CUCUMBER MAVEN

INTRODUCTION:

- Cucumber is a testing tool that supports Behavior Driven Development (BDD) framework. It defines application behavior using simple English text, defined by a language called Gherkin.
- Cucumber allows automation functional validation that is easily read and understood. Cucumber was initially implemented in Ruby and then extended to Java framework. Both the tools support native JUnit.

PREREQUISITES:

- Java(Latest)
- Eclipse
- Jenkins .war
- Smartest integration
- Tomcat
- Junit

CUCUMBER WORK FLOW:

- Cucumber reads the code written in plain English text in the feature file. then it finds the exact match of each step in the step definition.
- Runner file will pick the feature file and respective step definitions and then it will execute the scripts.

ADVANTAGES OF CUCUMBER:

- Cucumber supports different languages like Java.net and Ruby.
- It acts as a bridge between the business and technical language. We can accomplish this by creating a test case in plain English text.
- It allows the test script to be written without knowledge of any code, it allows the involvement of non-programmers as well.
- It serves the purpose of end-to-end test framework unlike other tools.
- Due to simple test script architecture, Cucumber provides code reusability.

FEATURE FILE:

Sample Feature file:

```
@TestAmazon
Feature: Login Amazon
      As a user login to site
  verify the details
    Scenario: Login to Amazon
        Given I am in Amazon landing page "http://www.amazon.in/"
        When I click on Sign in
        And I enter "Email" and "Password"
        And click on login button
```

A simple feature file consists of the following keywords/parts :

- **Feature** − Name of the feature under test.
- **Description** (optional) − Describe about feature under test.
- **Scenario** − What is the test scenario.
- **Given** − Prerequisite before the test steps get executed.
- **When** − Specific condition which should match in order to execute the next step.
- **Then** − What should happen if the condition mentioned in WHEN is satisfied.

Step Definitions:

Steps definition file stores the mapping between each step of the scenario defined in the feature file with a code of function to be executed. So, now when Cucumber executes a step of the scenario mentioned in the feature file, it scans the step definition file and figures out which function is to be called.

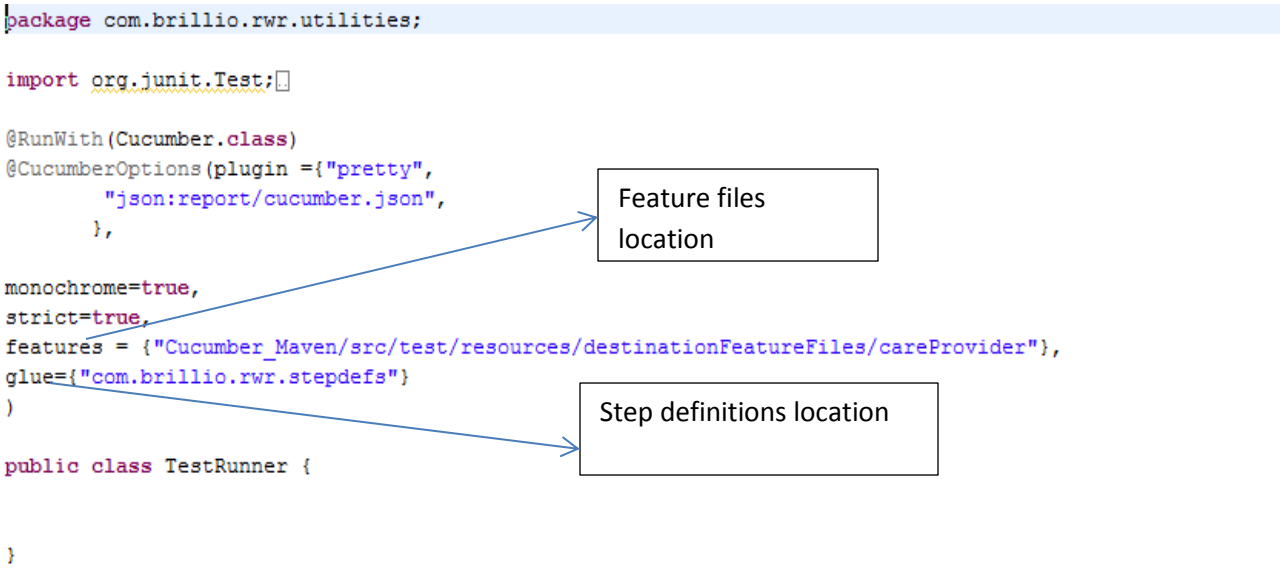
Sample Step definitions:

```
@Given("^I am in Amazon landing page \"([^\"]*)\"$")
public void i_am_in_amazon_landing_page_something(String strArg1) throws Throwable {
    System.out.println("User has logged in to Amazon page");

}

@When("^I click on Sign in$")
public void i_click_on_sign_in() throws Throwable {
    loginpage.clickOnAmazonSignin();

}
```

