What is Cucumber Options?

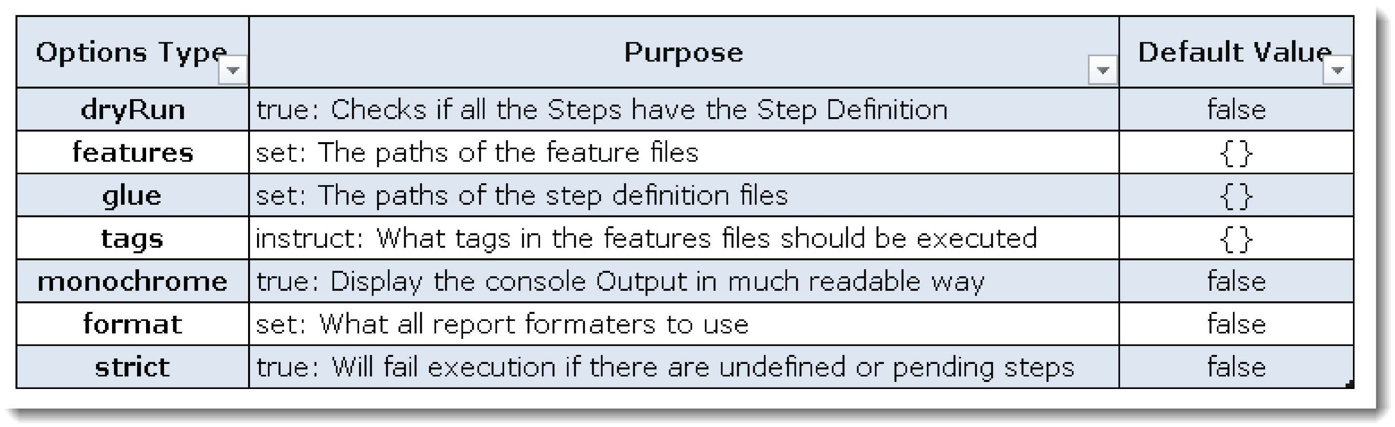
In layman language, ***@CucumberOptions*** are like property files or settings for your test. Basically ***@CucumberOptions*** enables us to do all the things that we could have done if we have used cucumber command line. This is very helpful and of utmost importance, if we are using IDE such eclipse only to execute our project. You must have noticed that we set a few options in the '*TestRunner*' class in the previous chapter.

TestRunner Class

```
package cucumberTest;

import org.junit.runner.RunWith;
import cucumber.api.CucumberOptions;
import cucumber.api.junit.Cucumber;

@RunWith(Cucumber.class)
@CucumberOptions(
                features = "Feature"
                ,glue={"stepDefinition"}
                )

public class TestRunner {

}
```

So in the above example, we have just set two different *Cucumber Options*. One is for *Feature File* and the other is for *Step Definition* file. We will talk about it in detail now but with this, we can say that @*CucumberOptions* are used to set some specific properties for the Cucumber test.

Following Main Options are available in Cucumber:



Options Type	Purpose	Default Value
dryRun	true: Checks if all the Steps have the Step Definition	false
features	set: The paths of the feature files	{}
glue	set: The paths of the step definition files	{}
tags	instruct: What tags in the features files should be executed	{}
monochrome	true: Display the console Output in much readable way	false
format	set: What all report formaters to use	false
strict	true: Will fail execution if there are undefined or pending steps	false

