API Framework Overview

Contents

[1.0 Purpose of the document 2](#_Toc455742744)

[2.0 Introduction about Think Space 2](#_Toc455742745)

[3.0 Automation Approach 2](#_Toc455742746)

[4.0 Technology used to design API Automation Framework. 3](#_Toc455742747)

[5.0 Framework Architecture and Execution Flow 3](#_Toc455742748)

[5.1 Rest Assured Framework 4](#_Toc455742749)

[5.2 Writing Textual Stories: 5](#_Toc455742750)

[5.3 Mapping Textual Scenario Steps to Java Methods via annotations 5](#_Toc455742751)

[5.4 Configuring Java Embeddable classes 6](#_Toc455742752)

[5.5 Junit 6](#_Toc455742753)

[5.6 Run Stories 7](#_Toc455742754)

[6.0 Keyword and its Functionality 8](#_Toc455742755)

[7.0 Reports: 9](#_Toc455742756)

# ****Purpose of the document****

This document gives the details about the API testing framework. The Key subjects addressed in this document include –

* Automation Approach
* Technology/Tools Used
* Framework architectural diagram
* Different component if the framework
* Keyword and its functionality.
* Test Result Format.

# ****Introduction about Think Space****

The Verizon Thing Space Connectivity Management API allows you to add and activate devices, check their status, monitor their usage, and perform other device connectivity management tasks through a RESTful API. You can use the API to add connectivity management to anything from small apps to enterprise software systems, such as enterprise resource planning (ERP), supply chain management and customer service management. The API provides a secure, standards-compliant REST interface to the web services at the Verizon M2M Data Center.

# ****Automation Approach****

Design

Develop

Consolidate

Maintenance

Discover

-Convert the manual test cases to automated scripts

-Review and Inspect

-Validate

-Maintain Test Suite

-Make changes as per requirement.

-Update Test scripts as per changes.

-Design the API Automation Framework

-Create Reusable Libraries

-Design Result format

-Prepare User Guide

-Execute the test suite

-Fix Defects

-Review the Test Cases

-Verify testability

-High Level Automation approach.

3.0.1 Five Phase Test Automation Approach

# ****Technology used to design API Automation Framework.****

The API framework uses java as the core programming language. The framework is build using a combination of Key word approach and BDD.

In Keyword approach there will be only one automation script which will crater to all the different Rest API’s. This approach facilitates domain experts to generate/build automation test cases according to the requirement without bothering about the code. They just need to fill in the data sheet with the required test data with the different API calls like POST, GET & DELETE etc.

The use of BDD (JBehave) is to facilitate the use of Rest Assured framework which is built on the BDD approach. Rest Assured framework provides all the libraries required to parse the JSON response and post the JSON request.

# ****Framework Architecture and Execution Flow****

This automation testing will be performed using following execution flow

* Click startup script to execute desired suite.
* Startup script will create an instance for execution and load story / step file.
* Will load required test data sheet, library files and configuration files.
* Test runner will execute based on story/features scenarios
* Step files will read data from excel to execute test suit from given set of test suits
* Data file specific to test suite will be used for test data.
* Test Step Scripts will be available as functions in .java file.
* While execution framework will generate result in form of excel sheet.
* Once execution of all test cases is over a HTML Result Report will be generated based on the data from the Excel sheet.