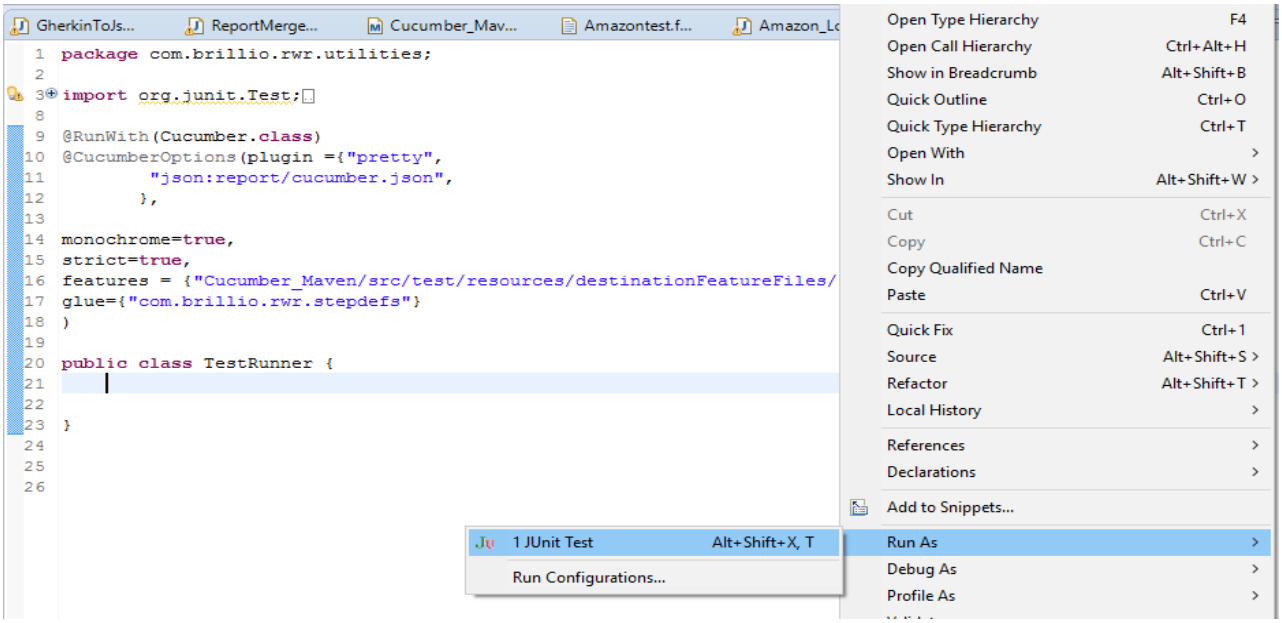
TRIGGERING THE EXECUTION:

a) Triggering execution from Runner file:

Runner file looks as follows:



To trigger the execution Run Runner file as Junit.

