TestNG Reporters

TestNG is a Framework and so far we have already seen the many different powerful features of TestNG. It almost gives you all the important things you are required to complete the Framework.

TestNG Reporter Logs

TestNG also gives us the logging facility for the test. For example during the running of test case user wants some information to be logged in the console. Information could be any detail depends upon the purpose. Keeping this in mind that we are using Selenium for testing, we need the information which helps the User to understand the test steps or any failure during the test case execution. With the help of TestNG Logs it is possible to enable logging during the Selenium test case execution.

In selenium there are two types of logging. **High level** logging and **Low level** logging. In low level logging you try to produce logs for the every step you take or every action you make in your automation script. In high level logging you just try to capture main events of your test.

Everybody has their own style of logging and I have mine too. I am also a big fan of Log4j logging and that's why I do not mix log4j logging with testng logging but on the same side I make to use of both of its. I perform **low level logging with log4j** and **high level logging with testng** reporter logs.

How to do it…

1) Write a test case for Sign In application and implement Log4j logging on every step.

2) Insert Reporter logs on the main events of the test.

```
1  package automationFramework;
2
3  import java.util.concurrent.TimeUnit;
4
5  import org.apache.log4j.Logger;
6
7  import org.apache.log4j.xml.DOMConfigurator;
8
9  import org.openqa.selenium.By;
10
11 import org.openqa.selenium.WebDriver;
12
13 import org.openqa.selenium.firefox.FirefoxDriver;
14
15 import org.testng.Reporter;
16
17 import org.testng.annotations.Test;
18
19 import utility.Log;
20
21 public class ReporterLogs {
22
23  private static WebDriver driver;
24
25  private static Logger Log = Logger.getLogger(Log.class.getName());
26
27     @Test
28
29  public static void test() {
30
31  DOMConfigurator.configure("log4j.xml");
```

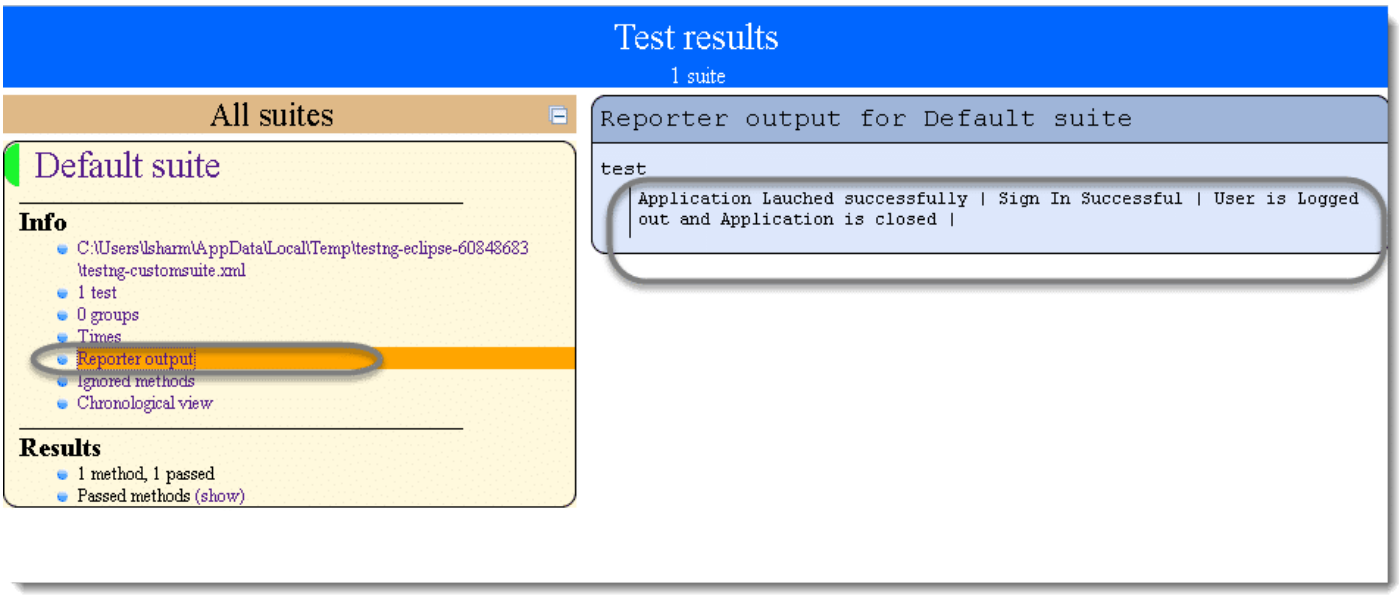
```
32
33          driver = new FirefoxDriver();
34
35          Log.info("New driver instantiated");
36
37          driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
38
39          Log.info("Implicit wait applied on the driver for 10 seconds");
40
41          driver.get("http://www.store.demoqa.com");
42
43          Log.info("Web application launched");
44
45          // Our first step is complete, so we produce a main event log here for our reports.
46
47          Reporter.log("Application Lauched successfully | ");
48
49          driver.findElement(By.xpath(".//*[@id='account']/a")).click();
50
51          Log.info("Click action performed on My Account link");
52
53          driver.findElement(By.id("log")).sendKeys("testuser_1");
54
55          Log.info("Username entered in the Username text box");
56
57          driver.findElement(By.id("pwd")).sendKeys("Test@123");
58
59          Log.info("Password entered in the Password text box");
60
61          driver.findElement(By.id("login")).click();
62
63          Log.info("Click action performed on Submit button");
64
65          // Here we are done with our Second main event
66
67          Reporter.log("Sign In Successful | " );
68
69          driver.findElement(By.id("account_logout"));
70
71          Log.info("Click action performed on Log out link");
72
73          driver.quit();
74
75          Log.info("Browser closed");
76
77          // This is the third main event
78
79          Reporter.log("User is Logged out and Application is closed | ");
80
81   }
82
83  }
```

