locatable) hoverElement;
        Mouse mouse = ((HasInputDevices) driver).getMouse();
        mouse.mouseMove(hoverItem.getCoordinates());

        // Click the element that is now visible after hover
        driver.findElement(By.xpath("//div[@class='visibleElementOnHover']"))
                .click();
    }
```

Move Slider to an offset

```
@Test
    public void dragSlider() {
        driver.get("http://someURl.com");
        // Get sliding element
        WebElement sliderElement = driver.findElement(By.id("sliderId"));
        // drag the slider
        Actions builder = new Actions(driver);
        builder = new Actions(driver);
        // Move 120px on x axis
        Action dragAndDrop = builder.clickAndHold(sliderElement)
                .moveByOffset(120, 0).release().build();
        dragAndDrop.perform();
    }
```

Generate logs

If you have more than 30-40 tests to run in a suite, then it will be very difficult to track which script is being executed at what time. Also error warnings, time elapsed, execution status etc. cannot be tracked. In order to overcome these limitations we introduce logging functionality to our automation framework.

You need to download log4j jar from Apache log4j 1.2 – Download Apache log4j 1.2. Place the log4j jar in your lib folder and add the same to build path. create a package with name testListener in your src folder. Create a class with name TestMonitor in this package and copy the below code to this class.

```
package testListener;

package testListener;

import java.io.PrintWriter;
import java.io.StringWriter;

import org.apache.log4j.Logger;
import org.apache.log4j.PropertyConfigurator;
import org.testng.ITestContext;
import org.testng.ITestResult;

public class TestMonitor implements org.testng.ITestListener {

    public Logger log = Logger.getLogger(this.getClass().getName());

    @Override
    public void onFinish(ITestContext arg0) {
        log.info("Test " + arg0.getName() + " Ends");
        log.info("-------------------------------------------------------");
    }

    @Override
    public void onStart(ITestContext arg0) {
        PropertyConfigurator.configure("src/log4j.properties");
        log.info("Test " + arg0.getName() + " Starts");
    }

    @Override
    public void onTestFailedButWithinSuccessPercentage(ITestResult arg0) {
        log.info("Test failed within success Percentage");
    }

    @Override
    public void onTestFailure(ITestResult arg0) {
        log.info("Test run has failed");
        log.error(arg0.getThrowable().getMessage());

        StringWriter sw = new StringWriter();
        arg0.getThrowable().printStackTrace(new PrintWriter(sw));
        String stacktrace = sw.toString();
        log.error(stacktrace.trim());
    }

    @Override
    public void onTestSkipped(ITestResult arg0) {
        log.info("Test skipped");
    }

    @Override
    public void onTestStart(ITestResult arg0) {
        log.info("Test Method " + arg0.getMethod().getMethodName()
                + " executing...");
    }

    @Override
    public void onTestSuccess(ITestResult arg0) {
        log.info("Test run is successful");
    }

}
```

Now create a file name log4j.properties in the src folder and copy paste the below lines to this file.

```
# This sets the global logging level and specifies the appenders
log4j.rootLogger=INFO, file, stdout
# Direct log messages to a log file
log4j.appender.file=org.apache.log4j.RollingFileAppender
log4j.appender.file.File=logsSeleniumLogs.log
log4j.appender.file.MaxFileSize=1MB
log4j.appender.file.MaxBackupIndex=1
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L - %m%n

# Direct log messages to stdout
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.Target=System.out
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L - %m%n
```

Now create a suite xml with added lines related to testListener. See sample below.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<suite name="Test_Suite">
   <listeners>
      <listener class-name="testListener.TestMonitor" />
   </listeners>
   <test name="T1" preserve-order="True">
      <classes>
         <class name="mockTests.Test1">
            <methods>
               <include name="Test1_1" />
               <include name="Test1_2" />
               <include name="Test1_3" />
            </methods>
         </class>
      </classes>
   </test>
   <test name="T2" preserve-order="True">
      <classes>
         <class name="mockTests.Test2">
            <methods>
               <include name="Test2_1" />
               <include name="Test2_2" />
               <exclude name="Test2_3" />
            </methods>
         </class>
      </classes>
   </test>
   <test name="T3" preserve-order="True">
      <classes>
         <class name="mockTests.Test3">
            <methods>
               <include name="Test3_1" />
               <exclude name="Test3_2" />
               <exclude name="Test3_3" />
            </methods>
         </class>
      </classes>
   </test>
</suite>
```

When your run this suite, the logs will be generated as and when a script is taken up for execution. these logs can be found in the directory seleniumLogs.

