



## ***Swift - Selenium User Guide***

Document Version [2.1] and Date [18-Jan-2016]

By Deepak S. Rupani,

Sachin V. Shengale

Tripti Mittal

Kuldeep K. Vasani

```
/*
 * Copyright(c) 2016 Mastek Ltd. All rights
reserved.
 *
 * SwiftLite is distributed in the hope that it
will be useful.
 *
 * This file is part of SwiftLite Framework:
Licensed under the Apache License,
 * Version 2.0 (the "License"); you may not use
this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
```



\*  
\* Unless required by applicable law or agreed to  
in writing, software  
\* distributed under the License is distributed on  
an "AS IS" BASIS,  
\* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND,  
either express or implied.  
\* See the License for the specific language  
governing permissions and  
\* limitations under the License.  
\*/

Division : Global Testing CoE	Doc. Name : Swift - Selenium User Guide	Version :2.1
Document Classification: NA	Date: 18-Jan-2016	Page : 2 of 23



# Table of Contents

<b>1. INTRODUCTION .....</b>	<b>5</b>
1.1 Objectives.....	5
1.2 Overview .....	5
Management's requirements.....	5
Manual testing team's requirements .....	5
Automation tester's approach.....	5
<b>2. FRAMEWORK STRUCTURE .....</b>	<b>6</b>
2.1 Library Architecture Framework .....	6
2.2 Keyword Driven Framework.....	6
2.3 Data Driven Framework.....	6
2.4 BDD (behaviour-driven development) support .....	6
<b>3. FRAMEWORK INSTALLATION .....</b>	<b>7</b>
3.1 Creating Folder Structure .....	7
3.2 Creating Java Project .....	7
<b>4. FRAMEWORK DETAILS .....</b>	<b>14</b>
4.1 Flowchart .....	14
4.2 Folder Structure .....	14
4.3 Key Components.....	14
Src .....	14
Projects.....	15
Common Resources .....	15
Resources.....	18
Input .....	18
Verification .....	19
Results .....	20
Templates .....	19
<b>5. Execution .....</b>	<b>20</b>
5.1 Interactive Mode .....	20
5.2 Batch Mode.....	20
<b>6. BDD Support.....</b>	<b>21</b>
6.1 BDD ON.....	21
a. features .....	22
b. glue .....	22

Division : Global Testing CoE	Doc. Name : Swift - Selenium User Guide	Version :2.1
Document Classification: NA	Date: 18-Jan-2016	Page : 3 of 23



c. tags .....	22
6.2 BDD OFF .....	22
<b>AMENDMENT HISTORY .....</b>	<b>23</b>

Division : Global Testing CoE	Doc. Name : Swift - Selenium User Guide	Version :2.1
Document Classification: NA	Date: 18-Jan-2016	Page : 4 of 23



# 1. INTRODUCTION

Mastek has successfully developed a highly configurable Hybrid Automation framework: “Swift” using Selenium tool that offers quick turnaround for Automation Suite Implementation. The Automation regression suite can be created or modified using tester friendly keywords for modules, screens, and verifications, thereby ensuring a more effective and efficient Test automation process and better ROI through the complete application life cycle.

## 1.1 Objectives

The objective of this document is to provide an insight of the Mastek’s Swift (hybrid) automation framework using Selenium tool.

## 1.2 Overview

A test automation framework is a set of functions, assumptions, concepts, and practices that provide support for automated software testing. Functions and concepts are developed for common test automation activities like UI data input, verification and reporting.

The framework caters to the goals and requirements of the management, the testing team and, of course, the automation testers.

### Management’s requirements

- Reuse the framework across the organization
- Reduction in total testing efforts, improvement in quality of deliverable
- Have a quick turnaround time while releasing newer versions
- Determine status of test execution by looking at the test results

### Manual testing team’s requirements

- Control test execution process
- Select verifications/checkpoints
- Understand how to run the tests and analyse the failures
- Execute script with minimal support from the automation development team

### Automation tester’s approach

- Develop and maintain scripts with minimum effort
- Reuse scripts across multiple projects
- Recover from script failure(s)

Division : Global Testing CoE	Doc. Name : Swift - Selenium User Guide	Version :2.1
Document Classification: NA	Date: 18-Jan-2016	Page : 5 of 23



## 2. FRAMEWORK STRUCTURE

Swift is a hybrid framework that derives benefits from the following automation framework methodologies.

### 2.1 Library Architecture Framework

Framework contains rich and robust library of generic functions for data input, verification and reporting. Library functions can be called from anywhere within the framework.

### 2.2 Keyword Driven Framework

Development of the functionality as step-by-step instruction set captured in external spreadsheets. These spreadsheets use keyword names that are relevant to the business users.

E.g. RegisterUser, EnterUserName, Address details, Save, VerifyUser etc.

The automated driver script interprets the instruction set to simulate user actions. This approach makes the framework independent of the application under test.

### 2.3 Data Driven Framework

Different data sets used for same functionality to increase regression test coverage. Test case input values will be read from a data file provided by business users. It Involves externalization of input data and verification data files provided by business users. Test case expected output values can be read from either input data or external file provided by business users.

The data files can be in any of the following formats: CSV, XLS, XML, etc.

### 2.4 BDD (behaviour-driven development) support

BDD is extension of Test Driven Development and it is used to test the system rather than testing the particular piece of code. BDD framework provides the ability to write acceptance tests for any application. It allows automation of functional validations in easily readable and understandable format (like plain English called 'Gherkin') for all stake holders a like viz. Business Analysts, Developers, Testers, etc. It aims scenarios expressed in the form 'given-when-then' format.

Swift uses Cucumber as tool to support BDD. Test scenarios are specified in feature files with extension: .feature.