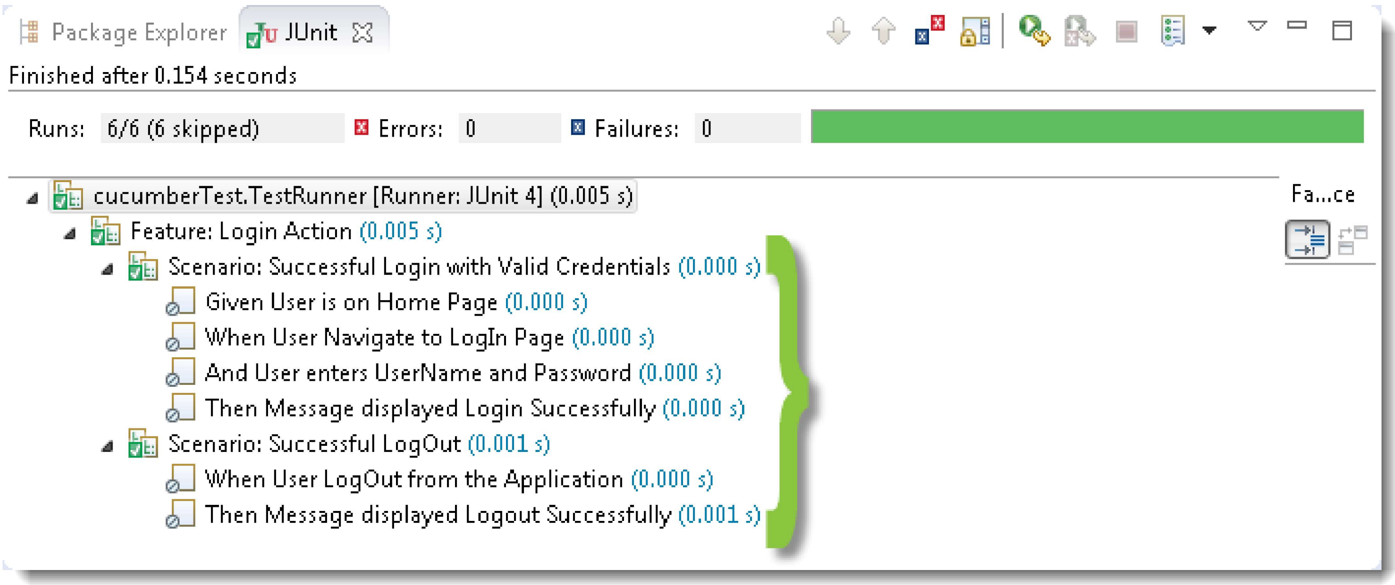
Dry Run

dryRun option can either set as ***true*** or ***false***. If it is set as true, it means that Cucumber will only check that every *Step* mentioned in the *Feature File* has corresponding code written in *Step Definition* file or not. So in case any of the functions are missed in the *Step Definition for any Step in Feature File*, it will give us the message. For practice just add the code '*dryRun = true*' in ***TestRunner*** class:

TestRunner Class

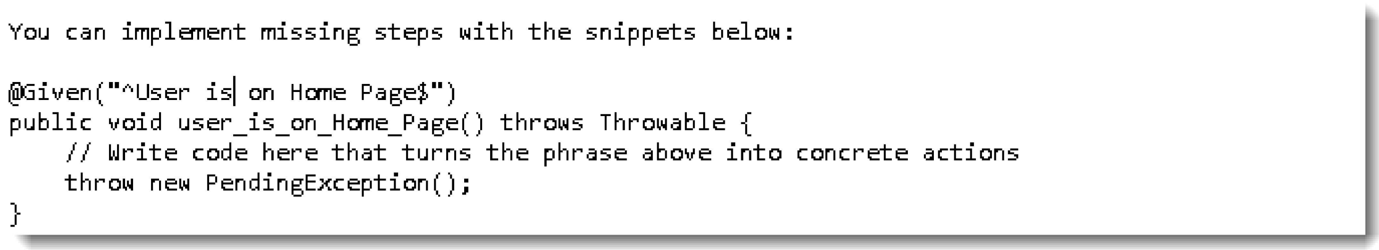
```
package cucumberTest;

import org.junit.runner.RunWith;
import cucumber.api.CucumberOptions;
import cucumber.api.junit.Cucumber;

@RunWith(Cucumber.class)
@CucumberOptions(
                features = "Feature"
                ,glue={"stepDefinition"}
                ,dryRun = true
                )

public class TestRunner {

}
```

