



specflow

tutorialspoint

SIMPLY EASY LEARNING

www.tutorialspoint.com



<https://www.facebook.com/tutorialspointindia>



<https://twitter.com/tutorialspoint>

About the Tutorial

SpecFlow is an open-source test automation tool based on BDD. It is commonly utilized to create automation scripts for projects in .NET. This tutorial will give a comprehensive understanding of SpecFlow.

Audience

This tutorial designed for professionals working in software who want to collaborate among the developers, testers, business analysts, and other stakeholders of the project. The tutorial will be helpful for both the developers and business analysts.

Prerequisites

Before reading this tutorial, you should have a fair knowledge on basics in testing. Also, an expertise on the language C# would be an added advantage for understanding the practical examples in SpecFlow discussed here.

Copyright & Disclaimer

© Copyright 2021 by Tutorials Point (I) Pvt. Ltd.

All the content and graphics published in this e-book are the property of Tutorials Point (I) Pvt. Ltd. The user of this e-book is prohibited to reuse, retain, copy, distribute or republish any contents or a part of contents of this e-book in any manner without written consent of the publisher.

We strive to update the contents of our website and tutorials as timely and as precisely as possible, however, the contents may contain inaccuracies or errors. Tutorials Point (I) Pvt. Ltd. provides no guarantee regarding the accuracy, timeliness or completeness of our website or its contents including this tutorial. If you discover any errors on our website or in this tutorial, please notify us at contact@tutorialspoint.com

Table of Contents

About the Tutorial	ii
Audience.....	ii
Prerequisites.....	ii
Copyright & Disclaimer	ii
Table of Contents	iii
1. SpecFlow — Introduction.....	1
Components	1
Editing Features in Visual Studio Extension.....	1
2. SpecFlow — Test Driven Development	4
Benefits of TDD.....	4
Drawbacks of TDD	5
Myths around TDD	5
3. SpecFlow - Behaviour Driven Development	6
BDD Methodologies.....	6
BDD Tool.....	6
4. SpecFlow — Visual Studio Installation	8
Installation.....	8
Project Configuration	13
5. SpecFlow - Visual Studio Extension Installation	17
Installation.....	17
6. SpecFlow — Project Set Up	19
Project Creation.....	19
Project Folder	21
7. SpecFlow — Other Project Dependencies	23
Add Project Dependencies	23
Building and Running Tests	24

8. SpecFlow — Runner Activation	26
Activate Link	27
9. SpecFlow — HTML Reports	31
Test Run from Explorer	31
Report Generation	32
10. SpecFlow — Binding Test Steps	35
Test Steps in Feature File	35
Test Step Binding	35
11. SpecFlow - Creating First Test	37
12. SpecFlow — Configure Selenium Webdriver	40
Configure NUnit Framework	40
Configure Selenium WebDriver	42
13. SpecFlow — Gherkin	45
Rules in Gherkin	45
Gherkin Example	45
14. SpecFlow — Gherkin Keywords	46
Feature	46
Tags	47
Scenario	49
Given	49
When	50
Then	50
But, And	50
The * symbol	51
Background	51
Scenario Outline	52
Data Table	53
15. SpecFlow — Feature File	54

Feature File Creation	54
16. SpecFlow — Step Definition File	58
Rules for Regular expressions.....	58
Rules for Step Definition Methods	58
Step Definition File Creation.....	58
17. SpecFlow — Hooks.....	63
SpecFlow+ Runner Limitations	63
18. SpecFlow — Background Illustration.....	65
Background Rules	65
19. SpecFlow — Data Driven Testing with Examples	71
20. SpecFlow — Data Driven Testing without Examples	75
21. SpecFlow — Table conversion to Data Table.....	77
22. SpecFlow — Table conversion to Dictionary	84
23. SpecFlow — Table with CreateInstance	89
24. SpecFlow — Table with CreateSet.....	94

1. SpecFlow — Introduction

SpecFlow is an open-source test automation tool built on BDD model. It is mostly used to build automation tests for projects built in .NET. This tutorial will provide knowledge on SpecFlow and its features.

Spec-Flow is primarily developed to build, monitor, and run human-readable acceptance test cases. It is created with **Gherkin**, which is a plain-text language. SpecFlow has the Gherkin parser which can run over 70 languages. We can execute our tests via SpecFlow's in-built test runner and **SpecFlow+ Runner**.

Components

SpecFlow's primary task is to bind Feature files written in Gherkin. SpecFlow+ Runner is the test runner which has the execution capabilities and reports generation. This also comes without cost and we need to create a SpecFlow account for it.

SpecFlow+ LivingDoc is a group of tools that keeps the Gherkin Feature File in a readable format. This can be shared with the stakeholders in the team who are not well versed with tools like Visual Studio.

SpecFlow+ LivingDoc Generator is a group of plugins and tools for SpecFlow to produce documentation from the Gherkin Feature File. This does not require an account to be created and can be easily shared with others.

SpecFlow+ LivingDoc Azure DevOps allows output to be viewed in the **Azure DevOps/TFS**. Any user who has the system access can see the specifications when required. It is free but requires a SpecFlow account.

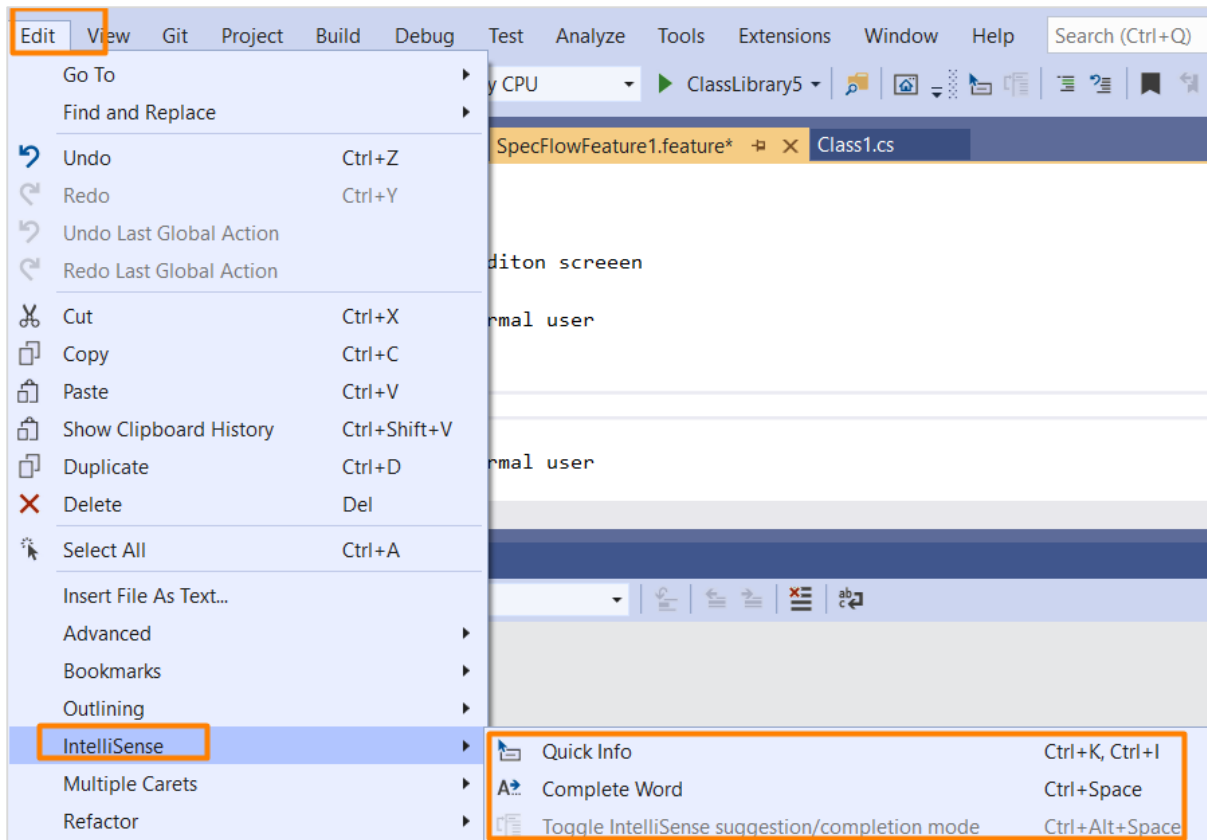
Editing Features in Visual Studio Extension

Besides, SpecFlow has the **Visual Studio Extension** that gives additional functionalities as described below:

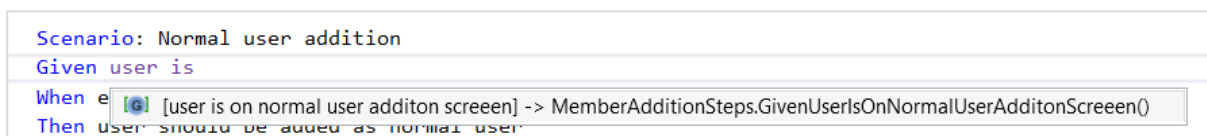
Intellisense (auto-completion) for Steps and Keywords

For easy usage of SpecFlow, intellisense provides the feature to find as we type to restrict the suggestion list. Intellisense is available for Gherkin Files, its keywords and code files as well.

In the Visual Studio, click on Edit, then select Intellisense to get the various options.



The below image shows Intellisense in the Gherkin File.



Gherkin syntax highlighting

To highlight the keywords, tags, comments, unbounded (purple colour) steps, bounded(black) steps, bounded steps parameter(grey italics).

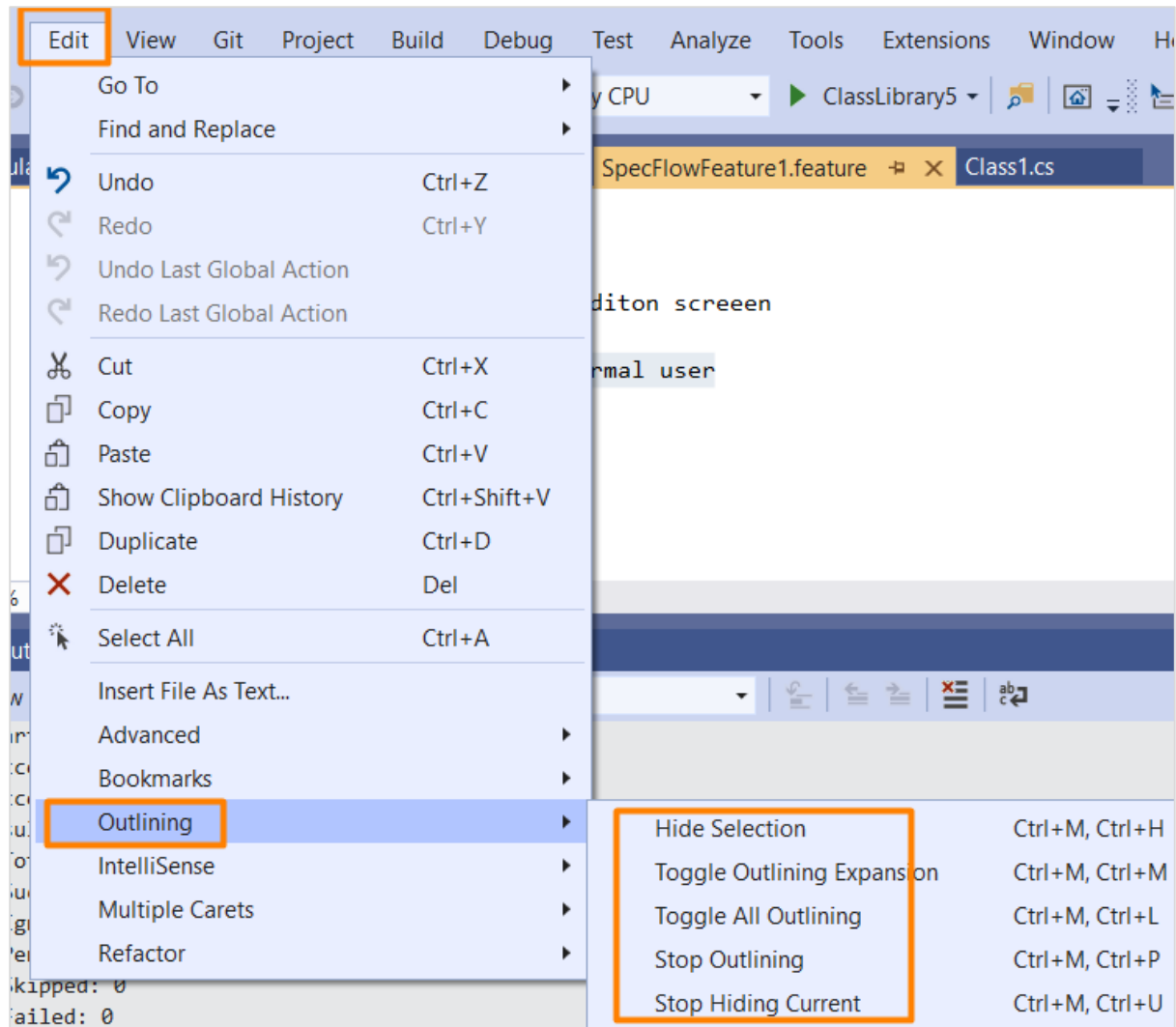
Feature Files outline and comments

In Visual Studio, most of the items in the **Edit** menu can add value to the Feature files in SpecFlow. We can comment and uncomment specific lines with **#** character, or with in-built shortcuts like the **CTRL+K CTRL+V** or **CTRL+K + CTRL+V**.

There are multiple options from the **Edit** menu to customize various sections of the Feature file.

Click on **Edit**, then select the option **Outlining**.

We can define our own feature file template to open when creating a new test case.



Formatting tables

We can modify the table size and format it automatically as we type the names of the column and enter its values.

```
When User types details
```

Name	Password
t1	pwd
t2	pwd1

But SpecFlow is not confined to Visual Studio only, it can be used with Mono and VSCode also.

2. SpecFlow — Test Driven Development

A **Test-Driven Development** is also known as the TDD. It consists of the below steps to be followed one-by-one:

Step 1: Create a Test.

Step 2: Verify if the test fails.

If the test passes, create the second test.

If the test fails, then move to Step 3.

Step 3: Fix the test to make it pass.

If the test passes, move to Step 4.

If the test fails, then jump to Step 3.

Step 4: Start code refactor and redo all the above steps till the development is done.

Benefits of TDD

The benefits of TDD are listed below:

- The developer is required to apprehend the requirements to know what the outcome of a scenario should be and how to test it.
- The implementation for a module is done only if all the test cases pass and code refactoring is complete. Thus, verification and refactoring should be done prior to moving it to the next test.
- After refactoring is done, the unit test suite is to run.
- The unit tests can be used as a live documentation.
- If a bug is found, a test is created to get the details of the bug. The script is updated, to pass the tests. Simultaneously, the other tests are also executed to ensure that existing features are not broken by the fix.
- A developer can participate in design decisions and improve it anytime during the test execution stage to ensure the application is working correctly. This is done to increase the maintainability of the product.
- A developer is sure of making any modifications. This is because if that affects any existing feature, it shall be reflected by executing the tests. This way bugs can be addressed quickly.
- On running the tests in succession, all the prior bug fixes are also verified, and the similar bugs can be avoided.

- Since major testing is conducted during the development phase, the test duration required prior to delivery is short.

Drawbacks of TDD

The drawbacks of TDD are listed below:

- The developers find it difficult to decide when to start testing.
- The developers get confused on what to test.
- The developers do not know if all the requirement specifications are being covered.
- The developers are unsure if their code is adding business values.

Myths around TDD

The myths around TDD are listed below:

Myths	Facts
TDD is only concerned with testing with automation.	TDD is a development technique following the Test First method.
TDD does not consist of design.	TDD has a thorough research and design depending on the requirements. The design is completed during the development phase.
TDD is done only for unit testing.	TDD is done for system and integration testing as well.
TDD cannot be adopted for orthodox test projects.	TDD is used for Agile development. But it can be adopted for conventional test projects as well.
TDD is considered a tool.	TDD is a development technique and post every new unit test pass, it is clubbed with the automation suite which is run whenever there is a modification in the code and post refactoring activity.

3. SpecFlow - Behaviour Driven Development

Behaviour Driven Development also known as BDD has the **features** listed below:

- Gives a shared method and tools which help to establish interaction with the developers, business analyst, and other stakeholders to work together for the product development.
- Ensures that the delivered product adds the necessary business value.
- Finds out the capabilities of the system and how it should be developed.
- Ensures that the product is presentable and has a good structure.
- Checks the functionalities of the software and ensures that the end user expectations are met.

BDD Methodologies

The primary methodologies adopted by BDD are listed below:

Specification By Example

It utilizes examples in interactions to describe the software characteristics and its business scenarios. This methodology helps to remove any knowledge gap on the business requirements among the developers, testers, product owners, business analysts and all other stakeholders in the team.

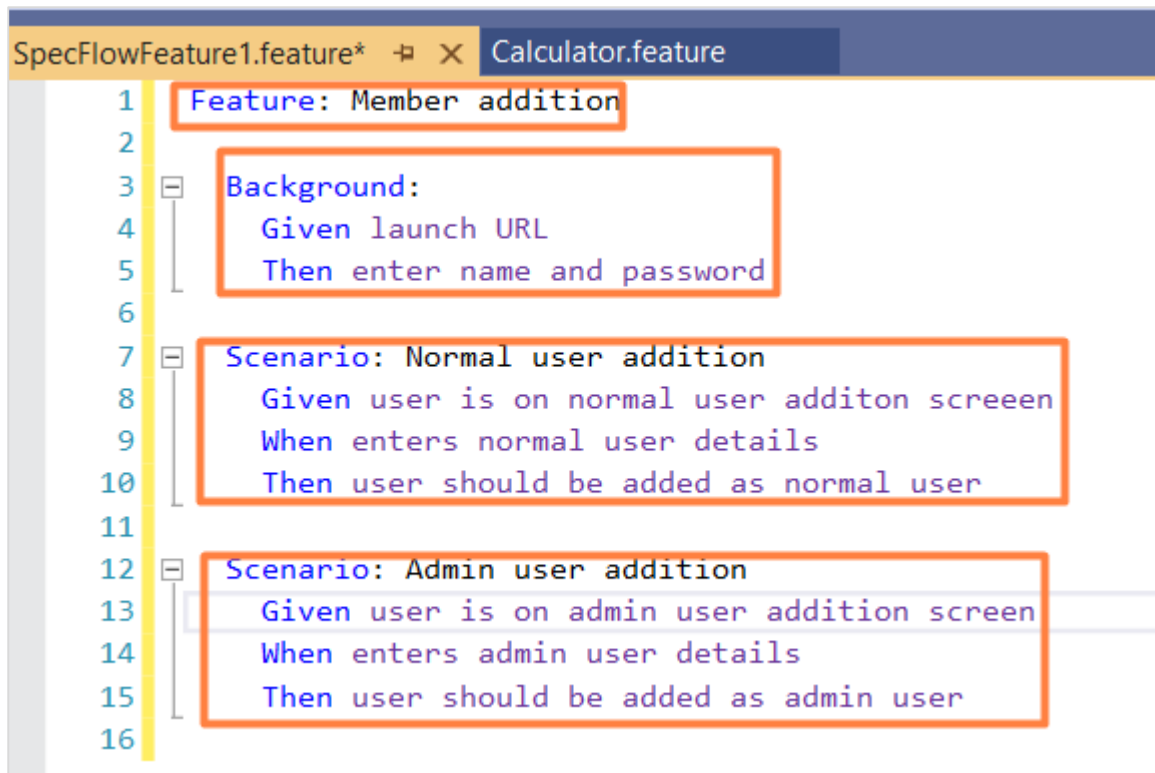
Test Driven Development

In the reference to BDD, Test Driven Development converts examples to plain text and executable specifications. The developers refer to this as a document while implementing the new features. It helps to develop a proper code base along with a regression suite. Thus, the overall maintenance cost lowers throughout the complete product lifecycle.

BDD Tool

SpecFlow is one of the BDD tools that is open source. It contains a **Feature** file which follows the **Gherkin** syntax. The source code of SpecFlow is hosted on GitHub. It is mostly used to build automation tests for projects built in .NET. It is similar to **Cucumber** in its functionalities.

Structure of a Feature file in SpecFlow:



The screenshot shows a code editor with two tabs: 'SpecFlowFeature1.feature*' and 'Calculator.feature'. The 'Calculator.feature' tab is active. The code is as follows:

```
1 Feature: Member addition
2
3 Background:
4     Given launch URL
5     Then enter name and password
6
7 Scenario: Normal user addition
8     Given user is on normal user additon screen
9     When enters normal user details
10    Then user should be added as normal user
11
12 Scenario: Admin user addition
13    Given user is on admin user addition screen
14    When enters admin user details
15    Then user should be added as admin user
16
```

Orange boxes highlight the 'Feature: Member addition' line, the 'Background' section, and the two 'Scenario' sections. A yellow vertical bar is on the left side of the editor.

It consists of the Feature, Background scenario, and two Scenarios. The Feature File consists of the acceptance standard for a Feature in the application.

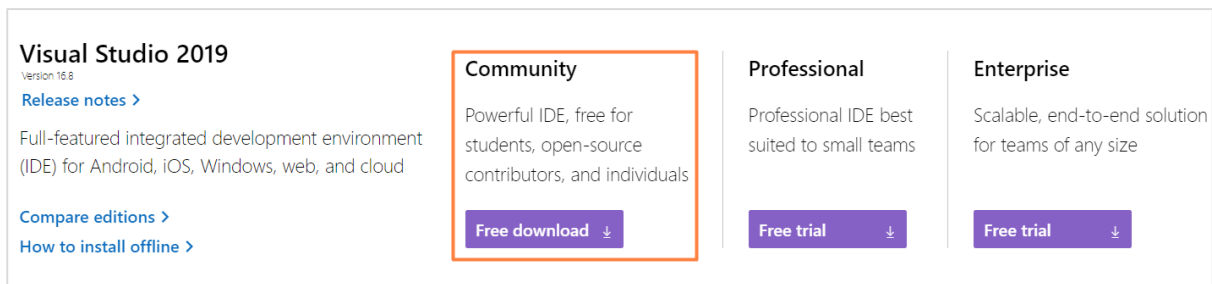
4. SpecFlow — Visual Studio Installation

In this chapter, we shall see the process of installation of Visual Studio and project configuration.

Installation

Navigate to the link: <https://www.visualstudio.com/en-us/downloads/download-visual-studio-vs.aspx#Community>

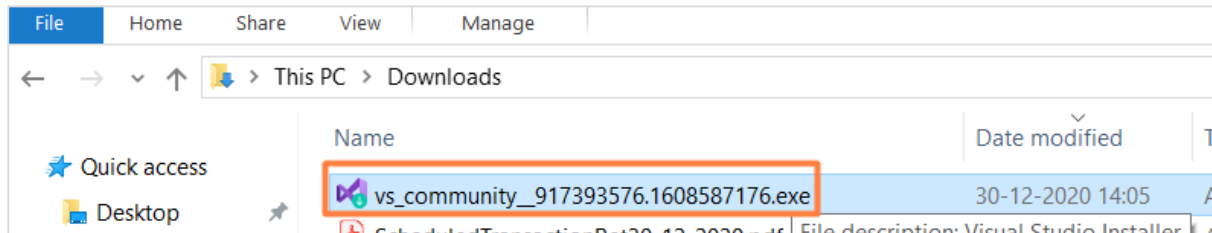
For the Community version of Visual Studio, click on **Free download** under the **Community** section.