Division : Global Testing CoE	Doc. Name : Swift - Selenium User Guide	Version :2.1
Document Classification: NA	Date: 18-Jan-2016	Page : 6 of 23



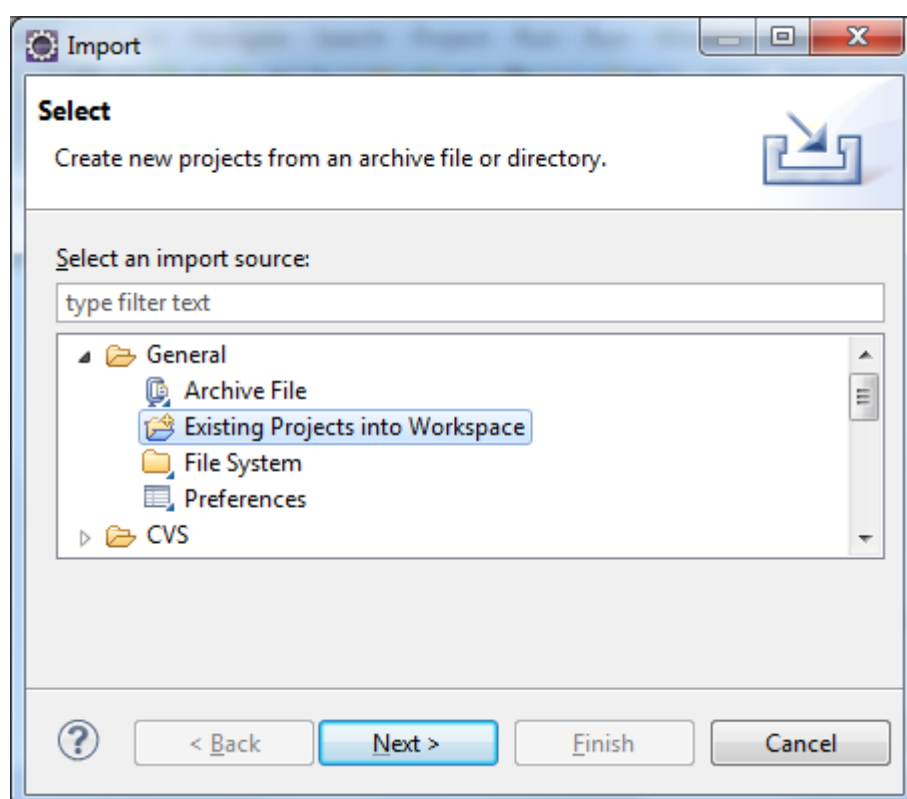
## 3. FRAMEWORK INSTALLATION

### 3.1 Creating Folder Structure

Swift framework is provided as zipped Java project containing all the requisite files, libraries, folders, etc. This zip file needs to be stored on a root drive. Now create a workspace folder (let's say '**Workspace - WS\_SwiftLite**') in root drive and extract the zip file into this workspace. The folder structure will be created by extraction of the zip file and it should not be changed.

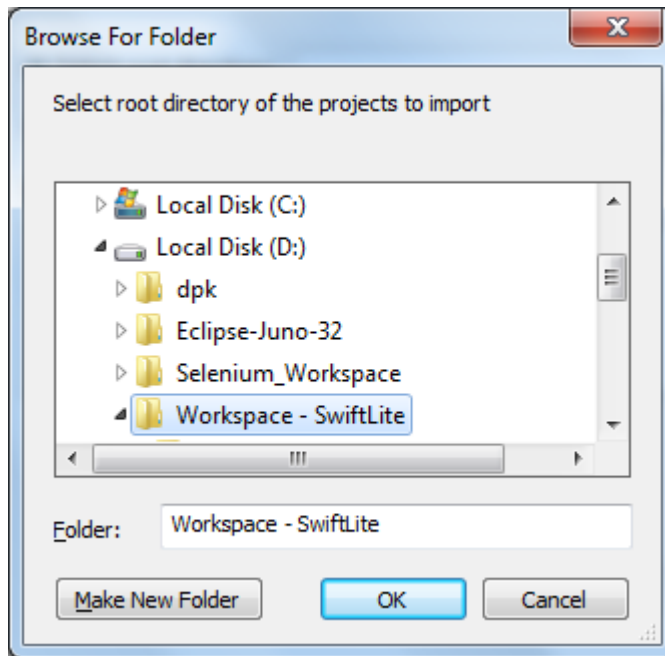
### 3.2 Creating Java Project

Once the folders are extracted, open Eclipse and select the above create workspace. Now import the above project (File >> Import) into the workspace as shown in the screenshots below:



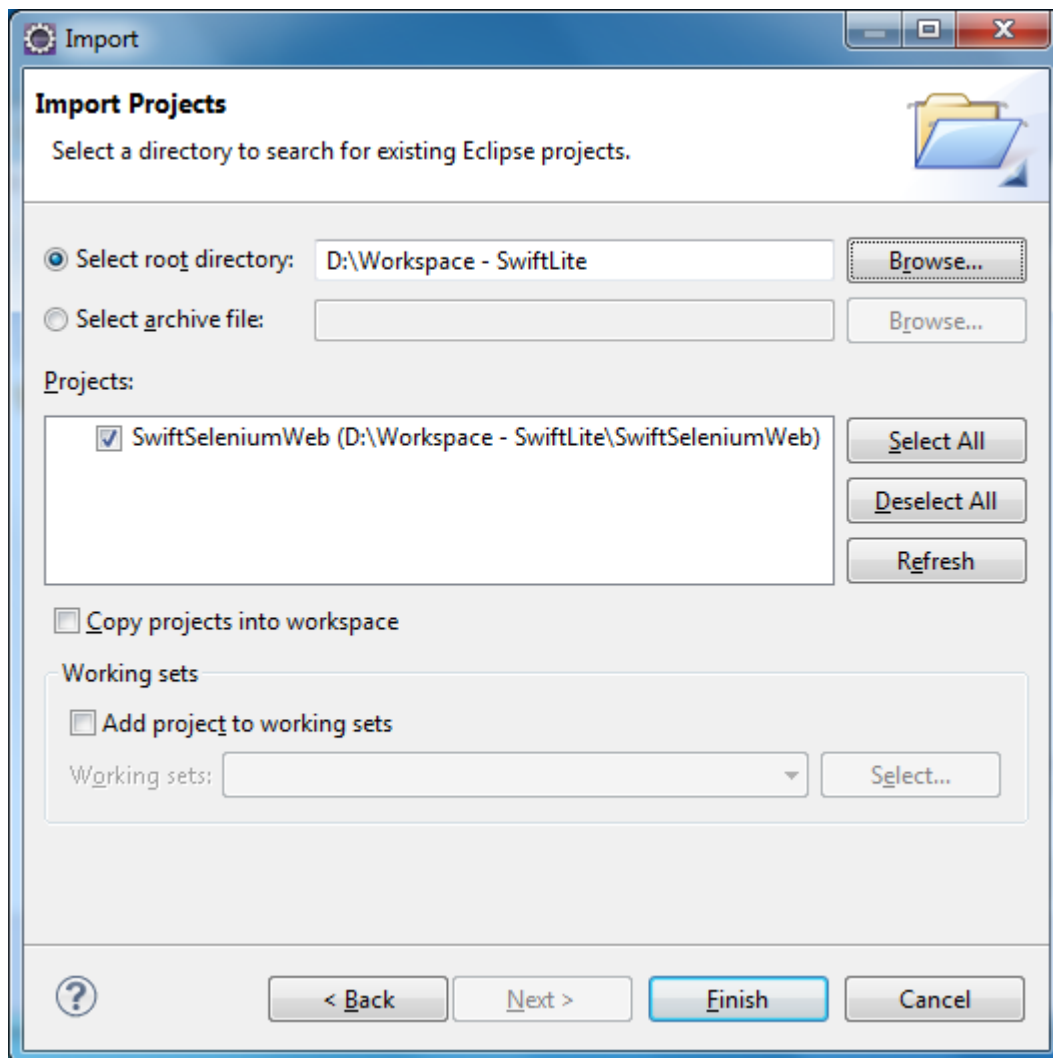
Select the option as shown above and click 'Next'. Then browse to the workspace and click 'OK'.