JavascriptExecutor

It’s possible to execute your custom JavaScript on a webpage. The code below helps in scrolling the page.

```
@Test
    public void pageScroll() {
        driver.get("http://someURl.com");

        // invoke javascript executor
        JavascriptExecutor executor = (JavascriptExecutor) driver;
        executor.executeScript("window.scrollBy(0,500);scrolldelay = setTimeout('pageScroll()',50);");
    }
```

TestSuite Parameterization

This tutorial assumes you know how to create a TestSuite. Please visit the post Preparing TestSuite to learn about TestSuite. The below Suite xml file passes the parameter “browser” with value “IE” to the script it’s calling.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<suite name="Test_Suite" verbose="-1" parallel="tests"
   thread-count="1">
   <test name="Parameter_Pass-IE" preserve-order="True">
      <parameter name="browser" value="IE" />
      <classes>
         <class name="testng.Parameter_Pass">
            <methods>
               <include name="testParameterPass" />
            </methods>
         </class>
      </classes>
   </test>
</suite>
```

The below Script accepts parameter browser from the Suite, it uses the value if it’s provided from the suite, else it sets to default value when it’s not set by Suite. Ideally when we run standalone script the default value will be used.

```
package testng;

import org.testng.annotations.AfterTest;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.Optional;
import org.testng.annotations.Parameters;
import org.testng.annotations.Test;

public class Parameter_Pass {

   @Parameters({ "browser" })
   @BeforeTest
   public void setup(@Optional("Firefox") String browser) {
      System.out.println("Browser setup here..");
      System.out.println("Accepted parameter value here : " + browser);
      System.out
            .println("Please note if the test is run without a parameter from suite the default "
                  + "value will be taken as Firefox as we have provided this "
                  + "argument to @optional annotation");
   }

   @Test
   public void testParameterPass() {
      System.out.println("All the test steps here..");
   }

   @AfterTest
   public void teardown() {
      System.out
            .println("Browser close and other closing activities to finish the Test execution here..");
   }
}
```

Object Repository

We use object repository to store all our page controls in a common class, this way whenever there are changes in the UI, the respective changes can be updated in the Object Repository, without having to go to the actual code. A person with lesser knowledge of java platform can also easily do this.

Below is the script without object Repository. We will prepare an OR (Object Repository) for the same script.

```
package com.selenium;

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.By;
import org.openqa.selenium.NoSuchElementException;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.testng.Assert;
import org.testng.annotations.AfterTest;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.Test;

public class WithoutOR {

   private WebDriver driver;
   private String baseUrl;

   @BeforeTest
   public void initialization() {
      driver = new FirefoxDriver();
      baseUrl = "http://docs.seleniumhq.org/";
      driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);
      driver.manage().window().maximize();
   }

   @Test
   public void testMethod() throws InterruptedException {
      driver.get(baseUrl + "/");
      driver.findElement(By.linkText("Projects")).click();
      driver.findElement(By.linkText("Download")).click();
      Assert.assertTrue(isElementPresent(By.name("submit")));
   }

   @AfterTest
   public void cleanup() {
      driver.quit();
   }

   private boolean isElementPresent(By by) {
      try {
         driver.findElement(by);
         return true;
      } catch (NoSuchElementException e) {
         return false;
      }
   }
}
```

Create A class named ObjectRepo as below.

```
package com.selenium;

public interface ObjectRepo {
    public String linkProejct = "Projects";
    public String linkDownload = "Download";
    public String donateBtn = "submit";

}
```

Update the script as below. The script is now totally free of any hardcoding. Whatever changes happen in the UI can be udpated in the object Repository.