Now give it a run by Right Click on ***TestRunner*** class and Click ***Run As > JUnit Test.*** Cucumber will run the script and the result will be shown in the left-hand side project *explorer window in JUnit tab*.



```
Finished after 0.154 seconds
Runs: 6/6 (6 skipped)    Errors: 0    Failures: 0

cucumberTest.TestRunner [Runner: JUnit 4] (0.005 s)
    Feature: Login Action (0.005 s)
        Scenario: Successful Login with Valid Credentials (0.000 s)
            Given User is on Home Page (0.000 s)
            When User Navigate to LogIn Page (0.000 s)
            And User enters UserName and Password (0.000 s)
            Then Message displayed Login Successfully (0.000 s)
        Scenario: Successful LogOut (0.001 s)
            When User LogOut from the Application (0.000 s)
            Then Message displayed Logout Successfully (0.001 s)
```

Take a look at the time duration at the end of the every *Steps*, it is (*0.000s*). It means none of the *Step* is executed but still, *Cucumber* has made sure that every *Step* has the corresponding method available in the *Step Definition file*. Give it a try, remove the '*@Given("^User is on Home Page$")*' statement from the ***Test_Steps*** class and run the ***TestRunner*** class again. You would get the following message:



```
You can implement missing steps with the snippets below:

@Given("^User is on Home Page$")
public void user_is_on_Home_Page() throws Throwable {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}
```

Monochrome

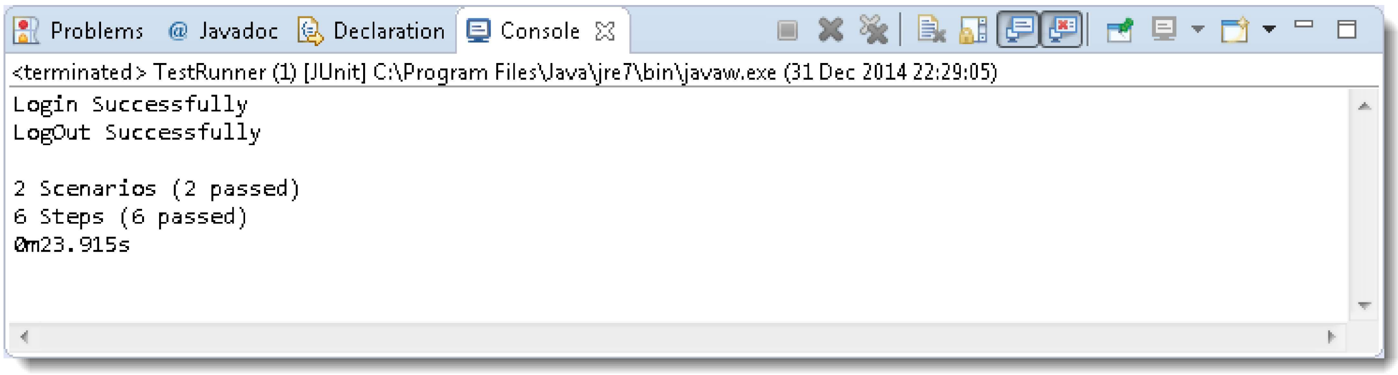
This option can either set as ***true*** or ***false***. If it is set as *true*, it means that the *console output* for the *Cucumber test* are much more readable. And if it is set as false, then the *console output* is not as readable as it should be. For practice just add the code '*monochrome = true*'
in ***TestRunner*** class:

TestRunner Class

```
package cucumberTest;

import org.junit.runner.RunWith;
import cucumber.api.CucumberOptions;
import cucumber.api.junit.Cucumber;

@RunWith(Cucumber.class)
```

```
@CucumberOptions(
                features = "Feature"
                ,glue={"stepDefinition"}
                ,monochrome = false
                )

public class TestRunner {

}
```