Division : Global Testing CoE	Doc. Name : Swift - Selenium User Guide	Version :2.1
Document Classification: NA	Date: 18-Jan-2016	Page : 7 of 23



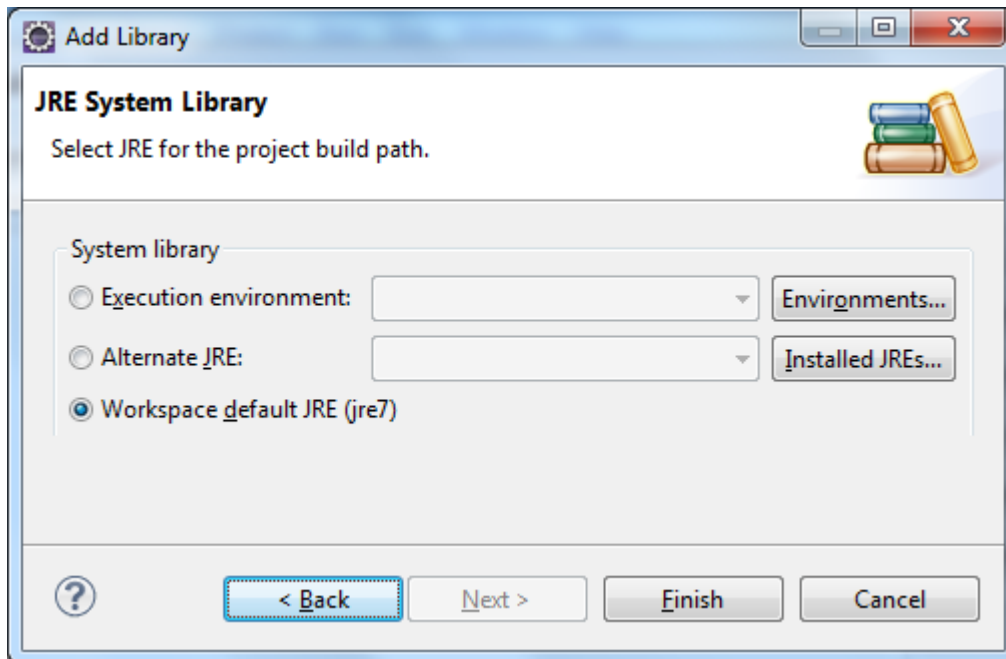
Division : Global Testing CoE	Doc. Name : Swift - Selenium User Guide	Version :2.1
Document Classification: NA	Date: 18-Jan-2016	Page : 8 of 23