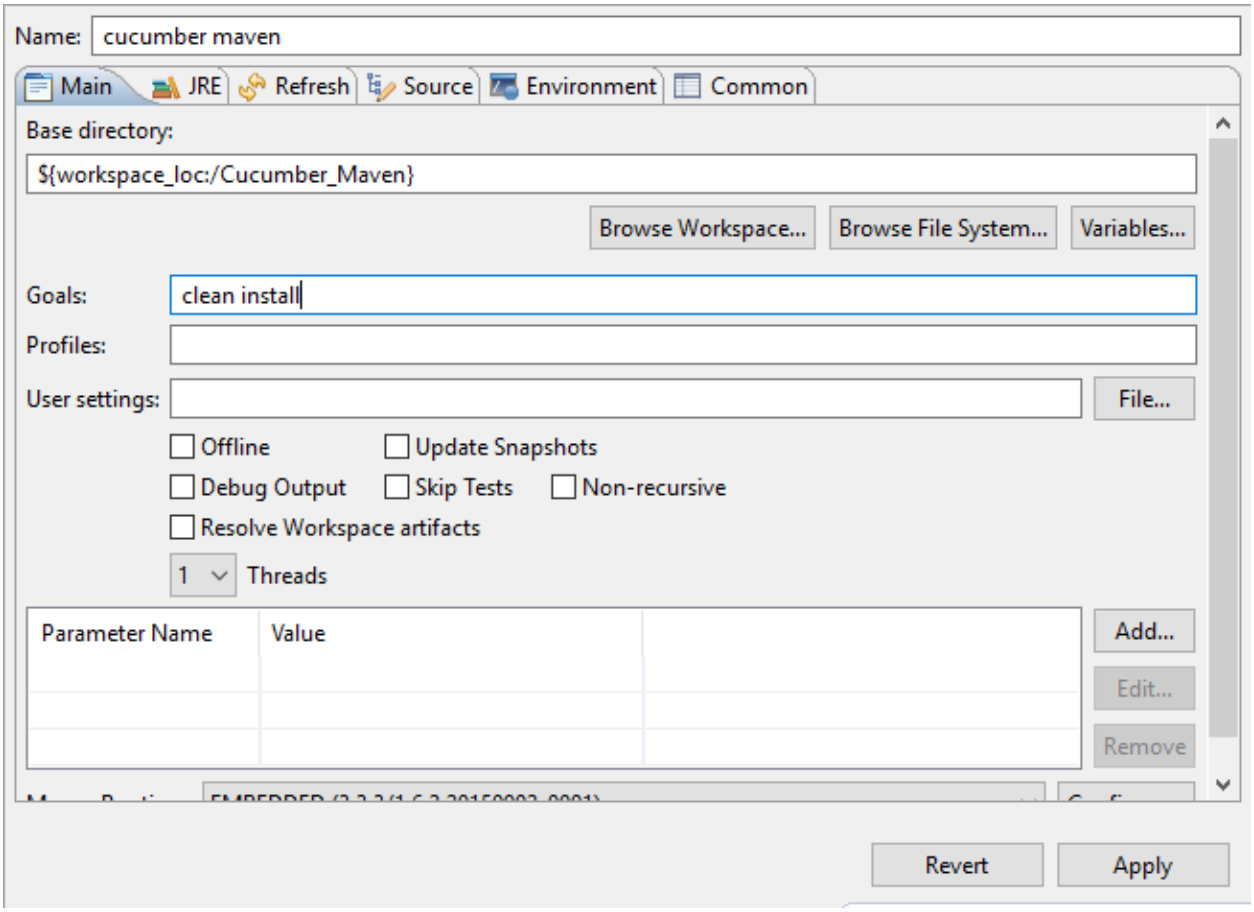


b) Triggering execution from POM.XML file

- Add cucumber-jvm-parallel-plugin. To trigger execution from pom.

```
<artifactId>cucumber-jvm-parallel-plugin</artifactId>
<version>2.0.2</version>
<executions>
    <execution>
        <id>generateRunners</id>
        <phase>validate</phase>
        <goals>
            <goal>generateRunners</goal>
        </goals>
        <configuration>
             <!-- Mandatory -->
            <!-- comma separated list of package names to scan for glue code -->
            <glue>com.brillio.rwr.stepdefs</glue>
            <!-- These are optional, with the default values -->
            <!-- Where to output the generated tests -->
            <outputDirectory>${project.build.directory}/generated-test-sources/cucumber</outputD
            <!-- The directory, which must be in the root of the runtime classpath,
                containing your feature files. -->
