CT-R1 RCM Automation:

Automation Framework Documentation

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sr No | Version | Date | Author | Remarks |
| 1 | 0.1 | 14-Oct-2019 | Dipti Shetty | Created initial draft of R1 automation guidelines document |
| 2 | 0.2 | 15-Oct-2019 | Vikram Rajagopal | Reviewed and suggested changes |
| 3 | 1.1 | 15-Oct-2019 | Vikram Rajagopal | Document sent to R1 for review |
| 4 | 1.1 | 16-Oct-2019 | Neha Agarwal | Modifications/Additions needed: -   * Explanation of flow between different modules * Video recording for execution of framework * Parallelism details to be incorporated |
| 5 | 1.2 | 18-Oct-2019 | Dipti Shetty | Added - Explanation of flow between modules |
| 6 | 1.3 | 23-Oct-2019 | Dipti Shetty / Rushikesh Deshmukh | Added - Parallelism implementation |
| 7 | 1.3 | 23-Oct-2019 | Rathulya Ravindran | Added – common modules section |
| 8 | 1.4 | 24-Oct-2019 | Rathulya Ravindran | Added – Recording of automation framework |
| 9 | 1.5 | 30-Oct-2019 | Dipti Shetty | Updates as per review comments by Neha |
| 10 | 1.6 | 31-Oct-2019 | Vidushi Shukla | Added – Egit implementation |

Table of Contents

[1. Introduction 3](#_Toc22757083)

[2. Framework Architecture: 3](#_Toc22757084)

[2.1 Explanation of Flow in the Framework 4](#_Toc22757085)

[3. Why Serenity: 5](#_Toc22757086)

[4. Pre-Requisites & Setup: 6](#_Toc22757087)

[5. Coding Guidelines: 8](#_Toc22757088)

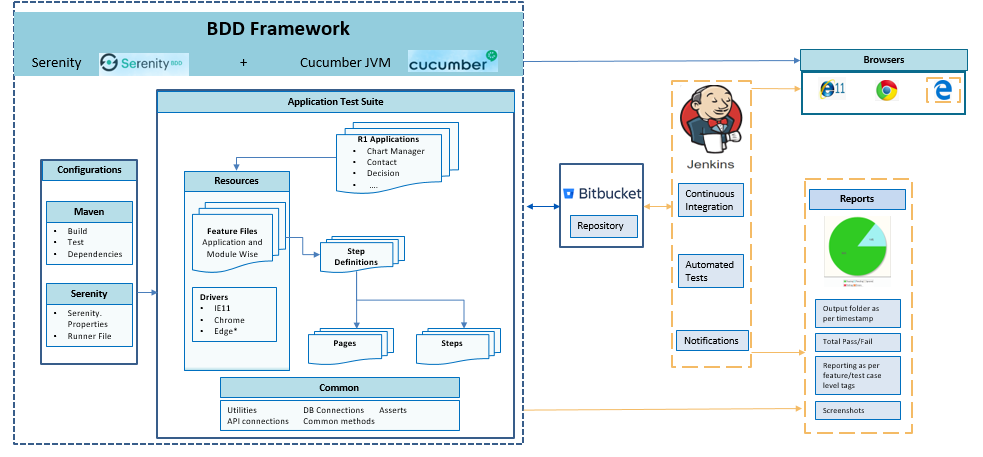
[6. Parallelism: 11](#_Toc22757089)

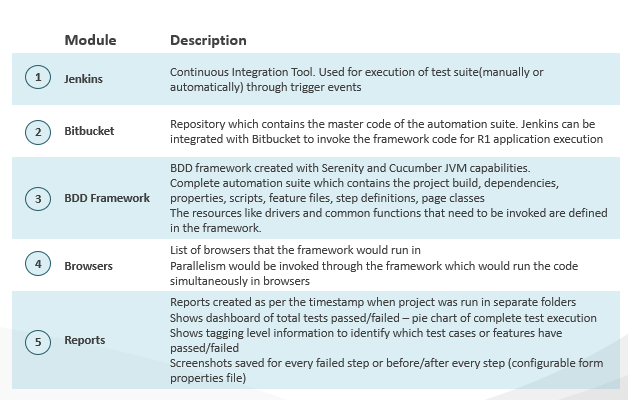
[7. Common Modules: 12](#_Toc22757090)

1. Introduction

The document will help users in understanding the existing framework architecture along with the required setup and coding guidelines to be followed for automating and executing TC’s with current framework.