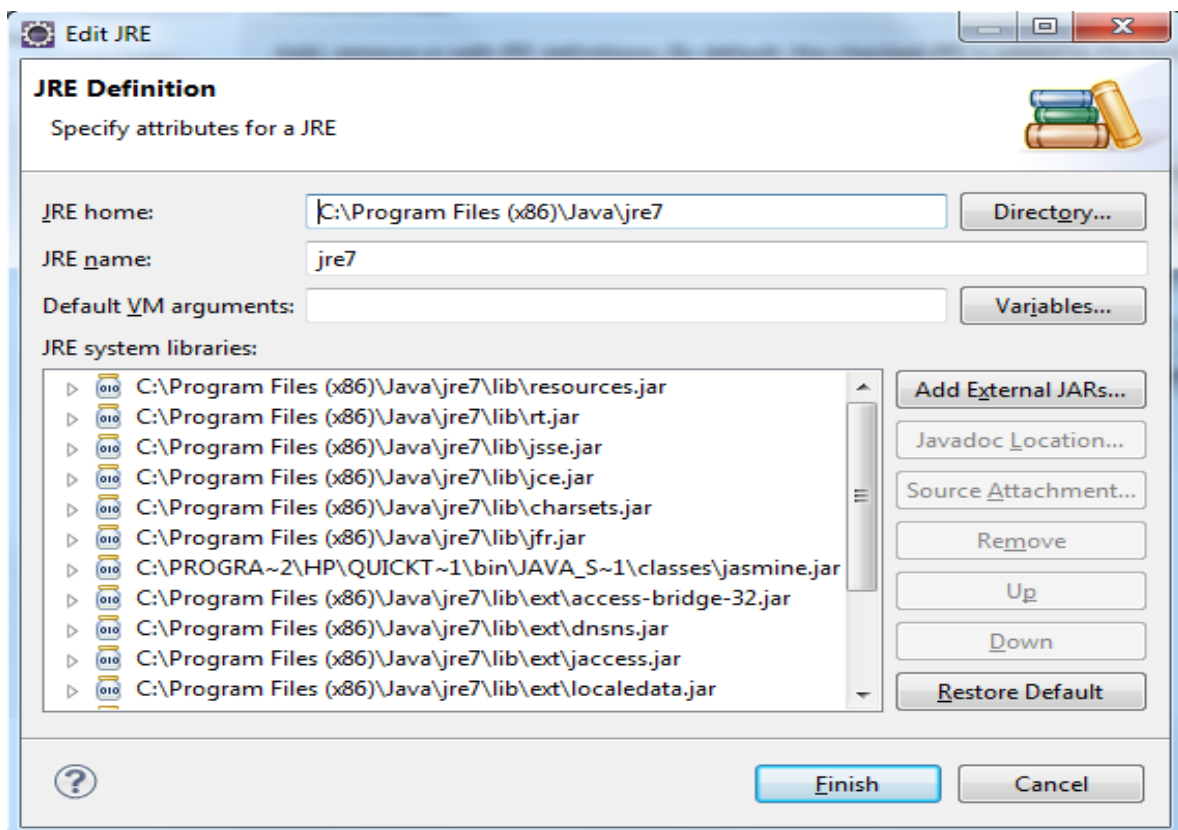


Click 'Finish'.

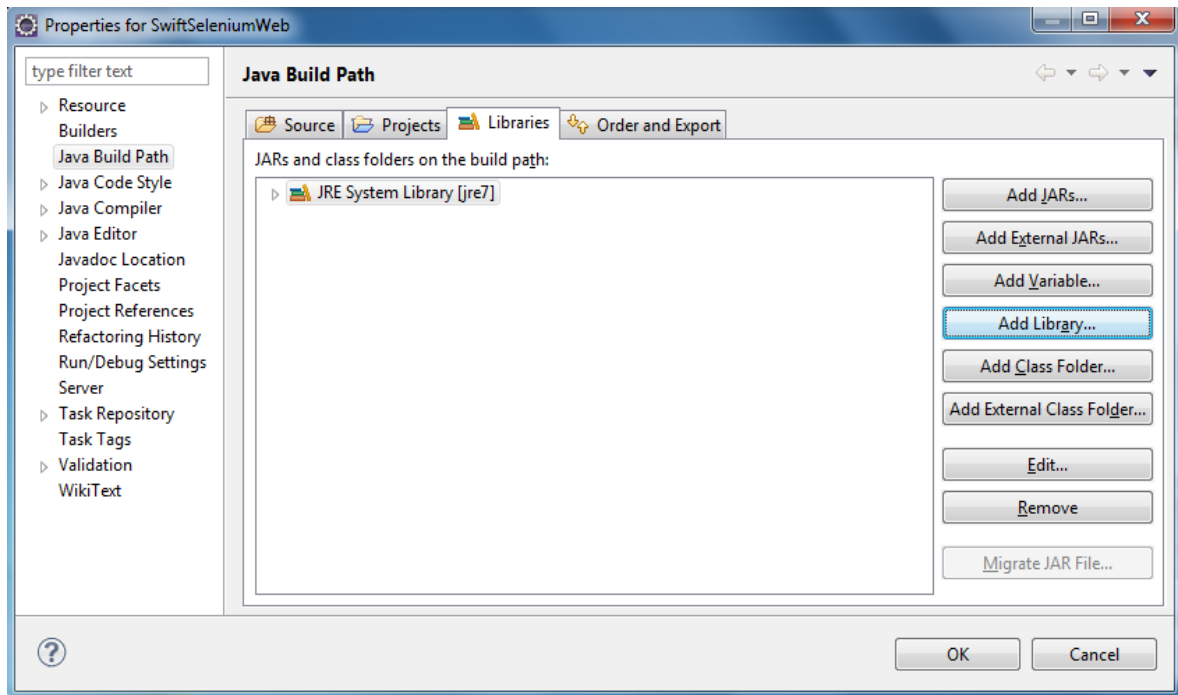
Division : Global Testing CoE	Doc. Name : Swift - Selenium User Guide	Version :2.1
Document Classification: NA	Date: 18-Jan-2016	Page : 9 of 23



Select 'Alternate JRE' option and click on 'Installed JREs' button and select the JRE installed on your machine.



Division : Global Testing CoE	Doc. Name : Swift - Selenium User Guide	Version :2.1
Document Classification: NA	Date: 18-Jan-2016	Page : 10 of 23



Now click on the 'OK' button, only a few warnings may remain, which can be ignored. The build is ready to use.

