**Steps to Configure or Set Up SpecFlow**

***SpecFlow*** tests are defined as “***features***” that have multiple scenarios. There is a small amount of project setup that needs to be done, but it’s not too bad and you can be up and running in just a few minutes.

1. Create a New Project
2. Install SpecFlow Visual Studio integration
3. Add SpecFlow to the Project Reference
4. Add NUnit Test Adapter to the Project Reference
5. Add NUnit Framework to the Project Reference

### Step 1 : Create a New Project

You might expect a SpecFlow template to appear somewhere in the project list.  Sorry, that’s not what you get.  You could create one yourself if you really wanted.  To create a SpecFlow project, all you really need to do is to create a ***Class Library project***.

1) Assuming that Visual studio is opened on the machine, Go to ***File -> New -> Project*** to create a new ***Demo Project*** for the practice.

2) Select ***Windows*** from the left tab, it will display all the windows application project types. Choose ***Class Library*** in the center window and name the project accordingly. To keep it simple, please follow the same name give below ‘***SpecFlowDemo***‘. Once done, click on the ***OK*** button.

3) A new project will be created with a dummy class ***‘Class1.cs***‘. The project will look like the below image. All the project files such as ***Property Files***, ***References*** and ***Class Files  & Folders*** display under the Solution Explorer at the right side.

### Step 2 : Install SpecFlow Visual Studio Integration, NUnit Adapter & NUnit Framework

1) The first thing should be done is to install the***SpecFlow Visual Studio integration***.  Do this by navigating to ***Tools –> Extension and Updates..***

2) Make sure you are in the ***Online*** branch on the left and type **SpecFlow** into the search area in the upper right corner of the dialog. Search will display the SpecFlow Tool at the center, select it and clickon ***Download***.

### Step 3 : Add SpecFlow to the Project Reference

And now, this is where I always get tripped up for a minute.  You can create a new feature file, but there is a menu item you should see that will allow you to add feature steps to a class file and you won’t see that until you add a reference to SpecFlow.

1) Now include the SpecFlow tool in the project reference. Navigate to ***Tools -> NuGet Package Manager -> Manager Nuget Packages for Solutions…***

2) Once again a dialog will display. In the search box, type **SpecFlow** and hit enter. This will display the SpecFlow tool, select the tool and click on ***Install*** button at the right side.

A ***Successfull Message***will display in the ***Console window*** at the successfull Installation.

3) The way install ***NUnit Adapter*** and ***NUnit Framework***. Still in case face any issue, please refer the chapter

***How to write C# Test using NUnit Framework.***In this chapter, installation for both tools is given, but still the installation is preety straight forward.

## Steps to Configure a SpecFlow Test

### Step 1 : Create a Feature File

1) Now that you have the environment setup, you can actually use SpecFlow. To create your first feature file, you can right click on the ***project*** and select***Add –> New Item…*** from the context menu.

2) Select***SpecFlow Feature File,***give it a logical name and click on ***Add***button. When I name my SpecFlow feature files, I try to name them similar to the feature it will be testing.

3) The feature file will have a sample of how feature will be setup. This is just a sample and the purpose it to display the pattern of the feature file. So, you will be changing these file a lot when you move forward in this tutorial.

### Step 2 : Generate Step Definition File

In order to test our scenario, the next thing to do is to create a ***Step Definition file***. This is a regular C# file with a***Binding attribute*** added.  The file that SpecFlow will generate will match the template they provided in the feature file. Yes, it can automatically generate a skeleton for the automation code that you can then extend as necessary:

1) Right-click on your feature file in the code editor and select ***Generate Step Definitions*** from the popup menu.

2) This display a Pop Up window, which will ask to select the statements for which Step Definition file is to be created. Select all and click on ***Generate***button.

***Note***: Always choose a sensible name for the files, this is just an example file, I am not bothering about the name, from next chapter we will follow the rules.

3) It will ask to specify the folder path to save the Step Definition file, let it be a ***default project folder*** and hit ***Save*** button.

4) All the statements will change the color now, it means these Feature statements are linked with Step Definitions.

5) To have a look at the attached definitions, click on the any statement and press ***F12*** button. This will open up the linked definition file and the cursor will be pointing to the linked definition.

### Executing a SpecFlow Test

1) To identify all the test in the Feature File, go to ***Test -> Windows -> Test Explorer***.

2) Notice, at the left side a new window is appeared called ***Test Explorer***. One test is also displayed in the Test Explorer window.

***Note***: There can be strong chances that the Test Explorer window will not show the default test. Kindly do a build by ***Build -> Build Solution***. if not succeeded, open App.config file from the Solution Explorer window and make sure that **unitTestProvider** name should contain only one entry, either **NUnit** or SpecFlow. take a help from above screenshot.