            <featuresDirectory>src/test/resources/destinationFeatureFiles/careProvider</features
            <!-- Directory where the cucumber report files shall be written -->
            <cucumberOutputDir>target/cucumber-parallel</cucumberOutputDir>
            <!-- comma separated list of output formats -->
            <format>json</format>
            <!-- CucumberOptions.strict property -->
            <strict>true</strict>
            <!-- CucumberOptions.monochrome property -->
            <monochrome>true</monochrome>
            <!-- The tags to run, maps to CucumberOptions.tags property -->
            <tags></tags>
            <!-- If set to true, only feature files containing the required tags
                shall be generated. -->
            <filterFeaturesByTags>false</filterFeaturesByTags>
```

- To trigger the execution : run following goal : **“clean install”**

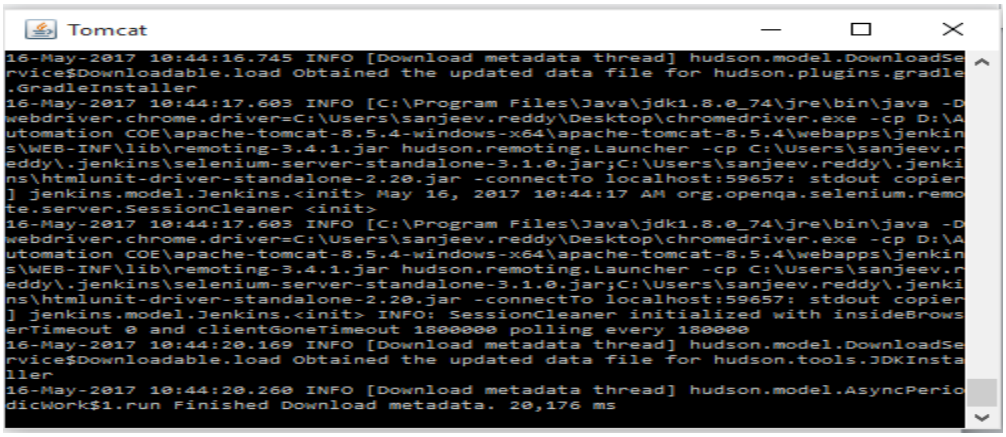


c) Triggering execution from Smartest + Jenkins:

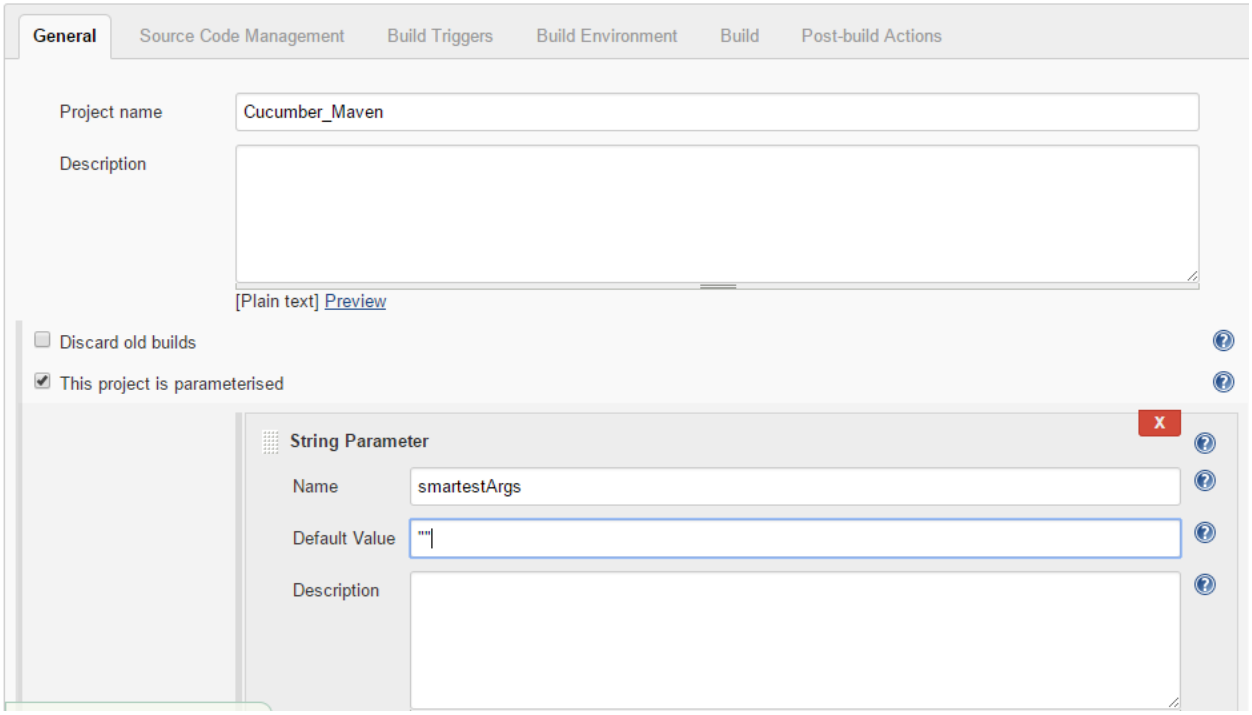
Smartest setup should be ready

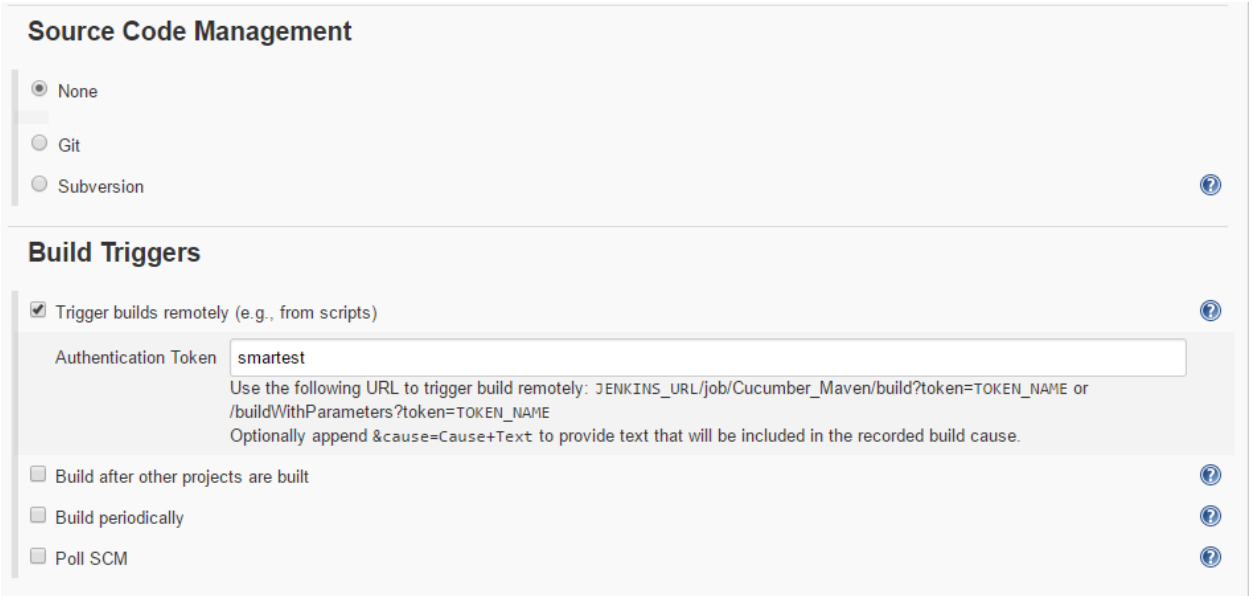
Jenkins set up :

- Download Jenkins .war file and place it in Tomcat webapps folder, Then Run Tomcat server.