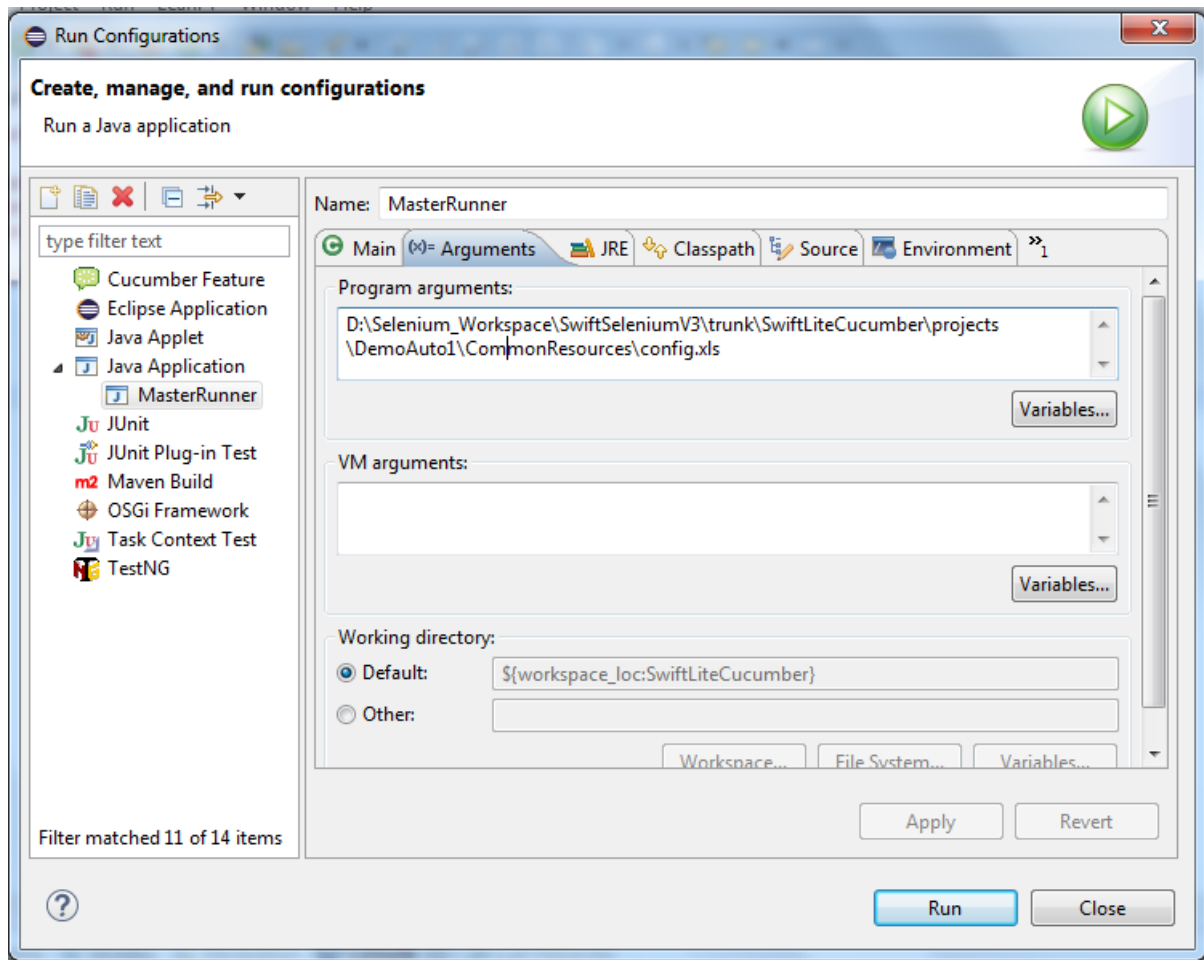
Execution starts with the file 'MasterRunner.java' based argument passed to this file as complete path for Config.xls. This file has option to make framework Cucumber BDD enabled with parameter 'BDD\_SUPPORT' to be set to TRUE, which is explained in the BDD support section.

Division : Global Testing CoE	Doc. Name : Swift - Selenium User Guide	Version :2.1
Document Classification: NA	Date: 18-Jan-2016	Page : 11 of 23



As Swift supports multiple configurations support, we need to specify which config.xls to pick up for the execution as shown in below fig.

Division : Global Testing CoE	Doc. Name : Swift - Selenium User Guide	Version :2.1
Document Classification: NA	Date: 18-Jan-2016	Page : 12 of 23



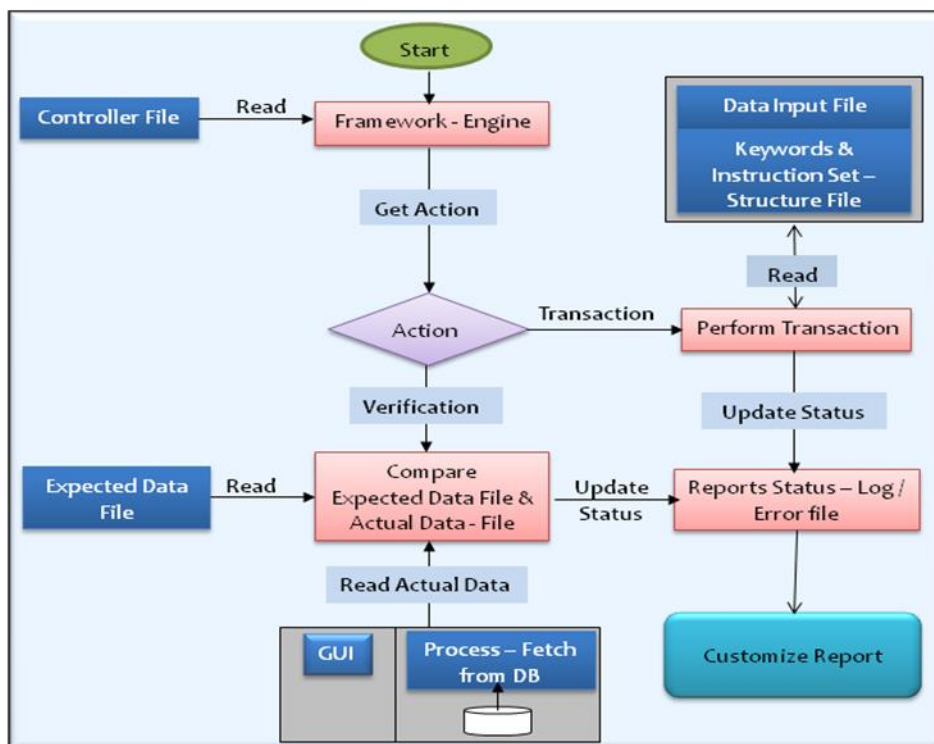
Division : Global Testing CoE	Doc. Name : Swift - Selenium User Guide	Version :2.1
Document Classification: NA	Date: 18-Jan-2016	Page : 13 of 23



## 4. FRAMEWORK DETAILS

### 4.1 Flowchart

The following schematic represents the functional flow of the automation framework



### 4.2 Folder Structure

Swift framework uses a specific folder structure for arranging the key components of the automation suite. The folder hierarchy aids the framework in attaining the expected functions. The following section explains the main components of the automation suite folder structure.

### 4.3 Key Components

#### Src

This folder contains the java code files which form the machinery of the Swift framework.

**MasterRunner.java** is the test execution file from where the execution starts. This is also the file to keep the custom code for any complex transactions.

Division : Global Testing CoE	Doc. Name : Swift - Selenium User Guide	Version :2.1
Document Classification: NA	Date: 18-Jan-2016	Page : 14 of 23



## Projects

This folder contains individual folders pertaining to each project which in turn contains input data spreadsheets and configuration files. Typically folders “CommonResources” and “Resources” are kept in specific project directory.

