Channel Test Automation Framework

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Contributor | Comments |
| 1.0 | 11/09/2012 | AFour | Initial Draft. |

Contents

[1. Document Intent 3](#_Toc382821756)

[2. Technology Stack 3](#_Toc382821757)

[3. System Overview 4](#_Toc382821758)

[i. Driver Script 4](#_Toc382821759)

[ii. Test Data 5](#_Toc382821760)

[iii. Global Library 5](#_Toc382821761)

[iv. Result Log 6](#_Toc382821762)

[4. Test Scripts Breakdown Structure 6](#_Toc382821763)

[5. Coding Standards 6](#_Toc382821764)

[6. Reports 7](#_Toc382821765)

[7. Delivery Mechanism 8](#_Toc382821766)

[8. Constraints 8](#_Toc382821767)

# Document Intent

The purpose of this document is to describe the framework intended for automating the Mind-Alliance Channels Portal test cases using Selenium WebDriver and Java.

It contains the information about:

* High level framework
* Interaction of components within the framework

# Technology Stack

**Language to be used :** JAVA

**Tool(s) :** Selenium – WebDriver

**IDE**  : Eclipse

**Data Storage** : XML

# System Overview

The automation framework is based on data-independent design keeping the data separate from the automation test script. This data can be configuration data required by the automation scripts as well as the test data. Automation framework allows selective execution of automated test scripts.

Test cases selected for automation would be written in such a way that they are independent of each other. Similarly, the test scripts developed for the test cases are also proposed to be independent of each other so that any script could be selected for execution and the dependency on the other scripts is nullified. *Figure 1 shows the schematic view of the proposed test automation framework*

Test Scripts

Driver Script

Report

Figure 1

Test Data Pool (Test data.xml)

Result Logs

**Global Library**

Configuration Data (config.xml)

DataController

Report Functions

Global Variables

ElementController

Log Functions

UI Actions

## Driver Script

It facilitates selection of the required set of test scripts for test automation and drives the automation according to the execution sequence provided. The Driver Scripts behaves as Test Plan. This test plan will be executed using ANT and JUNIT. Test case sheet which will decide which test cases will participate in the test execution.

## Test Data

It is kept in the ‘Test Data Pool’. Each test case owns a Separate xml. The Test Data <number>.xml is actually the xml file which will have the test data required for execution of the particular test case.

a. There will be one xml file per test case or common across all test cases.

b. The test cases will be independent of each other.

c. The framework will manage ‘Test Result’ logging.

The xml will look like this:

<?xml version=*"1.0"*?>

<Configuration>

<Credentials>

<URL>http://192.168.31.5:8080/channels/login.html</URL>

<ChannelsURL>http://192.168.31.5:8080/channels/login.html</ChannelsURL>

<UserName>admin</UserName>

<PassWord>admin</PassWord>

<Browser>Mozilla Firefox</Browser>

<title>Channels - Sign in</title>

</Credentials>

<TimeOutForFindingElementSeconds>30</TimeOutForFindingElementSeconds>

<TimeOutForFindingElementSecondsForClick>60

</TimeOutForFindingElementSecondsForClick>

</Configuration>

The test data would be read from the xml using the Apache Commons XMLConfiguration.Automation scripts

* There will be one test script file per test case.
* The test scripts can be executed independent of each other.
* The script file will have error/exception handling sections.
* The script file will manage ‘Test Result’ logging.

## Global Library

* **BrowserController** Creates the appropriate browser instance and set the Webdriver of browser.
* **LogFunctions** to log the result of script execution
* **ReportFunctions** to generate the report in html form
* **Configuration** is The Singleton Configuration class contains all the variables that are used throughout the application, and their getter/setter methods.etc.
* **GlobalVariables** This class is used to access the object of the Configuration(SingleTon) class throughout the application..
* **Data Controller**: The DataLocators class contains the methods for manipulating data such as reading test data from CSV file which we can use throughout the application.
* **ElementController**: The ElementController class contains the methods to find element on Web page and wait until the element is not found.
* **UIAutomationException:** User defined exception are defined in this class
* **The Config.xml** would look like:

<?xml version="1.0"?>

<Configuration>

<Credentials>

<URL>http://192.168.31.5:8080/channels/login.html/</URL>

<USERID>priyanka.gurav@afourtech.com</USERID>

<PASSWORD>**3!2@1#r$u%oZ^f&@\***<**/**PASSWORD>

</Credentials>

</Configuration>

The configuration data would be read from the xml using the Apache Commons XMLConfiguration.