```
package com.selenium;

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.By;
import org.openqa.selenium.NoSuchElementException;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.testng.Assert;
import org.testng.annotations.AfterTest;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.Test;

public class WithoutOR implements ObjectRepo{

    private WebDriver driver;
    private String baseUrl;

    @BeforeTest
    public void initialization() {
        driver = new FirefoxDriver();
        baseUrl = "http://docs.seleniumhq.org/";
        driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);
        driver.manage().window().maximize();
    }

    @Test
    public void testMethod() throws InterruptedException {
        driver.get(baseUrl + "/");
        // All the elements are fetched from Object Repository
        driver.findElement(By.linkText(linkProejct)).click();
        driver.findElement(By.linkText(linkDownload)).click();
        Assert.assertTrue(isElementPresent(By.name(donateBtn)));
    }

    @AfterTest
    public void cleanup() {
        driver.quit();
    }

    private boolean isElementPresent(By by) {
        try {
            driver.findElement(by);
            return true;
        } catch (NoSuchElementException e) {
            return false;
        }
    }
}
```

Using property files

For faster access and quick modifications we can store some of our configuration settings in property files. Below you will find the code to access information stored in property files.

Create a property file named Configuration.properties in the folder propertyFiles. and paste the below configuration in file.

applicationTimeout = 30 browser = Firefox

Now create a Class with name “ReadPropertyFile” and copy paste the below code in same. The class contains code to read properties like timeout and browser.

```
package com.selenium;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.Properties;

import org.testng.annotations.Test;
import org.testng.annotations.BeforeTest;

public class ReadPropertyFile {
   String browser;
   int timeOut;

   @BeforeTest
   public void initialization() {
      // Below is the code to read property file "Configuration.properties"
      // place in folder propertyFiles
      Properties readProp = new Properties();
      readProp = loadPropertyFile("Configuration.properties");
      browser = readProp.getProperty("browser");
      timeOut = Integer.parseInt(readProp.getProperty("applicationTimeout"));
   }

   @Test
   public void testMethod() {
      System.out.println("Browser is : " + browser);
      System.out.println("Timeout is : " + timeOut);
   }

   public Properties loadPropertyFile(String propFileName) {

      Properties props = new Properties();
      FileInputStream fis;
      try {
         fis = new FileInputStream(System.getProperty("user.dir")
               + "propertyFiles" + propFileName);
         props.load(fis);
      } catch (FileNotFoundException e) {
         e.printStackTrace();
      } catch (IOException e) {
         e.printStackTrace();
      }
      return props;
   }
}
```

Capture screenshot

In order to capture screenshot, copy the below method to your common functions Class. The method create a folder named screenShots in your project root folder and saves the screenShot in .png format. The capture method can be called at any line of code by provide driver object and screenshot name as arguments.

```
public void capture(WebDriver driver, String ImageName) {
      File screenshot = null;

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

      try {
         FileUtils.copyFile(screenshot, new File("screenShots/" + ImageName
               + ".png"));
      } catch (IOException e) {
         e.printStackTrace();
      }
   }
```

Below are few xpath combinations that can be used to make your xpaths more robust.

Using starts-with : it checks if text in td starts with ‘Returns’
.//*[@id='content']/table[1]/tbody/tr/td[starts-with(text(),'Returns')]

Using contains : it checks if the td contains text ‘number’.
.//*[@id='content']/table[1]/tbody/tr/td[contains(text(),'number')]

Using wildcard : Select all immediate child elements of the tag preceding /*
html/*

Finding matches by comparing attributes
.//*[@id='content']/table[@class='code-snippet']

Same can be written using contains
.//*[@id='content']/table[contains(@class, 'de-sni')]
And starts-with
.//*[@id='content']/table[starts-with(@class, 'code')]

Using axes
.//*[@id='main']/div[attribute::class]
.//*[@id='main']/div[child::*]
.//*[@id='main']/div[child::text()] select only text nodes
.//*[@id='main']/div[child::*/child::price]

Using axes in conjuction with contains
.//*[@id='main']/div[contains(attribute::class,'chapter')]

Using operators
.//*[@id='main']/table/tbody/tr[4]/td[2][string-length(text())-1+1*1 div 1=12 and 3<4 and 4>3 or 4>=4

String handling :
.//*[@id='main']/table/tbody/tr[4]/td[2][substring-after(text(),'Sub') = 'traction']

Selects all the title AND price elements of all book elements
//book/title | //book/price

Selects all the child nodes of the bookstore element
/bookstore/*

//* Selects all elements in the document
//title[@*] Selects all title elements which have any attribute

child with child
book[author/degree]

The second text node in each <p> element in the context node.
p/text()[2]

The first two <degree> elements that are children of the context node.
degree[position() < 3]

The first three books (1, 2, 3).
book[position() <= 3]

All <author> elements that contain at least one <degree> element child and at least one <award> eleme
author[degree][award]

The last <book> element of the current context node.
book[last()]

The last <author> child of each <book> element of the current context node.