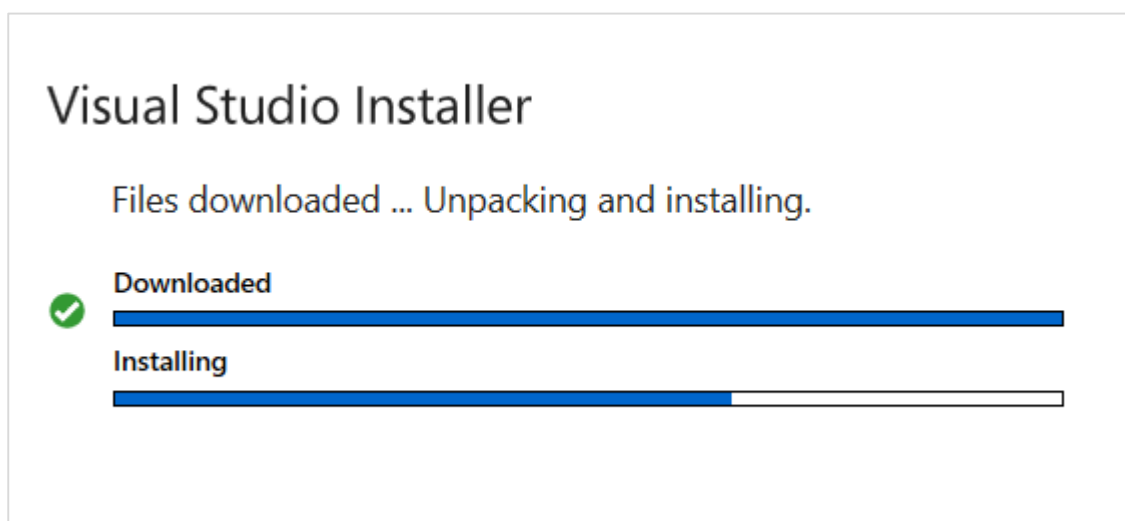
Visual Studio 2019
Version 16.8
[Release notes >](#)
Full-featured integrated development environment (IDE) for Android, iOS, Windows, web, and cloud
[Compare editions >](#)
[How to install offline >](#)

Community	Professional	Enterprise
Powerful IDE, free for students, open-source contributors, and individuals	Professional IDE best suited to small teams	Scalable, end-to-end solution for teams of any size
Free download ↓	Free trial ↓	Free trial ↓

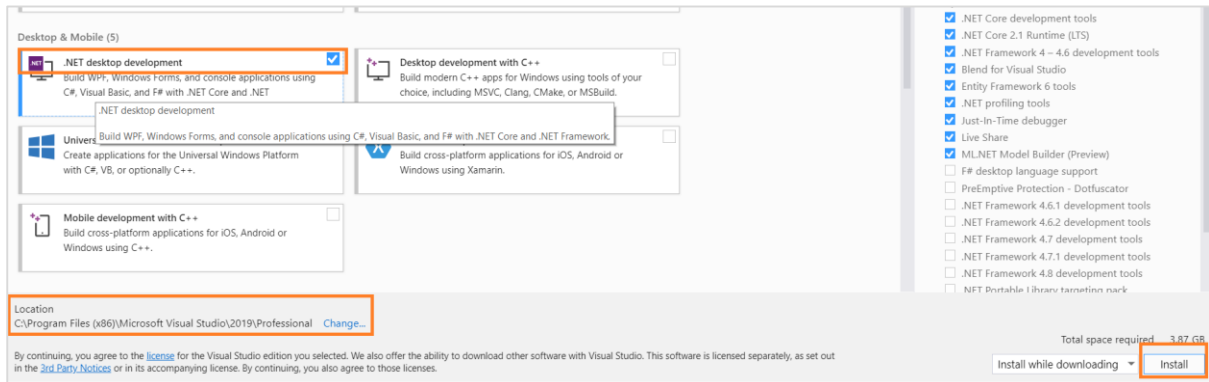
An .exe file gets downloaded to our system. Double-click on it.



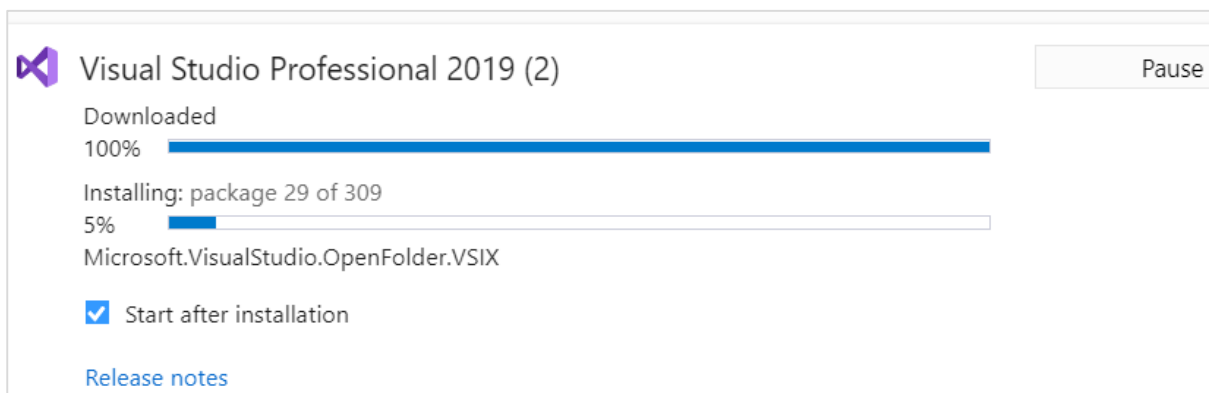
Visual Studio Installer pop-up comes up. Click on **Continue**. Download and installation process begins.



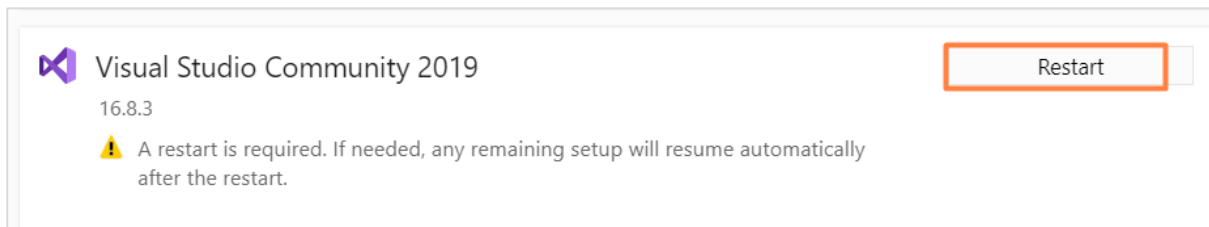
Once installation is done, select the option **.NET desktop development**. Then click on **Install**.



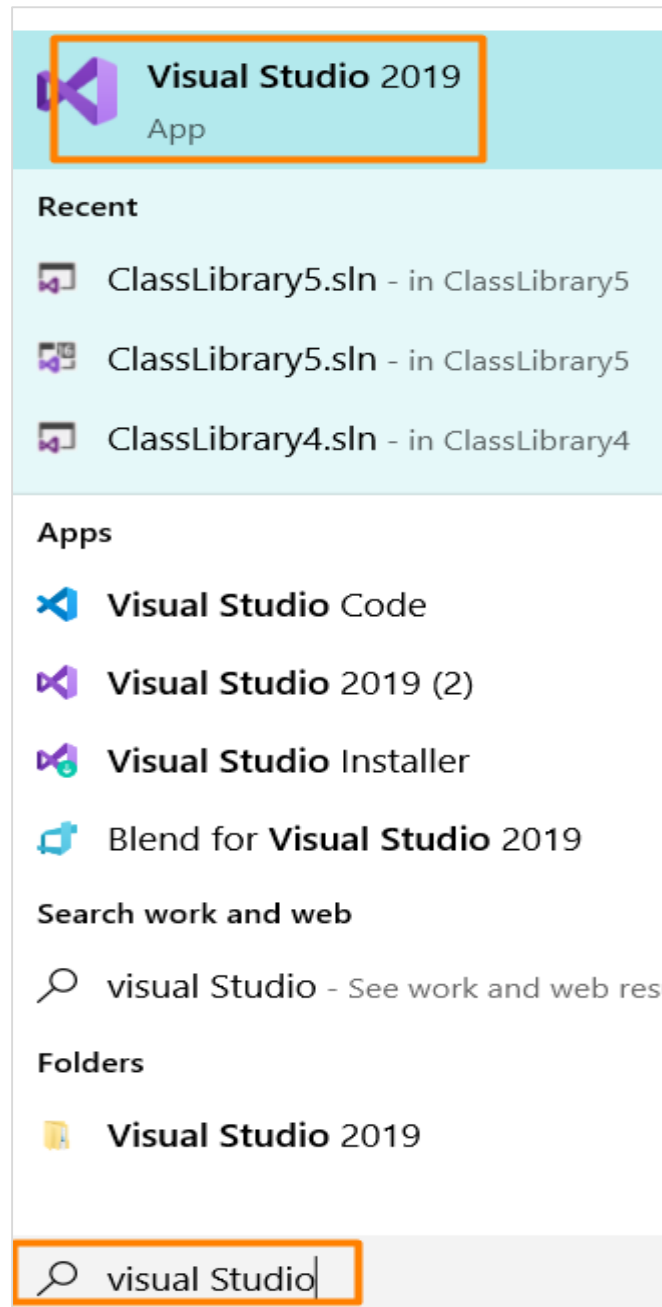
Download and installation of packages get started.



Once done, click on **Restart**.



From the start menu, type Visual Studio.



Click on Visual Studio, the welcome screen appears.



Along with it, **Visual Studio** pop-up appears. If you do not have an existing account, click on **Not now, may be later** link and proceed.

Visual Studio

Welcome!

Connect to all your developer services.

Sign in to start using your Azure credits, publish code to a private Git repository, sync your settings, and unlock the IDE.

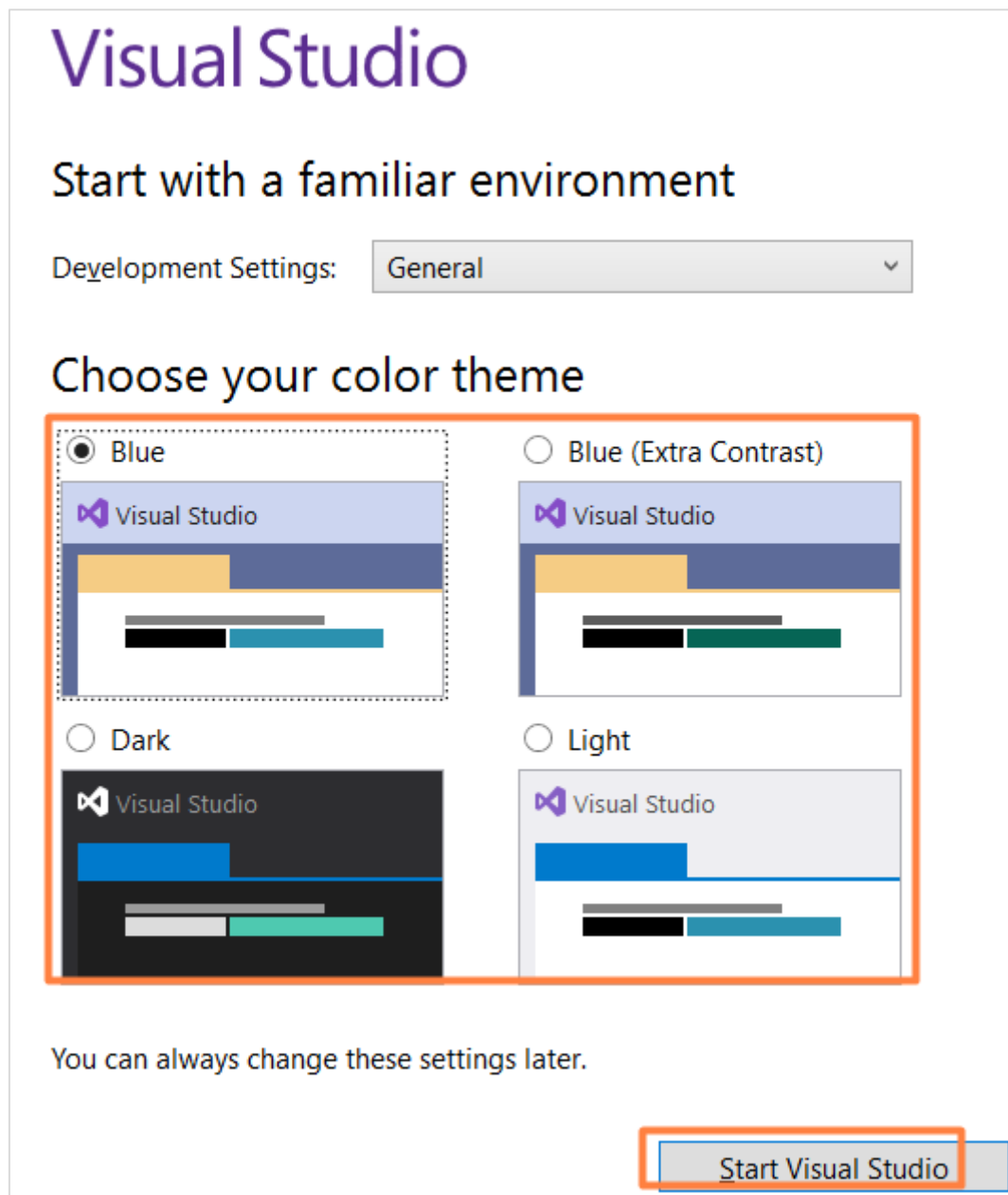
[Why should I sign in to Visual Studio?](#)

Sign in

No account? [Create one!](#)

[Not now, maybe later.](#)

Select a colour for theme and click on **Start Visual Studio**.



Project Configuration

Once the Visual Studio landing page gets opened, click on **Create a new project**.

Visual Studio 2019

Open recent

As you use Visual Studio, any projects, folders, or files that you open will show up here for quick access.

You can pin anything that you open frequently so that it's always at the top of the list.

Get started



Clone a repository

Get code from an online repository like GitHub or Azure DevOps



Open a project or solution

Open a local Visual Studio project or .sln file



Open a local folder

Navigate and edit code within any folder



Create a new project

Choose a project template with code scaffolding to get started

We shall create a new C# class library. Enter **class library core** in the search box. Choose the option **Class Library (.NET Core)** and click **Next**.

Create a new project

Recent project templates

A list of your recently accessed templates will be displayed here.

[Clear all](#)

All languages

All platforms

All project types



Class Library (.NET Framework)

A project for creating a C# **class library** (.dll)

C#

Windows

Library



Class Library (.NET Framework)

A project for creating a VB **class library** (.dll)

Visual Basic

Windows

Library



Class Library (.NET Core)

A project for creating a **class library** that targets .NET Core.

C#

Windows

Linux

macOS

Library



Class Library (.NET Core)

A project for creating a **class library** that targets .NET Core.

Visual Basic

Windows

Linux

macOS

Library



Class Library (.NET Core)

A project for creating a **class library** that targets .NET Core.

F#

Linux

macOS

Windows

Library



Class Library (.NET Standard)

Back

Next

Give a project name and location and then click on **Create**.

Configure your new project

Class Library (.NET Core) C# Windows Linux macOS Library

Project name
ClassLibrary1

Location
C:\Users\██████████\Desktop\Debomita\Java\ ...

Solution name ⓘ
ClassLibrary1

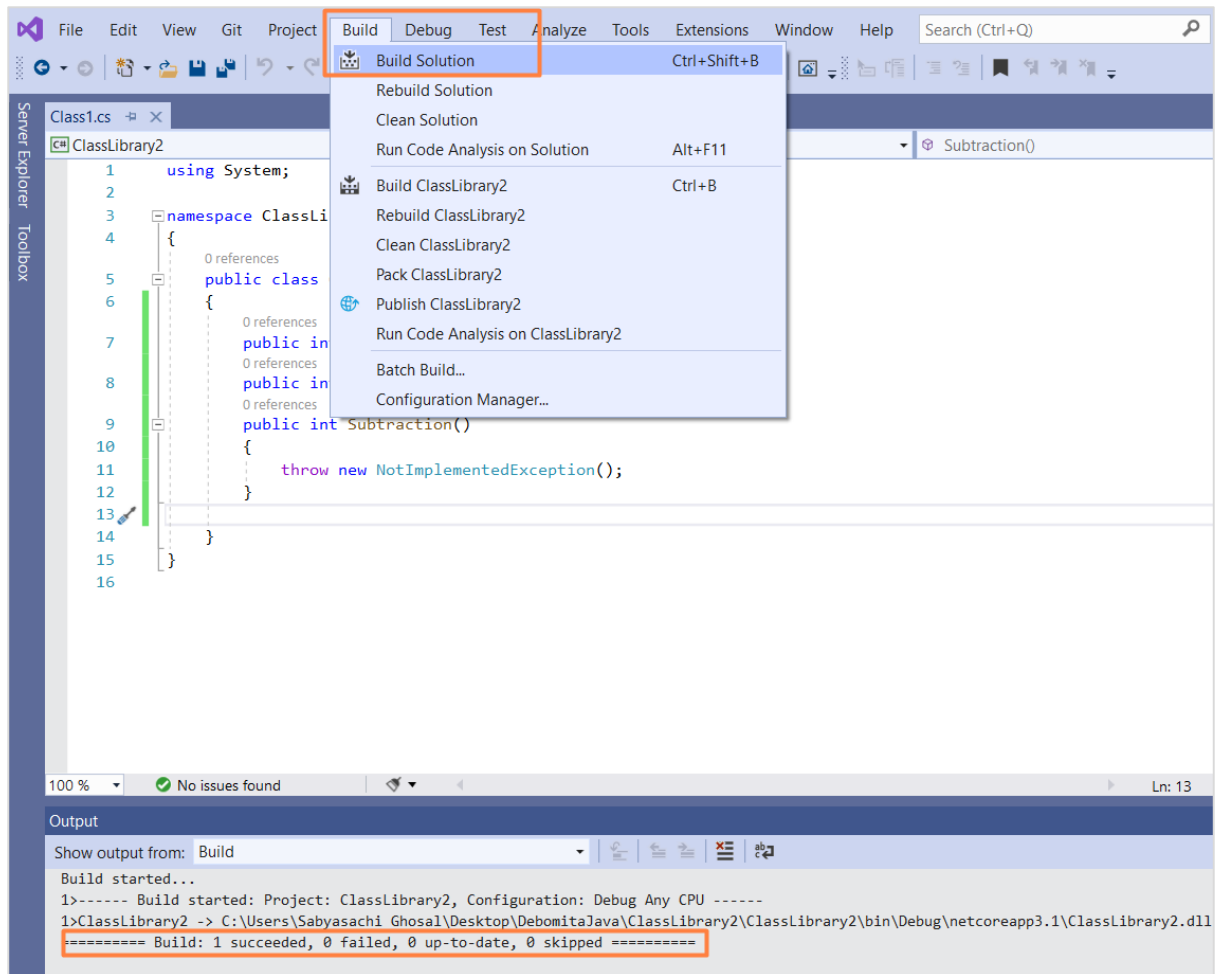
☐ Place solution and project in the same directory

Back Create

Code Implementation for Project

```
using System;
namespace ClassLibrary2
{
    public class Class1
    {
        public int Number1 { get; set; }
        public int Number2 { get; set; }
        public int Subtraction()
        {
            throw new NotImplementedException();
        }
    }
}
```

To build this solution, go to the **Build** menu, then select **Build Solution**.



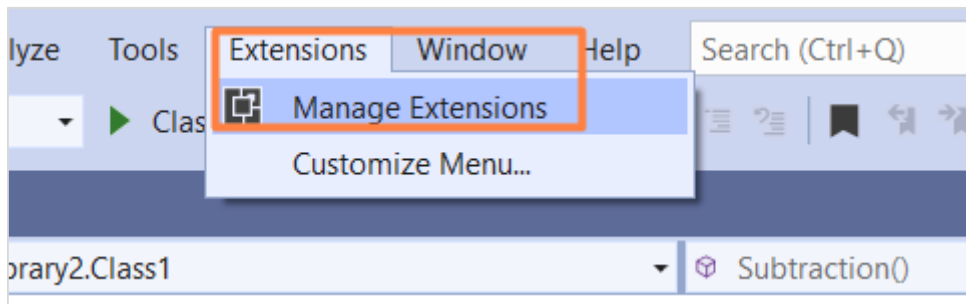
Build success message gets displayed and we have successfully created a project in Visual Studio.

5. SpecFlow - Visual Studio Extension Installation

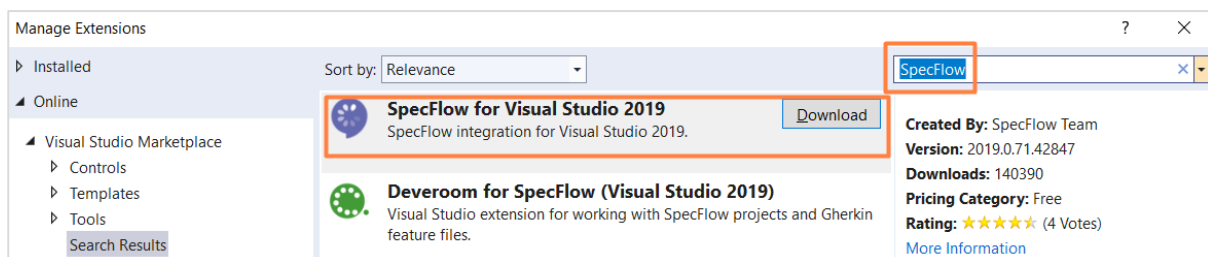
As mentioned earlier, **Visual Studio extension** allows a lot of added features required for test automation. This extension is available for Visual Studio 2017 and 2019.

Installation

Open Visual Studio and navigate to **Extensions** menu, then click on **Manage Extensions** option.

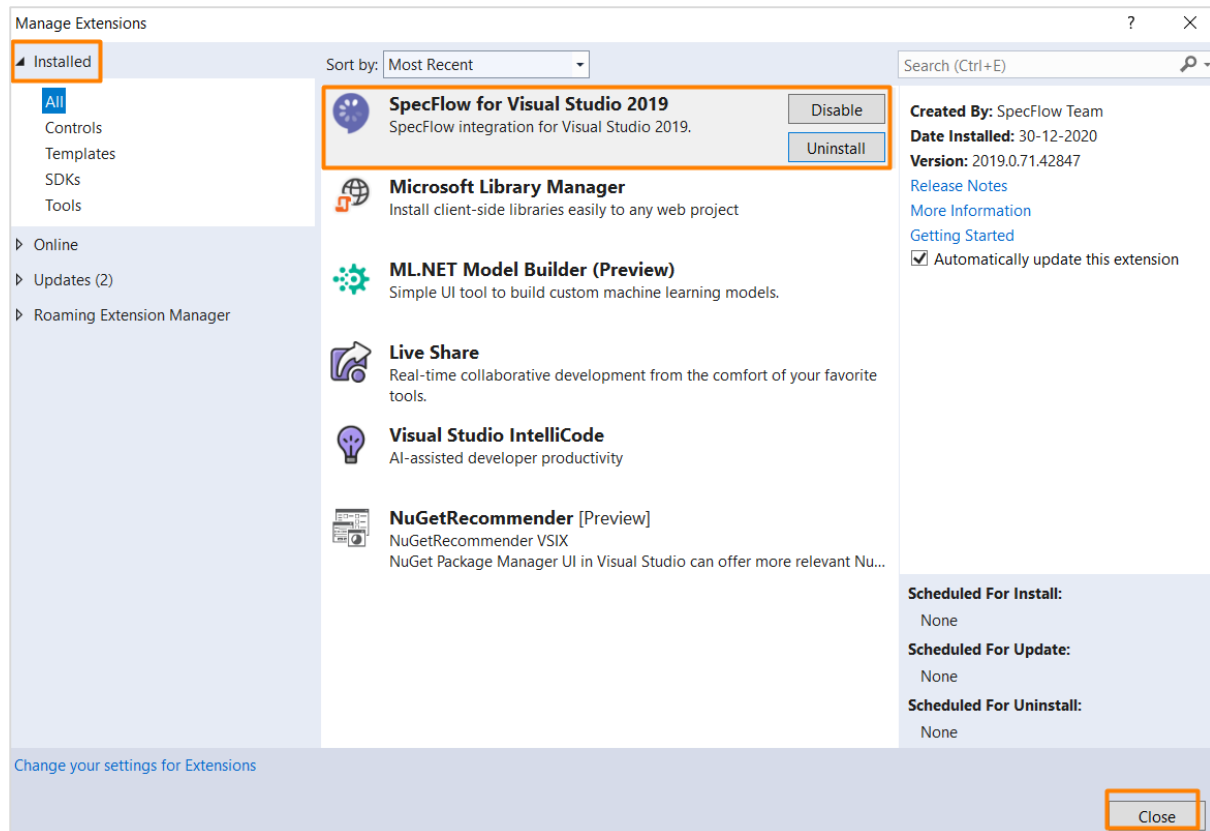


Manage Extensions pop-up comes up. Type **SpecFlow** in the search box. Once the search results get populated. Click on **Download**.



