JUnit Runner with @CucumberOptions

In this chapter we will discuss about cucumber options - “@CucumberOptions”.

**JUnit Runner**

Create one empty class with the @RunWith(Cucumber.class) annotation.  
Executing this class as any JUnit test class will run all features found on the classpath in the same package as this class.Name of the class could be anything like RunYoursTest.java

import org.junit.runner.RunWith;

import cucumber.api.junit.Cucumber;

@RunWith(Cucumber.class)

public class RunYoursTest

{

// This class will be empty

}

**JUnit Runner with @CucumberOptions**

@CucumberOptions annotation provides the same options as the cucumber jvm command line. for example: we can specify the path to feature files, path to step definitions, if we want to run the execution in dry mode or not etc.This is very helpful and importance if we are using IDE such eclipse only to execute our project.

Lets take above mentioned RunWith(Cucumber.class) annotation. The options to be used are defined with the @CucumberOptions.

import org.junit.runner.RunWith;

import cucumber.api.CucumberOptions;

import cucumber.api.junit.Cucumber;

@RunWith(Cucumber.class)

@CucumberOptions(//your cucumber options goes here)

public class RunYoursTest

{

// This class will be empty

}

There are different type of options present under @CucumberOptions. Lets look at each option one by one to understand it better.

**Plugin:***plugin Option is used to specify different formatting options for the output reports. Various options that can be used as for-matters are:*  
**Note –***Format option is deprecated . Use Plugin in place of that.*

***A-******Pretty****:*Prints the *Gherkin* source with additional colors and stack traces for errors.

@RunWith(Cucumber.class)

@CucumberOptions(plugin = {"pretty"}, strict = false)

public class RunYoursTest

{

// This class will be empty

}

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

@RunWith(Cucumber.class)

@CucumberOptions(plugin ={"pretty" , "html:Folder\_Name"})

public class RunYoursTest

{

// This class will be empty

}

**C-** **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.

@RunWith(Cucumber.class)

@CucumberOptions(plugin = {"pretty" ,

"json:Folder\_Name/cucumber.json"})

public class RunYoursTest

{

// This class will be empty

*}*

***D-******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.

@RunWith(Cucumber.class)

@CucumberOptions(plugin = {"pretty" ,

"junit:Folder\_Name/cucumber.xml"})

public class RunYoursTest

{

// This class will be empty

}

**We can also use these option together.**

@RunWith(Cucumber.class)

@CucumberOptions(plugin = {"pretty" ,"html:Folder\_Name" ,

"json:Folder\_Name/cucumber.json" ,

"junit:Folder\_Name/cucumber.xml"})

public class RunYoursTest

{

// This class will be empty

}

**DryRun:** This option can either set as true or false (default value is false). If it is set as true, it means that Cucumber will only checks that every Step mentioned in the Feature File have corresponding code written in Step Definition file or not. So in case any of the function is missed in the Step Definition for any Step in Feature File, it will give us the message. So If you writing scenarios first and then implementing step definitions then add dryRun = true.

@RunWith(Cucumber.class)

@CucumberOptions(plugin = {"pretty"}, dryRun = true)

public class RunYoursTest

{

// This class will be empty

}

**Strict:** if strict option is set to false then at execution time if cucumber encounters any undefined/pending steps then cucumber does not fail the execution and undefined steps are skipped and BUILD is SUCCESSFUL.

@RunWith(Cucumber.class)

@CucumberOptions(plugin = {"pretty"}, strict = false)

public class RunYoursTest

{

// This class will be empty

}

and if Strict option is set to true then at execution time if cucumber encounters any undefined/pending steps then cucumber does fails the execution and undefined steps are marked as fail and BUILD is FAILURE. This is what the Console output looks like:

@RunWith(Cucumber.class)

@CucumberOptions(plugin = {"pretty"}, strict = true)

public class RunYoursTest

{

// This class will be empty

}

**Monochrome**: This option can either set as true or false (default value is 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.

@RunWith(Cucumber.class)

@CucumberOptions(plugin = {"pretty"}, strict = true,

monochrome = true)

public class RunYoursTest

{

// This class will be empty

}

**Features:** Features Options helps Cucumber to locate the *Feature file* in the project folder structure.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 defined like:

@RunWith(Cucumber.class)

@CucumberOptions(

features = “Feature“

)

public class RunYoursTest

{

// This class will be empty

}

Or if the feature file is in the deep folder structure-

@RunWith(Cucumber.class)

@CucumberOptions(

features = “src/test/features“

)

public class RunYoursTest

{

// This class will be empty

}

**Glue:**It is almost the same think as *Features Option* but the only difference is that it helps Cucumber to locate theStep 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 defined like-

@RunWith(Cucumber.class)

@CucumberOptions(

features = “Feature“

glue = “stepDefinition“

)

public class RunYoursTest

{

// This class will be empty

}

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

@RunWith(Cucumber.class)

@CucumberOptions(

features = “src/test/features“

glue = “src/test/stepDeinition“

)

public class RunYoursTest

{

// This class will be empty

}

**Snippet Style:**Cucumber generates code snippets in Underscore style by default. If you want to change the format of cucumber snippets then you can set snippet type in your cucumber options. There are two types of snippets,

1- SnippetType.CAMELCASE  
2- SnippetType.UNDERSCORE

@RunWith(Cucumber.class)

@CucumberOptions(plugin = {"pretty"}, monochrome = true

Snippets = SnippetType.CAMELCASE)

public class RunYoursTest

{

// This class will be empty

}

**Running cucumber tests based on Tags**

The tags can be used when specifying what tests to run through any of the running mechanism.  
So, tags is just another parameter in the cucumber options annotation. We can also pass multiple tags as values separated by commas if we need so.

**Tag expressions-**

**Running Scenarios with a Tag**

Cucumber run scenarios with a particular tag.

@RunWith(Cucumber.class)

@CucumberOptions(

features = “src/test/features“,

tags ={“@Webdriver“},... )

public class RunYoursTest

{

// This class will be empty

}

**Running Scenarios without Tag**

Cucumber can exclude scenarios with a particular tag by inserting the tilde character before the tag.  
For the following command will run all Scenarios without the SoapUI tag.

@RunWith(Cucumber.class)

@CucumberOptions(

features = “src/test/features“,

tags ={“~@SoapUI“},... )

public class RunYoursTest

{

// This class will be empty

}

**Logical OR**

Separate a list of tags by commas for a Logical OR tag expression.

@RunWith(Cucumber.class)

@CucumberOptions(

features = “src/test/features“,

tags ={“@SoapUI,@Functional"},... )

public class RunYoursTest

{

// This class will be empty

}

**Logical AND**

Specifying multiple tag arguments creates a logical AND between each tag expression.

@RunWith(Cucumber.class)

@CucumberOptions(

features = “src/test/features“,

tags ={“@SoapUI","@Functional"},... )

public class RunYoursTest

{

// This class will be empty

}

**AND OR NOT**

Specifying multiple tag arguments creates a logical AND between each tag expression.

@RunWith(Cucumber.class)

@CucumberOptions(

features = “src/test/features“,

tags ={“@SoapUI,@Functional","~@Regression"},... )

public class RunYoursTest

{

// This class will be empty

}