## Common Resources

This folder stores all the driver spreadsheets and configuration files.

1. **Config.xls** is the configuration file that captures commonly used values for variables/constants. This allows adding and changing the variables, without the need to change the code or scripts. It also helps in creating a common location for most common variables as well as keeping the code independent of machines/servers. The file paths are stored in different environment variables, which will then be called via the automation scripts. These paths may need to be changed depending on the workspace folder path directory.

2. **MainController.xls** is the driver spreadsheet that controls the flow of execution. Each row in the Main Controller represents a test case or Scenario and each column defines corresponding transactions to complement the test case actions or steps. The flow of execution is from left to right and then top to bottom.

Each of these transaction keywords perform a set of specific actions defined in Transaction Input File (described later).

Each row in the Main Controller represents a test case or Scenario and each column defines corresponding transactions to complement the test case actions or steps. The flow of execution is from left to right and then top to bottom.

GroupName	TestCaseID	Test_Description	Execute				
Smoke	TC1	Registration	Y	<b>START</b>	Register	MercuryLogoff	MercuryLogin
Smoke	TC2	FlightBooking	Y		MercuryLogin	FlightBooking	TicketBooking
Regression	SC2	Masteknet Search	N		MastekSearch	VerifySearchResults	
Regression	SC3	Check flight records	Y		MercuryLogoff	MercuryLogin	FlightBooking
Integration	PC5	Verify travel data	N		MercuryLogin	FlightBooking	VerifyTravelEntry

## Column Description:

1. **GroupName:** This is type or category of the test case, or scenarios, like Smoke, Deployment etc. This determines the grouping of the results at the time of execution.
2. **TestCaseID:** This is the Test case or Scenario identifier helpful to identify the input or verification for respective Transaction Type to be specified in the same row later. This is required during execution as well as when results are being written. The Test Case ID should be unique and should not be repeated.
3. **TestDescription:** This is just an informatory field and defines the purpose of the test case.

Division : Global Testing CoE	Doc. Name : Swift - Selenium User Guide	Version :2.1
Document Classification: NA	Date: 18-Jan-2016	Page : 15 of 23



4. **ExecuteFlag:** This column value determines whether a test case should be executed or not. A value of 'Y' means that the corresponding row (test case) should be executed and a value of 'N' means that the corresponding row is to be completely ignored during execution.
5. **User defined keywords:** These keywords can start from the 6<sup>th</sup> column i.e. column 'F'. Each of these keywords performs a specific action, which is defined in the Transaction Input File.

Apart from these, there are two tags:

1. **START**
2. **PAUSE**

The execution starts from the START tag and continues (horizontally i.e. row-wise) till the PAUSE tag is reached. These tags can be placed anywhere beginning from the 6<sup>th</sup> column i.e. column 'F' onwards. The execution begins from the first START tag and stops at the first PAUSE tag. So if a test case is to be executed, it must have the Execute Flag as 'Y' and should fall within the START and PAUSE tags.

Since the entire flow of execution is maintained in a spreadsheet in the form of user defined keywords, a new test case execution can be easily configured, or an existing one amended, if the user has the knowledge of the required transaction keywords.

3. **Transaction\_Input\_Files.xls** determines what action a transaction keyword used in the Main Controller spreadsheet would perform. When the framework encounters a transaction keyword in the Main Controller, it searches for its associated file/action in the Transaction Input File

Created	TransactionCo	TransactionType	DirPath	InputSheet
RP	Login	LoginMercury	Login	Login.xls
RP		SelectFlight	FlightReservation	SelectFlight.xls
RP		BookFlight	FlightReservation	BookFlight.xls
RP		Confirmation	FlightReservation	
RP		Register,UpdateRegistration	FlightReservation	Register.xls
RP		VerifyFlightBooking		
DR		VerifyItinerary		
DR		EmpSearch	MastekEmpSearch	EmpSearch.xls
DR		VerifyEmpSearch		

Keywords are associated with one of the following actions

1. Data Input
2. Verification

A transaction keyword lets framework know what needs to be done as part of invoking that transaction type. Keywords are associated with one of the following actions, apart from also containing steps to be performed for navigating through the application:

1. **Data Input:** In this case, the transaction keyword points to **Input Datasheets** (described later) which contain data/values to be entered on a particular screen or page of the application. Field level verifications can also be done using these sheets.

Division : Global Testing CoE	Doc. Name : Swift - Selenium User Guide	Version :2.1
Document Classification: NA	Date: 18-Jan-2016	Page : 16 of 23





Example: A keyword 'CreateStudent' will contain instructions for creating a new student and also contain all the requisite input values (data) to be entered in the process of student creation.

2. **Verification:** In this case, the transaction keyword starts with prefix "Verify..." points to Verification Datasheets (described later). These keywords will invoke comparison of web table data based on the expected sheets and actual application values. These keywords will also publish the comparison results for the execution.

Example: The keyword 'VerifyAllStudentDetails' will invoke verification of all student details captured from application and compare them with expected sheets stored in the verification folder. The results of this comparison will then be published in summary and detailed results for users to validate/authenticate the run results.

Note: This full page Verification feature is not deployed in SwiftLite version

There are 4 columns in the Transaction Input File:

1. **Created By:** This column is not functional for documentation use only.
2. **Transaction Code:** This column is not required to be edited or entered by end users/SME. Internally used by automation framework only.
3. **Transaction Type:** This is same as the transaction keyword in the Main Controller file. Each transaction type should be unique, but can be associated with multiple datasheets.
4. **Dir Path:** This defines the directory path of the datasheets associated with a particular transaction type. This column lets the framework know which sheet has to be considered for data input actions and screen navigations. In case of verification transaction types, these fields may remain blank if there are no data entry or screen navigations happening for a particular transaction keyword.
5. **Input Sheet:** This column will contain the name of the spreadsheet associated with the transaction keyword. The value in this field will form the basis of data input or screen navigations for the application as well as providing specific selection/input criteria.

**4. VerificationTableList.xls** holds the information of the path and names of templates, expected results and actual results. All keywords with verification associated are listed in this sheet. These keywords invoke comparison of data from the expected sheets and actual application values. These keywords also publish the comparison results for the execution.

