Implementing first script with WebDriver

Therefore, motioning ahead with the consequent ***Selenium WebDriver tutorial***, we would be creating WebDriver script. We would also scatter the light on the basic and commonly used**WebDriver commands**. We would also learn about the **locating strategies of UI elements** and their inclusion in the test scripts. We would also study Get Commands in the detail.

**Script Creation**

For script creation, we would be using “Learning\_Selenium” project created in the previous tutorial and “gmail.com” as the application under test (AUT).

**Scenario:**

* Launch the browser and open “Gmail.com”.
* Verify the title of the page and print the verification result.
* Enter the username and Password.
* Click on the Sign in button.
* Close the web browser.

**Step 1:** Create a new java class named as “Gmail\_Login” under the “Learning\_Selenium” project.

**Step 2:** Copy and paste the below code in the “Gmail\_Login.java” class.

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

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

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

**import** org.openqa.selenium.firefox.FirefoxDriver;

**public** **class** Gmail\_Login {

/\*\*

\* **@param** args

\*/

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

// objects and variables instantiation

WebDriver driver = **new** FirefoxDriver();

String appUrl ="https://accounts.google.com";

driver.get(appUrl);

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

// declare and initialize the variable to store the expected title

String expectedTitle = " Sign in - Google Accounts ";

// fetch the title of the web page and save it into a string variable

String actualTitle = driver.getTitle();

// compare the expected title of the page with the actual title of the page

**if** (expectedTitle.equals(actualTitle))

{

System.***out***.println("Verification Successful");

}

**else**

{

System.***out***.println("Verification Failed");

}

// enter a valid username in the email textbox

WebElement username = driver.findElement(By.id("Email"));

username.clear();

username.sendKeys("TestSelenium");

// enter a valid password in the password textbox

WebElement password = driver.findElement(By.id("Passwd"));

password.clear();

password.sendKeys("password123");

// click on the Sign in button

WebElement SignInButton = driver.findElement(By.id("signIn"));

SignInButton.click();

// close the web browser

driver.close();

System.***out***.println("Test script executed successfully.");

// terminate the program

System.*exit*(0);

}

}

**IMPORTANT**

* *get()* method to launch a fresh web browser instance.
* *maximize()* to maximize the browser window.
* *clear()* to clear the value present in the textbox if any.
* *sendKeys()* to enter the specified value in the textbox.
* *Click()* to click on the web element.

***import****org.openqa.selenium.WebDriver* –WebDriver interface which is required to instantiate a new web browser.

***import****org.openqa.selenium.firefox.FirefoxDriver* –FirefoxDriver class that is required instantiate a Firefox specific driver on the browser instance instantiated using WebDriver interface.

***import****org.openqa.selenium.WebElement*– WebElement class which is required to instantiate a new web element.

***import****org.openqa.selenium.By –*References to the By class on which a locator type is called.

As and when our project would grow, it is evident and logical that we might have to introduce several other packages for more complex and distinct functionalities like excel manipulations, database connectivity, logging, assertions etc.

**Object Instantiation**

*WebDriver driver = new FirefoxDriver();*

We create a reference variable for WebDriver interface and instantiate it using FirefoxDriver class.

**Launching the Web browser**

*driver.get(appUrl);*

A *get()*method is called on the WebDriver instance to launch a fresh web browser instance. The string character sequence passed as a parameter into the *get()* method redirects the launched web browser instance to the application URL.

**Maximize Browser Window**

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

**Fetch the page Title**

*driver.getTitle();*

The *getTitle()* method is used to fetch the title of the current web page. Thus, the fetched title can be loaded to a string variable.

**Comparison between Expected and Actual Values:**

|  |  |
| --- | --- |
|  | // compare the expected title of the page with the actual title  **if** (expectedTitle.equals(actualTitle))  {  System.***out***.println("Verification Successful - The correct title is displayed on the web page.");  }  **else**  {  System.***out***.println("Verification Failed - An incorrect title is displayed on the web page.");  } |

**WebElement Instantiation**

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

In the above statement, we instantiate the WebElement reference with the help of *“driver.findElement(By.id(“Email”))”.*Thus, username can be used to reference the Email textbox on the user interface every time we want to perform some action on it.

**Clear Command**

*username.clear();*

The clear() method/command is used to clear the value present in the textbox if any. It also clears the default placeholder value.

**sendKeys Command**

*username.sendKeys(“TestSelenium “);*

The *sendKeys()* method/command is used to enter/type the specified value (within the parentheses ) in the textbox. Notice that the*sendKeys()* method is called on the WebElement object which was instantiated with the help of element property corresponding to the UI element.

The above block of code enters the string “TestSelenium” inside the Email textbox on the Gmail application.

*sendKeys* is one of the most popularly used commands across the WebDriver scripts.