Once the download is completed, we need to restart **Visual Studio**. As the installation is done, if we again go to the **Manage Extensions** pop-up, we can find this extension within the **Installed** tab.

Also, we can find the options to **Disable** and **Uninstall** now for the SpecFlow. Click on **Close** to exit.

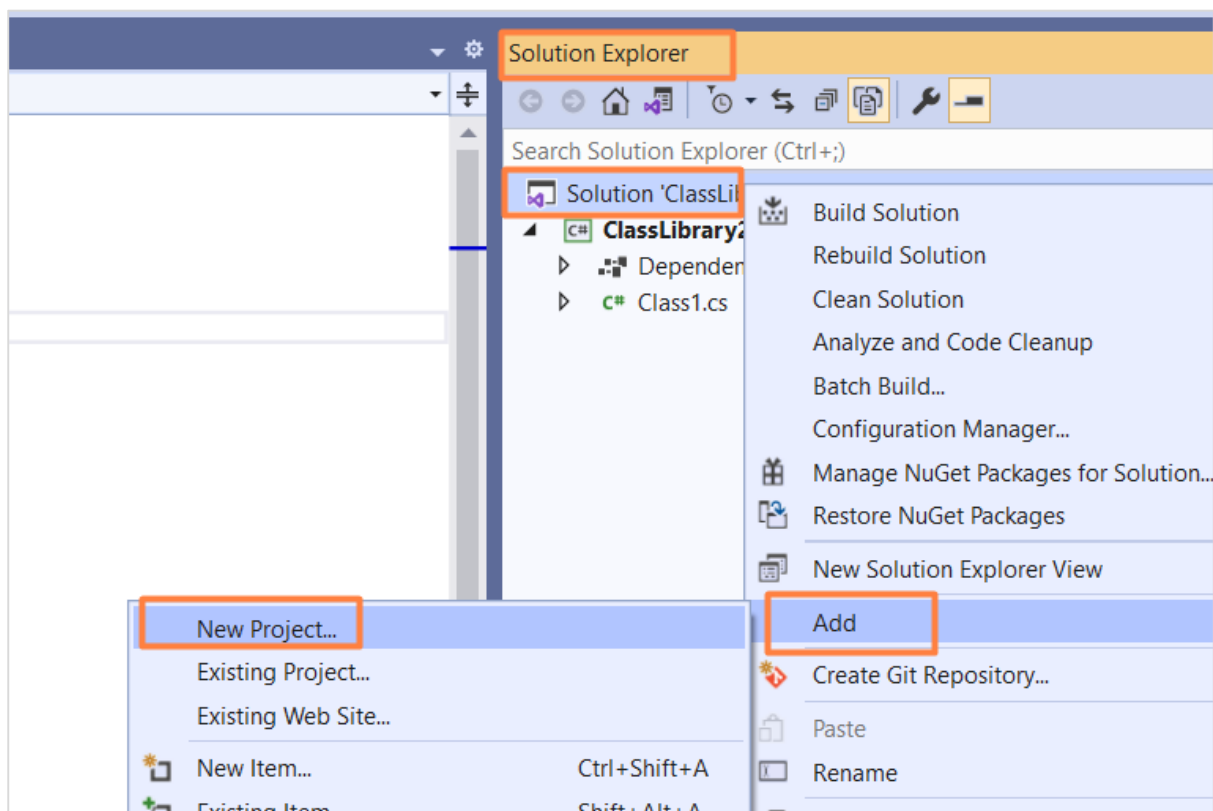


6. SpecFlow — Project Set Up

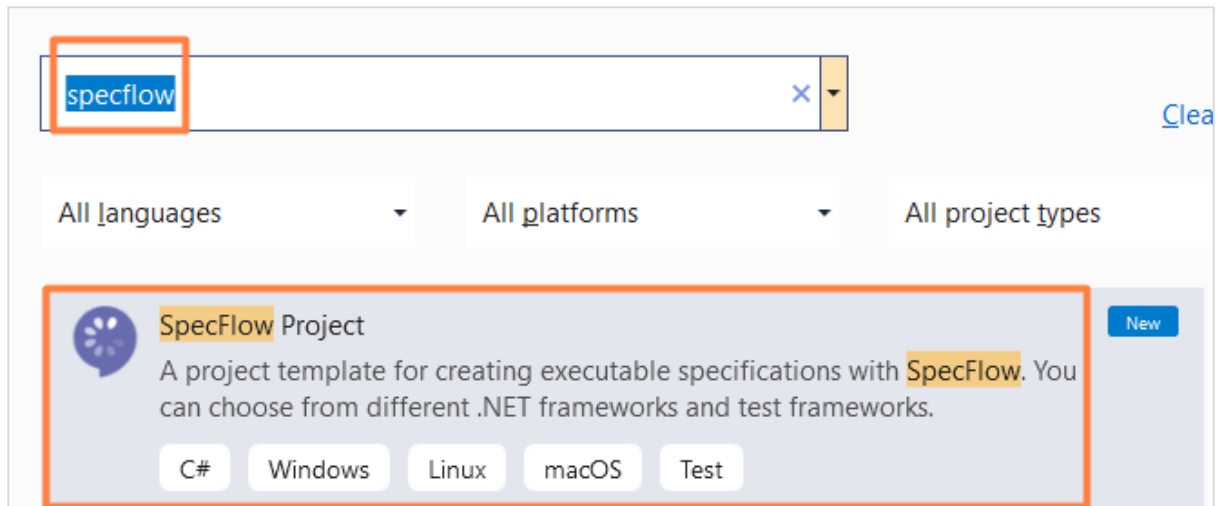
Now, we shall create a SpecFlow project within the same project we have built earlier.

Project Creation

Right-click on the **Solution Explorer** section. Click on the **Add** option. Then choose **New Project**.

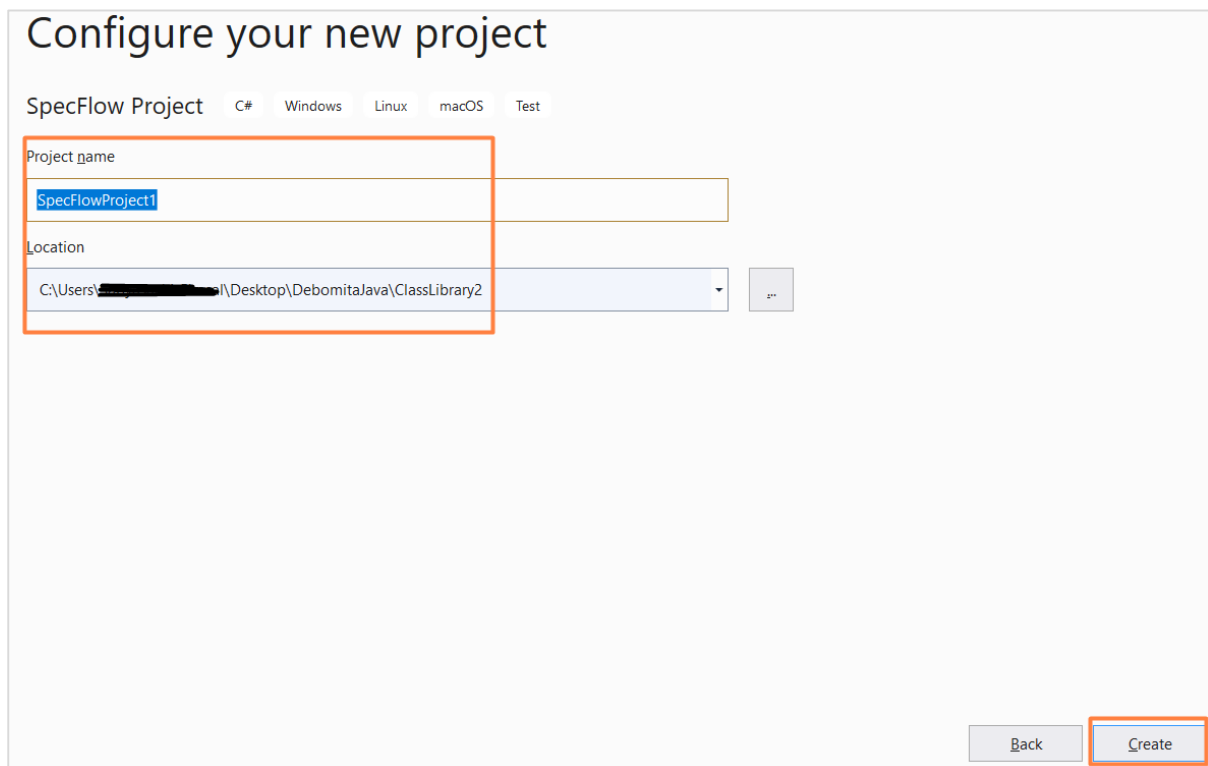


Type **SpecFlow** within the search box, SpecFlow Project gets displayed because of search results. Click on **Next** to proceed.



The screenshot shows the Visual Studio search interface. At the top, a search bar contains the text 'specflow'. Below the search bar, there are three filters: 'All languages', 'All platforms', and 'All project types'. A search result for 'SpecFlow Project' is highlighted with an orange border. The result includes a description: 'A project template for creating executable specifications with SpecFlow. You can choose from different .NET frameworks and test frameworks.' Below the description, there are five buttons: 'C#', 'Windows', 'Linux', 'macOS', and 'Test'. A 'New' button is located to the right of the result.

Enter the project name and location and then click on **Create**.



The screenshot shows the 'Configure your new project' dialog box. The title is 'Configure your new project'. Below the title, there are tabs for 'SpecFlow Project', 'C#', 'Windows', 'Linux', 'macOS', and 'Test'. The 'SpecFlow Project' tab is selected. The dialog has two input fields: 'Project name' and 'Location'. The 'Project name' field contains the text 'SpecFlowProject1'. The 'Location' field contains the text 'C:\Users\...\Desktop\DebomitaJava\ClassLibrary2'. At the bottom right, there are two buttons: 'Back' and 'Create'. The 'Create' button is highlighted with an orange border.

Select **SpecFlow+ Runner** option under the **Test Framework** dropdown from the **Create a new SpecFlow project** pop-up. Then click on **Create**.

Create a new SpecFlow project

Framework

.NET Core 3.1

Test Framework

SpecFlow+ Runner

☒ Add FluentAssertions library

Back

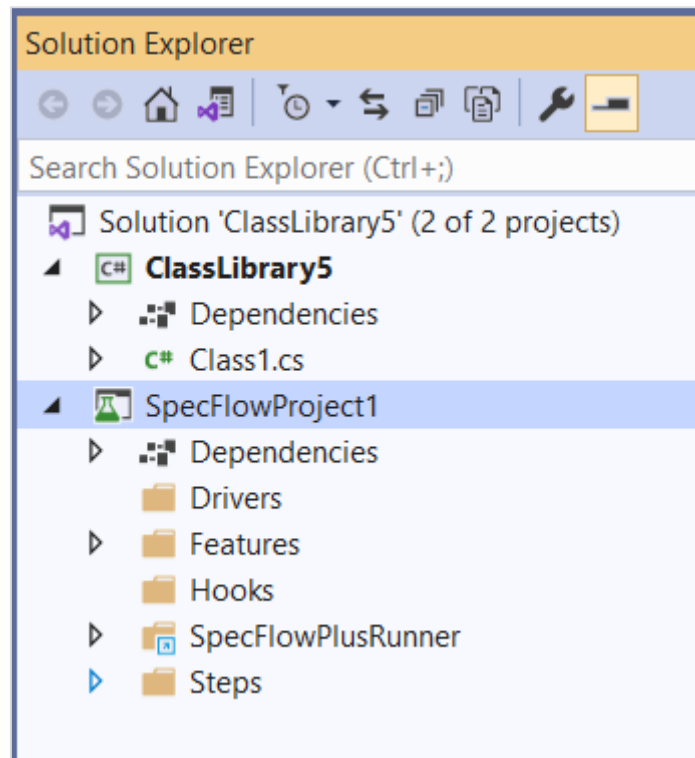
Create

The Solution Explorer shall now have a new project called the **SpecFlowProject1** created.



Project Folder

As the SpecFlow project is created, we shall also find a well-defined folder structure created for the project consisting of the Drivers, Dependencies, Features, Hooks, and so on.

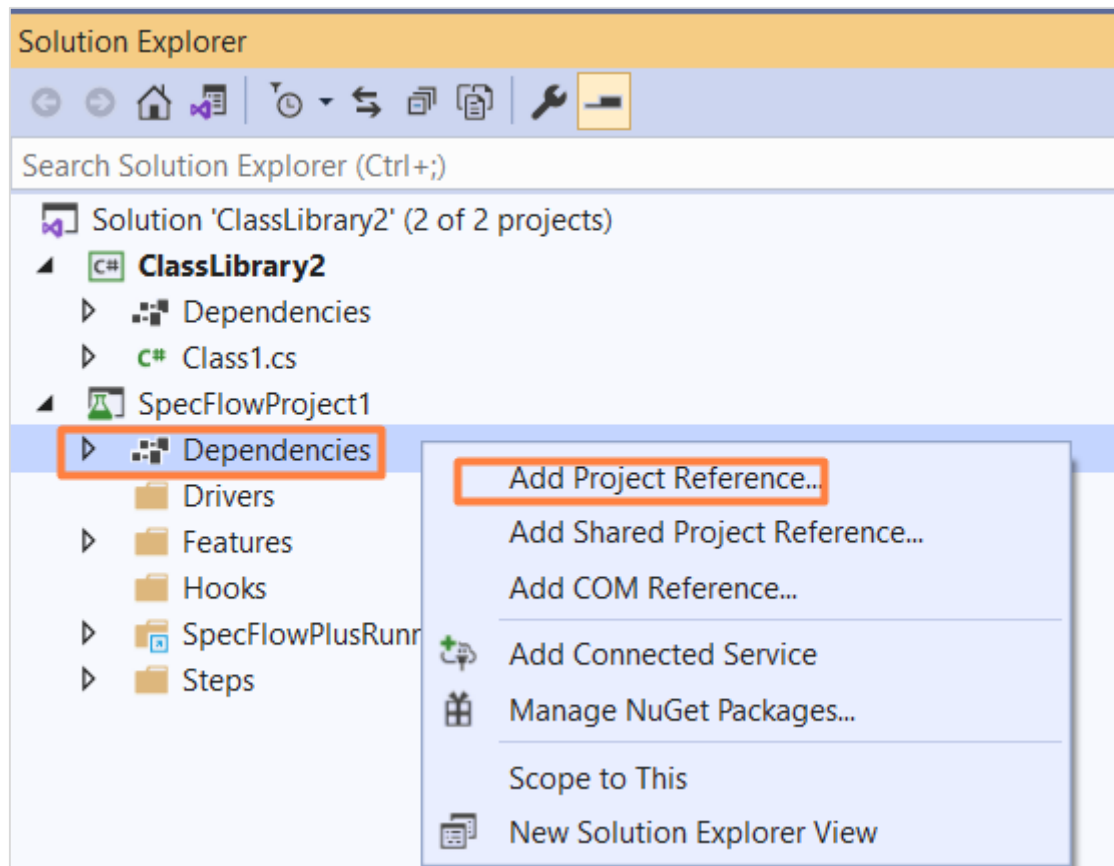


7. SpecFlow — Other Project Dependencies

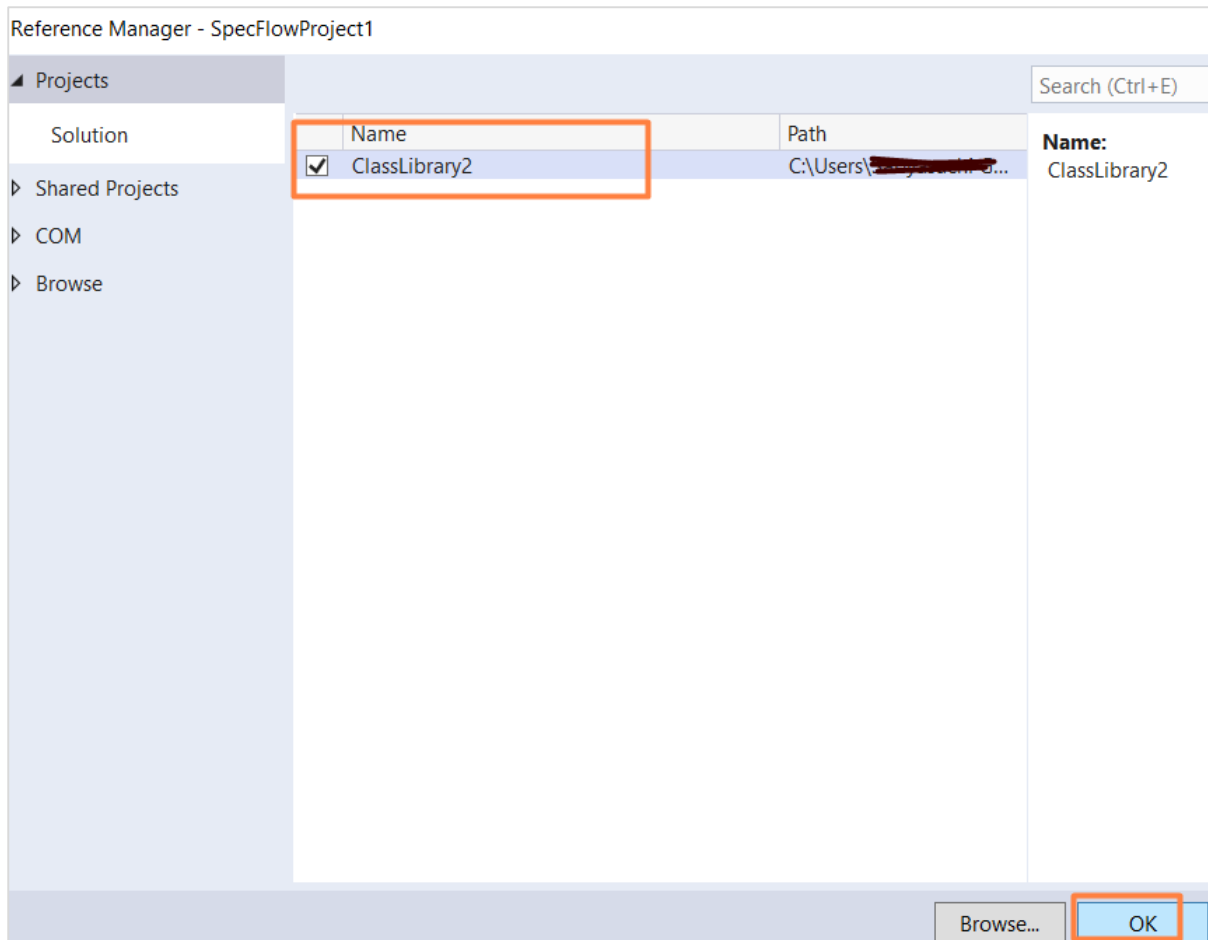
We need to have a project reference to the class library we have created for the SpecFlow project. This is important for testing the class within the class library in the project.

Add Project Dependencies

Click on the project **SpecFlowProject1** within **Solution Explorer**. Then right-click the folder **Dependencies**. Choose the option **Add Project Reference**.

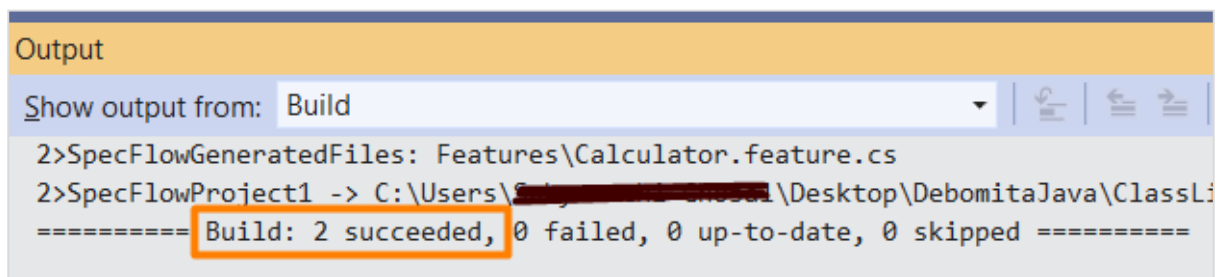


The **Reference Manager** pop-up opens. Select the checkbox for the class library and then click on **OK**.

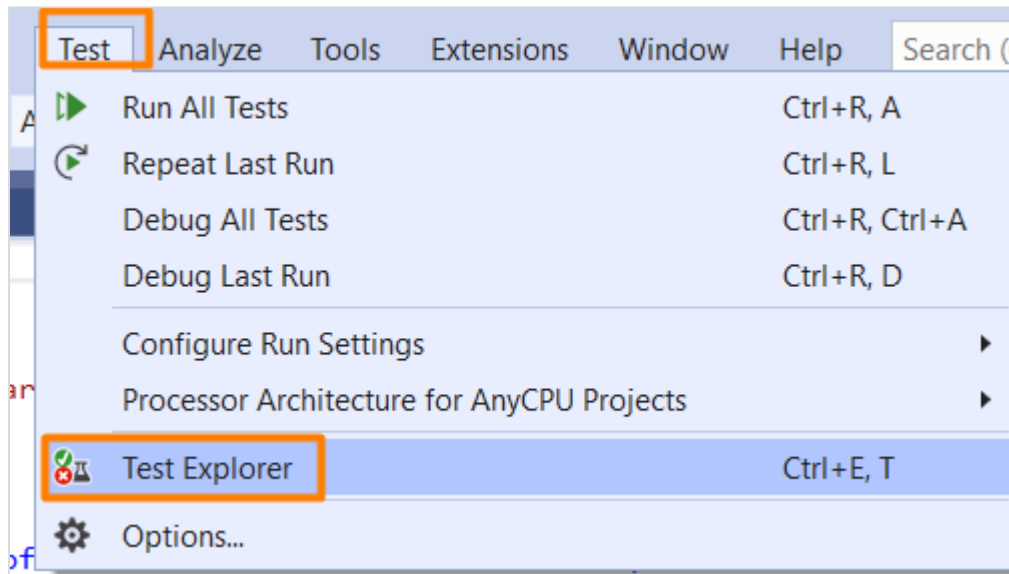


Building and Running Tests

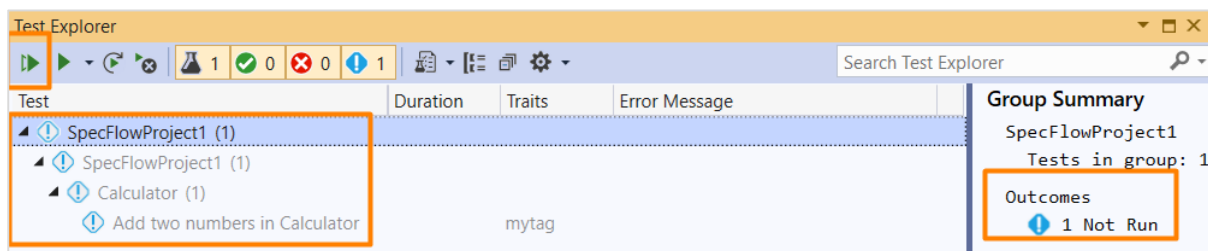
Next, go to the **Build** menu and select **Build Solution**. We should get Build succeeded message as output.



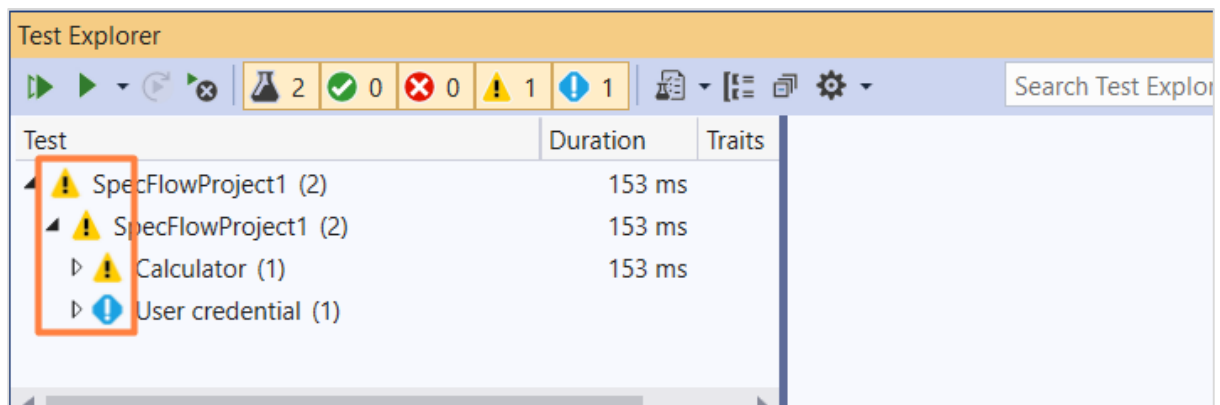
Navigate to the **Tests** menu and choose the **Test Explorer** option.