Note: The full page Verification feature is not deployed in SwiftLite version

TransactionType	VerificationType	TemplateAddition	TemplateSheet	ExpectedDataAddition	ExpectedData Sheet
VerifyBillingEnquiry	VerifyFlightDetails	BID	TC_POLINO_014_1.xls	BID	VerifyFlightDetails.xls
VerifyFlightDetails	VerifyFlightDetails	FlightBooking	Flight_Confirmation.xls	SearchResults	VerifyFlightDetails_Expected.xls
VerifyCruise	VerifyCruise	FlightBooking	Cruises.xls	SearchResults	VerifyCruise_Expected.xls
VerifySearchResults	VerifySearchResultsU	FlightBooking	SearchResultsUniform.xls	SearchResults	VerifySearchResults_Expected.xls
VerifyItinerary	Verify Flight Itinerary	FlightBooking	CruiseItinerary.xls	FlightBooking	CruiseItinerary_ExpectedValues.xls

Division : Global Testing CoE	Doc. Name : Swift - Selenium User Guide	Version :2.1
Document Classification: NA	Date: 18-Jan-2016	Page : 17 of 23



## Resources

This folder contains 3 types of data and hence is sub-divided into 3 folders:

### Input

This folder contains the input data sheets which co-relate to a particular screen or a group of screens in the application. You can further create sub-folders here that separate different modules of an application.

The input files define the execution flow on a particular screen or group of screens. These files contain navigation steps and expected values for input fields during the navigation. Each input sheet has 2 worksheets – 'Structure' and 'Values'. When a transaction input file that contains keywords and path invokes an input file, Selenium will look at the 'Structure' sheet for the specific input scenario and follow the navigation steps. 'Values' sheet contains field values that are to be considered/inserted in specific instances.

#### Structure sheet:

ExecuteFlag	Action	LogicalName	ControlName	ControlType	ControlID
Y	NC	Register	REGISTER	WebLink	LinkText
Y	I	FirstName	firstName	WebEdit	Name
Y	I	LastName	lastName	WebEdit	Name
Y	I	Phone_No	phone	WebEdit	Id
Y	I	eMail	userName	WebEdit	Id
Y	I	Pin_code	postalCode	WebEdit	Name
Y	I	Country	country	WebList	Name
Y	I	UserName	email	WebEdit	Name
Y	I	Password	password	WebEdit	Name
Y	I	ConfirmPwd	confirmPassword	WebEdit	Name
Y	NC		register	WebButton	Name
Y	V	VerifyText	//p/font/b	WebElement	XPath

#### Values Sheet:

TestCaseID	TransactionType	FirstName	LastName	Phone_No	eMail	Pin_code	Password	ConfirmPwd	UserName	VerifyText
TC1	Register	Deepak	Sharma	987654321	DeepakS@abc.com		123pass	123pass	Test	Dear Deepak
TC2	Register	Imran	Khan	123456789	ImranK@abc.com	223344	pass123	pass123	Test1	Dear Imran

Object identification is done using the data in columns '**ControlId**' and '**ControlName**'. So if control id is "LinkText" and control name is "REGISTER", the framework looks for an object on the screen whose "LinkText" is "REGISTER".

What type of action is to be taken on the identified object is defined by the column '**ControlType**'. So if control type is "WebLink", a click operation is performed and for "WebEdit", text is entered.

Division : Global Testing CoE	Doc. Name : Swift - Selenium User Guide	Version :2.1
Document Classification: NA	Date: 18-Jan-2016	Page : 18 of 23



**'LogicalName'** column forms the link between the Structure and the Values sheet. What data is entered for which control is determined by the logical name. Hence, Logical name should be kept as similar as the object's screen name so that data entry becomes easy.

**'Action'** specifies the action to be carried out. Action 'NC' stands for navigation control meaning the values from other columns in that row specify where the application is supposed to navigate. No data needs to be specified for a 'NC' action and this action will always be carried out once specified.

Action 'I' stands for Input, where input value for the parameter is specified in the 'Values' sheet. Every 'I' in the structure sheet must have a value specified or the framework will skip it and not perform any operation for that control.

Action 'V' is for field level verification i.e. verification of the value in a particular control like a label, a web edit etc. The actual value is captured from the screen and is compared with the expected value specified in the 'Values' sheet.

**'ExecuteFlag'** column specifies if the row needs to be considered or not. 'Y' means the row needs to be considered and 'N' means that the row should be skipped from execution.

## Verification

This folder stores the expected values sheet for table verification as per the test criteria in the folder specified in the verification table list for a given transaction type. This is the path where the actual data file created during execution will be stored for comparison.

Note: This full page Verification feature is not deployed in SwiftLite version

## Templates

This folder contains templates which act as a blue print for a web table that needs to be verified for its data contents. The template sheet holds information for identification of the table or the fields inside the table. Once the table is identified on the screen, data is extracted from it and stored as actual data and is then compared with expected data.

Below are the screenshot of a uniform table verification template.

TableType	TableID	TableIDType	StartRow	EndRow	ColumnName	Row	Column
Uniform	//div[@id='StaDiv']/div[2]/table	XPath	1		Status		0
					TransEffDate		1
					CancelType		2
					CancelMethod		3
					NoticeAmount		4
					CancelReason		5

The 'TableType' can be Uniform or Non-Uniform. 'TableIDType' and 'TableId' together identify the web table. 'Column' number determine which columns need to be considered for data extraction. 'ColumnName' is like the logical name in the input data sheets and is user defined matching the screen column names and act as a link with the column names in the expected data sheets. 'StartRow' determines which row to start with and 'EndRow' should be entered if certain rows from the end need not be extracted.

Division : Global Testing CoE	Doc. Name : Swift - Selenium User Guide	Version :2.1
Document Classification: NA	Date: 18-Jan-2016	Page : 19 of 23



Once the actual data sheet is created in the 'Verification' folder, as stated in previous section, its compared with the expected data sheet present in the same folder and results are published in the 'Results' folder as explained above.

## Results

This folder stores the execution results. There are 2 result files generated at the end of execution - Report.csv and DetailedResults.csv. The Report file contains one row per transaction of a test case and shows which transaction in a test case passed or failed. The 'DetailedReport' file contains specific verification results for both field level verification and table verification.