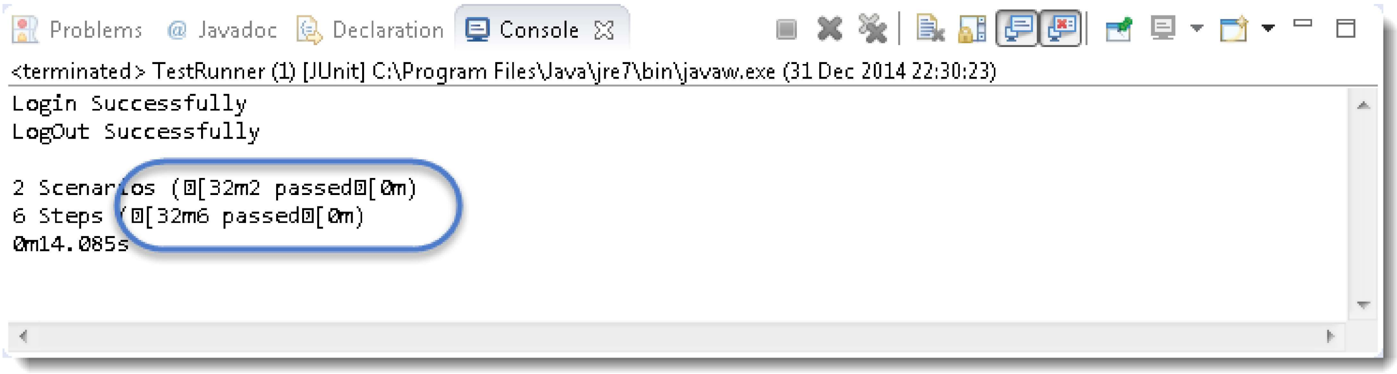
Now give it a run by Right Click on TestRunner class and Click Run As > JUnit Test. Cucumber will run the script and Console Output will display like this:



Problems @ Javadoc Declaration Console

```
<terminated> TestRunner (1) [JUnit] C:\Program Files\Java\jre7\bin\javaw.exe (31 Dec 2014 22:29:05)
Login Successfully
LogOut Successfully

2 Scenarios (2 passed)
6 Steps (6 passed)
0m23.915s
```

This time change the value from *true to false* and run the ***TestRunner*** class again. This time the *Console Output* will look like this:



Problems @ Javadoc Declaration Console

```
<terminated> TestRunner (1) [JUnit] C:\Program Files\Java\jre7\bin\javaw.exe (31 Dec 2014 22:30:23)
Login Successfully
LogOut Successfully

2 Scenarios (�[32m2 passed�[0m)
6 Steps (�[32m6 passed�[0m)
0m14.085s
```