- Open browser and go to:" **localhost:port/Jenkins".**
- In .jenkins folder create workspace folder and paste entire project source code.
- Create Jenkins job with following configurations.





Cucumber integration with smartest:

- Smartest will provide arguments in decoded format , To get encoded arguments we are using “smartestUtils-jar-with-dependencies.jar” .

```
smartestArgs = URLDecoder.decode(smartestargs, "UTF-8");
Gson gson = new Gson();
SmartestArgsVO smartestArgsVO =  gson.fromJson(smartestArgs, SmartestArgsVO.class);
System.out.println("Scenario names are:"+smartestArgsVO.getTestCaseNames());
String[] scenario_names = getTestcaseArray(smartestArgsVO.getTestCaseNames());
```

- We will get String array of scenarios names:

- Then “**cucumber_Runner.java”** file will search all feature files in **“/Cucumber_Maven/src/test/resources/SourceFeatureFiles/careProvider**” and then searches for scenario names and generates separate feature file with same scenario name in “**/Cucumber_Maven/src/test/resources/destinationFeatureFiles/careProvider**” folder.

- Then “Cucumber-jvm-parallel” plugin will pick the feature files from “**/Cucumber_Maven/src/test/resources/destinationFeatureFiles/careProvider**” folder and start the execution.

- Jenkins job got triggered after triggering execution from smartest, and smartest will send arguments , by using those arguments Jenkins will trigger execution.

Jenkins build console looks as follows:

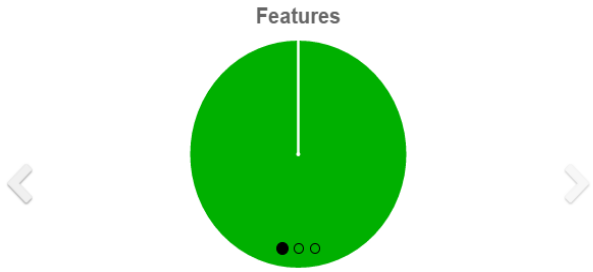

```
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running Parallel01IT
log4j:WARN No appenders could be found for logger (com.automation.framework.core.DriverScript).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
Starting ChromeDriver 2.29.461591 (62ebf098771772160f391d75e589dc567915b233) on port 20255
Only local connections are allowed.
User has logged in to Amazon page
1 Scenarios (1 passed)
4 Steps (4 passed)
1m2.748s

Tests run: 0, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 66.665 sec - in Parallel01IT
Running Parallel02IT
log4j:WARN No appenders could be found for logger (com.automation.framework.core.DriverScript).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
Starting ChromeDriver 2.29.461591 (62ebf098771772160f391d75e589dc567915b233) on port 34608
Only local connections are allowed.
User has logged in to Amazon page
1 Scenarios (1 passed)
3 Steps (3 passed)
0m56.968s
```

REPORTS:

After Execution report looks like:

Features Statistics

The following graphs show passing and failing statistics for features



	Steps						Scenarios			Features	
Feature	Passed	Failed	Skipped	Pending	Undefined	Total	Passed	Failed	Total	Duration	Status
Login Amazon	4	0	0	0	0	4	1	0	1	33s 438ms	Passed
Login Amazon	3	0	0	0	0	3	1	0	1	28s 179ms	Passed
2	**7**	**0**	**0**	**0**	**0**	**7**	**2**	**0**	**2**	**1m 1s 617ms**	
	100.00%	**0.00%**	**0.00%**	**0.00%**	**0.00%**		**100.00%**	**0.00%**			**100.00%**

REFERENCES:

1. Cucumber Tutorials: http://www.tutorialspoint.com/cucumber/