Start Execution

Read Excel to execute the test suite with required test data

Load Java Lib, driver, Config Files

Call Test Runner

Generate html Reports

Generate xls reports

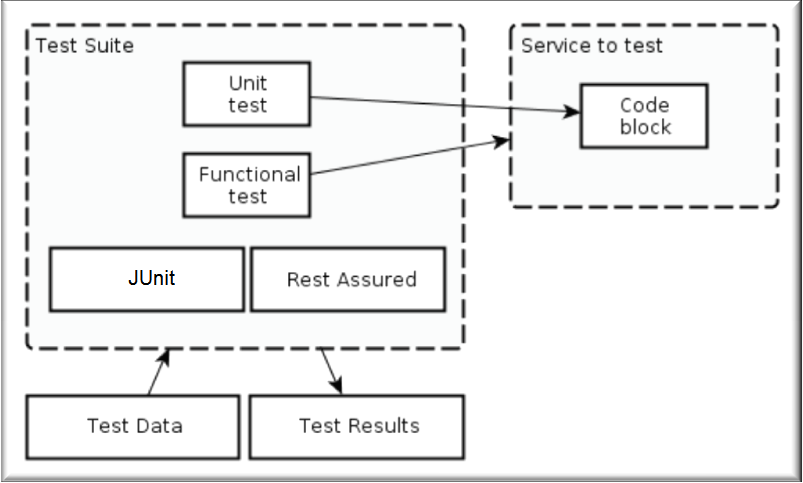
Execute Steps in Stories

5.0.1 API Frame Work Architectural Flow Diagram

There are different components of API framework as mentioned below –

# ****Rest Assured Framework****

It introduces simplicity of testing Rest web services from dynamic languages like groovy or Ruby to Java. It is a fast and lightweight tool used to build Rest assured framework. It can be used to validate both xml and Jason files easily. In addition, Rest Assured can be used in testing web services which requires authentication in order to be invoked.

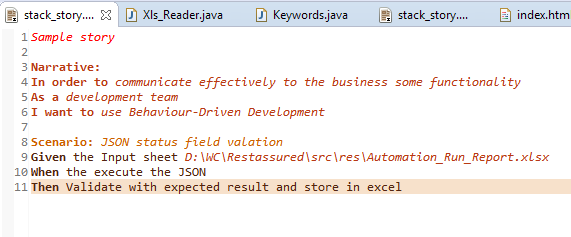


5.1.1 Rest Assured framework design

# ****Writing Textual Stories:****

Behavior –Driven Development encourages to start defining the stories via scenarios that express the desired behavior in a textual format.

The textual scenario should use the language of the business domain and shield away as much as possible the details of the technical implementation. Also, it should be given a name that is expressive of the functionality that is being verified.



5.2.1 Sample Story File

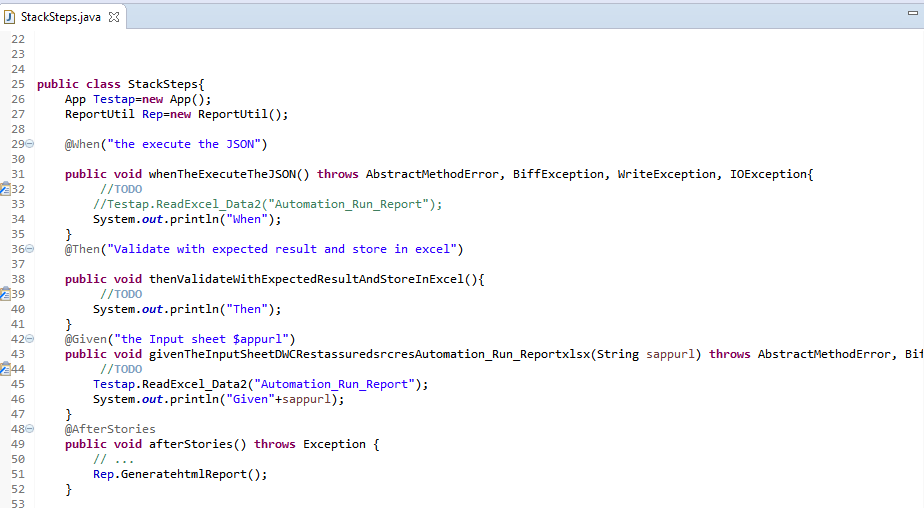
# ****Mapping Textual Scenario Steps to Java Methods via annotations****

JBehave maps textual steps to Java methods via [Candidate Steps](http://jbehave.org/reference/stable/javadoc/core/org/jbehave/core/steps/CandidateSteps.html). The scenario writer need only provide annotated methods that match, by regex patterns, the textual steps.

public interface CandidateSteps

Interface providing the list of step candidates, representing methods annotated with @Given, @When or @Then that can be matched.

It also provides the list of before and after steps, representing methods annotated with @BeforeStories, @AfterStory, @BeforeScenario, @AfterScenario, @AfterStories, @BeforeStory that do not require any matching.



5.3.1 Sample Mapping of Story File with Methods

# ****Configuring Java Embeddable classes****

Dependency: org. jbehave. core. embedder. Embedder

At the heart of the JBehave running of stories lies the [Embedder](http://jbehave.org/reference/stable/javadoc/core/org/jbehave/core/embedder/Embedder.html), which provides an entry point to all of JBehave's functionality that is embeddable into other launchers, such as IDEs or CLIs.