We should be able to find the Features added to the SpecFlow project. Execute them via the **Run All Tests in View** option. The status of the execution shows as **Not Run** as the tests have still not been executed.

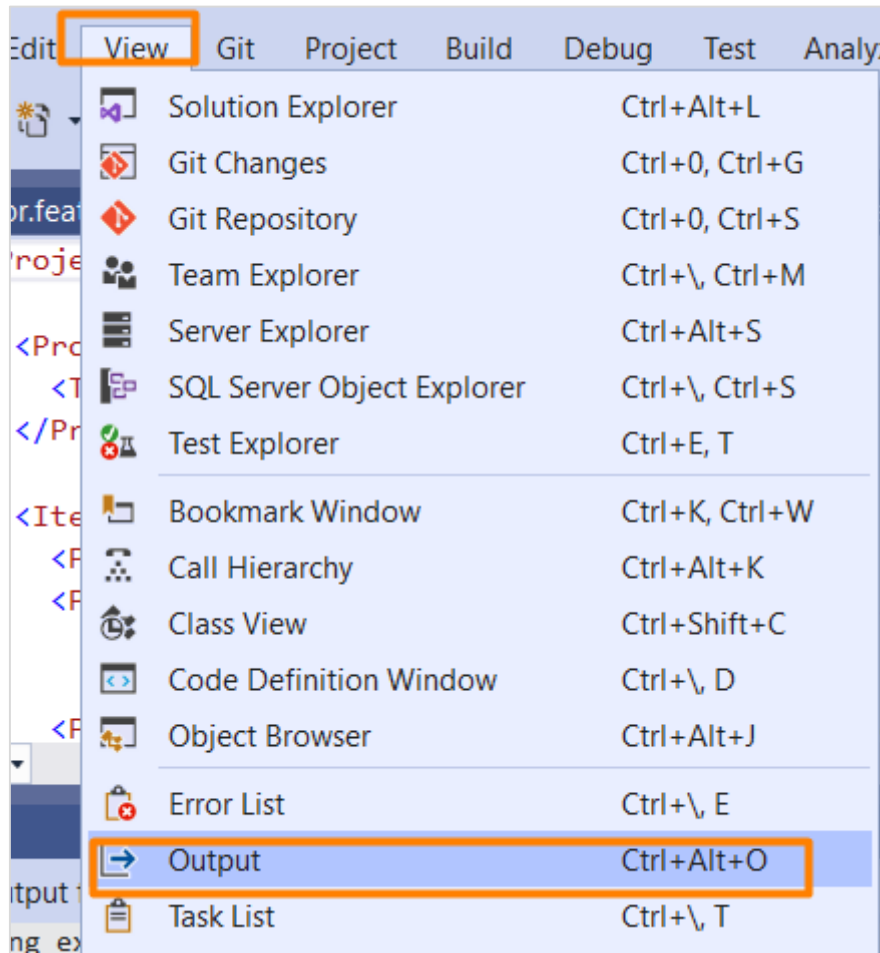


The **exclamation symbol** before a Feature suggests, test execution is pending for that Feature.



8. SpecFlow — Runner Activation

We have to perform the activation of **SpecFlow + Runner**. Navigate to **View** menu, then select the option **Output**.



Then choose **Tests** in the **Show output from:** dropdown.

We should obtain the test output along with the **activation link of the runner**. We should have this link available only if we have chosen **SpecFlow+ Runner** at the time of project set up.

```

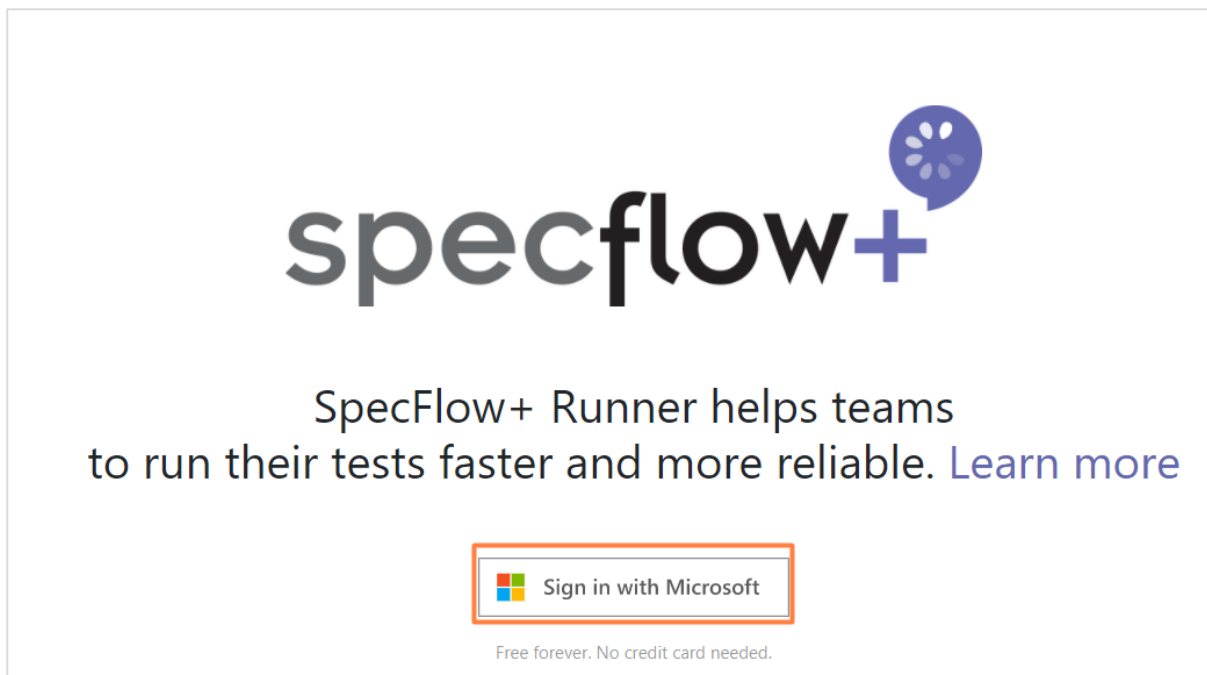
Output
Show output from: Tests
-----
*                               SpecFlow+ Runner                               *
-----
Please sign-up for your free SpecFlow account or sign-in with your already existing account here:
https://account.specflow.org/clientactivation/welcome/Runner/.../NotActivatedYet
to execute your scenarios again.
If you are reading this message on a build agent, we couldn't recognize the agent.
Please get in touch with us at support@specflow.org.
-----
No tests executed (activation needed)
Result: tests executed with warnings
Total: 0
Succeeded: 0
Ignored: 0
Pending: 0
Skipped: 0
Failed: 0

Execution Time: 00:00:00
Adding attachments to VSTest
Log file: file:///C:/Users/.../Desktop/Dehomitalava/Classlibrary2/TestResults/Unnamed%20project_2020-12-30T231638.log
SpecFlow+Runner execution finished
***** Test run finished: 0 Tests run in 2 sec (0 Passed, 0 Failed, 0 Skipped) *****

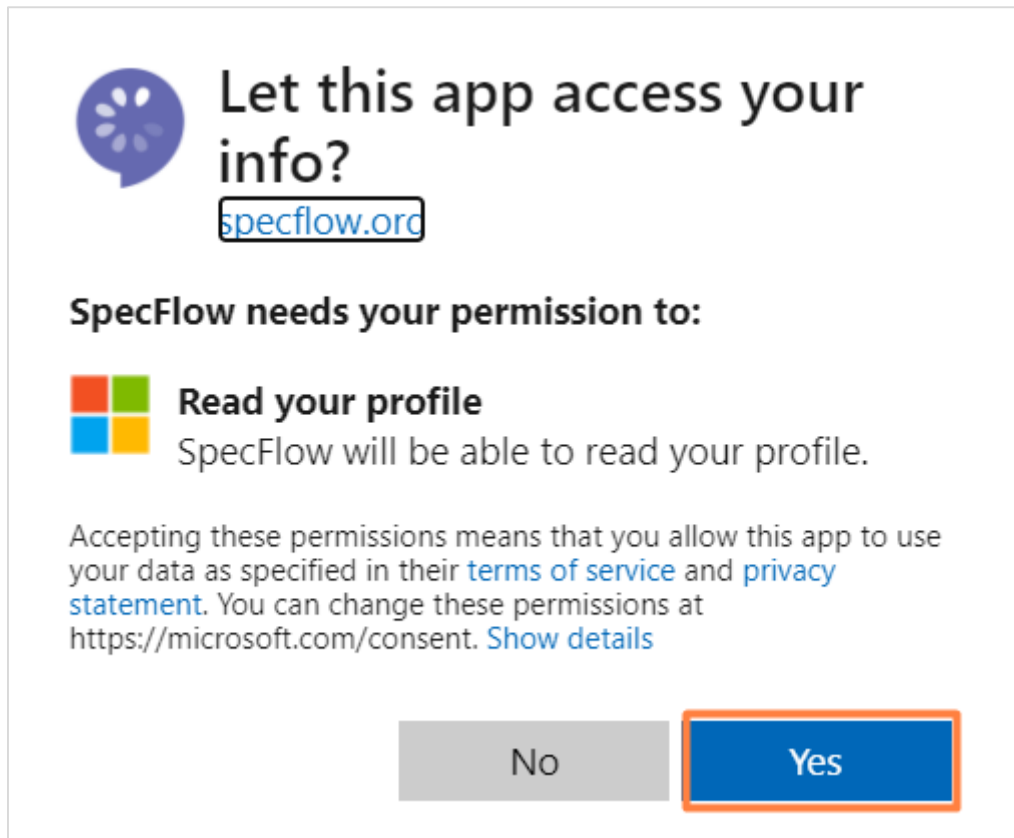
```

Activate Link

Open the activation link on a browser. We should get navigated to the SpecFlow landing page. Click on **Sign in with Microsoft**.



Click on **Yes** for letting **Microsoft** to access our SpecFlow account.



For setting up the account, provide the information needed. Then click on **Create Account**.

Let's set up your account

Where do you work? *

ABC

Where are you from? *

India

What's your role? (optional)

Select your role

- ☒ I have read, acknowledged and accept the [Terms and Conditions](#) and [Privacy Policy](#) and confirm that all indications above are true and correct. *
- ☒ I'm happy with receiving mails occasionally that help me, ⓘ but not spam me. - optional -

Create Account

We shall now have the SpecFlow account successfully activated.

Successfully activated

Welcome to SpecFlow+ Runner!

We're excited you've joined our growing community

You can now use SpecFlow+ Runner

- 1 Switch back to the SpecFlow+ Runner
- 2 Run your Scenarios again
- 3 Watch your Scenarios get executed

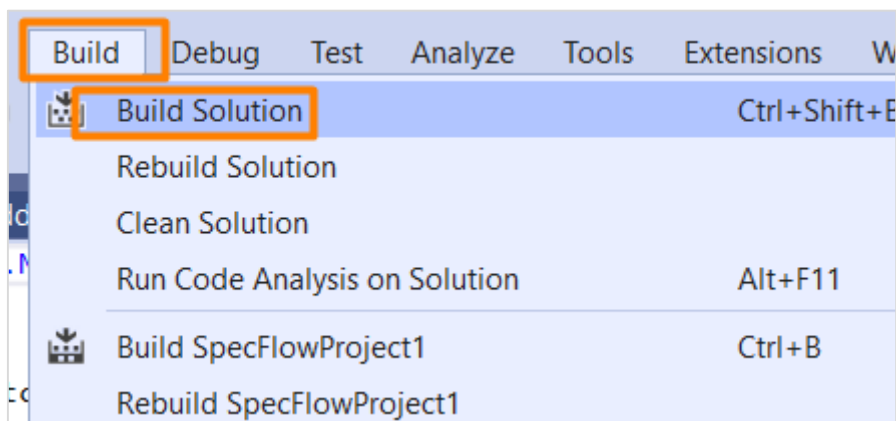
Now, if we again execute the test from the Text Explorer, it will display the proper results.

9. SpecFlow — HTML Reports

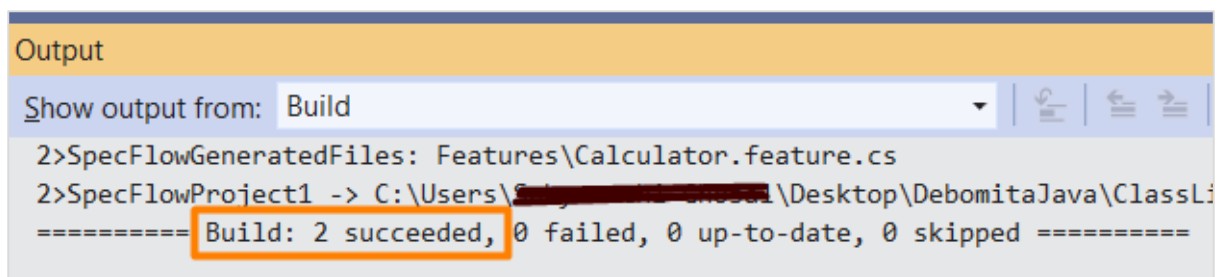
SpecFlow generates reports when all your tests completed executing and which includes breakdown of the test results.

Test Run from Explorer

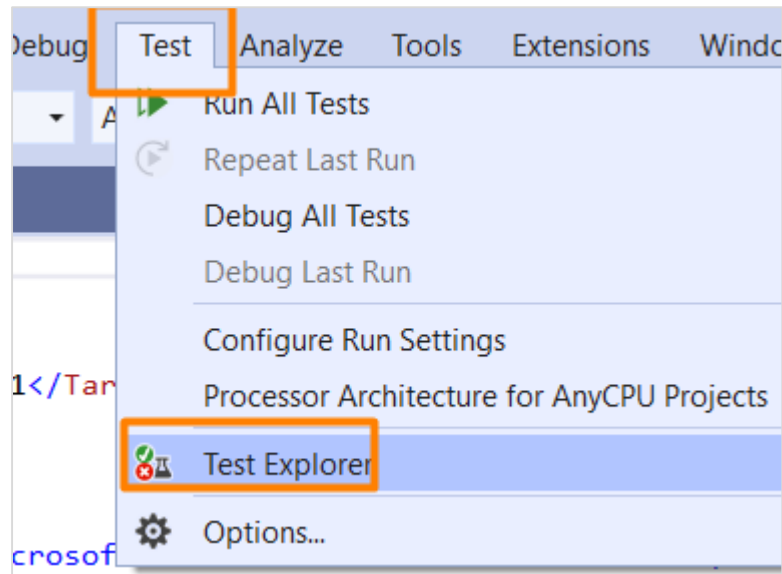
To build a solution, navigate to the **Build** menu, then click on **Build Solution**.



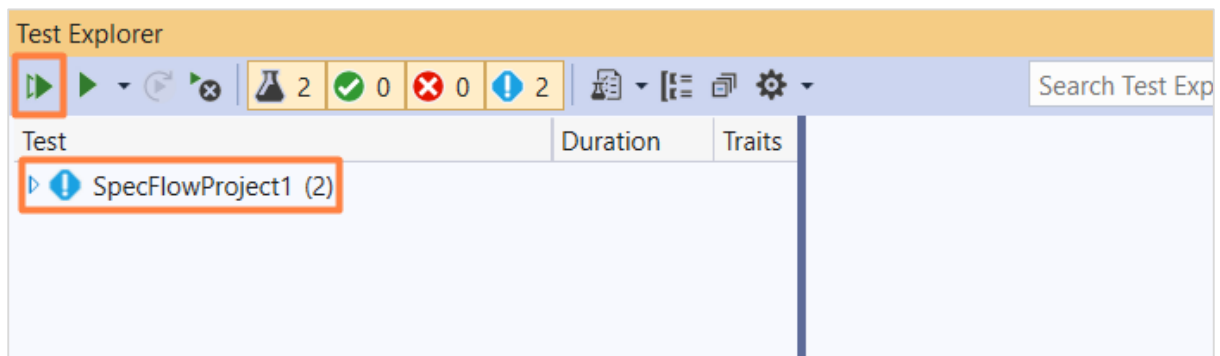
We should get a build success message.



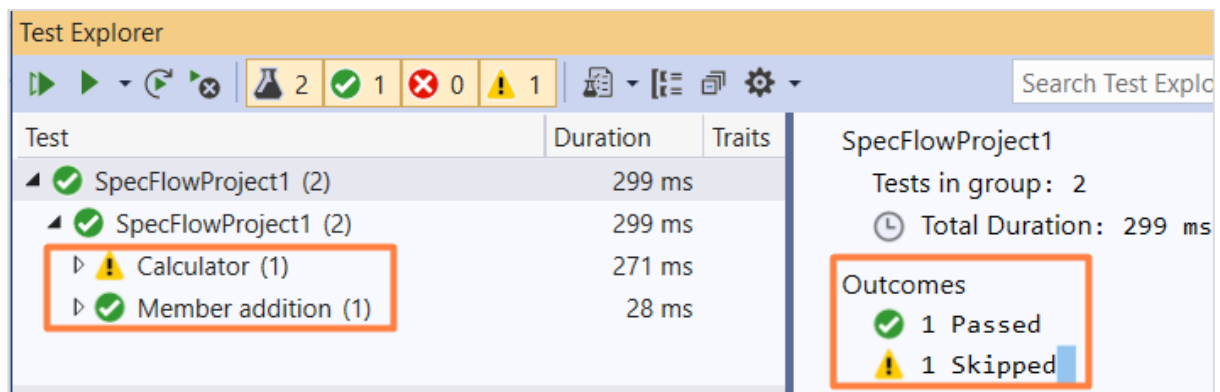
Navigate to the **Tests** menu and choose the **Test Explorer** option.



We should be able to find the Features added to the SpecFlow project. Execute that via the **Run All Tests in View** option.

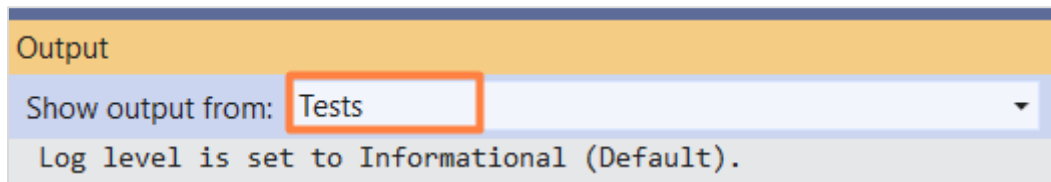


The result is displayed as highlighted in the image below.



Report Generation

Go to the **Output** menu and select **Tests** from the **Show output from:** dropdown.



The total execution results get displayed in the Output Console. It contains information about the count of the test cases, total succeeded, ignored, skipped, failed, and so on. The user and machine names where the execution happened are also captured.

Also, the execution duration is displayed along with the link to the HTML report and the log file path.

```
Result: 1 pending tests (1 succeeded)
Total: 2
Succeeded: 1
Ignored: 0
Pending: 1
Skipped: 0
Failed: 0

Execution Time: 00:00:02.4666886
Report file: file:///C:/Users/.../source/repos/ClassLibrary5/TestResults/Unnamed%20project_2021-01-06T232151.html
Adding attachments to VSTest
adding Test Execution Report 'C:\Users\Sabyasachi Ghosal\source\repos\ClassLibrary5\TestResults\Unnamed project_2021-01-06T232151.html'
Log file: file:///C:/Users/.../source/repos/ClassLibrary5/TestResults/Unnamed%20project_2021-01-06T232151.log
SpecFlow+Krunner execution finished
***** Test run finished: 2 Tests run in 7.4 sec (1 Passed, 0 Failed, 1 Skipped) *****
```

Copy the **Report file** path and open it on the browser. We shall get a detailed HTML report with the project name, configuration, execution start time, duration, number of threads, and so on.

It shall describe the Results, Test Timeline Summary and the complete Feature Summary.

Unnamed project Test Execution Report

- Project: Unnamed project
- Configuration:
- Test Assemblies: C:\Users\Sabyasachi Ghosal\source\repos\ClassLibrary5\SpecFlowProject1\bin\Debug\netcoreapp3.1\SpecFlowF
- Start Time: 06-01-2021 23:21:51
- Duration: 00:00:02.4666886
- Test Threads: 1

Result: 1 pending tests (1 succeeded)

Success rate	Tests	Succeeded	Failed	Pending	Ignored	Skipped
50%	2	1	0	1	0	0

Test Timeline Summary

thread
#0
0s

Test Result View

0.28s
0.14s
0s

Feature Summary

Feature	Success rate	Tests	Succeeded	Failed	Pending	Ignored	Skipped	Duration
Calculator	0%	1	0	0	1	0	0	00:00:00.2715987
Member addition	100%	1	1	0	0	0	0	00:00:00.0289364

The report also consists of the Error Summary and Scenario Summary as well. It contains the Success Rate for each test. To know the details of a particular Feature, we can click on the Scenario Name(provided as a link).

Error Summary

Test Success rate Tests Succeeded Failed Pending Ignored Skipped

Scenario Summary

Feature: Calculator

![[Calculator]](<https://specflow.org/wp-content/uploads/2020/09/calculator.png>)

In order to avoid silly mistakes

As a math idiot

I *want* to be told the **sum** of ***two*** numbers

Link to a feature: [Calculator](SpecFlowProject1/Features/Calculator.feature)

Further read: [**\[Learn more about how to generate Living Documentation\]\(https://docs.specflow.org/projects/specflow-livingdoc/en/latest/LivingDoc.html\)](https://docs.specflow.org/projects/specflow-livingdoc/en/latest/LivingDoc.html)

Test	Success rate	Tests Succeeded	Failed	Pending	Ignored	Skipped	Duration
Scenario: Add two numbers	0%	1	0	0	1	0	00:00:00.2715987

Feature: Member addition

Test	Success rate	Tests Succeeded	Failed	Pending	Ignored	Skipped	Duration
Scenario: Normal user addition	100%	1	1	0	0	0	00:00:00.0289364

Next, the Execution Details are captured for every step. Each step details are displayed with **Trace** and **Result**.