3) Now to run a Feature Test, Right click on the test in the Test Explorerwindow and select ***Run Selected Tests.***This will run the selected test and display the output in the console window.

### SpecFlow Basics

* ***SpecFlow Feature File***
* ***Gherkin Keywords***
* ***Step Definition File***

# Feature File

### Selenium Test Script

Selenium Script for logging in to ***www.store.demoqa.com*** application.

1. Launch the Browser
2. Navigate to Home Page
3. Click on the LogIn link
4. Enter UserName and Password
5. Click on Submit button
6. LogOut from the application
7. Close the Browser

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37 | using OpenQA.Selenium;  using OpenQA.Selenium.Firefox;    namespace ToolsQA  {      class FirstTestCase      {          static void Main(string[] args)          {              IWebDriver driver = new FirefoxDriver();                //Launch the Online Store Website              driver.Url = "http://www.store.demoqa.com";                // Find the element that's ID attribute is 'account'(My Account)              driver.FindElement(By.XPath(".//\*[@id='account']/a")).Click();                // Find the element that's ID attribute is 'log' (Username)              // Enter Username on the element found by above desc.              driver.FindElement(By.Id("log")).SendKeys("testuser\_1");                // Find the element that's ID attribute is 'pwd' (Password)              // Enter Password on the element found by the above desc.              driver.FindElement(By.Id("pwd")).SendKeys("Test@123");                // Now submit the form.              driver.FindElement(By.Id("login")).Click();                // Find the element that's ID attribute is 'account\_logout' (Log Out)              driver.FindElement(By.XPath(".//\*[@id='account\_logout']/a")).Click();                // Close the driver              driver.Quit();            }      }  } |

***Note*** : As I said before, prerequisite for SpecFlow with Selenium is to have the Basic Understanding of Selenium in C#. If you are not familiar with the above script, please go through the small tutorial on ***Selenium with C#***.

To convert the above Selenium Test in to SpecFlow Test, it is required to create a Feature file and write automation test statements in it. Now the question comes in mind is What is Feature File?

## What is SpecFlow Feature File?

A ***feature file*** is an entry point to the SpecFlow test. This is a file where you will describe your tests in Descriptive language (Like English). It is an essential part of SpecFlow, as it serves as an automation test script as well as live documents. A feature file can contain a scenario or can contain many scenarios in a single feature file but it usually contains a list of scenarios. Let’s create one such file.

Before moving head for writing the first script, let’s create a nice folder structure of the project.

### Create a Feature File Folder

It is always good to have a nice and clean folder structure in the project and each folder represents the content in it. So, first create a folder for the feature file.

1) Create a new ***Folder*** by right click on the ‘***Project***‘ and navigating to ***Add -> New Folder.***

2) Name the folder as ‘***Features***’ and hit enter.

Once the folder for feature file is created, we are good to go to create a feature file.

### Create a Feature File

1) On the ***Feature*** folder Right click and navigate to ***Add -> New Item…***

2) Select ***SpecFlow Feature File*** in the middle and give it a logical name, for the sake of this tutorial, please use the same name ‘***LogIn\_Feature***‘ referred in the below screenshot.

***Note***: In order for SpecFlow to automatically detect the stories (or ***features***, as they’re known in SpecFlow), you need to make sure that they carry the ‘***.feature***‘ file extension. For example, in this case, I’ve named my user story ‘***LogIn\_Feature.feature***‘. Every ‘.feature’ file conventionally consists of a single feature.

3) Write first feature file for LogIn Scenario.

***Feature File***

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | Feature: LogIn\_Feature  In order to access my account      As a user of the website      I want to log into the website    @mytag  Scenario: Successful Login with Valid Credentials  Given User is at the Home Page  And Navigate to LogIn Page  When User enter UserName and Password  And Click on the LogIn button  Then Successful LogIN message should display    Scenario: Successful LogOut  When User LogOut from the Application  Then Successful LogOut message should display |

***Note:***Don’t worry about the syntax if it looks strange to you.. Ideally you should be able to understand the intent of the test just by reading a test in feature file. We will discuss this in more details in next chapter on Gherkin Keywords but still go through the small intro on Gherkin & Keywords below.

### Gherkin

A language above is called ***Gherkin*** and it implements the principles of ***Business readable domain specific language(BRDSL)***. Domain specific language gives you the ability to describe your application behavior without getting into details of implementation. What does that mean? If we go back to our tutorial in ***TDD*** we saw that we wrote test code before writing any application code. In a way we described what is the expected behavior of our application in terms of tests. On TDD those tests were pure Java tests, in your case those might be a C++ or C# tests. But the basic idea is that those are core technical tests.

