**Selenium Framework Creation and Accessing Test Data from Excel**

In the last module, we familiarized you with the [**basics of test automation Frameworks**](http://www.softwaretestinghelp.com/test-automation-frameworks-selenium-tutorial-20/), its components and types. The frameworks illustrated in the previous module were a few amongst the most popular frameworks used by the testing fraternity.

We briefly discussed about Module based Frameworks, Library Architecture based framework, Keyword driven framework, Data driven Framework and Hybrid Framework. There are various other frameworks also in the place.

Please take a note that we would be adopting***Data Driven Test Automation Framework for the rest of our modules.***

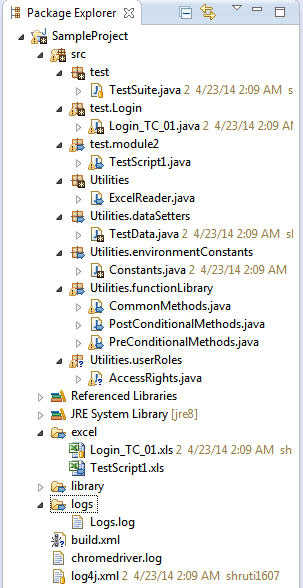
**In the current**[**module in this series**](http://www.softwaretestinghelp.com/selenium-tutorial-1/), we would make you acquainted with a **sample framework, the Excels which would store the test data and their Excel manipulations**. On the same lines, we would move forward and introduce new strategies and resources to mature our framework.

**So let’s learn:**

* Framework creation strategy using a sample project
* Access the test data stored in the external data source

Moving ahead, we would start with the description of the project hierarchy that we would be creating in order to segregate the various project components.

Refer the below image for the project hierarchy created for the sample project. The below java project can be easily created within the eclipse the way we have created the projects in the earlier modules.

[](http://cdn2.softwaretestinghelp.com/wp-content/qa/uploads/2014/11/Selenium-Framework.jpg)

### ****Selenium Project Folder Structure – Walkthrough****

**#1) src –** The folder contains all the test scripts, generics, readers and utilities. All these resources are nothing but the simple java classes. Under the source (src) folder, we have created a hierarchy of folders.

**a) test**– The “test” folder is constituted of majorly two ingredients – testsuite and the folders representing the various modules of the application under test. Thus, each of these folders contains the test scripts specific to the module to which it is associated. Testsuite is a logical combination of more than one test scripts. Thus, the user can mark an entry of any of the test script within the testsuite that he/she desires to execute in the subsequent runs.

**b) utilities** – The “utilities” folder is constituted of various generics, constants, Readers and classes for implementing user defined exceptions. Each of the folders under utilities has got its own significance.

* **Excel Reader –** A generic and common class has been created to read the test data (input parameters and expected results) from the Excel sheets
* **EnvironmentConstants –** The folder are integration of the java classes that stores the static variables referencing to the paths and other environmental details. These details can be Application URL, URL to the Databases, Credentials for Databases, and URL to any third party tool being used. The disparate application URLs can be set for different environments (dev, prod, test, master, slave etc).
* **DataSetters –** The folder incorporates the classes that implement the getters and setters of the test data fetched from the Excels. To lode multiple sets of Test data, we create ArrayLists.
* **UserRoles –** The folder accommodates the classes that take care of the Role based access criteria if any for instinct users.
* **FunctionLibrary –** The folder is constituted of the classes which contain functions and methods that can be shared and used amongst the multiple classes. Very often, we are suppose to perform certain procedures prior and aftermath to the actual test execution like login to the application, setting up environments, activities related to rolls, data manipulations, writing results, methods those generate pre/post-conditions to other methods. Since we tend to perform these activities for all or most of the test script. Thus it is always recommended to create a separate class for such activities instead of coding them repeatedly in each of the test script.
  + **PreConditionalMethods**
  + **PostConditionalMethods**

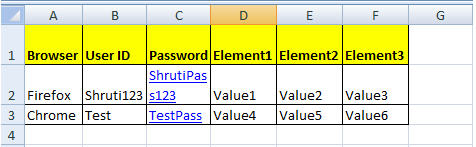
Very often, we are suppose to perform certain procedures prior and aftermath to the actual test execution like login to the application, setting up environments, activities related to user rolls, data manipulations, writing results, methods those generate pre/post-conditions to other methods. Since we tend to perform these activities for all or most of the test script, thus it is always recommended to create a separate class for such activities instead of coding them repeatedly in each of the test script.

**CommonMethods**

Like Pre and post conditions, there may be methods and functions those can be used by more than one test script. Thus, these methods are grouped together in a class. The testscript can access these methods using the object of the common class.