Execution Details

Scenario: Add two numbers (in SpecFlowProject1, Calculator)

tags: mytag

- Status: Pending
- Start time: 06-01-2021 23:21:53
- Execution time (sec): 0.2715987
- Thread: #0

Steps	Trace	Result
Given the first number is 50	pending: CalculatorStepDefinitions.GivenTheFirstNumberIs(50)	Pending in 0.2s
And the second number is 70	skipped because of previous errors	Skipped in 0.004s
When the two numbers are added	skipped because of previous errors	Skipped in 0.002s
Then the result should be 120	skipped because of previous errors Pending: One or more step definitions are not implemented yet. CalculatorStepDefinitions.GivenTheFirstNumberIs(50)	Skipped in 0.016s

Scenario: Normal user addition (in SpecFlowProject1, Member addition)

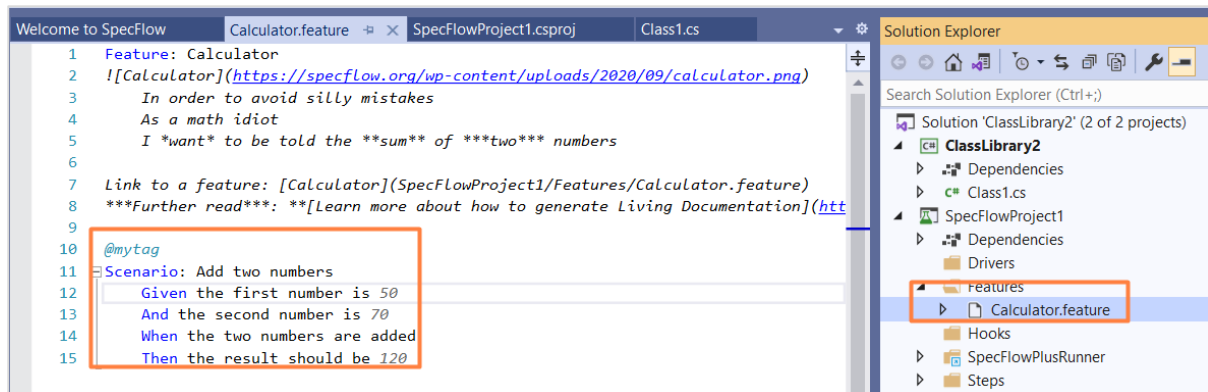
- Status: Succeeded
- Start time: 06-01-2021 23:21:53
- Execution time (sec): 0.0289364
- Thread: #0

Steps	Trace	Result
Given user is on normal user additon screen	User is on normal user addition screen done: MemberAdditionSteps.GivenUserIsOnNormalUserAdditonScreen() (0.0s)	Succeeded in 0.008s
When enters normal user details	User enters normal user details done: MemberAdditionSteps.WhenEntersNormalUserDetails() (0.0s)	Succeeded in 0.004s
Then user should be added as normal user	User should be added as normal user done: MemberAdditionSteps.ThenUserShouldBeAddedAsNormalUser() (0.0s)	Succeeded in 0.003s

10. SpecFlow — Binding Test Steps

Test Steps in Feature File

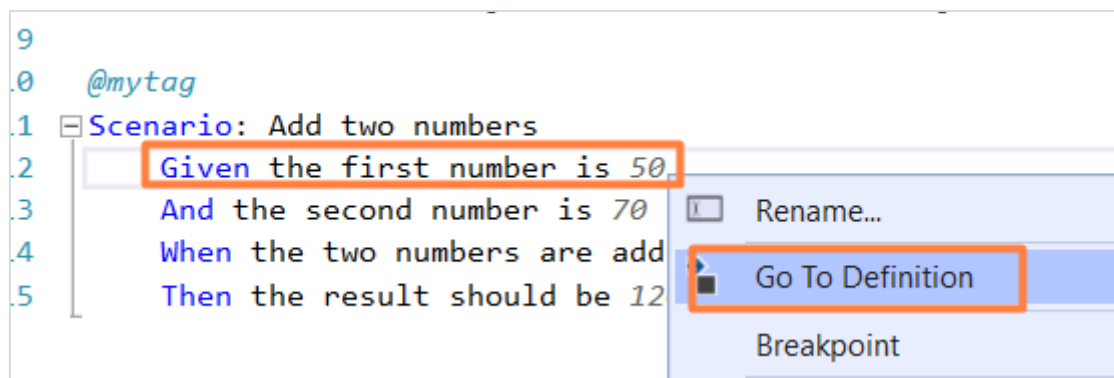
To access the steps in the Feature File, go to the SpecFlow project within the **Solution Explorer**. Then click on the **Features** folder. The Feature File shall be displayed.



A Feature File is useful for documenting the expected characteristics of an application in a format which is in plain text and can also be used for automation. The keywords **Given**, **Then**, **When**, and so on are used in SpecFlow to describe scenarios in **Gherkin** language.

Test Step Binding

The above Feature file has been added by default by the SpecFlow project. It also produces test methods that shall run scenarios defined within the feature file. Right-click on any line on the after the **Scenario** keyword. Then click on the **Go To Definition** option.



Visual Studio identifies the corresponding step definition to this step. In this example, it opens the class **CalculatorStepDefinitions** and moves to the **GivenTheFirstNumberIs** method.


```

[Binding]
1 reference
public sealed class CalculatorStepDefinitions
{
    // For additional details on SpecFlow step definitions see https://go.specflow.o

    private readonly ScenarioContext _scenarioContext;

    0 references
    public CalculatorStepDefinitions(ScenarioContext scenarioContext)
    {
        _scenarioContext = scenarioContext;
    }

    [Given("the first number is (.*)")]
    0 references
    public void GivenTheFirstNumberIs(int number)
    {
        //TODO: implement arrange (precondition) logic
        // For storing and retrieving scenario-specific data see https://go.specflow
        // To use the multiline text or the table argument of the scenario,
        // additional string/Table parameters can be defined on the step definition
        // method
    }
}

```

11. SpecFlow - Creating First Test

We shall now create a file in the class library which performs subtraction of two numbers.

```
using System;
namespace ClassLibrary2
{
    public class Class1
    {
        public int Number1 { get; set; }
        public int Number2 { get; set; }
        public int Subtraction()
        {
            return Number1 - Number2;
        }
    }
}
```

Feature File Implementation

```
9
10 @mytag
11 Scenario: Sub two numbers
12     Given the first number is 70
13     And the second number is 50
14     When the two numbers are subtracted
15     Then the result should be 20
```

Step Definition File Implementation

The corresponding Step Definition file of the above Feature file, along with usage of Class1 to perform subtraction.

```
using ClassLibrary2;
using FluentAssertions;
using TechTalk.SpecFlow;
namespace SpecFlowCalculator.Specs.Steps
{
```

```

[Binding]
public sealed class CalculatorStepDefinitions
{
    private readonly ScenarioContext _scenarioContext;
    //instantiating Class1
    private readonly Class1 _calculator = new Class1();
    private int _result;
    public CalculatorStepDefinitions(ScenarioContext scenarioContext)
    {
        _scenarioContext = scenarioContext;
    }
    [Given("the first number is (.*)")]
    public void GivenTheFirstNumberIs(int number)
    {
        _calculator.Number1 = number;
    }
    [Given("the second number is (.*)")]
    public void GivenTheSecondNumberIs(int number)
    {
        _calculator.Number2 = number;
    }
    [When("the two numbers are subtracted")]
    public void WhenTheTwoNumbersAreSubtracted()
    {
        _result = _calculator.Subtraction();
    }
    [Then("the result should be (.*)")]
    public void ThenTheResultShouldBe(int result)
    {
        _result.Should().Be(result);
    }
}
}

```

Executing the Test

Build the above solution, then execute the test after we obtain the build succeed message from **Test -> Test Explorer**.

Select the **SpecFlowProject1** feature and click on **Run All tests in View**.

The screenshot shows the Test Explorer interface. On the left, a tree view lists tests: SpecFlowProject1 (1), SpecFlowProject1 (1), Calculator (1), and Sub two numbers. The 'Run All tests in View' button (a green play icon) is highlighted with a red box. On the right, the 'Group Summary' for SpecFlowProject1 shows 'Tests in group: 1' and 'Total Duration: 151 ms'. Below this, the 'Outcomes' section shows '1 Passed' with a green checkmark, also highlighted with a red box.

The result shows as **1 Passed** along with execution duration. Click on the option **Open additional output for this result** to get result details.

This screenshot shows the 'Test Detail Summary' for the 'Sub two numbers' test. It displays the test name, duration (137 ms), and source ('Calculator.feature' line 11). A link 'Open additional output for this result' is highlighted with a red box.

The execution result for each test step is displayed.

The screenshot shows the 'Test Outcome' window for the 'Sub two numbers' test. It displays the test name, outcome (Passed), and a detailed 'Standard Output' section. The output contains Gherkin steps and their corresponding Step Definition method calls, all with a status of 'done' and a duration of (0.0s). The entire output section is highlighted with a red box.

```

Test Name: Sub two numbers
Test Outcome: Passed

Standard Output

-> -> Loading plugin C:\Users\...\Desktop\DebomitaJava\ClassLibra
-> -> Loading plugin C:\Users\...\Desktop\DebomitaJava\ClassLibra
-> -> Using default config

Given the first number is 70
-> done: CalculatorStepDefinitions.GivenTheFirstNumberIs(70) (0.0s)

And the second number is 50
-> done: CalculatorStepDefinitions.GivenTheSecondNumberIs(50) (0.0s)

When the two numbers are subtracted
-> done: CalculatorStepDefinitions.WhenTheTwoNumbersAreSubtracted() (0.0s)

Then the result should be 20
-> done: CalculatorStepDefinitions.ThenTheResultShouldBe(20) (0.0s)
  
```

All the steps in the Feature File get executed along with status as done. Also, the corresponding methods in the Step Definition File get displayed with the execution duration.

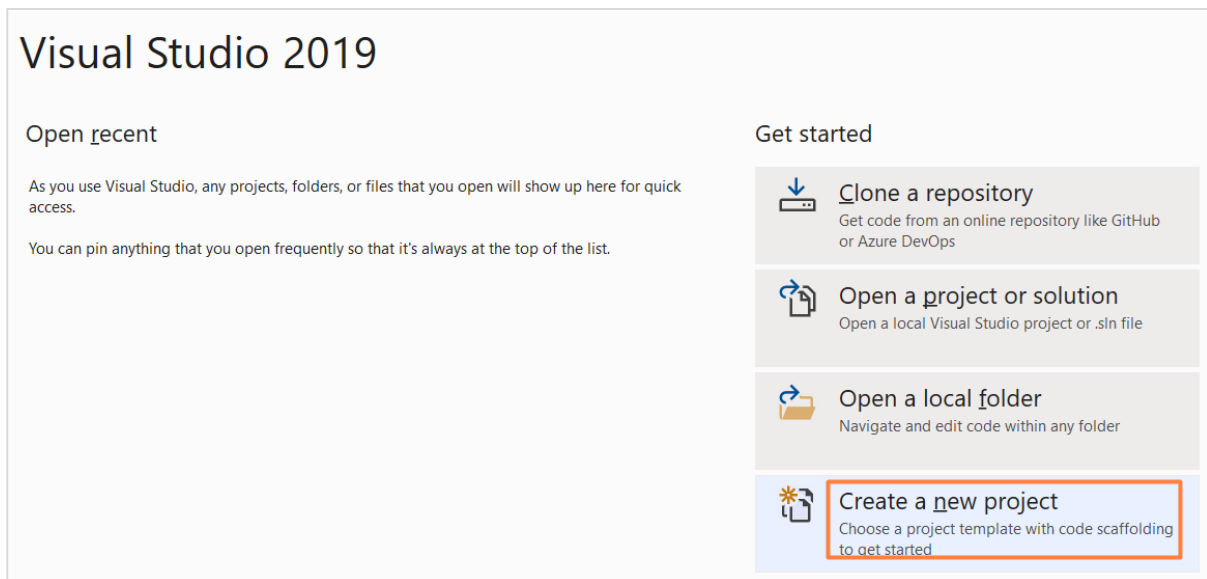
12. SpecFlow — Configure Selenium Webdriver

To configure Selenium Webdriver in Visual Studio, we shall take the help of **NUnit framework**. This framework allows to run Selenium tests in C#.

Configure NUnit Framework

We shall first create a NUnit project.

Once the Visual Studio landing page gets opened, click on **Create a new project**.



Type **NUnit** in the search box appearing in **Create a new project** pop-up. Select **NUnit Test Project(.NET Core)** from the search results. Click on **Next**.

Create a new project

[Clear all](#)

All languages ▾ All platforms ▾ All project types ▾

Recent project templates

- SpecFlow Project C#
- Class Library (.NET Core) C#
- NUnit Test Project (.NET Core) C#

NUnit Test Project (.NET Core)

A project that contains NUnit tests that can run on .NET Core on Windows, Linux and MacOS.

C# Linux macOS Windows Desktop Test Web

NUnit Test Project (.NET Core)

A project that contains NUnit tests that can run on .NET Core on Windows, Linux and MacOS.

Visual Basic Linux macOS Windows Desktop Test Web

NUnit Test Project (.NET Core)

A project that contains NUnit tests that can run on .NET Core on Windows, Linux and MacOS.

F# Linux macOS Windows Desktop Test Web

Not finding what you're looking for?
[Install more tools and features](#)

Back Next

Enter project name and location. Then click on **Create** to proceed.

Configure your new project

NUnit Test Project (.NET Core)

C# Linux macOS Windows Desktop

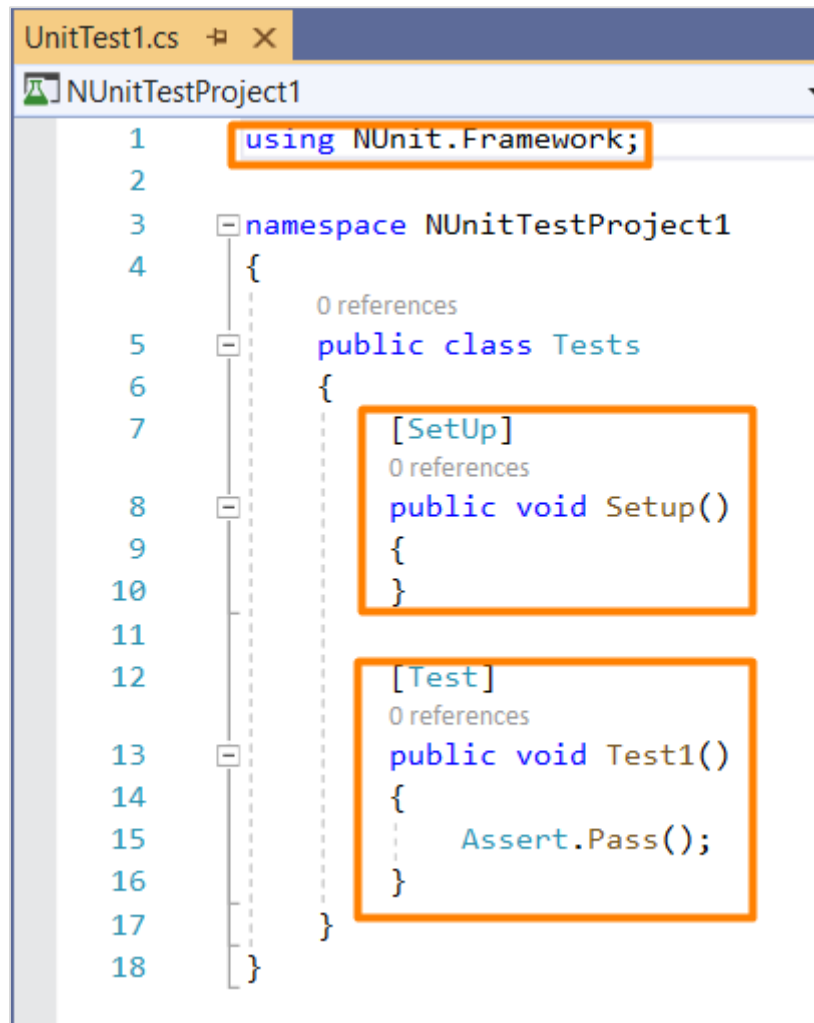
Project name

Location

Solution name ⓘ

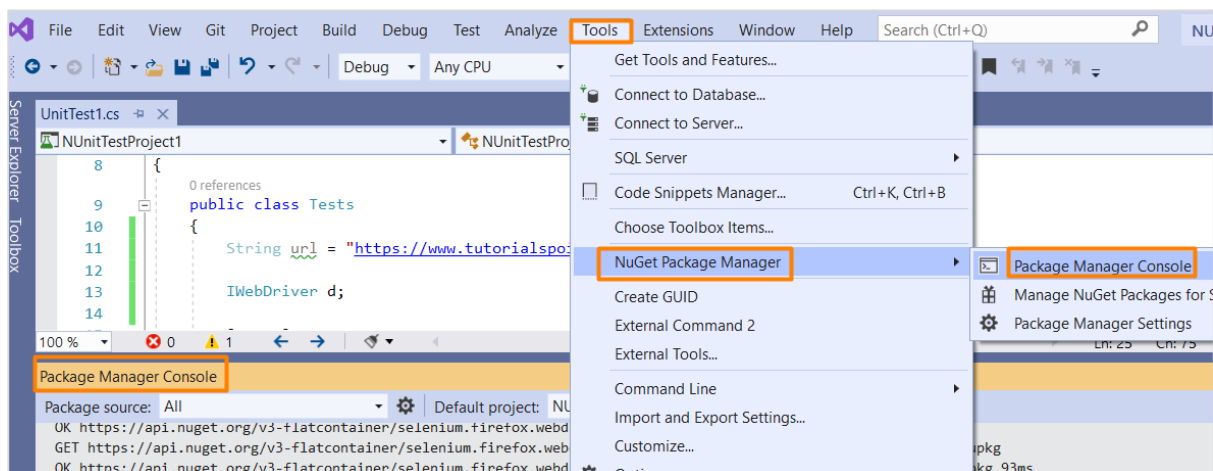
☐ Place solution and project in the same directory

As the project is set up on NUnit(.Net Core), the **Setup** and **Test** methods shall be defined by default. Also, the statement **using NUnit.Framework** should reflect at the top. This shall prove that NUnit Framework has been successfully configured.



Configure Selenium WebDriver

Once the NUnit framework is set, navigate to the **Tools** menu, select **NuGet Package Manager**, and then click on **Package Manager Console**.



We must execute the required **Package Manager** commands for installation of **Selenium Webdriver** and **NUnit**.

For Selenium installation, run the below commands in Package Manager Console:

```
Install-Package Selenium.WebDriver
Install-Package Selenium.Firefox.WebDriver
```

For NUnit installation, run the below commands in Package Manager Console:

```
Install-Package NUnit
Install-Package NUnit3TestAdapter
```

To check the installation status, run the command in Package Manager Console:

```
Get-Package
```

```
PM> Get-Package
```

Id	Versions	ProjectName
--	-----	-----
NUnit	{3.12.0}	NUnitTestProject1
NUnit3TestAdapter	{3.17.0}	NUnitTestProject1
Selenium.Chrome.WebDriver	{85.0.0}	NUnitTestProject1
Selenium.Firefox.WebDriver	{0.27.0}	NUnitTestProject1
Microsoft.NET.Test.Sdk	{16.5.0}	NUnitTestProject1
Selenium.WebDriver	{3.141.0}	NUnitTestProject1

Implementation with Selenium WebDriver in C#

```
using NUnit.Framework;
using OpenQA.Selenium;
using OpenQA.Selenium.Firefox;
using System;
namespace NUnitTestProject1
{
    public class Tests
    {
        String u = "https://www.tutorialspoint.com/index.htm";
        IWebDriver d;
        [SetUp]
        public void Setup()
        {
            //creating object of FirefoxDriver
            d = new FirefoxDriver();
        }
        [Test]
```



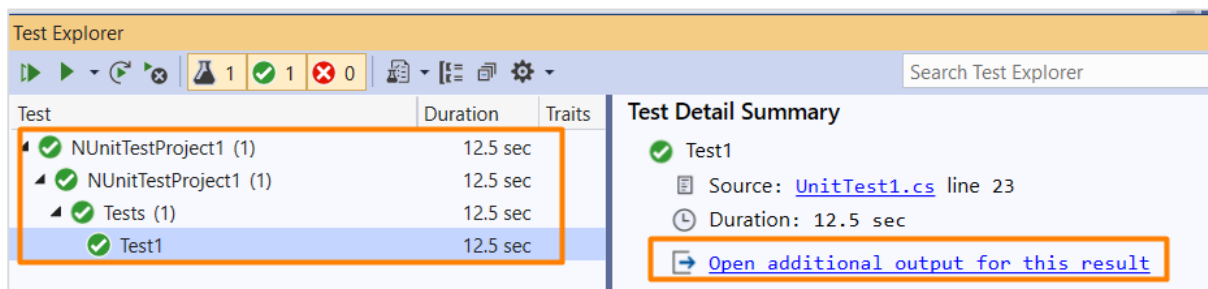
```

    public void Test1()
    {
        //launching URL
        d.Navigate().GoToUrl(u);
        Console.WriteLine("Url launched");
    }
    [TearDown]
    public void close_Browser()
    {
        d.Quit();
    }
}
}

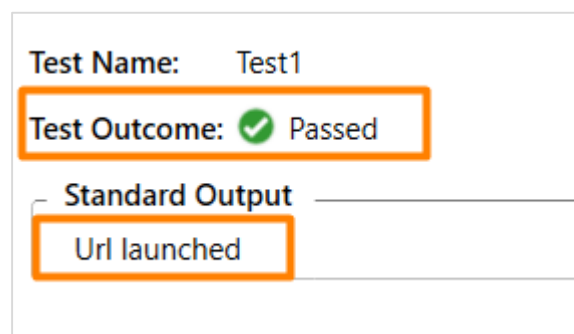
```

Execution & Output

Run the above code from **Test->Test Explorer**. The output in Test Explorer is:



Click on **Open additional output for this result** link, we should get the **Test Outcome** and **Standard Output**.



Url launched is obtained as an output as implemented with **Console.WriteLine** method in the code.

13. SpecFlow — Gherkin

A **Gherkin** is a group of important **keywords** to build a meaningful architecture for specifications. Every keyword is converted to plain spoken languages like English.

Rules in Gherkin

Some of the rules in Gherkin are listed below:

- A document in Gherkin begins with keywords.
- Comments can be added at the beginning of the new line in the Feature File. They start with or without spaces followed by **#** symbol and text. However, block comments cannot be added till now in SpecFlow.
- To indent the code, spaces or tabs can be used. It is recommended to have two spaces for indentation.
- The content after the keyword for each step has a corresponding block of code. This is known as the **Step Definition**.

Gherkin Example

Following is the Gherkin example:

```
Feature: Payment Functionality

    # Example 1
    Scenario: Member Payment Method
        When a member is on Payment screen
        Then the payment amount is displayed.

    # Example 2
    Scenario: Member Payment Dues
        When a member is on Payment Due screen
        Then the payable amount is displayed.
```

In the above example, **Feature**, **Scenario**, **Given**, **When**, and **Then** are known as the Gherkin keywords.

14. SpecFlow — Gherkin Keywords

The main Gherkin keywords are:

- Feature
- Scenario
- Rule(till Gherkin 6)
- Example or Scenario
- Background
- Scenario Outline
- Examples
- | for Data table
- "" for Document Strings
- @ for Tags
- # for Comments
- Given
- When
- Then
- But
- And

Gherkin uses localization for multiple languages and each of the above keywords has its equivalent terms in respective languages.

Let us explore some of the important Gherkin keywords:

Feature

A Feature is added to have an **overall description of the features** of the applications and to club connected scenarios. This is the most important keyword in a Gherkin document.

A Feature is followed by a **colon:** symbol and then a small description on the feature. We can add multiple lines for more description. These are not considered by SpecFlow at execution but are added in the html reports.

Feature: Payment Functionality

**The Payment is an important module.
There can be several modes of payment.**

Example 1

Scenario: Member Payment Method

When a member is on Payment screen

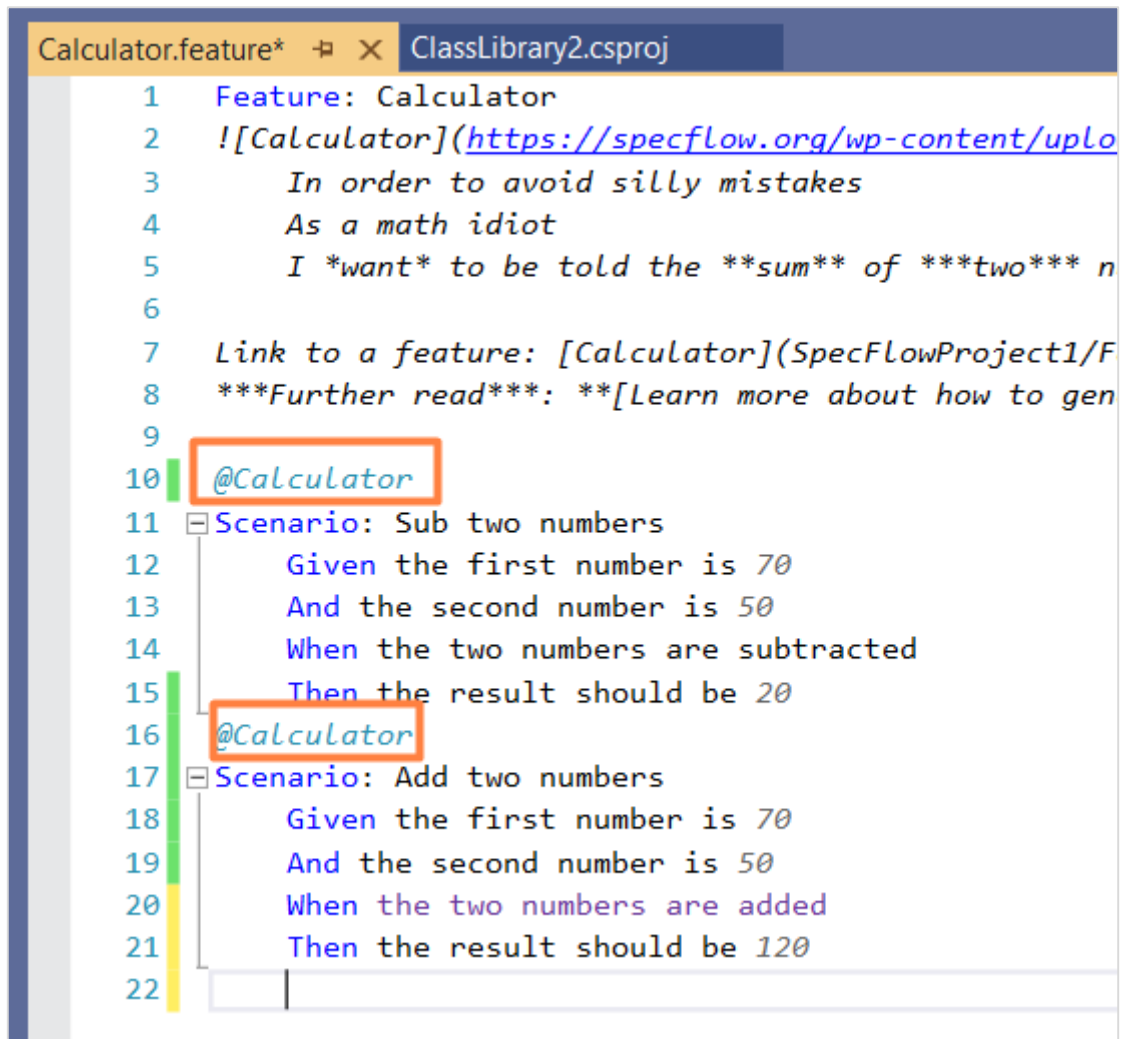
Then the payment amount is displayed.