If we now come back to BDD/BRDSL we will see that we are able to describe tests in a more readable format. In the above test it’s quite clear and evident, just by reading, what test would do. At the same time of being a test it also documents the behavior of application. This is the true power of BDD/BRDSL and it will become the power of cucumber eventually because cucumber works on the same principles.

### Keywords

Now moving forward we have just defined a test. You will notice colored part of the tests (***Feature, Scenario, Given, When, And and Then***). These are keywords defined by **Gherkin**. Gherkin has more keywords and we will discuss those in following tutorials. But to start off we can quickly explain some of the keywords in one line. Note this is not complete listing of Keywords:

***Feature:***Defines what feature you will be testing in the tests below

***Given:***Tells the pre-condition of the test

***And:***Defines additional conditions of the test

***When***: Defines the action of the test

***Then:***States the post condition. You can say that it is expected result of the test

# Gherkin Keywords

***Gherkin*** is not necessarily used to write automated tests. *Gherkin* is primarily used to write ***structured***tests which can later be used as project documentation. The property of being *structured* gives us the ability to automate them. This automation is done by ***Cucumber/SpecFlow***. In the ***Gherkin – Business Driven Development*** we saw a simple Gherkin Keyword test and why *Gherkin* is important to use.

***Note:****Cucumber/SpecFlow understands Gherkin hence we can say that this is a Cucumber/SpecFlow test.*

***Feature****: LogIn Action Test*  
*Description: This feature will test a LogIn and LogOut functionality*

***Scenario****: Successful Login with Valid Credentials*  
***Given****User is on Home Page*  
***When****User Navigate to LogIn Page*  
***And****User enters UserName and Password*  
***Then****Message displayed Login Successfully*

You will quickly notice that there are some colored words. These words are *Gherkin keywords* and each keyword holds a meaning. Now we will discuss these keywords one by one. Here is the list of keywords that *Gherkin*supports:

* ***Feature***
* ***Background***
* ***Scenario***
* ***Given***
* ***When***
* ***Then***
* ***And***
* ***But***
* ***\****

**Feature: Keyword**

Each *Gherkin* file begins with a ***Feature*** keyword. *Feature* defines the logical test functionality you will test in this feature file. For e.g if you are testing a payment gateway your *Feature* will become *Payment Gateway* or if you are testing the *LogIn* functionality then the *Feature* will become *Login*. The idea of having a feature file is to put down a summary of what you will be testing. This will serve as the documentation for your tests as well as a good point to start for a new team member. Note that a feature keyword is present at the starting of the feature file.

***Feature****: LogIn Action Test*

*Or*

***Feature****: LogIn Action Test*  
*Description: This feature will test a LogIn and LogOut functionality*

*Or*

***Feature****: LogIn Action Test*  
*This feature will test a LogIn and LogOut functionality*

Notice that whatever comes after the ***Feature: keyword,***will be considered as the feature description. Feature description can span across multiple lines like shown above in second example. Everything after *Feature:* till the next Keyword is encountered is considered as feature description.

***Note:****Description is not a keyword of Gherkin.*

Take a look at the example of ***Cucumber Feature*** file and ***SpecFlow Feature*** file

**Background: Keyword**

***Background***keyword is used to define steps which are common to all the tests in the feature file. For example to purchase a product, you need to do following steps:

* *Navigate to Home Page*
* *Click on the LogIn link*
* *Enter UserName and Password*
* *Click on Submit button*

After these steps only you will be able to add a product to your *cart/basket*and able to perform the payment. Now as we are in a feature file where we will be testing only the *Add to Cart* functionality, these tests become common for all tests. So instead of writing them again and again for all tests we can move it under the background keyword. This is how it will look like:

***Feature****: Add to Cart*  
*This feature will test functionality of adding different products to the User basket from different flow*

***Background:****User is Logged In*

***Scenario****: Search a product and add the first result/product to the User basket*  
***Given****User searched for Lenovo Laptop*  
***When****Add the first laptop that appears in the search result to the basket*  
***Then****User basket should display with 1 item*

Take a look at the example of ***Cucumber Background***

**Scenario: Keyword**

Each Feature will contain some number of tests to test the feature. Each test is called a ***Scenario***and is described using the *Scenario:* keyword.