3) Run the test by right click on the test case script and select **Run As** > **TestNG Test**.

Your Log4j logging output will look like this:

```
2014-04-27 13:30:34,111 INFO  [Log] web application launched
2014-04-27 13:30:40,489 INFO  [Log] Click action performed on My Account link
2014-04-27 13:30:40,537 INFO  [Log] Username entered in the Username text box
2014-04-27 13:30:40,580 INFO  [Log] Password entered in the Password text box
2014-04-27 13:30:41,181 INFO  [Log] Click action performed on Submit button
2014-04-27 13:30:44,062 INFO  [Log] Click action performed on Log out link
2014-04-27 13:30:44,621 INFO  [Log] Browser closed
```

But your Reporters log will look like this:



Log4j logging will help you to report a bug or steps taken during the test, on the other hand reporters log will help you to share the test status with leadership. As leadership is just interested in the test results, not the test steps.

I also use reporter's logs on the verification during the test. For example

```
1  if(Text1.equals(Text2)){
2
3  Reporter.log("Verification Passed forText");
4
5  }else{
6
7  Reporter.log("Verification Failed for Text");
8
9  }
```

TestNG Asserts

TestNG also gives us the power to take decisions in the middle of the test run with the help of **Asserts**. With this we can put various checkpoints in the test. Asserts are the most popular and frequently used methods while creating Selenium Scripts. In selenium there will be many situations in the test where you just like to check the presence of an element. All you need to do is to put an assert statement on to it to verify its existence.

Different Asserts Statements

1) Assert.assertTrue() & Assert.assertFalse()

```
1  package automationFramework;
2
3  import java.util.concurrent.TimeUnit;
4
5  import org.openqa.selenium.By;
6
7  import org.openqa.selenium.WebDriver;
8
9  import org.openqa.selenium.WebElement;
10
11 import org.openqa.selenium.firefox.FirefoxDriver;
12
13 import org.testng.Assert;
14
```

```
import org.testng.annotations.Test;

public class Asserts {

 private static WebDriver driver;

  @Test

  public void f() {

   driver = new FirefoxDriver();

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

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

      // Here driver will try to find out My Account link on the application

      WebElement myAccount = driver.findElement(By.xpath(".//*[@id='account']/a"));

      //Test will only continue, if the below statement is true

      //This is to check whether the link is displayed or not

      Assert.assertTrue(myAccount.isDisplayed());

      //My Account will be clicked only if the above condition is true

      myAccount.click();

  }

}
```

Note: Assert true statement fails the test and stop the execution of the test, if the actual output is false. Assert.assertFalse() works opposite of Assert.assertTrue(). It means that if you want your test to continue only if when some certain element is not present on the page. You will use Assert false, so it will fail the test in case of the element present on the page.

2) Assert.assertEquals()

```
  @Test

  public void test() {

   String sValue = "Lakshay Sharma";

   System.out.println(" What is your full name");

   Assert.assertEquals("Lakshay Sharma", sValue);

   System.out.println(sValue);

  }
```

It also works the same way like assert true and assert fail. It will also stop the execution, if the value is not equal and carry on the execution, if the value is equal.