Once the description of a Feature is completed, we should begin a new line with keywords Background, Example, and so on. We can add tags above Feature to club similar features, irrespective of the structure of file or directory.

Tags

Tags are markers added to Scenarios or Features. Giving a tag to a Feature is like marking that tag to every Scenario within that Feature file. A tag name is mentioned after the @ symbol.

We can filter and club tests to be run with the tags. For instance, we can tag an urgent test with **@important** and run it quite often. SpecFlow considers the **@ignore** tag as an important one and produces an ignored unit test method out of the Scenarios with this tag.

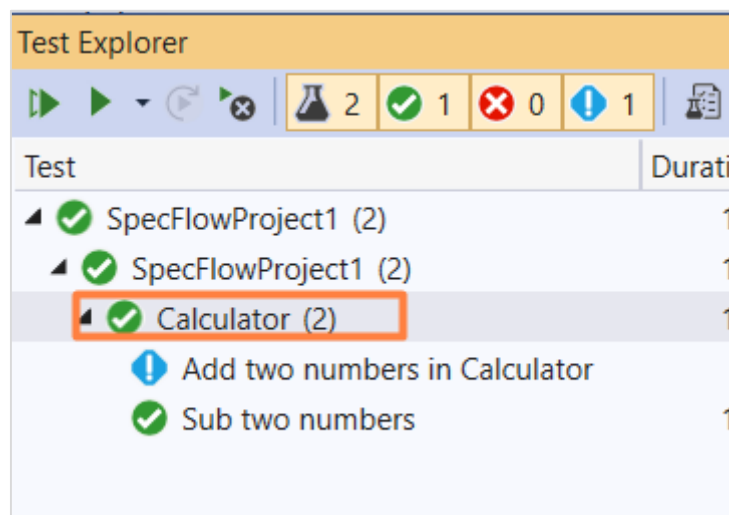


```

1  Feature: Calculator
2  ![Calculator](https://specflow.org/wp-content/uplo
3      In order to avoid silly mistakes
4      As a math idiot
5      I *want* to be told the **sum** of ***two*** n
6
7  Link to a feature: [Calculator](SpecFlowProject1/F
8  ***Further read***: **[Learn more about how to gen
9
10 @Calculator
11 Scenario: Sub two numbers
12     Given the first number is 70
13     And the second number is 50
14     When the two numbers are subtracted
15     Then the result should be 20
16 @Calculator
17 Scenario: Add two numbers
18     Given the first number is 70
19     And the second number is 50
20     When the two numbers are added
21     Then the result should be 120
22

```

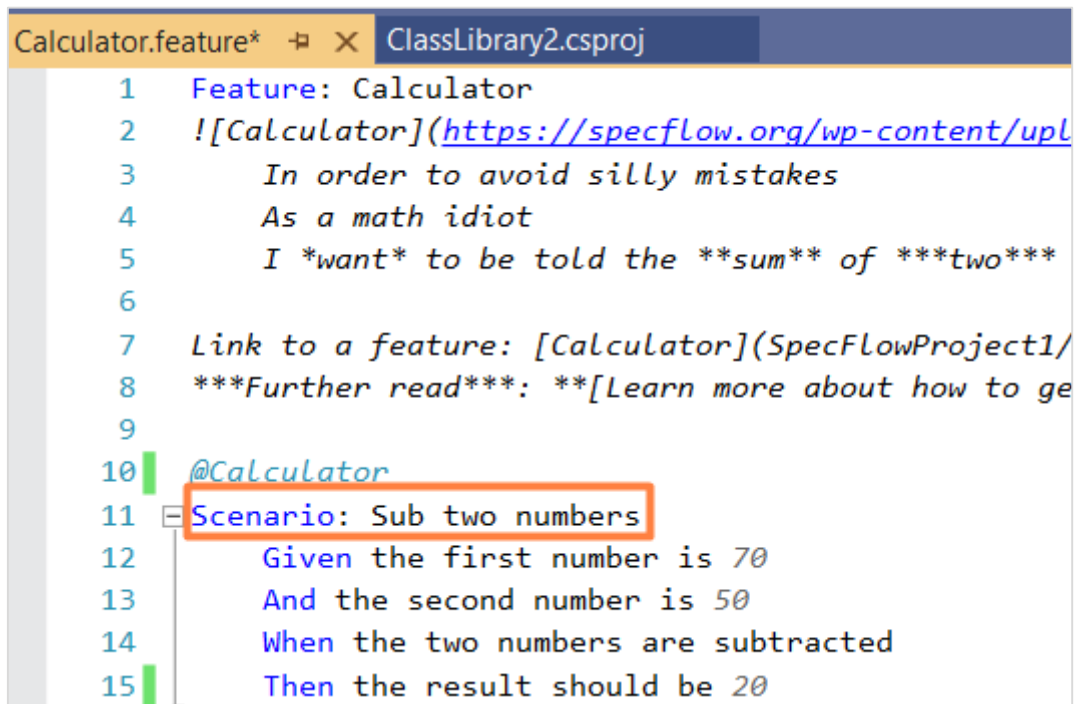
Here, the Feature File contains two scenarios with **@Calculator** tag. The same shall also be reflected in the **Test Explorer**, to pick and choose the test to be run.



Scenario

Scenario is a complete instance that describes a business logic. It has multiple steps. It is often considered a synonym of keyword **Example**. A Scenario does not have a fixed number of steps. But it is recommended to have 3 to 5 steps per Scenario.

If there are too many steps, it may lose its value to be used as specification and documentation. A Scenario is like a test in a development lifecycle. Also, it can be divided into a precondition, test step and verification.



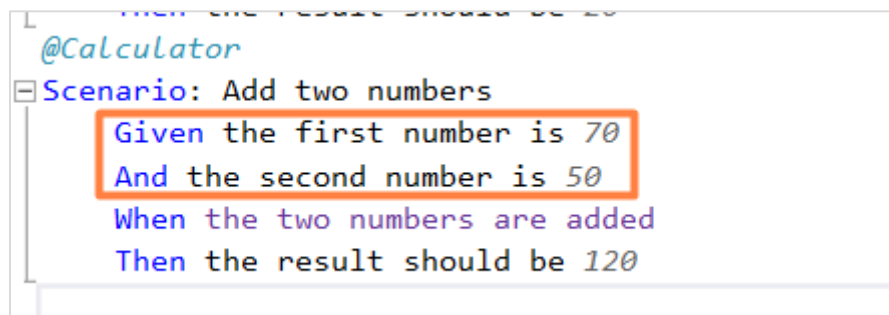
```

Calculator.feature*  X  ClassLibrary2.csproj
1  Feature: Calculator
2  ![Calculator](https://specflow.org/wp-content/upl
3      In order to avoid silly mistakes
4      As a math idiot
5      I *want* to be told the **sum** of ***two***
6
7  Link to a feature: [Calculator](SpecFlowProject1/
8  ***Further read***: **[Learn more about how to ge
9
10 @Calculator
11 Scenario: Sub two numbers
12     Given the first number is 70
13     And the second number is 50
14     When the two numbers are subtracted
15     Then the result should be 20
  
```

Given

Given are steps used for describing the **pre-existing** condition of the system. It typically deals with the events that have occurred in the past. As a Given step is executed, it shall set the objects, test data in the database and put the system in a proper state.

Thus, the Given step helps to define the system in a known condition prior to the interaction of the user with the system. We can have multiple Given steps. Two or more Given steps can be used with **And** keyword. In short, it is used to have the preconditions defined.



```

@Calculator
Scenario: Add two numbers
    Given the first number is 70
    And the second number is 50
    When the two numbers are added
    Then the result should be 120
  
```

When

When is a step used for describing an action or an incident. This can either be an interaction of the person with the system or an incident caused by another system. It is a good practise to have a single When step in a Scenario.

If we are forced to have multiple When steps, we should ideally break the Scenario into smaller ones.

```

    Then the result should be 20
  @Calculator
  Scenario: Add two numbers
    Given the first number is 70
    And the second number is 50
    When the two numbers are added
    Then the result should be 120
  
```

Then

Then is a step used for describing an expected result. The corresponding step definition of a Then step should have an **assertion** to verify actual result against the expected result.

Thus, it basically deals with the output obtained from the tests (message, report, and so on) and not on the internal characteristics of the system, for instance a database record. In other words, it is used for an outcome that is noticeable from the end user perspective.

```

  @Calculator
  Scenario: Add two numbers
    Given the first number is 70
    And the second number is 50
    When the two numbers are added
    Then the result should be 120
  
```

But, And

If we have repeated Given, When and Then steps, then we can make the Scenarios more organized by replacing the consecutive Given, When, Then steps with And, But steps.

```

@Calculator
Scenario: Add two numbers
    Given the first number is 70
    And the second number is 50
    When the two numbers are added
    Then the result should be 120
    But the result should not be negative

```

The above example shows the usage of And and But.

The * symbol

The * symbol is used in place of another step keyword. This can be used for steps that represent a list of items. It is more like a bullet point. For the below example, two And steps have appeared one after the other.

```

@Calculator
Scenario: Add two numbers
    Given the first number is 70
    And the second number is 50
    And the third number is 30
    When the two numbers are added
    Then the result should be 150
    But the result should not be negative

```

The consecutive And steps should be represented like this:

```

@Calculator
Scenario: Add two numbers
    Given the first number is 70
    * the second number is 50
    * the third number is 30
    When the two numbers are added
    Then the result should be 150
    But the result should not be negative

```

Background

Sometimes, we may require repeating the same steps for all Scenarios within the Feature file. We may shift these steps to the backdrop by clubbing them under the Background segment.

It helps to add context to a scenario. It can have more than one Given step. Thus, it shall execute prior to execution of each Scenario, but post any **Before hooks**.

A Background is kept prior to the first **Example** or **Scenario**, at the similar indentation level. In short, Background is used for declaring the common steps to all the tests.

```
Feature: Member addition

Background:
    Given launch URL
    Then enter name and password

Scenario: Normal user addition
    Given user is on normal user additon screen
    When enters normal user details
    Then user should be added as normal user

Scenario: Admin user addition
    Given user is on admin user addition screen
    When enters admin user details
    Then user should be added as admin user
```

In the above example, having two Scenarios, the Background steps shall run once before execution of each of these scenarios.

Scenario Outline

Scenario Outline is used to replicate the same Scenario with a different data set. Writing the same tests with different values is cumbersome and time taking. For instance,

```
Feature: User credential

Scenario: Normal User Login
    Given user enters user1 as user and pwd1 as password
    Then user should be logged in as normal

Scenario: Admin user login
    Given user enters user2 as user and pwd2 as password
    Then user should be logged in as admin
```

We can club the above two scenarios with the **Scenario Outline**.

Feature: User credential

Scenario Outline: Login module

Given user types <username> and <password>

Then user should be able to login

Examples:

username	password
user1	pwd1
user2	pwd2

Thus, we see that a Scenario Outline should be accompanied with keyword **Examples**. A Scenario Outline is executed once for each of the rows appearing below the Examples segment.

Also, we have seen that the Given step has the <> delimiter. It points to the header of the Examples table. SpecFlow shall put the values within this table prior to the task of matching a step with a Step Definition.

Data Table

Data Table is used to send a group of values in the form of a list to the Step Definition file. It is useful to deal with large data sets. SpecFlow has a rich API for table manipulation in the Step Definition File.

SpecFlow Assist Helpers packages are used to work on tables. Also, we have to add namespace **TechTalk.SpecFlow.Assist** to our code.

Feature: User credential

Scenario: Login module

When User types details

Name	Password
t1	pwd
t2	pwd1

Then user should be able to login

15. SpecFlow — Feature File

The SpecFlow test execution begins from the **Feature File**. Here all the Features and their corresponding Scenarios are explained in plain text. It has a dual role of serving as an automation element as well as for documentation. A Feature File consists of one or more Scenarios in form of a list. The extension for a Feature File should always be **.feature**.

A Feature is added to have an **overall description of the features** of the applications and to club connected scenarios. This is the most important keyword in a Gherkin document.

A Feature is followed by a **colon:** symbol and then a small description on the feature. We can add multiple lines for more description. These are not considered by SpecFlow at execution but are added in the html reports.

```
Feature: Payment Functionality

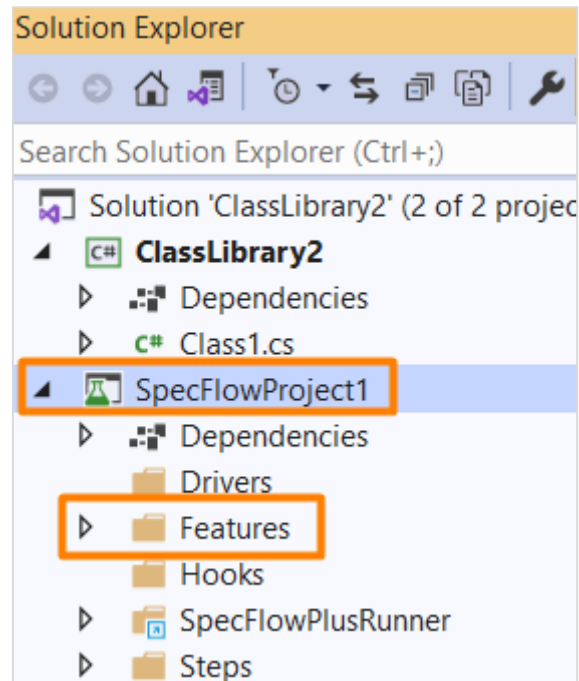
    The Payment is an important module.
    There can be several modes of payment.

    # Example 1
    Scenario: Member Payment Method
        When a member is on Payment screen
        Then the payment amount is displayed.
```

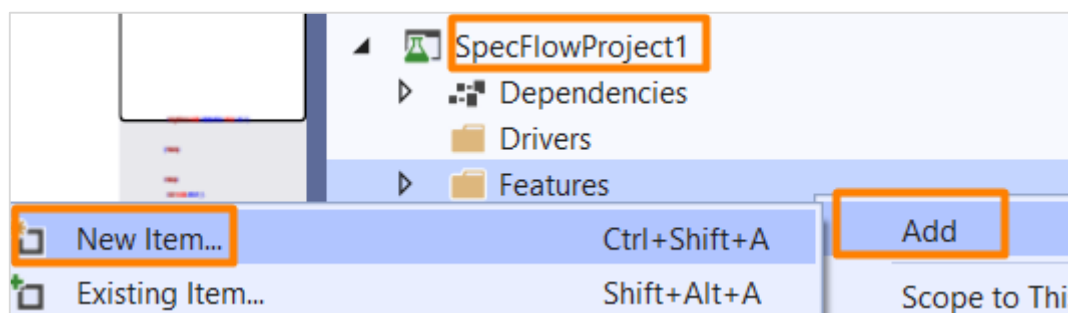
Once the description of a Feature is completed, we should begin a new line with keywords Background, Example, and so on. We can add tags above Feature to club similar features, irrespective of the structure of file or directory.

Feature File Creation

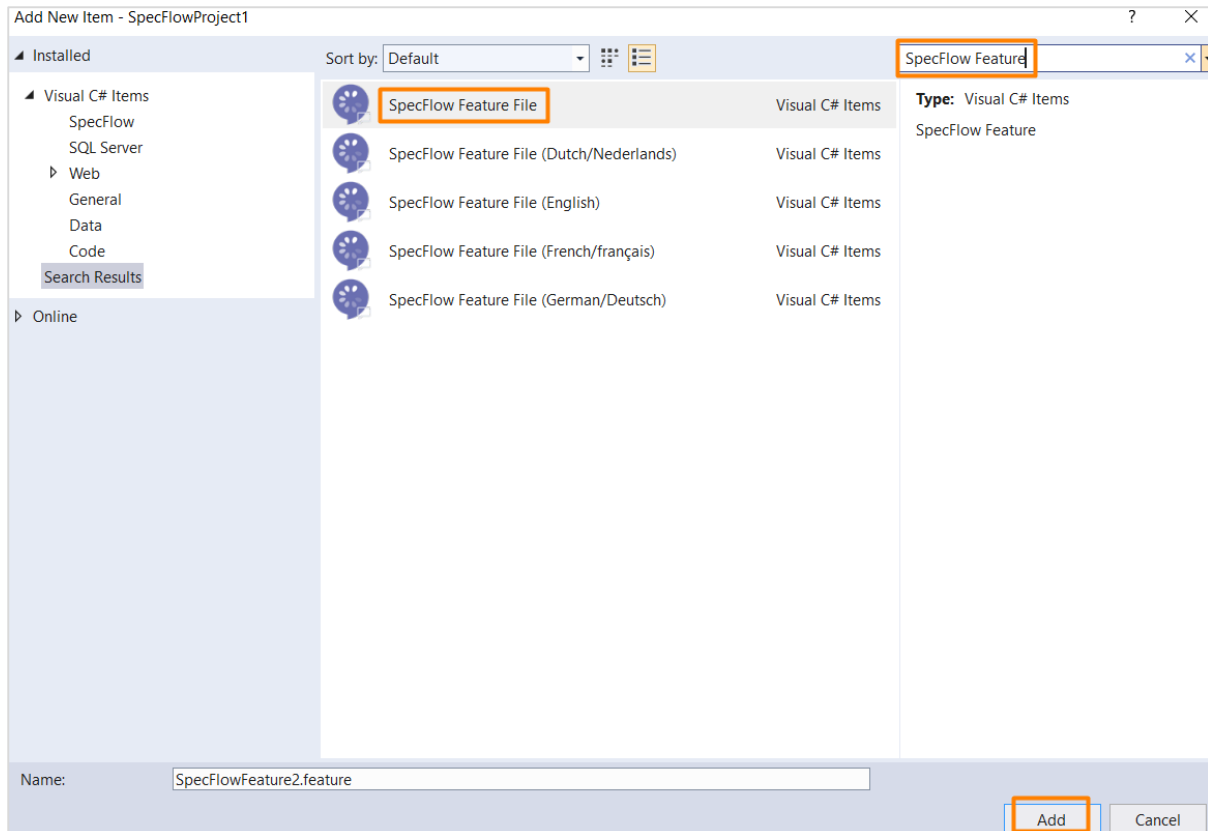
Once a SpecFlow project is created, go to the **Solution Explorer**, and expand it.



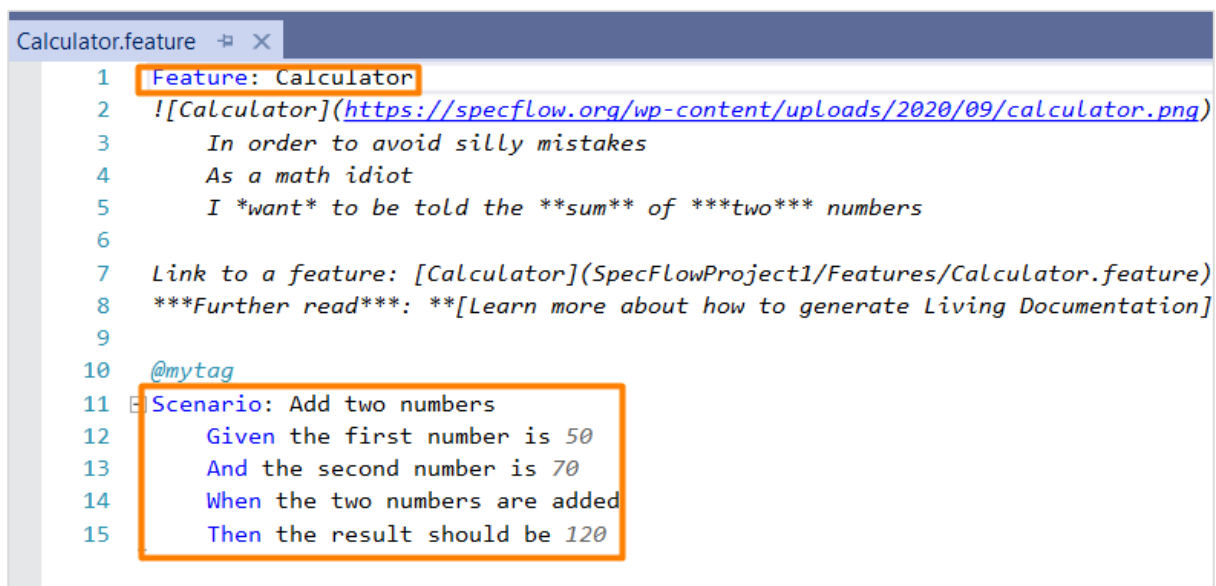
Right-click on **Features** folder. Click on **Add**, then select the option **New Item**.



Add New Item pop-up comes up. Type **SpecFlow Feature** in the search box. Select the option **SpecFlow Feature File** from the search results. Click on **Add** and proceed.

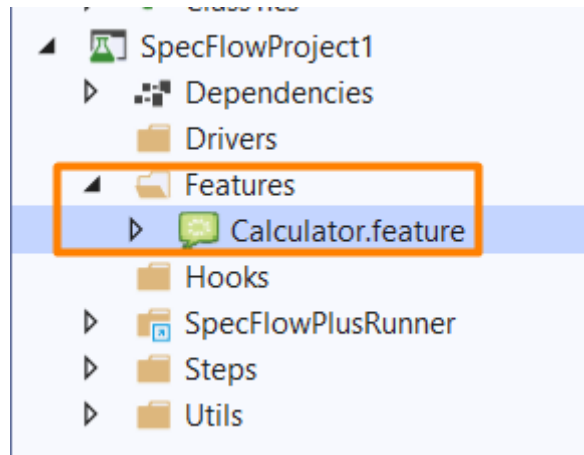


The Feature File gets generated with few steps created by SpecFlow by default.



Project Folder after Feature File Creation

Following is the project folder after the feature file is created.



A Feature File is mainly composed of the Gherkin Keywords to take a form of a Feature having one or multiple Scenarios.

16. SpecFlow — Step Definition File

To execute the Feature file, we must add the implementation logic for each of the steps. To add the definition of the step in SpecFlow, the C# language is used. Thus, a **Step Definition File** contains methods developed in C# within a Class.

The methods have annotations along with a pattern to connect the Step Definition to every matching step. The SpecFlow shall run the code to execute the keywords in Gherkin.

A Step Definition file is a link between the application interfaces and Feature File. For providing readability features, the Step Definition File can have parameters. This signifies that it is not required to have a step definition for each step that has a minor difference.

For instance, **Given Login to admin application** and **Given Login to payment application** steps can be automated with one step definition by passing **admin** and **payment** as parameters. The regular expression (.*) is used to declare parameters for a method.