***Scenario****: Search a product and add the first result/product to the User basket*

*Or*

***Scenario****: Successful LogIn with Valid Credentials*

A scenario is equivalent to a test in our regular development process. Each scenario/test can be basically broken down into three parts:

* ***Precondition****to the test, which represent with (****Given****) keyword*
* ***Test step****execution, which represent with (****When****) keyword*
* ***Verification****of the output with expected result, which represent with (****Then****)*

**Given Keyword**

***Given***defines a precondition to the test. For e.g. In shopping website, assume that the *LogIn page* *link* is only present on the Home Page, so the precondition for clicking the*LogIn link* is that the user is at the Home Page. If user is not at the Home Page, user would not be able to enter *Username* & *Password*. This precondition can be expressed in *Gherkin* like this:

***Scenario****: Successful LogIn with Valid Credentials*

***Given****User is on Home Page*  
***When****User Navigate to LogIn Page*

**When Keyword**

***When*** keyword defines the test action that will be executed. By test action we mean the user input action.

***Scenario****: Successful LogIn with Valid Credentials*

***Given****User is on Home Page*  
***When****User Navigate to LogIn Page*

Here user is performing some action using *When* keyword, clicking on the LogIn link. We can see that when defines the action taken by the user. It’s the event that will cause the actual change in state of the application.

**Then Keyword**

***Then***keyword defines the Outcome of previous steps. We can understand it best by looking at the test above and adding a Then step there.

***Feature****: LogIn Action Test*  
*Description: This feature will test a LogIn and LogOut functionality*

***Scenario****: Successful Login with Valid Credentials*  
***Given****User is on Home Page*  
***When****User Navigate to LogIn Page*  
***And****User enters UserName and Password*  
***Then****Message displayed LogIn Successfully*

Here we can see that ***Then***is the outcome of the steps above. The reader of this test would easily be able to relate to *Then* step and would understand that when the above conditions are fulfilled then the *Then* step will be executed.

**And Keyword**

***And*** keyword is used to add conditions to your steps. Let’s look at it by modifying our example a little

***Feature****: LogIn Action Test*  
*Description: This feature will test a LogIn and LogOut functionality*

***Scenario****: Successful Login with Valid Credentials*  
***Given****User is on Home Page*  
***When****User Navigate to LogIn Page*  
***And****User enters UserName and Password*  
***Then****Message displayed Login Successfully*

Or

***Feature****: LogIn Action Test*  
*Description: This feature will test a LogIn and LogOut functionality*

***Scenario****: Successful Login with Valid Credentials*  
***Given****User is on Home Page*  
***And****LogIn Link displayed*  
***When****User Navigate to LogIn Page*  
***And****User enters UserName and Password*  
***Then****Message displayed Login Successfully*  
***And****LogOut Link displayed*

Here you would see that *And* is being used to add more details to the *Given*step, it’s simply adding more conditions. We have just added three conditions. Use it when you have specified more than one condition. *And* is used to add more conditions to *Given*, *When* and *Then* statements.

**But Keyword**

***But***keyword is used to add negative type comments. It is not a hard & fast rule to use but only for negative conditions. It makes sense to use *But* when you will try to add a condition which is opposite to the premise your test is trying to set. Take a look at the example below:

***Feature****: LogIn Action Test*  
*Description: This feature will test a LogIn and LogOut functionality*

***Scenario****: Unsuccessful Login with InValid Credentials*  
***Given****User is on Home Page*  
***When****User Navigate to LogIn Page*  
***And****User enters UserName and Password*  
***But****The user credentials are wrong*  
***Then****Message displayed Wrong UserName & Password*

Here you can see how adding ***But***has helped define a negative test, in this test we will try to test failure conditions. Where a wrong credentials are a failure condition.

**\* Keyword**

This keyword is very special. This keyword defies the whole purpose of having Given, When, Then and all the other keywords. Basically Cucumber doesn’t care about what Keyword you use to define test steps, all it cares about what code it needs to execute for each step. That code is called a ***step definition*** and we will discuss about it in the next section. At this time just remember that all the keywords can be replaced by the***\* keyword*** and your test will just work fine. Let’s see with example, we had this test earlier:

***Feature****: LogIn Action Test*  
*Description: This feature will test a LogIn and LogOut functionality*

***Scenario****: Successful Login with Valid Credentials*  
***Given****User is on Home Page*  
***When****User Navigate to LogIn Page*  
***And****User enters UserName and Password*  
***Then****Message displayed Login Successfully*

***Using \* Keyword***

***Feature****: LogIn Action Test*  
*Description: This feature will test a LogIn and LogOut functionality*

***Scenario****: Successful Login with Valid Credentials*  
***\*****User is on Home Page*  
***\*****User Navigate to LogIn Page*  
***\*****User enters UserName and Password*  
***\*****Message displayed Login Successfully*

# Step Definition File

The next target is to test or run the feature file and in order to test the feature file, it is required to***write the implementation or step definition***for each step in the ***Step Definition File in C Sharp***. When SpecFlow executes a Step in a Scenario it will look for a matching Step Definition to execute.

## What is SpecFlow Step Definition file?

A Step Definition file is a small piece of code with a pattern attached to it or in other words a Step Definition is a C# method in a class with an annotation above it. An annotation followed by the pattern is used to link the Step Definition to all the matching Steps, and the code is what SpecFlow will execute when it sees a Gherkin Keywords.