Report file:

GroupName	Iteration	TestCaseID	TransactionType	TestCaseDescription	StartDate	EndDate	Status
Smoke	Cycle3	TC1	Shakespeare_WS_1	Web Service	7/31/2014 19:35	7/31/2014 19:35	PASS
Smoke	Cycle3	TC1	Shakespeare_WS_1	Web Service	7/31/2014 19:35	7/31/2014 19:35	PASS
Regression	Cycle3	TC2	CreateProduct	Products Menu	9/12/2014 10:48	9/12/2014 10:49	FAIL
Regression	Cycle3	TC2	CreateProduct	Products Menu	9/15/2014 19:36	9/15/2014 19:36	FAIL
Regression	Cycle3	TC2	CreateProduct	Products Menu	9/15/2014 19:39	9/15/2014 19:39	PASS
Regression	Cycle3	TC2	PAUSE		9/15/2014 19:39	9/15/2014 19:39	

Detailed Results file:

Iteration	TestCaseID	TransactionType	CurrentDate	RowType	Status	PassCount	FailCount	Comment
Cycle3	SC1_TC1	AddEntry	10/10/2014 9:37	Field: VerifyHeader	FAIL	0	1	FAIL Imran Deepak
Cycle3	SC1_TC1	UpdateEntry	10/10/2014 9:37	Field: VerifyHeader	PASS	1	0	VIP Database Test Web Application

## 5. EXECUTION

There are two modes of execution: Interactive and Batch mode. The Config.xls parameter 'USERINTERACTION' determines the mode of execution.

### 5.1 Interactive Mode

If 'USERINTERACTION' is set to "TRUE", execution happens in interactive mode. Alerts are displayed on screen in case of any issues and execution does not proceed unless a response is provided to the alerts.

### 5.2 Batch Mode

If 'USERINTERACTION' is set to "FALSE", execution happens in non-interactive mode. In case of any issues during execution, the error message is logged in the result files and the execution proceeds with the next test case. This mode is not recommended till the automation suite has reached some maturity or stability.

Division : Global Testing CoE	Doc. Name : Swift - Selenium User Guide	Version :2.1
Document Classification: NA	Date: 18-Jan-2016	Page : 20 of 23



## 6. BDD SUPPORT

The Config.xls parameter 'BDD\_SUPPORT' governs the support for BDD in Swift. There are two ways you can utilize the BDD support

### ➤ Default BDD

In this mode, you can pass the data to the test scenarios from the feature file itself as shown below.

```
1 Feature: User Creation Feature File
2 @SmokeTest
3 Scenario: User Creation scenario test for User1
4
5 Given navigate to Ranorex page
6 When user enters "John" as firstname and "Patric" as lastname
7 Then User "John" as firstname and "Patric" as lastname should get added
8
```

### ➤ BDD with Swift

In this mode, execution happens in the Swift way only instead of MainController.xls; test scenario flow gets driven by feature files. We have to specify Transaction/Keyword name in angular brackets i.e. <Login>.

```
1 Feature: Smoke Test Feature
2 @SmokeTest
3 Scenario: <SC1_TC1> Test data entry into sample text boxes
4 Given User on Tutorials Page
5 When User enters data into <HtmlEditText>
6 Then Verify user details
7
8 @SmokeTest
9 Scenario: <SC1_TC2> To test check boxes data
10 Given User navigates to check boxes page to test <HtmlCheckBoxes>
11
12 @SmokeTest
13 Scenario: <SC1_TC3> To test Select Boxes
14 Given User Navigates to Select Boxes page
15 Then Enter the selection option from list box <HtmlSelectBox>
```

### 6.1 BDD ON

If 'BDD\_SUPPORT' is set to "YES", support for BDD is turned on and execution happens with the feature files mentioned in CucumberRunner.java. There are 3 parameters which assist in the BDD execution.

Division : Global Testing CoE	Doc. Name : Swift - Selenium User Guide	Version :2.1
Document Classification: NA	Date: 18-Jan-2016	Page : 21 of 23



```
@CucumberOptions(plugin = {  
    "pretty",  
    "html:target/cucumber",  
    "json:target_json/cucumber.json",  
    "junit:target_junit/cucumber.xml"  
}, features = {"Feature/test.feature"}, glue={"swift.cucumber"}, tags="@SmokeTest")
```

### a. features

This parameter defines which feature files to be executed. We can specify comma separated feature files as well the entire directory to pick the feature files for the execution.

E.g. features = {"Feature/test1.feature", "Feature/test2.feature"}

### b. glue

We need be careful while specifying this parameter as this defines the package from which the definitions of feature files to be read so if you are specifying a particular feature file for the execution then appropriate package should be specified for the glue otherwise execution will fail saying no matching step definitions found.

E.g. glue= {"swift.cucumber"}

### c. tags

This parameter defines which test scenarios to be executed from the feature files. We can specify comma separated tags for the execution.

E.g. tags= {"@SmokeTest", "@SanityTest"}

**Note:** We can install cucumber plugin for eclipse by referring the following link; it works like a syntax highlighter. It highlights all the main syntax in the feature file which makes it more readable.

<http://www.toolsqa.com/cucumber/install-cucumber-eclipse-plugin/>

## 6.2 BDD OFF

If 'BDD\_SUPPORT' is set to "NO", execution happens in normal mode i.e. Test selection made in the MainController file.

Division : Global Testing CoE	Doc. Name : Swift - Selenium User Guide	Version :2.1
Document Classification: NA	Date: 18-Jan-2016	Page : 22 of 23



#### AMENDMENT HISTORY

Version No.	Date	Amendment History	Remarks
0.1	04/11/2014	Draft	Draft version
0.1	07/11/2014	Review	Reviewed
1.0	07/11/2014	Baseline	Baseline version
2.0	12/02/2015	Review	Reviewed
2.1	18/01/2016	Review for BDD addition	Added BDD Support

Division : Global Testing CoE	Doc. Name : Swift - Selenium User Guide	Version :2.1
Document Classification: NA	Date: 18-Jan-2016	Page : 23 of 23