Rules for Regular expressions

The rules for regular expressions are listed below:

- It is matched with the complete step, even though we are not using the markers ^ and \$.
- The capturing groups in the regular expression describe the parameters for the method in order.

Rules for Step Definition Methods

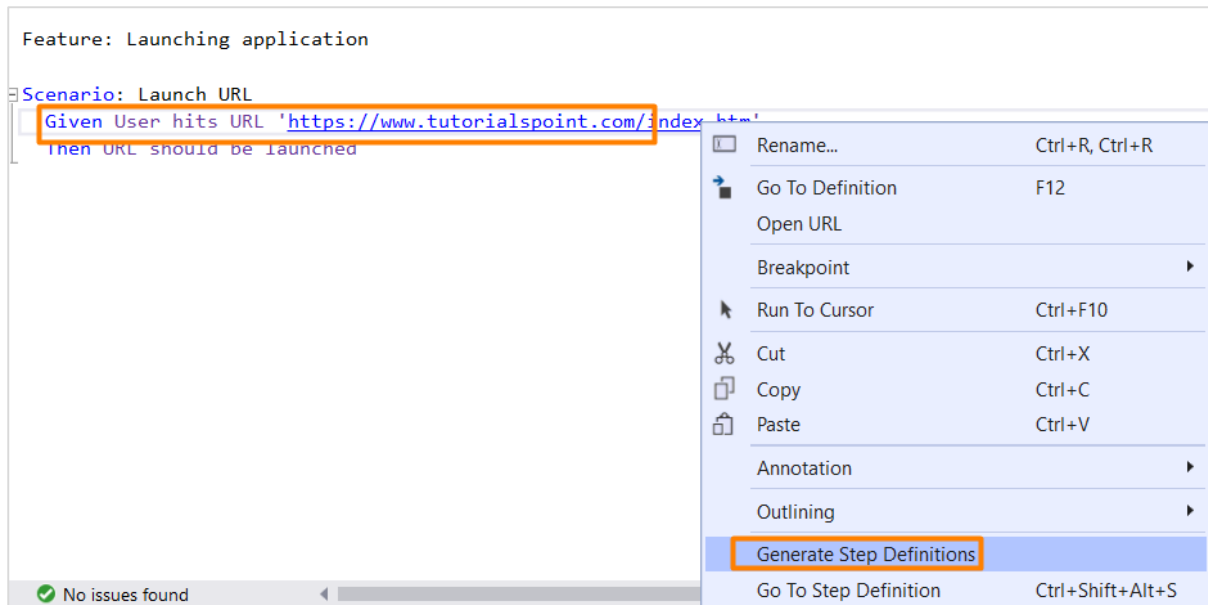
The rules to be followed for Step Definition methods are listed below:

- It should have a **[Binding]** attribute and reside within a public class.
- It should be public.
- It can either have a static or non-static method. If it is a non-static method, an object should be instantiated once for every scenario of the class where it resides.
- It should not have **ref** or **out** parameters.
- It cannot have a return type.

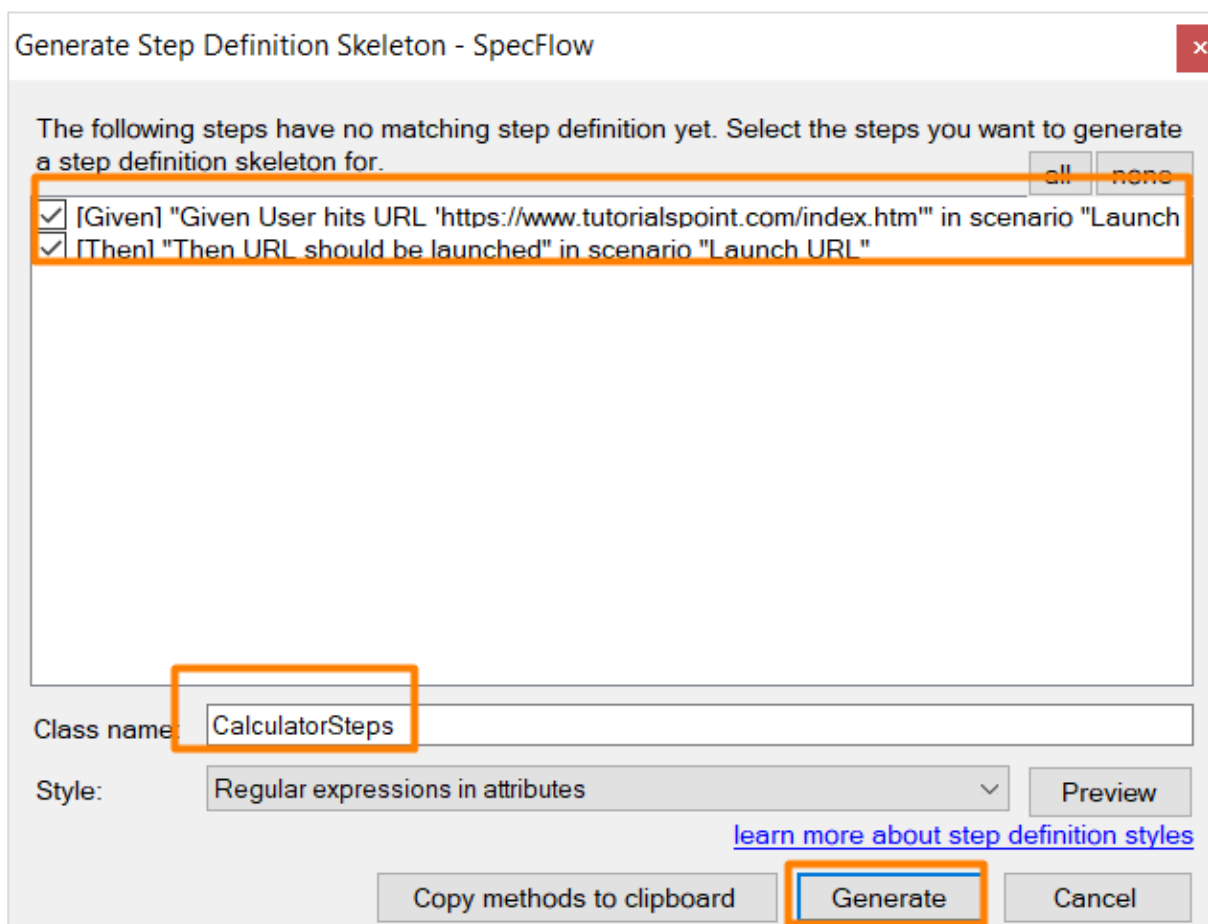
Step Definition File Creation

Right-click on any step of the Feature File, then click on **Generate Step Definitions** option.

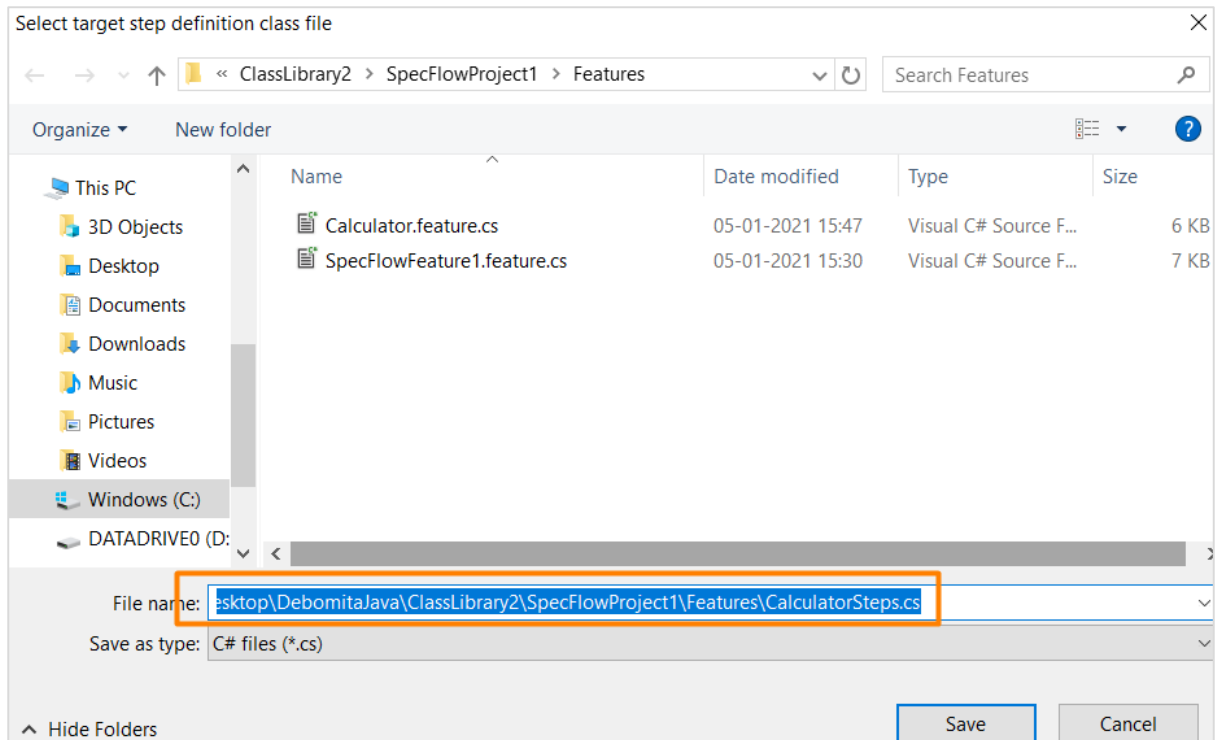
The details of how to create a Feature File is discussed in detail in the Chapter – Feature File.



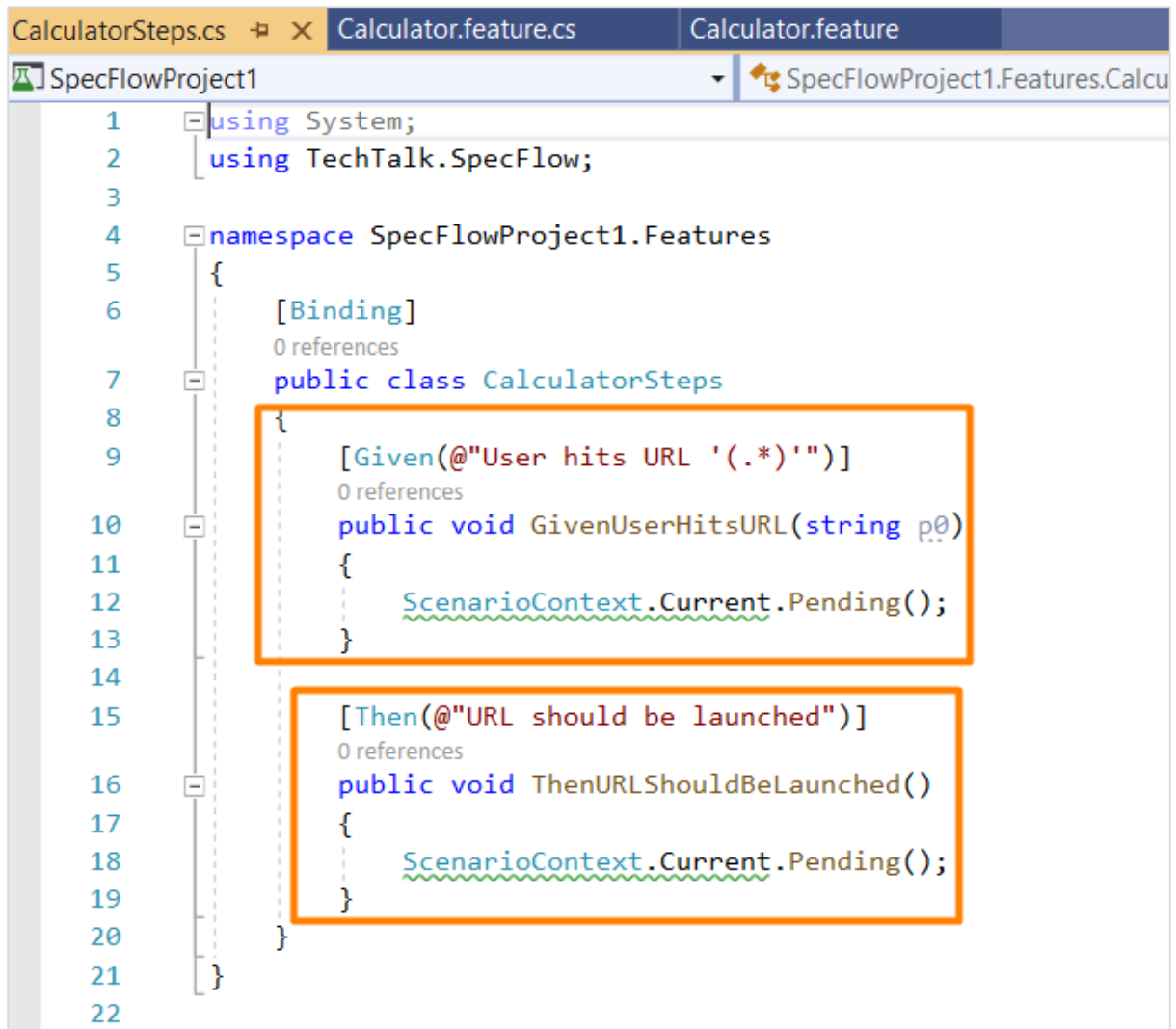
In the **Generate Step Definition Skeleton** pop-up, check the steps for which we want to generate the implementation. Add a **Class Name**, then click on the **Generate** button.



Give the location of saving the Step Definition File and then click on **Save**.



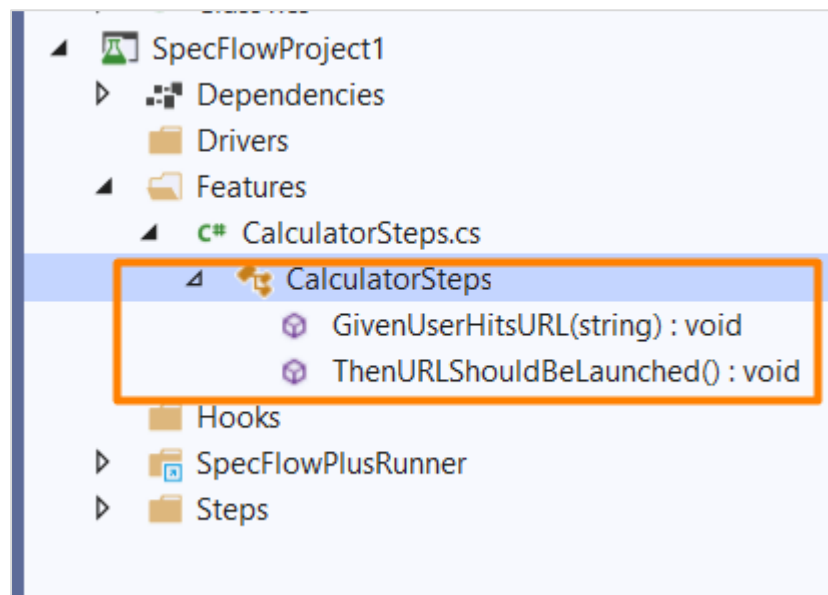
The Step Definition File gets opened with for all the matching steps in the Feature File. It also contains regular expression attributes.



```
1  using System;
2  using TechTalk.SpecFlow;
3
4  namespace SpecFlowProject1.Features
5  {
6      [Binding]
7      0 references
8      public class CalculatorSteps
9      {
10         [Given(@"User hits URL '(.*)'")]
11         0 references
12         public void GivenUserHitsURL(string p0)
13         {
14             ScenarioContext.Current.Pending();
15         }
16
17         [Then(@"URL should be launched")]
18         0 references
19         public void ThenURLShouldBeLaunched()
20         {
21             ScenarioContext.Current.Pending();
22         }
23     }
24 }
```

Project Folder after Step Definition File Creation

Following is the project folder after the step definition file is created:



17. SpecFlow — Hooks

Hooks are event bindings to add more automation logic at certain steps. For example, for any step which is needed to be run prior to a specific Scenario. To introduce, hooks in the code we have to add the **[Binding]** attribute.

Hooks have global access. But it can be made available to a Features and Scenarios by declaring a scoped binding. The scoped binding can be filtered with the tags.

SpecFlow+ Runner Limitations

If we are executing tests from more than one thread with SpecFlow+ Runner, the After and Before hooks like the **BeforeTestRun** and **AfterTestRun** are run only once for each thread.

Hook Attributes

The Hook attributes are listed below:

BeforeTestRun/AfterTestRun: This is used to run an automation logic prior/post to the complete test execution. The method it is applicable to should be static.

BeforeFeature/AfterFeature: This is used to run an automation logic prior/post to individual Feature execution. The method it is applicable to should be static.

BeforeScenario or Before/AfterScenario or After: This is used to run an automation logic prior/post to individual Scenario or Scenario Outline execution.

BeforeScenarioBlock/AfterScenarioBlock: This is used to run an automation logic prior/post to individual Scenario block execution. (in between the When and Given steps).

BeforeStep/AfterStep: This is used to run an automation logic prior/post to individual Scenario step execution.

Hook Execution Sequence

The hooks of similar type, for example two AfterScenario hooks, are run in a random sequence. To make execution in a specific sequence, we have to add the **Order** property in the hook attribute.

Example

```
[AfterScenario(Order = 1)]
public void CloseBrowser()
{
    // we require this method to execute first...
}

[AfterScenario(Order = 2)]
```

```
public void VerifySessionIdAfterBrowserClose()
{
    // ...so we require this method to execute after the CloseBrowser //method
    is run
}
```

The number signifies order which means that the hook with the lowest number is run first. If the number is omitted, the default value is 10000. It is not a good practise to depend on it and rather mention the order for individual hooks.

Also, if an unhandled exception is thrown, all the following hooks of similar type will be skipped. In order to prevent that, we should handle all the exceptions.

18. SpecFlow — Background Illustration

Background keyword is applied to replicate the same steps before all Scenarios within a Feature File. We may shift these steps to the backdrop by clubbing them under the Background segment.

It helps to add context to a scenario. It can have more than one Given step. Thus, it shall execute prior to execution of each Scenario, but post any **Before hooks**.

A Background is kept prior to the first **Example** or **Scenario**, at the similar indentation level. In short, it is used for declaring the common steps to all the tests.

```
Feature: Member addition

  Background:
    Given launch URL
    Then enter name and password

  Scenario: Normal user addition
    Given user is on normal user additon screen
    When enters normal user details
    Then user should be added as normal user

  Scenario: Admin user addition
    Given user is on admin user addition screen
    When enters admin user details
    Then user should be added as admin user
```

In the above example, having two Scenarios, the Background steps shall run once before execution of each of these scenarios.

Background Rules

Let us describe some of the rules while applying Background:

- It should be used for defining simple steps unless we are forced to bring the application to a state which requires complicated steps to be carried out. As requested by the stakeholders of the project.
- It should be brief and realistic.
- All the Scenarios should also be short and to the point.

Background Example

Let us see an example where we have used Background steps to be executed before all the tests in the Feature File. For instance, to add a normal and admin user for an application, we require the below steps to be run before the execution of the Scenario Normal user addition:

- Launch the application URL.
- Submit the username and password.

Step 1: Create a Feature File

The details of how to create a Feature File is discussed in detail in the Chapter – Feature File.

```
Feature: Member addition

Background:
    Given launch URL
    Then enter name and password

Scenario: Normal user addition
    Given user is on normal user additon screen
    When enters normal user details
    Then user should be added as normal user

Scenario: Admin user addition
    Given user is on admin user addition screen
    When enters admin user details
    Then user should be added as admin user
```

Step 2: Create a Step Definition File

The details of how to create a Step Definition File is discussed in detail in the Chapter – Step Definition File.

```
using System;
using TechTalk.SpecFlow;
namespace SpecFlowProject1.Features
{
    [Binding]
    public class MemberAdditionSteps
    {
        [Given(@"launch URL")]
        public void GivenLaunchURL()
```

```

{
    Console.WriteLine("Url launched");
}
[Given(@"user is on normal user additon screen")]
public void GivenUserIsOnNormalUserAdditonScreen()
{
    Console.WriteLine("User is on normal user addition screen");
}
[Given(@"user is on admin user addition screen")]
public void GivenUserIsOnAdminUserAdditionScreen()
{
    Console.WriteLine("User is on admin user addition screen");
}
[When(@"enters normal user details")]
public void WhenEntersNormalUserDetails()
{
    Console.WriteLine("User enters normal user details");
}
[When(@"enters admin user details")]
public void WhenEntersAdminUserDetails()
{
    Console.WriteLine("User enters admin user details");
}
[Then(@"enter name and password")]
public void ThenEnterNameAndPassword()
{
    Console.WriteLine("User enters name and password");
}
[Then(@"user should be added as normal user")]
public void ThenUserShouldBeAddedAsNormalUser()
{
    Console.WriteLine("User should be added as normal user");
}
[Then(@"user should be added as admin user")]
public void ThenUserShouldBeAddedAsAdminUser()
{
    Console.WriteLine("User should be added as admin user");
}

```



```

    }
}
}

```

Step 3: Execution & Results

Select **SpecFlowProject(2)**, then click on **Run All Tests in View**.

Test	Duration	Traits
SpecFlowProject1 (2)	143 ms	
SpecFlowProject1 (2)	143 ms	
Member addition (2)	143 ms	
Admin user addition	16 ms	
Normal user addition	127 ms	

Group Summary

SpecFlowProject1

Tests in group: 2

⌚ Total Duration: 143 ms

Outcomes

✓ 2 Passed

Select **Normal user addition Scenario**, then click on **Open additional output for this result** link.

Member addition (2)	143 ms
Admin user addition	16 ms
Normal user addition	127 ms

⌚ Duration: 127 ms

[Open additional output for this result](#)

Test Name: Normal user addition

Test Outcome:  Passed

Standard Output

```
-> -> Loading plugin C:\Users\... \Desktop\DebomitaJava\ClassLibrary1\Sp
-> -> Loading plugin C:\Users\... \Desktop\DebomitaJava\ClassLibrary1\Sp
-> -> Using default config
```

Given launch URL

```
-> Url launched
-> done: MemberAdditionSteps.GivenLaunchURL() (0.0s)
```

Then enter name and password

```
-> User enters name and password
-> done: MemberAdditionSteps.ThenEnterNameAndPassword() (0.0s)
```

Given user is on normal user additon screen

```
-> User is on normal user addition screen
-> done: MemberAdditionSteps.GivenUserIsOnNormalUserAdditonScreen() (0.0s)
```

When enters normal user details

```
-> User enters normal user details
-> done: MemberAdditionSteps.WhenEntersNormalUserDetails() (0.0s)
```

Then user should be added as normal user

```
-> User should be added as normal user
-> done: MemberAdditionSteps.ThenUserShouldBeAddedAsNormalUser() (0.0s)
```

In the above output, the Background steps - **Given Url launched** and **Then enter name and password** got executed prior to the actual normal user Scenario.

Select **Admin user addition** Feature, then click on **Open additional output for this result** link.

Test Name: Admin user addition

Test Outcome:  Passed

Standard Output

Given launch URL

-> Url launched

-> done: MemberAdditionSteps.GivenLaunchURL() (0.0s)

Then enter name and password

-> User enters name and password

-> done: MemberAdditionSteps.ThenEnterNameAndPassword() (0.0s)

Given user is on admin user addition screen

-> User is on admin user addition screen

-> done: MemberAdditionSteps.GivenUserIsOnAdminUserAdditionScreen() (0.0s)

When enters admin user details

-> User enters admin user details

-> done: MemberAdditionSteps.WhenEntersAdminUserDetails() (0.0s)

Then user should be added as admin user

-> User should be added as admin user

-> done: MemberAdditionSteps.ThenUserShouldBeAddedAsAdminUser() (0.0s)

In the above output as well, the Background steps - **Given Url launched** and **Then enter name and password** got executed prior to the actual admin user Scenario.

19. SpecFlow — Data Driven Testing with Examples

We can perform data driven testing with the help of keyword **Examples**. We shall also take the help of keyword **Scenario Outline** to execute the same Scenario over multiple values.

The data sets to be taken into consideration shall be passed below the Examples section one after another separated by | symbol. So, if there are three rows, we shall have three test cases executed from a Single scenario.