## How to Create a Step Definition file?

It is always good to have a nice and clean folder structure in the project and each folder represents the content in it. So, first create a folder for the ***Step Definition*** file.

### Create a Step Definition File Folder

1) Create a new ***Folder*** by right click on the ‘***Project***‘ and navigating to ***Add -> New Folder.***

2) Name the folder as ‘***Steps***’ and hit enter.

### Create a Step Definition File

1) Notice, the color of the statements mentioned in the feature file is violet. It means these statements do not have any definition attached to it. It is very easy to implement all the steps. Right-click on your feature file in the code editor and select ***Generate Step Definitions*** from the popup menu.

2) This display a Pop Up window, which will ask to select the statements for which Step Definition file is to be created. Select all and click on ***Generate***button. Do not forget to give it a logical name. In this case I named it as ‘***Login\_Steps***‘, as all the steps in this will file will be related to user Login.

3) It will ask to specify the folder path to save the Step Definition file. As a new ***Steps*** folder was created to keep the definition file, mention the path for the same folder with the name of the Steps file. In this case I have named it as ‘***LogIn\_Steps***‘. Hit ***Save*** button to move forward.

4) All the statements in the feature file will change the color now, it means these Feature statements are linked with Step Definitions.

5) To have a look at the attached definitions, click on the any statement and press***F12*** button. This will open up the linked definition file and the cursor will be pointing to the linked definition.

## Step Definition method Implementation

This stuff is really easy if you know how to work with ***Selenium***. In the last chapter of ***SpecFlow Feature File***, we took a simple example of***Selenium Test script for LogIn functionality.***Now start picking up the code from the same Selenium Test Script and fit it in to ***SpecFlow steps***.

### Add Selenium C# code in the Step Definition methods

1) Take the code for following steps and fit it in to the first method of Step Definition File:

‘***Given User is at the Home Page***‘.

* ***Launch the Browser***
* ***Navigate to Home Page***

Method will look like this now:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | [Given(@"User is at the Home Page")]          public void GivenUserIsAtTheHomePage()          {              driver = new FirefoxDriver();              driver.Url = "http://www.store.demoqa.com";          } |

2) Take the code for following step and fit it in to the second method of Step Definition File:

‘***And Navigate to LogIn Page***‘.

* ***Click on the LogIn link***

Method will look like this now:

|  |  |
| --- | --- |
| 1  2  3  4  5 | [Given(@"Navigate to LogIn Page")]          public void GivenNavigateToLogInPage()          {              driver.FindElement(By.XPath(".//\*[@id='account']/a")).Click();          } |

3) Take the code for following step and fit it in to the second method of Step Definition File:

‘***When User enter UserName and Password***‘.

* ***Enter UserName and Password***

Method will look like this now:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | [When(@"User enter UserName and Password")]          public void WhenUserEnterUserNameAndPassword()          {              driver.FindElement(By.Id("log")).SendKeys("testuser\_1");              driver.FindElement(By.Id("pwd")).SendKeys("Test@123");          } |

4) Take the code for following step and fit it in to the second method of Step Definition File:

‘***And Click on the LogIn button***‘.

* ***Click on Submit button***

Method will look like this now:

|  |  |
| --- | --- |
| 1  2  3  4  5 | [When(@"Click on the LogIn button")]          public void WhenClickOnTheLogInButton()          {              driver.FindElement(By.Id("login")).Click();          } |

5) Take the code for following step and fit it in to the second method of Step Definition File:

‘***Then Successful LogIN message should display***‘.

* ***LogOut Button should be displayed***

***Note***: In the demo application, we do not have any successful message on positive LogIn, So we can verify that if the LogOut button is displayed once user is logged in. In case of failure, LogOut button will never display.

Method will look like this now:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | [Then(@"Successful LogIN message should display")]          public void ThenSuccessfulLogINMessageShouldDisplay()          {              //This Checks that if the LogOut button is displayed              true.Equals(driver.FindElement(By.XPath(".//\*[@id='account\_logout']/a")).Displayed);          } |