## Result Log

It contains script execution logs which can be used to keep track of how the test automation execution behaves. Result of test case will consist of the test case name, pass/ fail status, details of pass/fail and error/exception details.

We also get JUnit infrastructure for logging and reporting

# Test Scripts Breakdown Structure

Test cases will be taken from the Test Case sheet. For test automation, these test cases will have to be decomposed / updated to have the following information and then each test case will have one corresponding automated test script

* Each of the test step or the expected result for a condition will be as per the current implementations of the Channels application.
* Every test case will have a unique test case ID
* For every test step in the test case, there has to be an expected result and a unique step ID. This will help in troubleshooting and maintenance of the scripts.

# Coding Standards

For all the coding we shall follow the coding guidelines as given in:

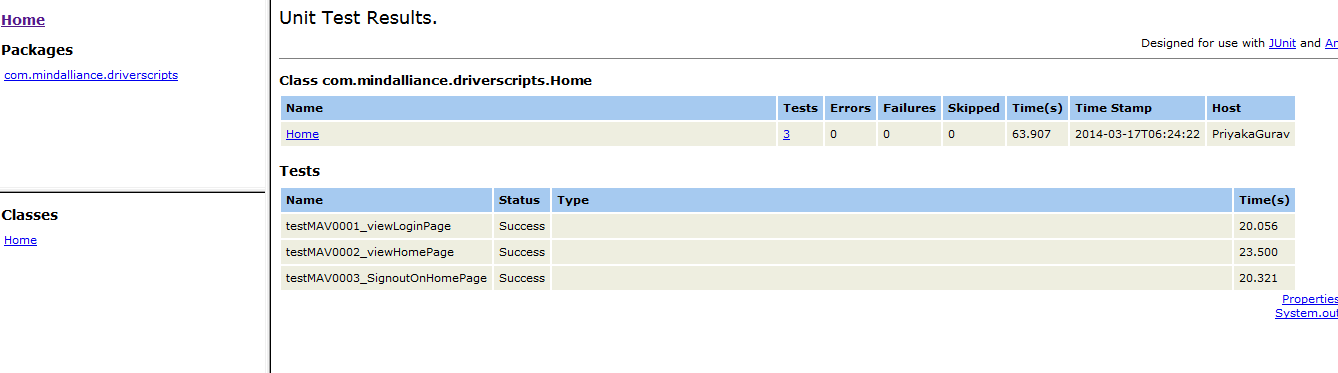
<http://www.oracle.com/technetwork/java/codeconvtoc-136057.html>

# Reports

The reports will be generated in HTML format for better readability. A report would generate for every run of a test case/ test suite and will be available at a pre specified location.

JUnit reports (XML) is also generated and many tool can reads these for different formats

A typical report after running a set of test cases/ suite JUNIT Report would look like:



And HTML report will look like



# Delivery Mechanism

The entire automation project will be delivered as the automation work. The user will need to import that project in the tool and run the scripts (after setting up the automation environment).

# Constraints

* The pre-requisites for execution of automation stated in this document must be followed before executing any automated script.
* There can be UI changes over builds. There is a possibility that the automation will fail if these changes are not incorporated in the script. These failures are due to the possible changes in description of window or its object.
* Any changes in functionality or the UI of the application can cause failures in the automated test scripts hence automation needs to be maintained to an extent. Good design of framework (e.g. tolerant way of finding objects) will minimize maintenance of test automation when UI undergo changes in future.