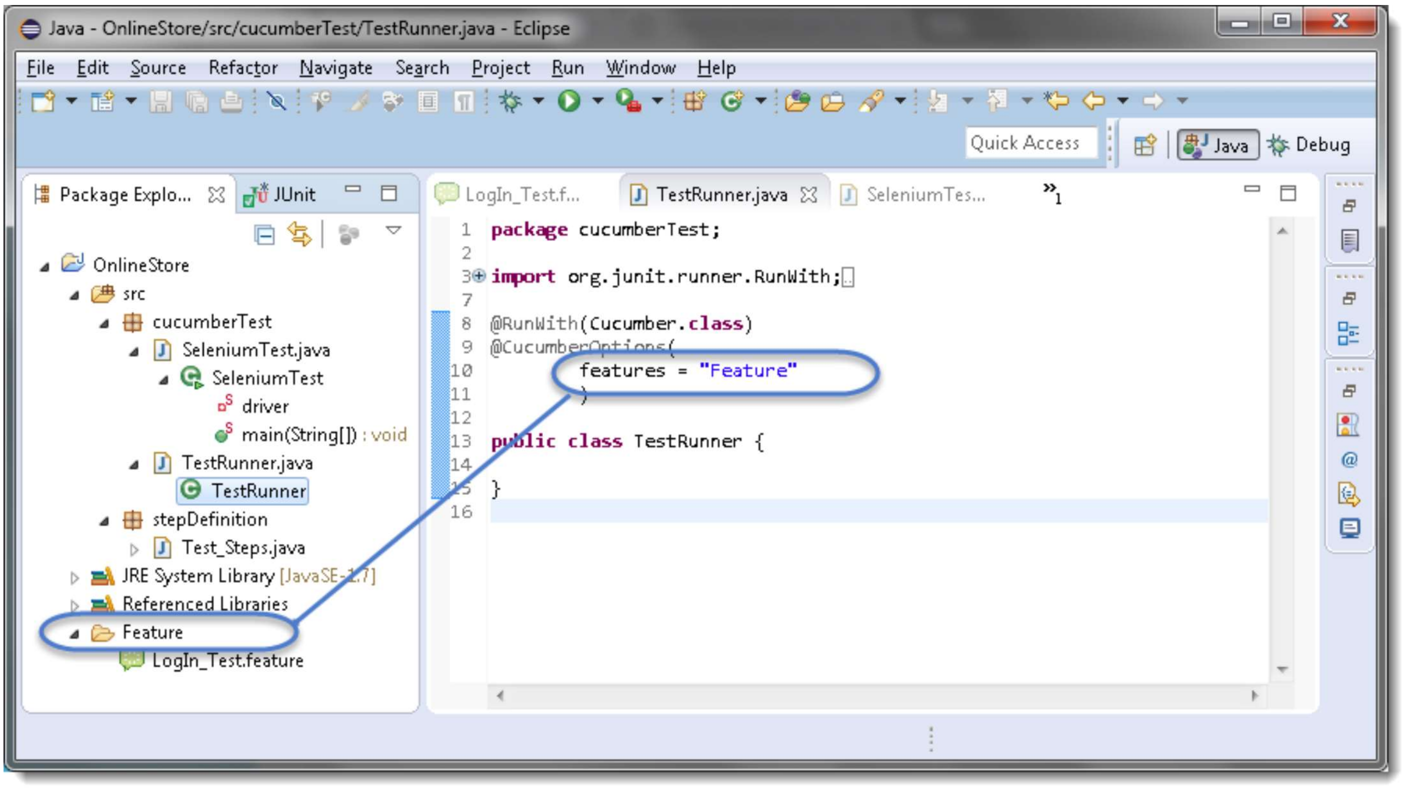
Features

Features Options helps *Cucumber* to locate the *Feature file* in the project folder structure. You must have notices that we have been specifying the *Feature Option* in the ***TestRunner*** class since the first chapter. All we need to do is to specify the folder path and Cucumber will automatically find all the '*.features*' extension files in the folder. It can be specified like:

features = "Feature"

Or if the Feature file is in the deep folder structure

features = "src/test/features"



```
package cucumberTest;

import org.junit.runner.RunWith;

@RunWith(Cucumber.class)
@CucumberOptions(
        features = "Feature"
        )

public class TestRunner {

}
```