JBehave complements the Embedder with an [Embeddable](http://jbehave.org/reference/stable/javadoc/core/org/jbehave/core/Embeddable.html) which represents a runnable facade to the Embedder.

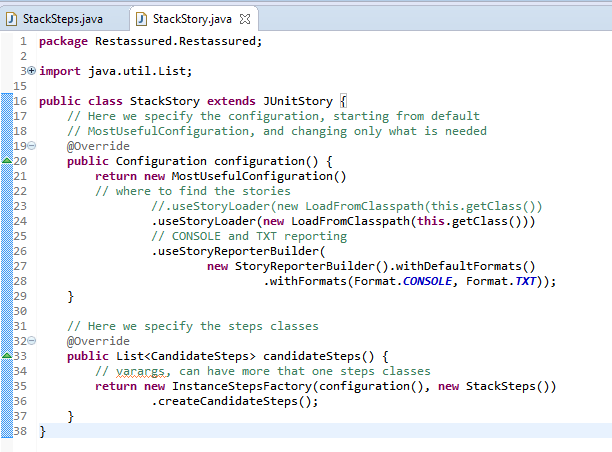
JBehave provides two main Embeddable implementations:

[Configurable Embedder](http://jbehave.org/reference/stable/javadoc/core/org/jbehave/core/ConfigurableEmbedder.html): allows the specification of the Configuration and Candidate Steps.

[Inject able Embedder](http://jbehave.org/reference/stable/javadoc/core/org/jbehave/core/InjectableEmbedder.html): allows the injection of a fully specified Embedder.

# **Junit**

[JUnit](http://junit.org/) is supported out-of-the-box via several Embeddable implementations: [JUnit Stories](http://jbehave.org/reference/stable/javadoc/core/org/jbehave/core/junit/JUnitStories.html) provides a many-to-one mapping with the textual story path.



5.5.1 Sample JUnit File

# ****Run Stories****

You can also run stories using the JBehave Ant tasks or Maven goals which provide a facade around the Embedder functionality.

Stories can be run in different modes:

As embeddable: An embeddable is an abstraction of a running mode, which can run a single story, e.g. if extending Junit Story or can run multiple stories, e.g. if extending Junit Stories. The Embeddable instances allow their configuration.

As paths: once we have specified an instance of an Embedder, with its configuration, we run stories specified by paths.

With annotated embedder runner: An Annotated Embedder Runner is an implementation of the JUnit Runner, which allows annotated specification of the Embedder to be used to run the stories.

With annotated path runner: An Annotated Path Runner is an extension of Annotated Embedder Runner which creates a JUnit suite containing each story path and resolving the path to a human readable form that plays nicely with any IDE that provides JUnit integration

# ****Keyword and its Functionality****

**Input Sheet File Name: Automation\_Run\_Report.xlsx**

**TestSuite Sheet:** This sheet maintains the scenario details for the test

* TestSuiteID: The ID of the test suite to which this test case belongs. A test suite contains detailed instructions or goals for each collection of test cases. Each TestSuite have a ID “TestSuiteID”
* RunMode: If this RunMode flag is set to ''**YES**” then start executed the TestSuite

Keyword: YES / NOS

* SuiteName: It mention the Scenario or Module Name

It’s should be same as the TestCase Input sheet name

E.g. SuiteName- sample then the Automation\_Run\_Report.xlsx file TestCase sheet name should be sample

Description: Details of particular scenario or Module

**Test Case sheet**: This sheet maintains the Test Case Input details

* + TCID: The ID of the test case. Each test case has unique test case ID
  + RunMode: This is execution flag. If this RunMode flag is set to ''**YES**” then start executed the Test cases. RunMode Keyword: YES / NOS
  + WebServices: Webservices to be tested. Keywords: API, UI and Database
  + URL: WebServices URL
  + Details: Description of the test case
  + RequestType: Type of file Request file format. Keyword – JSON/ XML
  + MethodType: Type of Methods. Keyword – POST, GET, PUT & DELETE
  + Param\_Keys: JSON Field name / XML Tag Name.