***Complete Step Definition: LogIn\_Steps***

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56 | using OpenQA.Selenium;  using OpenQA.Selenium.Firefox;  using TechTalk.SpecFlow;    namespace SpecFlowDemo.Steps  {      [Binding]      public class LogIn\_Steps      {          public IWebDriver driver;          [Given(@"User is at the Home Page")]          public void GivenUserIsAtTheHomePage()          {              driver = new FirefoxDriver();              driver.Url = "http://www.store.demoqa.com";          }            [Given(@"Navigate to LogIn Page")]          public void GivenNavigateToLogInPage()          {              driver.FindElement(By.XPath(".//\*[@id='account']/a")).Click();          }            [When(@"User enter UserName and Password")]          public void WhenUserEnterUserNameAndPassword()          {              driver.FindElement(By.Id("log")).SendKeys("testuser\_1");              driver.FindElement(By.Id("pwd")).SendKeys("Test@123");          }            [When(@"Click on the LogIn button")]          public void WhenClickOnTheLogInButton()          {              driver.FindElement(By.Id("login")).Click();          }            [When(@"User LogOut from the Application")]          public void WhenUserLogOutFromTheApplication()          {              ScenarioContext.Current.Pending();          }            [Then(@"Successful LogIN message should display")]          public void ThenSuccessfulLogINMessageShouldDisplay()          {              //This Checks that if the LogOut button is displayed              true.Equals(driver.FindElement(By.XPath(".//\*[@id='account\_logout']/a")).Displayed);          }            [Then(@"Successful LogOut message should display")]          public void ThenSuccessfulLogOutMessageShouldDisplay()          {              ScenarioContext.Current.Pending();          }      }  } |

***Note***: Please create your own UserName and Password on ***www.store.demoqa.com***.

***Note***: In case get error on IWebDriver or FirefoxDriver keyword, it means ***Selenium dll*** is not yet added to Project Reference. Please take a look to ***Set Up Selenium on Visual Studio***.

## Run the SpecFlow Test

1. To identify all the test in the Feature File, go to ***Test -> Windows -> Test Explorer***.
2. Notice, at the left side a new window is appeared called Test Explorer.
3. Now to run a Feature Test, Right click on the test in the Test Explorer window and select Run Selected Tests. This will run the selected test and display the output in the console window.

***Note:***There can be strong chances that the Test Explorer window will not show the default test. Kindly do a build by***Build -> Build Solution***.

***Note***: Feature File can also be run by Right-clicking in the feature and choosing Run SpecFlow Scenarios. But sometimes it creates issue.

SpecFlow starts it’s execution by reading the feature file steps. As soon as it reaches to the first step for e.g. Given statement of Scenario, it looks for the same statement in the Step Definition file, the moment it find the statement, it executes the piece of code written inside the function.

### ***Data Driven Testing***

* ***Data Driven Testing in SpecFlow***
* ***Data Driven Testing Using Examples Keyword***

# Data Driven Testing in SpecFlow

Most commercial automated software tools on the market support some sort of ***Data Driven Testing***, which allows to automatically run a test case multiple times with different input and validation values. As Selenium WebDriver is more an automated testing framework than a ready-to-use tool. It takes extra efforts to support data driven testing in automated tests.

This is a very often requirement in any automated test to pass data or to use same test again with different data set. And the good part is that the ***SpecFlow*** inherently supports ***Data Driven Testing using Scenario Outline***. There are different ways to use the data insertion with in the SpecFlow and outside the SpecFlow with external files.

***Data Driven Testing in SpecFlow***

* Parameterization without Example Keyword

***Data Driven Testing in SpecFlow using Scenario Outline***

* Parameterization with Example Keyword
* Parameterization using Tables

***Data Driven Testing in SpecFlow using External Files***

* Parameterization using Excel Files
* Parameterization using Json
* Parameterization using XML

***Scenario Outline*** – This is used to run the same scenario for 2 or more different set of test data. ***E.g***. In our scenario, if you want to register another user you can data drive the same scenario twice.

***Examples*** – All scenario outlines have to be followed with the Examplessection. This contains the data that has to be passed on to the scenario.

## Data Driven Testing in SpecFlow

In the series of previous chapters, we are following the LogIn scenario. To demonstrate how parametrizing works, I am taking the same scenario again. It is important for you to be on the same page in term of project code, else you may get confused. Let’s take a look at the current state of the project. In case you find it confusing, I would request you to go through the following tutorials

* ***SpecFlow Feature File***
* ***Gherkin Keywords***
* ***Step Definition File***

The project folder structure and code should be in the below state.

***Package Explorer***

***Note:*** The last two circled files (SpecFlowFeature1.feature & SpecFlowFeature1Steps.cs) were introduced as an example in the first chapter, please delete these two files if you still have these in your project.