**Click Command**

*SignInButton.click();*

Like *sendKeys(), click()* is another excessively used command to interact with the web elements. *Click()* command/method is used to click on the web element present on the web page.

The above block of code clicks on the “Sign in” button present on the Gmail application.

**Notes:**

* Unlike sendKeys() method, click() methods can never be parameterized.
* At times, clicking on a web element may load a new page altogether. Thus to sustain such cases, click() method is coded in a way to wait until the page is loaded.

**Close the Web Browser**

*driver.close();*

The close() is used to close the current browser window.

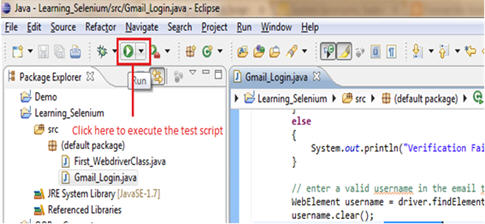
**Terminate the Java Program**

*System.exit(0);*

**Test Execution**

**The test script or simply the java program can be executed in the following ways:**

**#1.** Under the Eclipse’s menu bar, there is an icon to execute the test script. Refer the following figure.

[](http://cdn2.softwaretestinghelp.com/wp-content/qa/uploads/2014/10/WebDriver-tutorial-1.jpg)

Make a note that only the class which is selected would be executed.

**#2.** Right click anywhere inside the class within the editor, select “Run As” option and click on the “Java Application”.

**#3.** Another shortcut to execute the test script is – Press ctrl + F11.

At the end of the execution cycle, the print statement “Test script executed successfully.” can be found in the console.

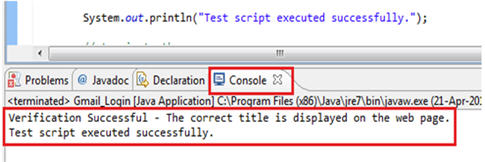
**Locating Web Elements**

Web elements in WebDriver can be located and inspected in the same way as we did in the previous tutorials of Selenium IDE. Selenium IDE and Firebug can be used to inspect the web element on the GUI. It is highly suggested to use Selenium IDE to find the web elements. Once the web element is successfully found, copy and paste the target value within the WebDriver code. The types of locators and the locating strategies are pretty much the same except for the syntax and their application.

In WebDriver, web elements are located with the help of the dynamic finders (findElement(By.locatorType(“locator value”))).

Sample Code:

*driver.findElement(By.id(“Email”));*

[](http://cdn2.softwaretestinghelp.com/wp-content/qa/uploads/2014/10/WebDriver-tutorial-2.jpg)

**Locator Types and their Syntax**

| **Locator Type** | **Syntax** | **Description** |
| --- | --- | --- |
| id | driver.findElement (By.id(“ID\_of\_Element”)) | Locate by value of  the “id” attribute |
| className | driver.findElement (By.className (“Class\_of\_Element”)) | Locate by value of  the “class” attribute |
| linkText | driver.findElement (By.linkText(“Text”)) | Locate by value of the text of the hyperlink |
| partialLinkText | driver.findElement (By.partialLinkText (“PartialText”)) | Locate by value of the sub-text of the hyperlink |
| name | driver.findElement (By.name (“Name\_of\_Element”)) | Locate by value of the “name” attribute |
| xpath | driver.findElement (By.xpath(“Xpath”)) | Locate by value  of the xpath |
| cssSelector | driver.findElement (By.cssSelector (“CSS Selector”)) | Locate by value of the CSS selector |
| tagName | driver.findElement (By.tagName(“input”)) | Locate by value of its tag name |

**Conclusion**

In this tutorial, we developed an automation script using WebDriver and Java. We also discussed the various components that constitute a WebDriver script.

**Here are the cruxes of this Selenium WebDriver Tutorial:**

* A *get()* method used to launch a fresh web browser instance. The character sequence passed as a parameter into the get() method redirects the launched web browser instance to the application URL.
* The *maximize()* method is used to maximize the browser window.
* The *clear()* method is used to clear the value present in the textbox if any.
* The *sendKeys()* method is used to enter the specified value in the textbox.
* *Click()* method is used to click on the web element present on the web page.
* In WebDriver, web elements can be located using Dynamic finders.
* The following are the available locator types:
  + id
  + className
  + name
  + xpath
  + cssSelector
  + linkText
  + partialLinkText
  + tagName

Moving ahead, in the next tutorial, we would shift our focus towards a framework that aids to Automation testing known as TestNG. We would have a detailed study on the various kinds of the annotations provided by the framework.

[**Next tutorial #11**](http://www.softwaretestinghelp.com/selenium-junit-framework-selenium-tutorial-11/)**:** Before diving deep into Frameworks we will see details about JUnit – an open source unit testing tool. Most of the programmers use JUnit as it is easy and does not take much effort to test. This tutorial will give an insight about JUnit and its usage in selenium script.