```
book/author[last()]

The last <author> element from the entire set of <author> children of <book> elements of the current c
(book/author)[last()]

all div that do not contain style attribute
//div[not(@style)]
```

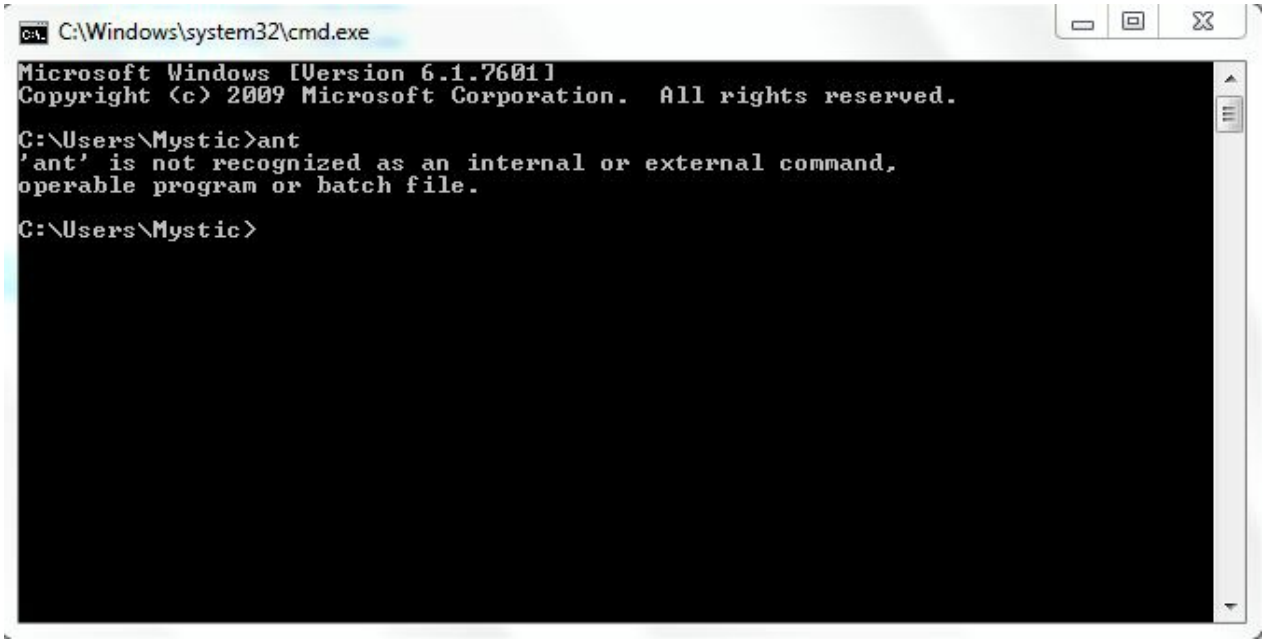
Continuous Integration

We will be using ANT for creating hook with our test framework that can be utilized by continuous integration server.

ANT Basics

Apache ANT – “Another Neat Tool” is a build tool provided by Apache Foundation. The tool is used to compile java source code, package and deploy. The tool can be downloaded from Apache ANT link. Once the ANT zip file is downloaded, extract the zip to a folder. The path can be something link C:ANT. Once the contents are extracted we need to set classpath variables. Follow the below steps carefully, to get this done.

Step 1. Prior to ANT installation, when you type ant in the command prompt you will get message as shown in below picture.

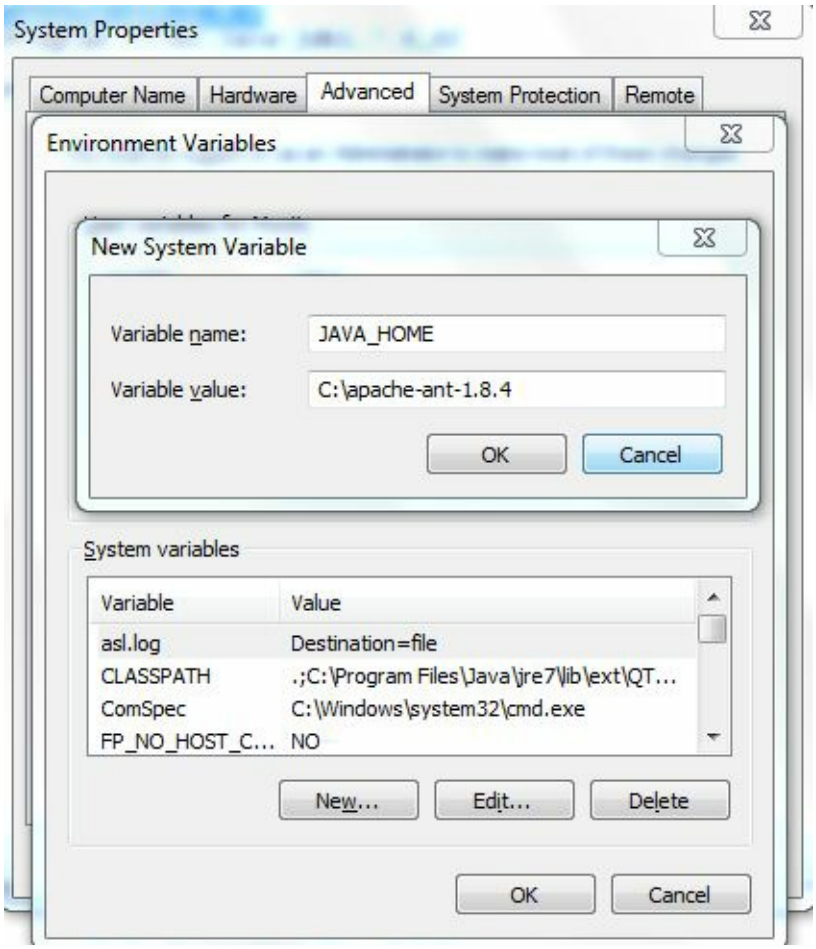


Step 2. Get the folder locations for ANT and jdk. On my machine I have below paths. Apache Installation –> C:apache-ant-1.8.4 jdk Installation –> C:Program FilesJavajdk1.7.0_02

Step 3. update the below text as per your above paths. JAVA_HOME : C:Program FilesJavajdk1.7.0_02 ANT_HOME : C:apache-ant-1.8.4

Step 4. Open your environment variables window (Right click Computer > properties >Advance System Settings > Click Environment Variable)

Step 5. Under SystemVariables click New, fill in the details as shown in below picture and click OK.

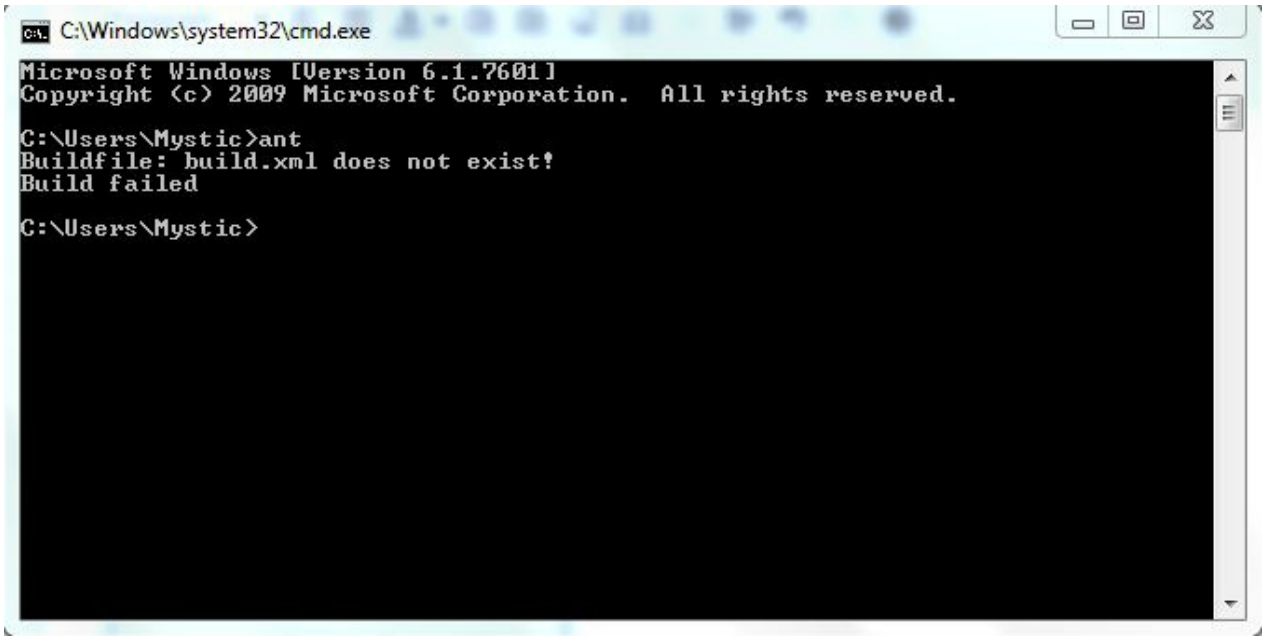