**#2) excelFiles –** The excel files are considered to be the data source/data providers for test script execution. These files store the test data into key value pairs. Make a note that we create a separate excel sheet for each of the test script i.e. each test script has its own test data file. The name of the test script and the corresponding test data files/ excel sheet has been kept same for the traceability perspective. Check out the sample test data format below:

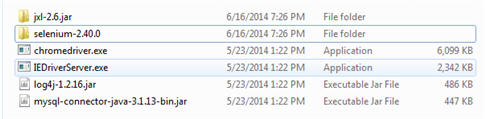
**Test Data Format**

[](http://cdn.softwaretestinghelp.com/wp-content/qa/uploads/2014/11/Selenium-Framework-2.jpg)

Each of the columns represents a key and each of the rows represents a test data/value. Specify the multiple rows in order to execute the same test script with multiple data sets.

Mark that the test data formats are solely user defined. Thus based on your requirements, you can customize the test data files.

**#3) library –** The folder acts as a repository/artifactory for all the required jar files, libraries, drivers etc to successfully build the test environment and to execute the test scripts. Refer the following figure to check out the libraries we would be employing within our project.

[](http://cdn.softwaretestinghelp.com/wp-content/qa/uploads/2014/11/Selenium-Framework-3.jpg)

**#4) logs –** The folder contains a .txt file that stores the logging information upon each execution.

**#5) testMaterial –** The folder contains the actual test data that needs to be uploaded if any. This folder would come into picture when we come across test scenarios where the user is required to upload files, documents, pictures, reports etc.

**#6) build.xml –**The xml file is used by the “Ant Server” to automate the entire build process.

**#7) log4j.xml –**This xml file is used by a Java based utility named as “Log4j” to generate the execution logs.

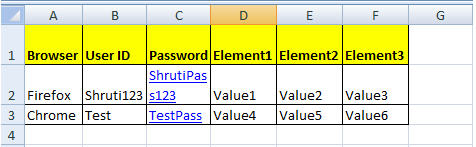
**Note**: We would study more about the logs, user defined exceptions and Ant in detail in the upcoming modules. So don’t panic if you get confused between the notions.

**Now, as we move forward let us understand the phenomenon where we access the excel files and populate the test data into our test scripts.**

In order to comprehend the process easily, we would break down the process into the following steps.

### ****Test Data Creation****

**Step 1:** The first and the foremost step is to create the test data with which we would be executing the test scripts. Considering the aforementioned test data format, let us create an excel file named as “TestScript1”. Furnish the values in the elements.

[](http://cdn.softwaretestinghelp.com/wp-content/qa/uploads/2014/11/Selenium-Framework-2.jpg)

**Step 2:** The next step is to download a standard java based API/Library named as “Java excel Library” (jxl) to be able to access the already created generic methods for Excel Manipulation.

**Step 3:** Create a generic excel reader class named as “ExcelReader.java”. Copy the below code in the ExcelReader.java.

------------

|  |  |
| --- | --- |
| 1 | package Utilities; |
| 2 | import java.io.File; | |

|  |  |
| --- | --- |
| 3 | import java.io.IOException; |
| 4 | import java.util.Hashtable; |

|  |  |
| --- | --- |
| 5 | import jxl.Sheet; |
| 6 | import jxl.Workbook; | |

|  |  |  |
| --- | --- | --- |
| 7 | import jxl.read.biff.BiffException; | |
| 8 |  |

|  |  |  |
| --- | --- | --- |
| 9 | /\*\* | |
| 10 | | \* This is a utility class created to read the excel test data file before performing the test steps. | |

|  |  |  |
| --- | --- | --- |
| 11 | \* This class loads the excel file and | |
| 12 | \* reads its column entries. |

|  |  |
| --- | --- |
| 13 | \* |
| 14 | \*/ | |

|  |  |
| --- | --- |
| 15 |  |
| 16 | public class ExcelReader { | |

|  |  |
| --- | --- |
| 17 | /\*\* |
| 18 | \* The worksheet to read in Excel file | |

|  |  |  |
| --- | --- | --- |
| 19 | \*/ | |
| 20 |  |

|  |  |  |
| --- | --- | --- |
| 21 | public static Sheet wrksheet; | |
| 22 | /\*\* |

|  |  |  |
| --- | --- | --- |
| 23 | \* The Excel file to read | |
| 24 | \*/ |

|  |  |
| --- | --- |
| 25 |  |
| 26 | public static Workbook wrkbook = null; | |