***LogIn\_Feature.fetaure***

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | Feature: LogIn\_Feature  In order to access my account      As a user of the website     I want to log into the website    @mytag  Scenario: Successful Login with Valid Credentials  Given User is at the Home Page  And Navigate to LogIn Page  When User enter UserName and Password  And Click on the LogIn button  Then Successful LogIN message should display    Scenario: Successful LogOut  When User LogOut from the Application  Then Successful LogOut message should display |

***LogIn\_Steps.cs***

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56 | using OpenQA.Selenium;  using OpenQA.Selenium.Firefox;  using TechTalk.SpecFlow;    namespace SpecFlowDemo.Steps  {      [Binding]      public class LogIn\_Steps      {          public IWebDriver driver;          [Given(@"User is at the Home Page")]          public void GivenUserIsAtTheHomePage()          {              driver = new FirefoxDriver();              driver.Url = "http://www.store.demoqa.com";          }            [Given(@"Navigate to LogIn Page")]          public void GivenNavigateToLogInPage()          {              driver.FindElement(By.XPath(".//\*[@id='account']/a")).Click();          }            [When(@"User enter UserName and Password")]          public void WhenUserEnterUserNameAndPassword()          {              driver.FindElement(By.Id("log")).SendKeys("testuser\_1");              driver.FindElement(By.Id("pwd")).SendKeys("Test@123");          }            [When(@"Click on the LogIn button")]          public void WhenClickOnTheLogInButton()          {              driver.FindElement(By.Id("login")).Click();          }            [When(@"User LogOut from the Application")]          public void WhenUserLogOutFromTheApplication()          {              ScenarioContext.Current.Pending();          }            [Then(@"Successful LogIN message should display")]          public void ThenSuccessfulLogINMessageShouldDisplay()          {              //This Checks that if the LogOut button is displayed              true.Equals(driver.FindElement(By.XPath(".//\*[@id='account\_logout']/a")).Displayed);          }            [Then(@"Successful LogOut message should display")]          public void ThenSuccessfulLogOutMessageShouldDisplay()          {              ScenarioContext.Current.Pending();          }      }  } |

### Parameterizing without Example Keyword

Now the task is to ***Parameterizing the UserName and Password***. Which is quite logical, why would anybody want to hard code the UserName & Password of the application. As there is a high probability of changing both.

1) Go to the ***Feature File*** and change the statement where passing Username & Password as per below:

***When User enter ‘testuser\_1′ and ‘Test@123′***

In the above statement, we have passed Username & Password from the Feature File which will feed in to Step Definition of the above statement automatically. SpecFlow will do the trick for us. After the above changes, the code will look like this:

***LogIn\_Feature.feature***  
  
2) Changes in Step Definition file is also required to make it understand the Parameterization of the feature file. So, it is required to update the Test Stepin the Step Definition file which is linked with the above changed Feature file statement.

It can either be done through editing old Step Definition or Create new one. It is good to create new one with the help of SpecFlow to avoid any syntax error. To do that, simply bring the cursor on the above changed Step for which need to create a definition and Click ***F-12*** button. This will pop up a dialog box which display the outline code for the step.

Click on the ***Yes*** button to copy the generated skeleton step.

3) Now do ***CTRL + V*** to paste the code, paste the code just above the old step definition.

Copy the inner code from the old definition and paste it in the new definition. Do not forget to delete the old definition.

4) Now the new definition will look like this:

5) Same parameters should also go in to the associated Test\_Step. As the Test step is nothing but a simple C Sharp method, syntax to accept the parameter in the C# method is like this:

***public void WhenUserEnterAnd(string username, string password)***

***Note***: Just given the sensible name to p0 & p1 and changed to username & password

6) Now the last step is to feed the parameters in the actual core statements of Selenium WebDriver. Use the below code:

5) Now to run a Feature Test, Right click on the test in the Test Explorerwindow and select ***Run Selected Tests.***This will run the selected test and display the output in the console window.

***Note***: Feature File can also be run by Right-clicking in the feature and choosing Run SpecFlow Scenarios. But sometimes it creates issue.

# Data Driven Testing Using Examples Keyword In SpecFlow

In the last chapter of ***Parameterization in SpecFlow***, we learned how to parameterize data. But with that trick only limited functionality can be achieved of Data Driven. As the test can be run multiple times. But by now that you know the anatomy of a Data-Driven test, here’s a trick that simplifies the process of ***Data Driven testing using SpecFlow***. SpecFlowinherently supports Data Driven testing by the use of the ***Scenario Outline***and ***Examples*** section. It is with these keywords that SpecFlow allows for easy Data Driven testing to be completed where no changes need to be made to the Java file. In this tutorial we learn, How to ***Implement Scenario Outline in Data Driven testing using Examples Keyword?***

Example keyword can only be used with the Scenario Outline Keyword.

* ***Scenario Outline*** – This is used to run the same scenario for 2 or more different set of test data. E.g. In our scenario, if you want to register another user you can data drive the same scenario twice.
* ***Examples*** – All scenario outlines have to be followed with the Examples section. This contains the data that has to be passed on to the scenario.

## Data Driven Testing Using Examples Keyword

If you understood the concept of ***Parameterization in SpecFlow***, you would find this one very easy. In this tutorial as well I am taking the same LogIn test scenario.