Step 6. Under System Variables click New again and create ANT_HOME providing correct variable value.

Step 7. Search for a variable named “PATH” under System variables list. Select this variable and click Edit. append the below line to the existing content. Remove the first semicolon from the below text if there is already a semicolon at the end of the opened text and click OK.

;%JAVA_HOME%bin;%ANT_HOME%bin;

Step 8. Close the command prompt if previously open. Open again ant type ant. Now you should be able to view the below message which implies that the ANT is working properly.



Once ANT is installed and working, you can refer the tutorial Tutorial: Hello World with Apache Ant. to get and idea about ANT scripting.

ANT Build.xml

We assume in this tutorial that you have ANT installed on your system. Follow the tutorial ANT Basics to know about how to get it installed. Before running the selenium testSuite via ANT, Download zip file from this location Downloads – testng-xslt – TestNG XSL Reports – Google Project Hosting. Extract the below files from this zip. saxon-8.7.jar SaxonLiaison.jar testng-results.xsl

Create a suite file with name suite.xml in the root directory of your project.

Copy the two jar files to lib folder and add them to build path. copy the testng-results.xsl to the root folder of your project. Once these files are in place create a new file with name Build.xml at the root directory of your project. Copy paste the contents below to this Build.xml

The build can be run through by right click on Build.xml > Run As > Ant Build. Or alternately using command prompt with command line : ant -buildfile **physicalPathofBuild.xml**