|  |  |
| --- | --- |
| 27 | /\*\* |
| 28 | \* Store the column data | |

|  |  |  |
| --- | --- | --- |
| 29 | \*/ | |
| 30 |  |

|  |  |  |
| --- | --- | --- |
| 31 | public static Hashtable<String, Integer> dict = new Hashtable<String, Integer>(); | |
| 32 | /\*\* |

|  |  |  |
| --- | --- | --- |
| 33 | \* Create a Constructor | |
| 34 | \* |

|  |  |
| --- | --- |
| 35 | \* @param ExcelSheetPath |
| 36 | \* @throws BiffException |

|  |  |  |
| --- | --- | --- |
| 37 | \* @throws WeblivException | |
| 38 | \*/ |

|  |  |
| --- | --- |
| 39 |  |
| 40 | public ExcelReader(String ExcelSheetPath)throws IOException, BiffException { | |

|  |  |
| --- | --- |
| 41 |  |
| 42 | // Initialize | |

|  |  |
| --- | --- |
| 43 | try { |
| 44 | wrkbook = Workbook.getWorkbook(new File(ExcelSheetPath)); | |

|  |  |  |
| --- | --- | --- |
| 45 | wrksheet = wrkbook.getSheet("Sheet1"); | |
| 46 | } catch (IOException e) { |

|  |  |  |
| --- | --- | --- |
| 47 | throw newIOException(); | |
| 48 | } |

|  |  |
| --- | --- |
| 49 | } |
| 50 | /\*\* | |

|  |  |  |
| --- | --- | --- |
| 51 | \* Returns the Number of Rows | |
| 52 | \* |

|  |  |  |
| --- | --- | --- |
| 53 | \* @return Rows | |
| 54 | \*/ |

|  |  |
| --- | --- |
| 55 |  |
| 56 | public static int RowCount() { | |

|  |  |  |
| --- | --- | --- |
| 57 | return wrksheet.getRows(); | |
| 58 | } |

|  |  |
| --- | --- |
| 59 | /\*\* |
| 60 | \* Returns the Cell value by taking row and Column values as argument | |

|  |  |
| --- | --- |
| 61 | \* |
| 62 | \* @param column | |

|  |  |
| --- | --- |
| 63 | \* @param row |
| 64 | \* @return Cell contents | |

|  |  |  |
| --- | --- | --- |
| 65 | \*/ | |
| 66 |  |

|  |  |
| --- | --- |
| 67 | public static String ReadCell(int column,int row) { |
| 68 | returnwrksheet.getCell(column, row).getContents(); | |

|  |  |
| --- | --- |
| 69 | } |
| 70 | /\*\* | |

|  |  |  |
| --- | --- | --- |
| 71 | \* Create Column Dictionary to hold all the Column Names | |
| 72 | \*/ |

|  |  |
| --- | --- |
| 73 | public static void ColumnDictionary() { |
| 74 | // Iterate through all the columns in the Excel sheet and store the | |

|  |  |
| --- | --- |
| 75 | // value in Hashtable |
| 76 | for (int col = 0; col < wrksheet.getColumns(); col++) { | |

|  |  |  |
| --- | --- | --- |
| 77 | dict.put(ReadCell(col,0), col); | |
| 78 | } |

|  |  |
| --- | --- |
| 79 | } |
| 80 | /\*\* | |

|  |  |  |
| --- | --- | --- |
| 81 | \* Read Column Names | |
| 82 | \* |

|  |  |  |
| --- | --- | --- |
| 83 | \* @param colName | |
| 84 | \* @return value |

|  |  |  |
| --- | --- | --- |
| 85 | \*/ | |
| 86 |  |

|  |  |  |
| --- | --- | --- |
| 87 | public static int GetCell(String colName) { | |
| 88 | try { |

|  |  |
| --- | --- |
| 89 | int value; |
| 90 | value = ((Integer) dict.get(colName)).intValue(); | |

|  |  |
| --- | --- |
| 91 | returnvalue; |
| 92 | } catch(NullPointerException e) { | |

|  |  |  |
| --- | --- | --- |
| 93 | return(0); | |
| 94 | } |

|  |  |  |
| --- | --- | --- |
| 95 | } | |
| 96 | } |

**Step 4:** Create a generic class –“CommonMethods.java”. Create a common method within the class that would read the cells from the excel sheet using the methods implemented in ExcelReader.java.