For POST Method: To Create/Insert new value in the post request. Multiple value insertion used with delimiter |.

For PUT Method: To update value in the put request. Multiple value update used with delimiter |.

* Param\_Values: JSON Value / XML Tag Value.

Corresponding Values for the Param\_Keys.Multiple value used with delimiter |.

* GetOutputKey:

For POST Method: The output of the get requests. And its store into the “Output\_Values” Column

For GET Method: Validate the Get Response. Multiple Field validation used with delimiter |.

* Expected: The expected result of the test.

The Corresponding expected value for the GetOutputKey column.

* SwitchingMode: This is used for Chaining response functionality.

Run time passing parameter from one webservices to another webservices in POST method. Used to condition specific.

Format: Fieldname|ResponseFieldvalue1:TCID1| ResponseFieldvalue2:TCID2

Response Field value validated. If it’s true and start execute the corresponding TCID.

e.g: type|health:TC03|irregation:TC05

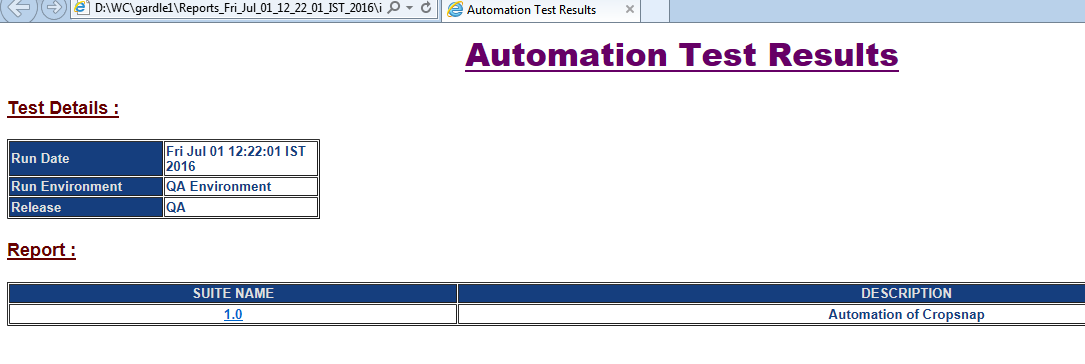
* Status: Status of the Test Case. PASS or FAIL
* Description: Any comments on the test case or test execution
* Output\_Values : Run time Output values updated in the column “Output\_Values”
* Response: Store Webservices Response file path link

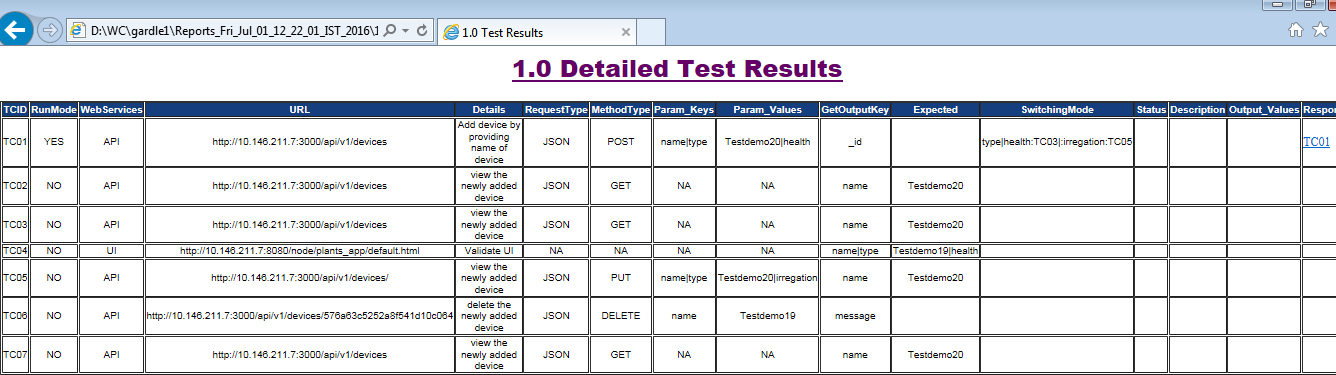
**Automation\_Run\_Report.xlsx**



# ****Reports:****

After the Test Execution the html report is generated in the following format





7.0.1 Html report format

**Sample HTML Reports: **