```
<project name="ANT_with_TestNGXslt" default="run" basedir=".">

    <property name="classes.dir" value="bin" />
    <property name="src.dir" value="src" />
    <property name="or.dir" value="objectRepo" />
    <property name="suite.dir" value="scripts" />
    <property name="report.dir" value="test-output" />
    <property name="reportXslt.dir" value="testng-xslt" />
    <property name="lib.dir" value="lib" />

    <path id="build.class.path">
        <pathelement location="${classes.dir}" />
        <pathelement location="${lib.dir}/testng-6.8.jar" />
        <pathelement location="${lib.dir}/selenium-server-standalone-2.25.0.jar" />
        <pathelement location="${lib.dir}/saxon-8.7.jar" />
        <pathelement location="${lib.dir}/SaxonLiaison.jar" />
        <pathelement location="${lib.dir}/log4j-1.2.17.jar" />
        <pathelement location="${lib.dir}/selenium-java-2.25.0.jar" />
        <pathelement location="${lib.dir}/poi/dom4j-1.6.1.jar" />
        <pathelement location="${lib.dir}/poi/poi-3.9-20121203.jar" />
        <pathelement location="${lib.dir}/poi/poi-ooxml-3.9-20121203.jar" />
        <pathelement location="${lib.dir}/poi/poi-ooxml-schemas-3.9-20121203.jar" />
        <pathelement location="${lib.dir}/poi/stax-api-1.0.1.jar" />
        <pathelement location="${lib.dir}/poi/xmlbeans-2.3.0.jar" />
    </path>