1. Framework Architecture:





2.1 Explanation of Flow in the Framework

* **Serenity properties** helps us in specifying the browser specifying webdriver driver along with other internal serenities properties [eg. serenity.project.name; webdriver.driver; webdriver.base.url; webdriver.chrome.driver; webdriver.timeouts.implicitlywait; webdriver.wait.for.timeout; serenity.browser.maximized; serenity.resized.image.width; story.timeout.in.secs; etc] required for testcase execution, reporting and many more purposes
* Apache **Maven** provides support for managing the full lifecycle of a test project. Maven is used to define project structure, dependencies, build, and test management. Using pom.xml (Maven) helps us in configuring dependencies needed for building, testing and running code
* All the required browsers drivers are listed inside the framework under resources-drivers which the framework would run in. It would also be called from Serenity Properties file
* Testcases are read from **feature files** and these feature file calls **step definition** for further testcase execution
* **Step definitions** connects with **pages** and **steps** folder for resolving the methods called in **step definition** files. Basically, these methods have all the **business logics** written required for executing the testcases
* The **generic** or common logic which are commonly used across the projects are written inside **common** folder, hence many a times call is done to the **common** folder for extracting those **business logics.** Things like DB connection calls, API calls, asserts which can be commonly used across the project are written here
* The code from Bitbucket is pulled from the repository of the individual project created and then worked upon locally. Once all changes are done (self and peer reviewed) the final automation code changes are pushed into Bit Bucket repository and onto the development branch
* Also, in future implementation, the latest framework will be pulled in from Bit Bucket repository for executing those testcases on Jenkins and final in-depth reporting will be done with the help of serenity framework
* Recording of project walkthrough:

<https://mcrschicago.sharepoint.com/:v:/r/sites/CT-R1QAAutomationTeam/Shared%20Documents/KTs%20and%20Demo/ProjectWalkthrough.wmv?csf=1&e=JQ8MVc>

1. Why Serenity:

There are many BDD frameworks in the market – out of which Serenity is one such framework which is robust and intuitive. Few qualities of Serenity framework that, in comparison with other frameworks: -

* Reporting is one of the Serenity’s fortes. Serenity not only reports on whether a test passes or fails, but documents what it did, in a step-by-step narrative format that includes test data and screenshots for web tests as well
* Serenity BDD plays well with JUnit as well as more specialized BDD frameworks such as Cucumber and JBehave.
* It has strong support for both web testing with Selenium, and API testing using RestAssured.
* It has defined PageObject class which hides WebDriver logic in “Page objects” due to which the number of coding lines are reduced
* Serenity helps user in writing a clean and more maintainable automated acceptance and regression tests faster
* Key advantage of using Serenity BDD in framework is that user doesn’t have to invest time in building and maintaining own automation framework.
* User can integrate Serenity with requirements stored in an external source (such as JIRA or any other test cases management tool), or just use a simple directory-based approach to organize the requirements

1. Pre-Requisites & Setup:

|  |  |
| --- | --- |
| 1 | Software Installation |
| 2 | Environment variables setup for Java, Maven, Git |
| 3 | Fetch project from Bit Bucket |
| 4 | Import project from eclipse |
| 5 | Cucumber plugins in Eclipse |
| 6 | Running Tests from eclipse |
| 7 | Running tests from command line |