|  |  |
| --- | --- |
| 1 | /\*\* |
| 2 | \* Read the test data from excel file | |

|  |  |
| --- | --- |
| 3 | \* |
| 4 | \* @param data The TestData data object | |

|  |  |  |
| --- | --- | --- |
| 5 | \*/ | |
| 6 |  |

|  |  |
| --- | --- |
| 7 | public void readExcelData (TestData data) { |
| 8 | ArrayList<String> browser = new ArrayList<String>(); | |

|  |  |  |
| --- | --- | --- |
| 9 | ArrayList<String> username = new ArrayList<String>(); | |
| 10 | | ArrayList<String> password = new ArrayList<String>(); | |

|  |  |
| --- | --- |
| 11 | ArrayList<String> element1 = new ArrayList<String>(); |
| 12 | ArrayList<String> element2 = new ArrayList<String>(); |

|  |  |  |
| --- | --- | --- |
| 13 | ArrayList<String> element3 = new ArrayList<String>(); | |
| 14 |  |

|  |  |
| --- | --- |
| 15 | // Get the data from excel file |
| 16 | for (int rowCnt = 1; rowCnt < ExcelReader.RowCount(); rowCnt++) { | |

|  |  |
| --- | --- |
| 17 | browser.add(ExcelReader.ReadCell(ExcelReader.GetCell("Browser"), rowCnt)); |
| 18 | username.add(ExcelReader.ReadCell(ExcelReader.GetCell("User ID"), rowCnt)); |

|  |  |
| --- | --- |
| 19 | password.add(ExcelReader.ReadCell(ExcelReader.GetCell("Password"), rowCnt)); |
| 20 | element1.add(ExcelReader.ReadCell(ExcelReader.GetCell("Element1"), rowCnt)); |

|  |  |  |
| --- | --- | --- |
| 21 | element2.add(ExcelReader.ReadCell(ExcelReader.GetCell("Element2"), rowCnt)); | |
| 22 | | element3.add(ExcelReader.ReadCell(ExcelReader.GetCell("Element3"), rowCnt)); |

|  |  |
| --- | --- |
| 23 | } |
| 24 | data.setBrowser(browser); | |

|  |  |  |
| --- | --- | --- |
| 25 | data.setLoginUser(username); | |
| 26 | data.setPassword(password); |

|  |  |
| --- | --- |
| 27 | data.setElement1(element1); |
| 28 | data.setElement2(element2); |

|  |  |  |
| --- | --- | --- |
| 29 | data.setElement3(element3); | |
| 30 | } |

**Step 5:** Create a new java class named as “TestData.java”. This class would act as a getter and setter for excel data. Copy and paste the following code in the TestData.java class.

|  |  |  |
| --- | --- | --- |
| 1 | package Utilities.dataSetters; | |
| 2 | import java.util.ArrayList; |

|  |  |
| --- | --- |
| 3 | public class TestData { |
| 4 | private ArrayList<String> loginUser = null; | |

|  |  |  |
| --- | --- | --- |
| 5 | private ArrayList<String> password = null; | |
| 6 | private ArrayList<String> browser = null; |

|  |  |
| --- | --- |
| 7 | private ArrayList<String> element1 = null; |
| 8 | private ArrayList<String> element2 = null; |

|  |  |  |  |
| --- | --- | --- | --- |
| 9 | private ArrayList<String> element3 = null; | | |
| 10 | | /\*\* |

|  |  |  |
| --- | --- | --- |
| 11 | \* @return loginUser | |
| 12 | \*/ |

|  |  |  |
| --- | --- | --- |
| 13 | public ArrayList<String> getLoginUser() { | |
| 14 | return loginUser; |

|  |  |
| --- | --- |
| 15 | } |
| 16 | /\*\* | |

|  |  |  |
| --- | --- | --- |
| 17 | \* @param loginUser | |
| 18 | \*/ |

|  |  |  |
| --- | --- | --- |
| 19 | public void setLoginUser(ArrayList<String> loginUser) { | |
| 20 | this.loginUser = loginUser; |

|  |  |
| --- | --- |
| 21 | } |
| 22 | /\*\* | |

|  |  |  |
| --- | --- | --- |
| 23 | \* @return password | |
| 24 | \*/ |

|  |  |  |
| --- | --- | --- |
| 25 | public ArrayList<String> getPassword() { | |
| 26 | return password; |

|  |  |
| --- | --- |
| 27 | } |
| 28 | /\*\* | |

|  |  |  |
| --- | --- | --- |
| 29 | \* @param password | |
| 30 | \*/ |

|  |  |  |
| --- | --- | --- |
| 31 | public void setPassword(ArrayList<String> password) { | |
| 32 | this.password = password; |