    <target name="run">
        <!--<antcall target="startSeleniumServer" />-->
        <antcall target="clean" />
        <antcall target="compile" />
        <antcall target="runTests" />
        <antcall target="generateXsltReport" />
        <!--<antcall target="stopSeleniumServer" />-->
    </target>

    <!-- Start the server -->
    <target name="startSeleniumServer">
        <echo>Starting Selenium Server...</echo>
        <java jar="${lib.dir}selenium-server-standalone-2.25.0.jar" fork="true" spawn="true">
            <arg line="-singlewindow -log ${logs.dir}selenium_server_log.txt" />
        </java>
    </target>
```

```
    <!-- Delete old data and create new directories -->
    <target name="clean">
       <echo>Initlizing...</echo>
       <delete dir="${classes.dir}" />
       <mkdir dir="${classes.dir}" />
       <delete dir="${report.dir}" />
       <mkdir dir="${report.dir}" />
       <delete dir="${reportXslt.dir}" />
       <mkdir dir="${reportXslt.dir}" />
       <mkdir dir="screenShots" />
    </target>

    <!-- Compiles the java files -->
    <target name="compile">
       <echo>Compiling...</echo>
       <javac debug="true" srcdir="${or.dir}" destdir="${classes.dir}" classpathref="build.class.path" ir
       <javac debug="true" srcdir="${src.dir}" destdir="${classes.dir}" classpathref="build.class.path" i
       <javac debug="true" srcdir="${suite.dir}" destdir="${classes.dir}" classpathref="build.class.path
       <copy todir="${classes.dir}">
          <fileset dir="${src.dir}" excludes="**/*.java" />
       </copy>
    </target>

    <!-- Runs the file and generate report -->
    <target name="runTests" description="Running tests">
       <echo>Running Tests...</echo>
       <taskdef name="testng" classname="org.testng.TestNGAntTask" classpathref="build.class.path" />
       <testng outputdir="${report.dir}" classpathref="build.class.path" workingdir="${basedir}">
          <xmlfileset dir="${basedir}" includes="suite.xml" />
       </testng>
    </target>

    <!-- Target to make XSLT reports -->
    <target name="generateXsltReport">
       <xslt in="${report.dir}/testng-results.xml" style="testng-results.xsl" out="${reportXslt.dir}/index.h
          <param name="testNgXslt.outputDir" expression="${basedir}/${reportXslt.dir}/" />
          <param name="testNgXslt.showRuntimeTotals" expression="true" />
       </xslt>
    </target>

    <!-- Stop the selenium Server -->
    <target name="stopSeleniumServer">
       <echo> Trying to stop selenium server ... </echo>
       <get taskname="selenium-shutdown" src="http://localhost:4444/selenium-server/driver/?cmd=shut
       <echo taskname="selenium-shutdown" message="Shutdown complete.." />
    </target>

 </project>
```

When the suite is run this way, at the end of your execution you should have two test output folders namely – test-output and testng-xslt.

Workarounds

IE not working

There are several reasons why Internet explorer might not open, when you run your webdriver code. Please verify the below things taken care of before running the code.

1. IEDriverServer.exe is the file which is most likely to get corrupt when file is transferred from one machine to another. Please ensure the exe is a fresh download from seleniumhq.org. Also note there are different version of IEDriverServer for 32 bit and 64 bit. download the correct version as per your system configuration.

2. A very irritating InternetExplorer setting for running selenium code : **Security settings**, Please ensure all the security settings are either checked or unchecked.

3. A rather unnoticed but important setting is the browser resolution. Please ensure that the resolution is set to 100%, the code will not work if the resolution is not 100%. You will not even know why your script is not working, if this setting is not correct.

Presence of iframe

If the html source contains an iframe tag then the usual way of locating elements does not work. We get the below exception in such cases.

```
org.openqa.selenium.NoSuchElementException: The element could not be found
(WARNING: The server did not provide any stacktrace information)
```

To tackle the issue use the below line of code, before searching for an element in iframe.

```
driver.switchTo().frame(driver.findElement(By.id("frameID")));
```