* Software Installation
  + Below are the software’s that need to be installed in your local machine:
* Eclipse IDE for Java Developers- Eclipse Oxygen 4.7
* Maven 3.5
* Java JDK 8
* Drivers (Chrome, IE, Mozilla Firefox) drivers to be version compatible with the browser
* Git 2.21
* Environment Variables Setup
  + Set environment variables for JAVA
    - Install Java JDK 8
    - Go to Control Panel and search Environment Variables
    - In the System Properties window, go to “Advanced tab” and click “Environment Variables” button
* In the Environment Variables window, under “System Variables” section, create new variable “JAVA\_HOME” and value as “C:\Program Files\Java\jdk1.8.0\_212”. Click OK
* In the Environment Variables window, under “System Variables” section, edit the “PATH” variable and add value as “C:\Program Files\Java\jdk1.8.0\_212\bin”
* Open command line and type “java -version” to verify installation is complete
  + Set environment variables for Maven
    - Place Maven folder in local directory
    - In the System Properties window, go to “Advanced tab” and click “Environment Variables” button
    - In the Environment Variables window, under “System Variables” section, create new variable “MAVEN\_HOME” and set value as path of the Maven folder “C:\test\softwares\apache-maven-3.5.0”. Click OK
    - Select “PATH” variable and click on “Edit”.
    - Click on “New” and enter value “%MAVEN\_HOME%\bin”
    - Click OK and save the changes
    - Open command prompt and type “mvn -version” and verify whether installation is complete
  + Configure git ( for using GitBash):
    - When you've successfully started the installer, you should see the **Git Setup** wizard screen. Follow the **Next** and **Finish** prompts to complete the installation.
    - In the System Properties window, go to “Advanced tab” and click “Environment Variables” button
    - In the Environment Variables window, under “System Variables” section, edit the “PATH” variable and add value as “C:\Program Files\Git\cmd”
    - Open command prompt and run the below commands with your username and password
      * git config --global user.name "Emma Paris"
      * git config --global user.email “[eparis@atlassian.com](mailto:eparis@atlassian.com)”
  + Configure git ( As Eclipse plugin):
    - Eclipse  Help  Eclipse Market place  Search & Install Egit plugin
* Fetch the project from Bit Bucket
  + Create local folder and open command prompt inside that folder location
  + Enter command “git init”
  + Enter command “**git clone /path/to/repository**”

* Import Project to eclipse from local repository (if used gitbash)
* Open Eclipse IDE
* Click on File  Import  Under “Git” select “Projects from Git”  Click Next  Select “Existing local repository’  Click Next
* Click “Add” button  Browse to the repository cloned from Git  Select and click Finish
* The project is successfully imported to Eclipse
* Import Project to eclipse using Egit Plugin
* Open Eclipse IDE
* Click on File  Import  Under “Git” select “Projects from Git”  Click Next  Select “Clone URI”  Click Next
* Provide the git URL from bitbucket repo and password  Click Next  Select the repository to be cloned from Git  Click Next
* Chose the local directory to pull the project  Click Finish
* Cucumber plugins in Eclipse
  + In Eclipse, click on “Help” and select “Eclipse MarketPlace”
  + Enter “cucumber” in the Find options and click “go”
  + The search result fetches “Natural 0.7.6” and “Cucumber Eclipse Plugin 0.0.23”
  + Install both the plugins one after other
  + Restart eclipse
  + The changes would be reflected in the feature files.

**Note**: These set of plugins provides a smooth experience in editing and maintaining BDD/ATDD files.

* Running Tests from eclipse
  + In Eclipse, expand the project folder and under “src/test/java” package, expand the “runner” folder
  + Open the runner file that needs to be executed
  + Right click  Run As Click on “Maven build”
* Running tests from command line
  + Open command line and go to Project Folder using command:
    - $ cd projectname
  + Enter command “mvn clean verify” – All the tests will be executed one after other
  + To run specific tags use command “mvn clean verify -Dcucumber.options="--tags @UserLogin”
  + Here @UserLogin is the tag mentioned at the scenario in the feature file