|  |  |
| --- | --- |
| 33 | } |
| 34 | /\*\* | |

|  |  |  |
| --- | --- | --- |
| 35 | \* @return browser | |
| 36 | \*/ |

|  |  |  |
| --- | --- | --- |
| 37 | public ArrayList<String> getBrowser() { | |
| 38 | return browser; |

|  |  |
| --- | --- |
| 39 | } |
| 40 | /\*\* | |

|  |  |  |
| --- | --- | --- |
| 41 | \* @param browser | |
| 42 | \*/ |

|  |  |  |
| --- | --- | --- |
| 43 | public void setBrowser(ArrayList<String> browser) { | |
| 44 | this.browser = browser; |

|  |  |
| --- | --- |
| 45 | } |
| 46 | /\*\* | |

|  |  |  |
| --- | --- | --- |
| 47 | \* @return element1 | |
| 48 | \*/ |

|  |  |  |
| --- | --- | --- |
| 49 | public ArrayList<String> getElement1() { | |
| 50 | return element1; |

|  |  |
| --- | --- |
| 51 | } |
| 52 | /\*\* | |

|  |  |  |
| --- | --- | --- |
| 53 | \* @param element1 | |
| 54 | \*/ |

|  |  |  |
| --- | --- | --- |
| 55 | public void setElement1(ArrayList<String> element1) { | |
| 56 | this.element1 = element1; |

|  |  |  |
| --- | --- | --- |
| 57 | } | |
| 58 | /\*\* |

|  |  |  |
| --- | --- | --- |
| 59 | \* @return element2 | |
| 60 | \*/ |

|  |  |  |
| --- | --- | --- |
| 61 | public ArrayList<String> getElement2() { | |
| 62 | return element2; |

|  |  |
| --- | --- |
| 63 | } |
| 64 | /\*\* | |

|  |  |  |
| --- | --- | --- |
| 65 | \* @param element2 | |
| 66 | \*/ |

|  |  |  |
| --- | --- | --- |
| 67 | public void setElement2(ArrayList<String> element2) { | |
| 68 | this.element2 = element2; |

|  |  |
| --- | --- |
| 69 | } |
| 70 | /\*\* | |

|  |  |  |
| --- | --- | --- |
| 71 | \* @return element3 | |
| 72 | \*/ |

|  |  |  |
| --- | --- | --- |
| 73 | public ArrayList<String> getElement3() { | |
| 74 | return element3; |

|  |  |
| --- | --- |
| 75 | } |
| 76 | /\*\* | |

|  |  |  |
| --- | --- | --- |
| 77 | \* @param element3 | |
| 78 | \*/ |

|  |  |  |
| --- | --- | --- |
| 79 | public void setElement3(ArrayList<String> element3) { | |
| 80 | this.element3 = element3; |

|  |  |  |
| --- | --- | --- |
| 81 | } | |
| 82 | } |

**Step 6:** The next step is to create instances of “TestData.java” and “CommonMethods.java” java classes within the test script in order to access and populate the test data. Refer the below code snippet for object initialization, reading excel data and populating the values wherever required.

|  |  |
| --- | --- |
| 1 | // Create Objects |
| 2 | public ExcelReader excelReaderObj; | |

|  |  |  |
| --- | --- | --- |
| 3 | CommonMethods commonMethodobj = new CommonMethods(); | |
| 4 | TestData td = new TestData(); |

|  |  |
| --- | --- |
| 5 |  |
| 6 | // Load the excel file for testing | |

|  |  |  |
| --- | --- | --- |
| 7 | excelReaderObj = new ExcelReader(Path of the excel); | |
| 8 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 9 | // Load the Excel Sheet Col in to Dictionary for use in test cases | | |
| 10 | | excelReaderObj.ColumnDictionary(); |

|  |  |
| --- | --- |
| 11 |  |
| 12 | // Get the data from excel file | |

|  |  |  |
| --- | --- | --- |
| 13 | commonMethodobj.readExcelData (td); | |
| 14 |  |

|  |  |  |
| --- | --- | --- |
| 15 | | // Populate the username |
| 16 | driver.findElement(By.id("idofElement")).sendKeys(data.getLoginUser().get(0)); | | |

**Therefore using the instance of testData.java class in conjunction with getters, any test data value can be populated within the script.**

### ****Conclusion:****

The module mainly revolved around the notions like Framework Creation and Accessing test data from the excels. We made you acquainted with the Framework creation strategy using a sample project. We briefly laid light on the various components and aspects of our framework.

In order to access the test data stored in the external data source, we used a java based API – jxl. We also created the sample code for reading and populating the excel data into the testscripts.