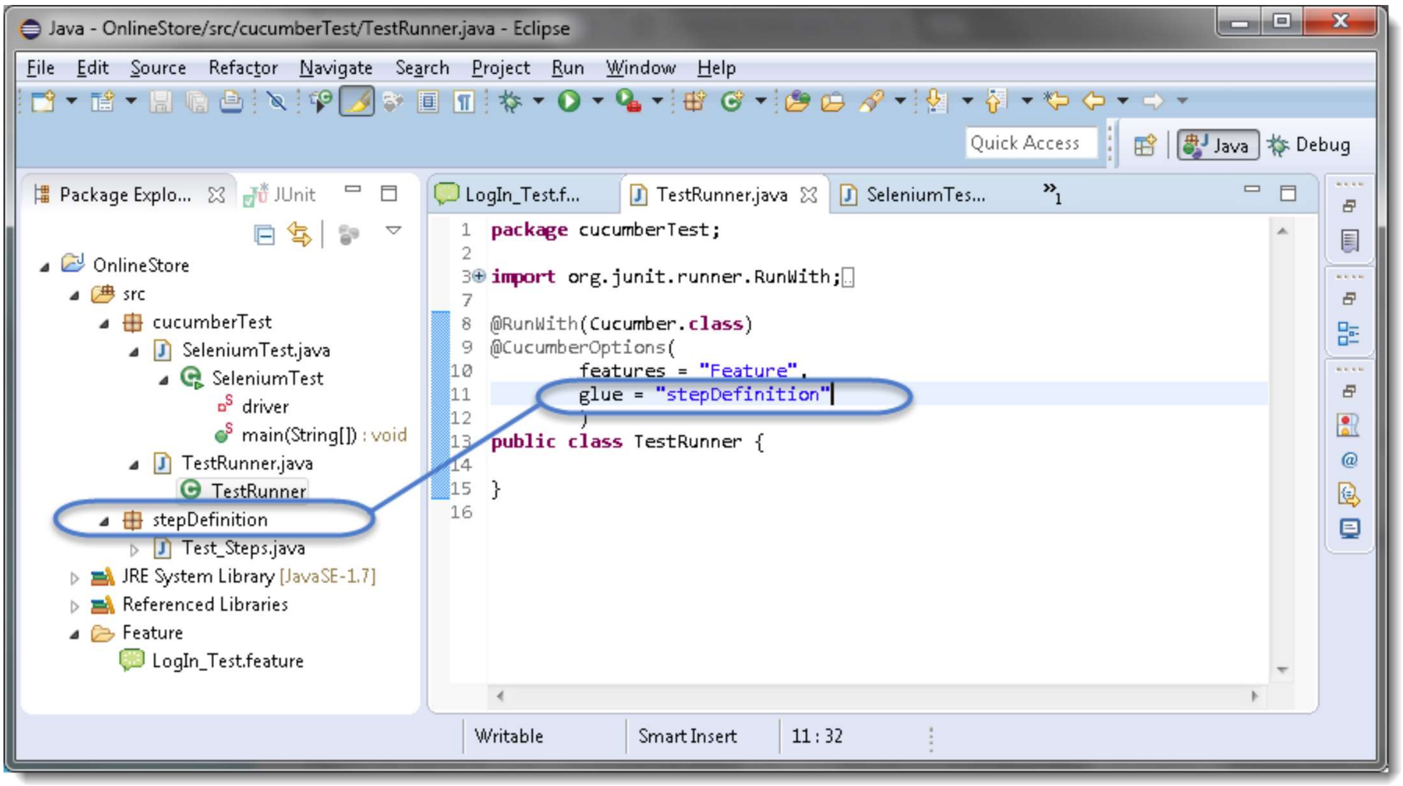
Glue

It is almost the same think as Features Option but the only difference is that it helps *Cucumber* to locate the ***Step Definition file***. Whenever *Cucumber* encounters a *Step*, it looks for a *Step Definition* inside all the files present in the folder mentioned in ***Glue Option***. It can be specified like:

glue = "stepDefinition"

Or if the Step Definition file is in the deep folder structure

glue = "src/test/stepDeinition"



```
package cucumberTest;

import org.junit.runner.RunWith;

@RunWith(Cucumber.class)
@CucumberOptions(
        features = "Feature",
        glue = "stepDefinition"
        )
public class TestRunner {

}
```

Format

Format Option is used to specify different formatting options for the output reports. Various options that can be used as for-matters are:

Pretty: Prints the *Gherkin* source with additional colors and stack traces for errors. Use below code:

format = {"pretty"}

HTML: *This will generate a HTML report at the location mentioned in the for-matter itself. Use below code:*

format = {"html:Folder_Name"}

JSON*: This report contains all the information from the gherkin source in JSON Format. This report is meant to be post-processed into another visual format by 3rd party tools such as Cucumber Jenkins. Use the below code*:

format = {"json:Folder_Name/cucumber.json"}

JUnit*: This report generates XML files just like Apache Ant's JUnit report task. This XML format is understood by most Continuous Integration servers, who will use it to generate visual reports. use* *the below code:*

format = { "junit:Folder_Name/cucumber.xml"}