1. Coding Guidelines:

|  |  |
| --- | --- |
| **#** | **Check Points** |
| **Section 1** | **Style** |
| 1.1 | Define a proper folder structure to store all shared components: a) log b) Results c) Test data d) Test script |
| 1.2 | Always use Tab to indent instead of Spaces. Tab character should be set to 4 spaces. |
| 1.3 | Use whitespace and newlines to ensure readability, but don't overdo it. |
| 1.4 | Code should be indented well to clearly differentiate code and loops. |
| 1.5 | Use Calibri font throughout test data sheet and test script sheet with font size set to 10 |
| 1.6 | Column header in test data sheet and test script sheet should be bold |
| 1.7 | Used range in test data sheet and test script sheet should have cell border set to "All Border" |
| 1.8 | Each test script should logout of the application to ensure that it has no impact on any other subsequent test scripts. Flush all the objects and call the logout reusable script. |
| **Section 2** | **Naming Convention** |
| 2.1 | All variables used in scripts should be declared at top of script. |
| 2.2 | Variable names should follow variable naming conventions. |
| 2.3 | "To enhance readability and consistency, use the following prefixes with descriptive names for variables in your java code. No Subtype Prefix Example. boolean blnFound Double dblTolerance Integer intQuantity String strFirstName" |
| 2.4 | "Object Naming Conventions In order to make the object easily searchable, and to avoid duplicity, it has to be recorded with a meaningful name for example to represent a Close button on Edit/View page, it can be represented as button\_EditView\_Close (ObjectType\_Page\_ObjectName)." |
| 2.5 | Constants, if used, need to be implemented as variables and distinguished from other variables using all uppercase characters. Multiple words were separated using the underscore (\_) character |
| 2.6 | Function names should follow pascal casing. |
| 2.7 | Function name should be such that they briefly describe the purpose of the function. |
| 2.8 | All test data (input data and data to be verified) should be fetched from a centralized repository of external test data file and should be parameterized |
| 2.9 | Reusable component name should be chosen in such a way that it is clear from the name what functionality that component is going to performs. |
| 2.10 | Any external files if used should follow a standard naming convention which will specify the purpose of the file. |
| 2.11 | Use intention revealing names and pronounceable, searchable and readable English names. |
| **Section 3** | **Commenting** |
| 3.1 | Don't Comment Unused Code, Delete Dead Code |
| 3.2 | Comments should describe the code in simple, and clear terms |
| 3.3 | Avoid obvious comments that provide no new information. |
| 3.4 | Every class should have detailed description for its purpose and methods it provides. Also, every method other than small ones like getter setter etc. should have comments to indicate their purpose |
| **Section 4** | **Design** |
| 4.1 | Always Favor the Explicit over the Implicit. |
| 4.2 | "Each automated test case should be independent: a) Test Case should not depend on any test case which is running before this test case in batch b) Test Case should not depend on any test data which is created by any other test case" |
| 4.3 | Script should not have any hard-coded path in the script. |
| 4.4 | Every verification step should be automated. |
| 4.5 | In case any step/ verification is not feasible for automation, TC should be blocked for automation or accordingly intimated to manual team. |
| 4.6 | Code should be optimal. |
| 4.7 | No infinite loops in the code. |
| 4.8 | No unnecessary declarations of variables. |
| 4.9 | Release memory for objects created. |
| 4.10 | Avoid using hardcoded wait time for Sync purpose |
| 4.11 | Ensure appropriate reusable components are called in sequential manner as similar lines of manual test case |
| 4.12 | Error handling should be done at script level for all expected errors that user is aware of. |
| 4.13 | All the browser windows closed successfully before a fresh script starts and close after the script ends |
| 4.14 | Avoid lines longer than 80 characters, since they're not handled well by many terminals and tools |
| 4.15 | Initialization to avoid NPEs (Null Pointer Exceptions) |
| **Section 5** | **Logging** |
| 5.1 | Proper and consistent logging format should be used |
| 5.2 | The error messages should be clear and detailed |
| 5.3 | Every manual step and its corresponding verification should be reported and should have a Pass/ Fail in the report |
| 5.4 | Logging functions should log precise details on "what step worked and what did not work", but in a very user-friendly simplified message. |
| 5.5 | Any unhandled exception should be caught. A user-friendly message should be logged indicating some problem with a UI object followed by the short exception message |
| **Section 6** | **Reusability** |
| 6.1 | There should not be any repeated code in the script. Appropriate functions or reusable actions should be created for the same. |
| 6.2 | Each automation script should be independent of the other automation script. |
| 6.3 | There should be no manual intervention required for executing the automation scripts |
| **Section 7** | **Exception Handling** |
| 7.1 | Always write a catch block for handling exceptions |
| 7.2 | Make sure to add a logging message or the stack trace in the catch block |
| 7.3 | Avoid catching the general exception and have a specific exception |
| 7.4 | The cleanup code should be added in the <finally> block. This provides a single location for the cleanup and it’s guaranteed to run |