1) Change the Scenario keyword to ***Scenario Outline*** in the Feature file.

2) Enter the***Example Data*** just below the LogIn Scenario of the FeatureFile.

***Examples:***  
***| username | password |***  
***| testuser\_1 | Test@123 |***  
***| testuser\_2 | Test@153 |***

***Note***: The table must have a header row corresponding to the variables in the Scenario Outline steps.

The Examples section is a table where each argument variable represents a column in the table, separated by “***|***”. Each line below the header represents an individual run of the test case with the respective data. As a result if there are 3 lines below the header in the Examples table, the script will run 3 times with its respective data.

3) Need to update the Statement in the feature file, which tells SpecFlow to enter Username & Password.

***And User enter <username> and <password>***

SpecFlow understands the above statement syntax and look for the ***Examples*** Keyword in the test to read the Test Data.

***The complete code will look like this***:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | Scenario Outline: Successful Login with Valid Credentials  Given User is at the Home Page  And Navigate to LogIn Page  When User enter <username> and <password>  And Click on the LogIn button  Then Successful LogIN message should display  Examples:  | username   | password |  | testuser\_1 | Test@123 |  | testuser\_2 | Test@153 | |

4) There is a little change in the syntax of Definition file for the corresponding method.

Change the below line:

***[When(@”User enter ‘(.\*)’ and ‘(.\*)'”)]***

To

***[When(@”User enter (.\*) and (.\*)”)]***

***Note***: it looks similar, but if you notice carefully there is a little change in the regular expression.

### The updated Feature file and Step Definition file for the Test:

***LogIn\_Feature.feature***

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | Feature: LogIn\_Feature  In order to access my account      As a user of the website     I want to log into the website    @mytag  Scenario Outline: Successful Login with Valid Credentials  Given User is at the Home Page  And Navigate to LogIn Page  When User enter <username> and <password>  And Click on the LogIn button  Then Successful LogIN message should display  Examples:  | username   | password |  | testuser\_1 | Test@123 |  | testuser\_2 | Test@153 |    Scenario: Successful LogOut  When User LogOut from the Application  Then Successful LogOut message should display |

***LogIn\_Steps.cs***

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65 | Scenario Outline: Successful Login with Valid Credentials  Given User is at the Home Page  And Navigate to LogIn Page  When User enter <username> and <password>  And Click on the LogIn button  Then Successful LogIN message should display  Examples:  | username   | password |  | testuser\_1 | Test@123 |  | testuser\_2 | Test@153 |using OpenQA.Selenium;  using OpenQA.Selenium.Firefox;  using TechTalk.SpecFlow;    namespace SpecFlowDemo.Steps  {      [Binding]      public class LogIn\_Steps      {          public IWebDriver driver;          [Given(@"User is at the Home Page")]          public void GivenUserIsAtTheHomePage()          {              driver = new FirefoxDriver();              driver.Url = "http://www.store.demoqa.com";          }            [Given(@"Navigate to LogIn Page")]          public void GivenNavigateToLogInPage()          {              driver.FindElement(By.XPath(".//\*[@id='account']/a")).Click();          }            [When(@"User enter (.\*) and (.\*)")]          public void WhenUserEnterAnd(string username, string password)          {              driver.FindElement(By.Id("log")).SendKeys(username);              driver.FindElement(By.Id("pwd")).SendKeys(password);          }            [When(@"Click on the LogIn button")]          public void WhenClickOnTheLogInButton()          {              driver.FindElement(By.Id("login")).Click();          }            [When(@"User LogOut from the Application")]          public void WhenUserLogOutFromTheApplication()          {              ScenarioContext.Current.Pending();          }            [Then(@"Successful LogIN message should display")]          public void ThenSuccessfulLogINMessageShouldDisplay()          {              //This Checks that if the LogOut button is displayed              true.Equals(driver.FindElement(By.XPath(".//\*[@id='account\_logout']/a")).Displayed);          }            [Then(@"Successful LogOut message should display")]          public void ThenSuccessfulLogOutMessageShouldDisplay()          {              ScenarioContext.Current.Pending();          }      }  } |

5) Now to run a Feature Test, Right click on the test in the Test Explorerwindow and select ***Run Selected Tests.***This will run the selected test and display the output in the console window.

***Note***: Feature File can also be run by Right-clicking in the feature and choosing Run SpecFlow Scenarios. But sometimes it creates issue.

This takes the parameterization one step further: now our scenario has “***variables***” and they get filled in by the values in each row. To be clear: by defining this, the scenario will run two times, passing in one row at a time. This makes it very easy to define a lot of examples, edge cases and special outcomes. Instead of hardcoding the test data, variables are defined in the Examples section and used in Scenario Outline section.