1. Parallelism:

The framework created can be configured to have the test cases run simultaneously in different environments (care/imh) and in multiple threads of browser. The project can be run via command prompt using below mentioned commands where we can specify the environment on which the scripts need to be run. Different commands for different environments can be run using a batch file. This batch file can be integrated with Jenkins jobs to run at a scheduled time – which will automatically trigger the commands and run tests in parallel.

By this below implementation the test cases will run in two different environments, with facility configured from the configuration file, and will run in parallel threads in each environment – thus attaining parallelism within and cross environments through the same framework.

* Specify dependency of ‘maven-failsafe-plugin’ in pom file
* Serenity.conf file is used which holds details of multiple environments for parallel execution
* Single command can be used for running TC’s parallelly on single environment

eg. mvn verify -Dcontext=PNTX -Denvironment=care -Dinjected.tags=”Environment:care”

Details of above command:

* -Dcontext=PNTX: This attribute holds any meaningful text and it can be related to environment
* -Denvironment=care: This attribute helps serenity to select environment specific configuration for parallelism which are specified in serenity.conf file
* -Dinjected.tags=”Environment:care”: This attribute helps to identify the environment where execution is performed in reports
* BatchCommand file contains multiple execution commands for multiple environments
* Runner files holds details of the TC’s with tag names that will run in parallel
* Depending on threads the number of runner files need to be specified with the respective tags of test cases to be run in parallel execution in single environment and these details needs to be specified in pom file as below:



1. Common Modules:

|  |  |  |
| --- | --- | --- |
| Java Class Name | Method | Purpose |
| DBConn.java | serverConn()   * input: Expects server host, database name and query * output: DB connection is established with database | Common method to make DB connections |
| CommonMethods.java | loadQuery()   * input: file name, query name * output: query value | Common method used to load the DB query from properties file |
| loadProperties()   * input: Serenity property name * output: property value | Common method used to fetch specific property from Serenity.properties |
| handleMultipleWindows()   * output: returns main window | Common method used to handle multiple tabs |
| getRandom()   * input: size of list * output: returns random integer | Common method to fetch a random integer, which is further used to select a random element from a list |
| generateTickleDate()   * input: current date, number of days * output: returns tickle date | Method used to calculate TickleDate |
| fastestRandomStringWithMixedCase()   * input: accepts integer length * output: random string of input length | Method used to create random String |
| hoverOverElement()   * input: webelementfacade * output: performs hover action | Method used to hover on specific element |
| Hooks.java |  | Common file where the @before and @after criteria can be mentioned. In this project, @before method is used to implement login specific to testcase. |
|  | log()   * input: string to be printed in report | Common method used to print message in the Serenity report. This method can be placed in any Steps class |
| Login.java | clickSignInLink() verifyUsernameTextBox() enterUsername()   * input: accepts user name   enterPassword()   * input: accepts password   loginBtnClick() isProceedLinkVisible() clickOnProceedFurther() | These methods in Login.java is used across projects to log into the application. |