Scenario Outline is used to replicate the same Scenario with a different data set. Writing the same tests with different values is cumbersome and time taking. For instance,

```
Feature: User credential

Scenario: Normal User Login
    Given user enters user1 as user and pwd1 as password
    Then user should be logged in as normal

Scenario: Admin user login
    Given user enters user2 as user and pwd2 as password
    Then user should be logged in as admin
```

We can club the above two scenarios with the **Scenario Outline**.

```
Feature: User credential

Scenario Outline: Login module
    Given user types <username> and <password>
    Then user should be able to login

Examples:
    | username | password |
    | user1    | pwd1     |
    | user2    | pwd2     |
```

Thus, we see that a Scenario Outline should be accompanied with keyword **Examples**. A Scenario Outline is executed once for each of the rows appearing below the Examples segment.

Also, we have seen that the Given step has the <> delimiter. It points to the header of the Examples table. SpecFlow shall put the values within this table prior to the task of matching a step with a Step Definition.

To verify a Login module, we require the below steps to be executed:

- User types the username and password.
- Verify users should be able to login.

We shall incorporate the above steps to the Feature File.

Step 1: Create a Feature File

The details of how to create a Feature File is discussed in detail in the Chapter – Feature File.

```
Feature: User credential
Scenario Outline: Login module
    Given user types <username> and <password>
    Then user should be able to login

Examples:
    | username          | password |
    | tutorialspoint1 | pwd      |
    | tutorialspoint2 | pwd1     |
```

Step 2: Step Definition File

The details of how to create a Step Definition File is discussed in detail in the Chapter – Step Definition File.

```
using System;
using TechTalk.SpecFlow;
namespace SpecFlowProject1.Features
{
    [Binding]
    public class UserCredentialSteps
    {
        //regular expression used to point to data
        [Given(@"user types (.*) and (.*)")]
        public void GivenUserTypesUserAndPwds(string username, string password)
        {
            Console.WriteLine(username);
            Console.WriteLine(password);
        }
        [Then(@"user should be able to login")]
        public void ThenUserShouldBeAbleToLogin()
```

```

    {
        Console.WriteLine("User should be able to login");
    }
}
}

```

Step 3: Execution & Results

Select User credential(2), then click on Run All Tests in View.

Test	Duration	Traits
SpecFlowProject1 (4)	188 ms	
SpecFlowProject1 (4)	188 ms	
Member addition (2)	169 ms	
User credential (2)	19 ms	
Login module, tutorialspoint1	12 ms	
Login module, tutorialspoint2	7 ms	

Test Detail Summary

- Login module, tutorialspoint1 in User credential
- Source: [SpecFlowFeature2.feature](#) line 9
- Duration: 12 ms
- [Open additional output for this result](#)

Select Login module, tutorialspoint1 Scenario, then click on Open additional output for this result link.

User credential (2)	19 ms
Login module, tutorialspoint1	12 ms
Login module, tutorialspoint2	7 ms

[Open additional output for this result](#)

Test Name: Login module, tutorialspoint1

Test Outcome: Passed

Standard Output

```

Given user types tutorialspoint1 and pwd
-> tutorialspoint1
-> pwd
-> done: UserCredentialSteps.GivenUserTypesUserAndPwds("tutorialspoint1", "pwd") (0.0s)

Then user should be able to login
-> User should be able to login
-> done: UserCredentialSteps.ThenUserShouldBeAbleToLogin() (0.0s)

```

The Scenario got executed with **username – tutorialspoint1** and **password – pwd** as specified in Examples(1st row).

Select Login module, tutorialspoint2 scenario, then click on Open additional output for this result link.

✓ User credential (2)	19 ms	Open additional output for this result
✓ Login module, tutorialspoint1	12 ms	
✓ Login module, tutorialspoint2	7 ms	

Test Name: Login module, tutorialspoint2

Test Outcome: ✓ Passed

Standard Output

Given user types tutorialspoint2 and pwd1

-> tutorialspoint2

-> pwd1

-> done: UserCredentialSteps.GivenUserTypesUserAndPwds("tutorialspoint2", "pwd1") (0.0s)

Then user should be able to login

-> User should be able to login

-> done: UserCredentialSteps.ThenUserShouldBeAbleToLogin() (0.0s)

The test got executed with **username – tutorialspoint2** and **password – pwd1** as specified in Examples(2nd row).

20. SpecFlow — Data Driven Testing without Examples

We can perform data driven testing without the help of keyword **Examples**. This can be done by passing the data directly to the steps within the Feature File enclosed in (""). It will then be provided as an input to the Step Definition File.

Let us verify a module, for which the below steps need to be executed:

- User launches URL
- URL should open

Step 1: Create a Feature File

The details of how to create a Feature File is discussed in detail in the Chapter – Feature File.

Feature: Launching application

Scenario: Launch URL

Given User hits URL 'https://www.tutorialspoint.com/index.htm'

Then URL should be launched

Step 2: Create a Step Definition File

The details of how to create a Step Definition File is discussed in detail in the Chapter – Step Definition File.

```
using System;
using TechTalk.SpecFlow;
namespace SpecFlowProject1.Features
{
    [Binding]
    public class LaunchingApplicationSteps
    {
        [Given(@"User hits URL '(.*)'")]
        public void GivenUserHitsURL(string url)
        {
            Console.WriteLine(url);
        }
        [Then(@"URL should be launched")]
        public void ThenURLShouldBeLaunched()
```



```

    {
        Console.WriteLine("URL should be launched");
    }
}
}

```

Step 3: Execution & Results

Select Launching Application Feature, then click on Run All Tests in View.

The screenshot shows the SpecFlow Test Explorer interface. The test results table is as follows:

Test	Duration	Traits
SpecFlowProject1 (2)	263 ms	
SpecFlowProject1 (2)	263 ms	
Calculator (1)	239 ms	
Launching application (1)	24 ms	

The 'Group Summary' panel on the right indicates:

- Launching application
- Tests in group: 1
- Total Duration: 24 ms
- Outcomes: 1 Passed

Select Launch URL Scenario, then click on Open additional output for this result link.

The screenshot shows the 'Launch URL' test result selected. The 'Open additional output for this result' link is highlighted in the right-hand panel.

The screenshot shows the 'Launch URL' test results. The 'Test Outcome' is 'Passed'. The 'Standard Output' section shows the following steps:

```

Given User hits URL 'https://www.tutorialspoint.com/index.htm'
-> https://www.tutorialspoint.com/index.htm
-> done: LaunchingApplicationSteps.GivenUserHitsURL("https://www.tutor...") (0.0s)

Then URL should be launched
-> URL should be launched
-> done: LaunchingApplicationSteps.ThenURLShouldBeLaunched() (0.0s)

```

In the above output, the url (<https://www.tutorialspoint.com/index.htm>) is obtained which is passed directly from the Feature File within the Given step.

21. SpecFlow — Table conversion to Data Table

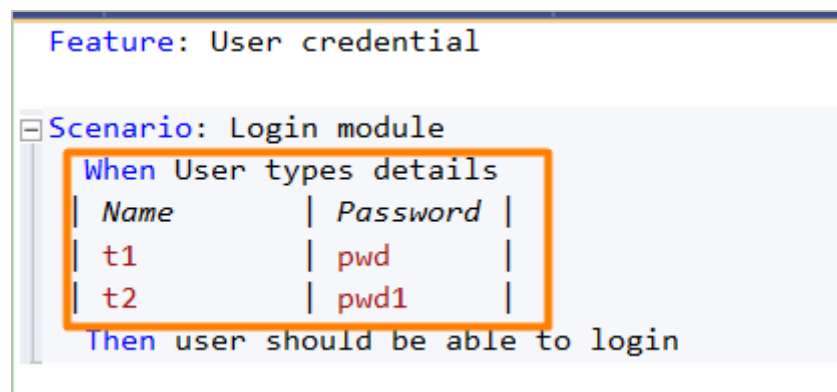
A **Table** is often confused with a **Scenario Outline**. While a Scenario Outline is applicable for the complete test, a Table is only for a single step under which it is defined.

However, a programming logic needs to be built to comprehend the data and then it can be incorporated in our test. An **Examples** keyword is used for a Scenario Outline, but no keywords are required for Data Table.

There are multiple methods available in Table in SpecFlow, let us see how to convert a Table to Table via **Table headers**.

Table is used to send a group of values in the form of a list to the Step Definition file. It is useful to deal with large data sets. SpecFlow has a rich API for table manipulation in the Step Definition File.

SpecFlow Assist Helpers packages are used to work on tables. Also, we have to add namespace **TechTalk.SpecFlow.Assist** to our code.



The screenshot shows a SpecFlow feature file with the following structure:

```
Feature: User credential

Scenario: Login module

    When User types details
        | Name      | Password |
        | t1         | pwd      |
        | t2         | pwd1     |
    Then user should be able to login
```

The table within the 'When User types details' step is highlighted with an orange border.

Step 1: Create a Feature File

The details of how to create a Feature File is discussed in detail in the Chapter – Feature File.

```
Feature: User credential
```

```
Scenario: Login module
```

```
    When User types details
```

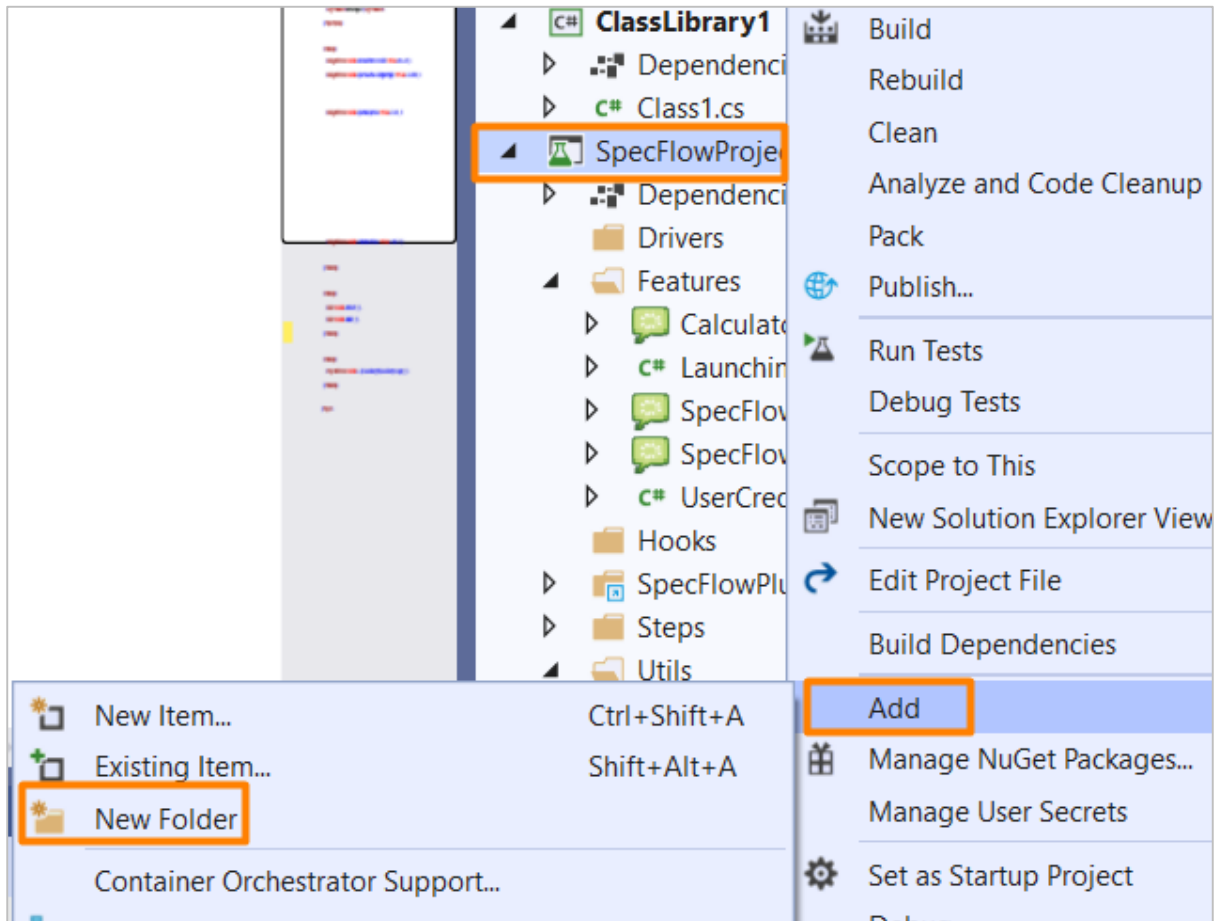
Name	Password
t1	pwd
t2	pwd1

```
    Then user should be able to login.
```

Step 2: Create C# File to access Table Data

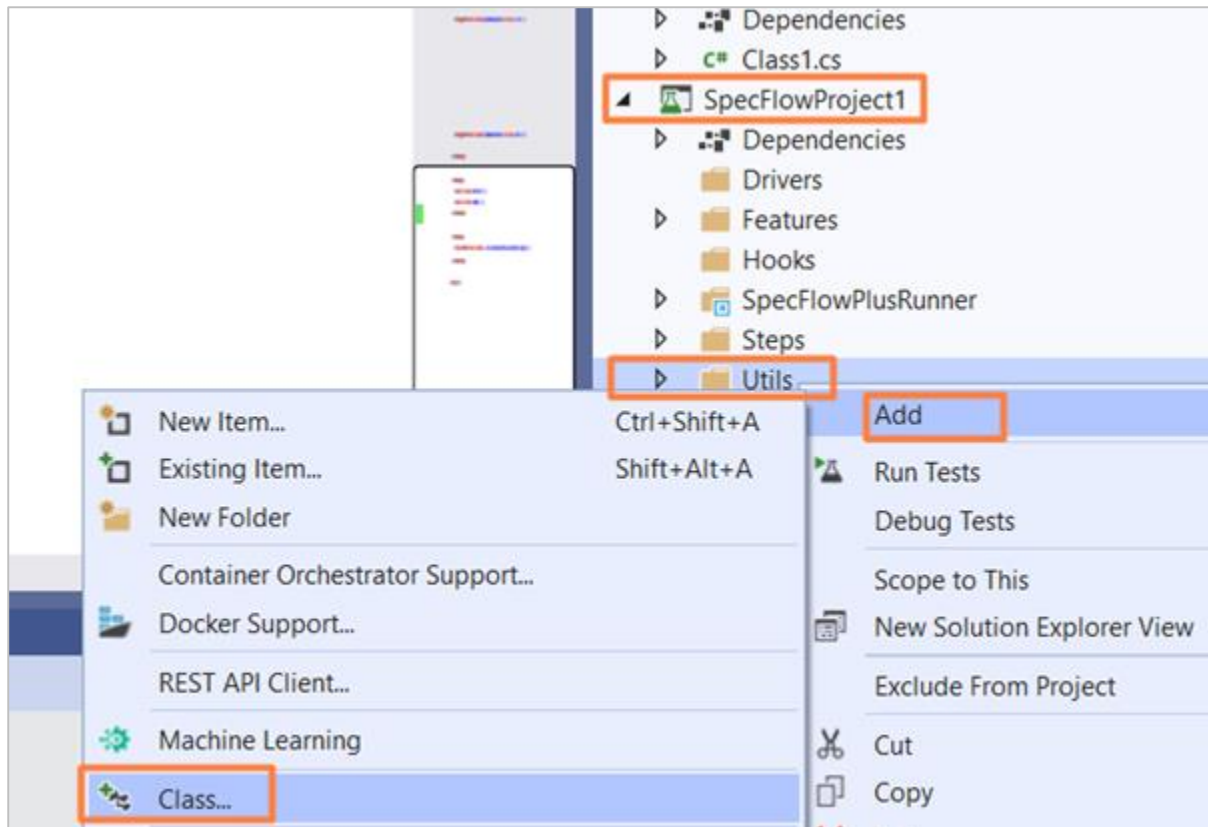
We must convert a Table to a Data Table via **System.Data** package. We shall create a new folder within the project and have a C# file in it. Right-click on the SpecFlow Project, then click on Add.

Select the option **New Folder**.

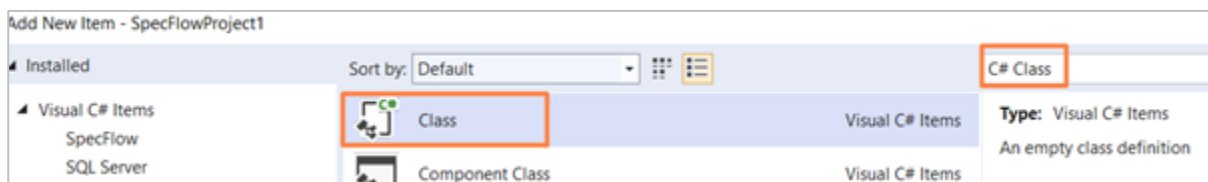


Give the name **Utils** to the folder.

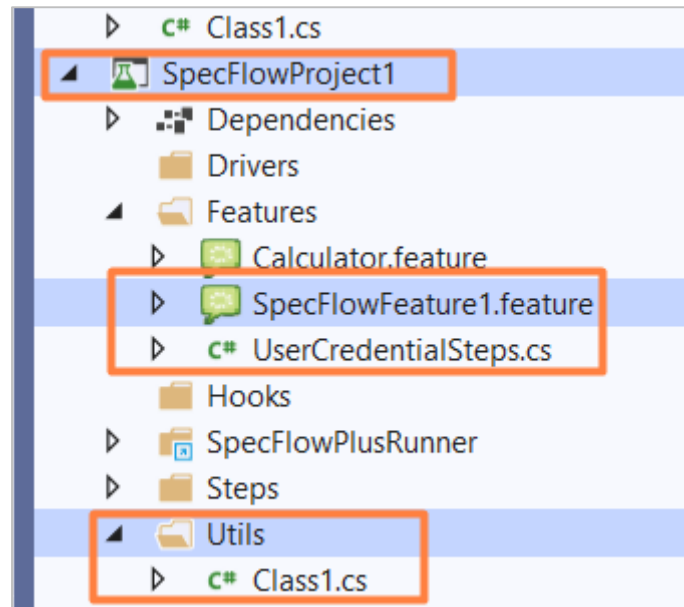
Right-click on the new Folder created, then select the option **Add**. Click on **Class**.



Type **C# Class** in the search box and search. Select the option **Class** from the search result and then click on **Add** to proceed.



Project Folder Structure



C# Class Implementation

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Data;
using TechTalk.SpecFlow;
namespace SpecFlowProject1.Utils
{
    class Class1
    {
        public static DataTable DT(Table t)
        {
            var dT = new DataTable();
            foreach (var h in t.Header)
            {
                dT.Columns.Add(h, typeof(string));
            }
            // iterating rows
            foreach (var row in t.Rows)
            {
                var n = dT.NewRow();
                foreach (var h in t.Header)
                {
```

```

        n.SetField(h, row[h]);
    }
    dT.Rows.Add(n);
}
return dT;
}
}
}

```

Step 3: Create a Step Definition File

The details of how to create a Step Definition File is discussed in detail in the Chapter – Step Definition File.

```

using System;
using System.Data;
using TechTalk.SpecFlow.Assist;
using TechTalk.SpecFlow;
namespace SpecFlowProject1.Features
{
    [Binding]
    public class UserCredentialSteps
    {
        [When(@"User types details")]
        public void WhenUserTypesDetails(Table t)
        {
            //Accessing C# class method from Step Definition
            var dTable = Utils.Class1.DT(t);
            //iterating rows
            foreach (DataRow r in dTable.Rows)
            {
                Console.WriteLine(r.ItemArray[0].ToString());
                Console.WriteLine(r.ItemArray[1].ToString());
            }
        }
        [Then(@"user should be able to login")]
        public void ThenUserShouldBeAbleToLogin()
        {
            Console.WriteLine("User should be able to login");
        }
    }
}

```

```

    }
}
}

```

Step 4: Execution & Results

Select **User credential(1)** Feature, then click on **Run All Tests in View**.

Test Explorer

Run All Tests in View

Test	Duration	Traits
SpecFlowProject1 (3)	295 ms	
SpecFlowProject1 (3)	295 ms	
Calculator (1)	236 ms	
Launching application (1)	27 ms	
User credential (1)	32 ms	

Group Summary

User credential

Tests in group: 1

Total Duration: 32 ms

Outcomes

1 Passed

Select **Login Module Scenario**, then click on **Open additional output for this result** link.

Launching application (1)	27 ms
User credential (1)	32 ms
Login module	32 ms

[Open additional output for this result](#)

Test Name: Login module

Test Outcome: Passed

Standard Output

When User types details

```

--- table step argument ---
| Name | Password |
| t1   | pwd      |
| t2   | pwd1     |
-> t1
-> pwd
-> t2
-> pwd1
-> done: UserCredentialSteps.WhenUserTypesDetails(<table>) (0.0s)

Then user should be able to login
-> User should be able to login
-> done: UserCredentialSteps.ThenUserShouldBeAbleToLogin() (0.0s)

```

The Scenario got executed with data passed from a Table (converted to a Data Table) in the Feature File within the When step.

22. SpecFlow — Table conversion to Dictionary

Tables can hold data in a horizontal and vertical direction in the Feature File. With a **Dictionary** object, we shall see how to access data in the Feature File vertically in a key-value pair.

Step 1: Create a Feature File

The details of how to create a Feature File is discussed in detail in the Chapter – Feature File.

Feature: User credential

Scenario: Login module

When User types details

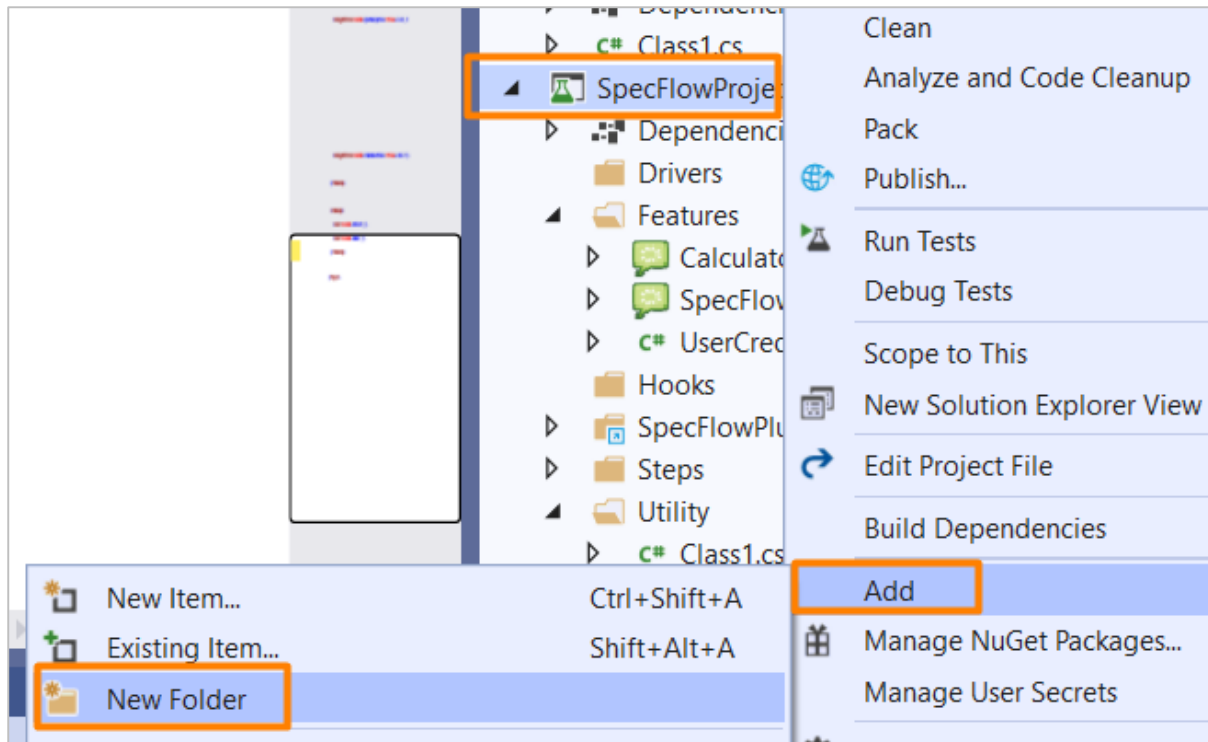
KY		Val			
	username		tutorialspoint		
	password		pwd1		

Then user should be able to login

Step 2: Create C# File to access Table Data

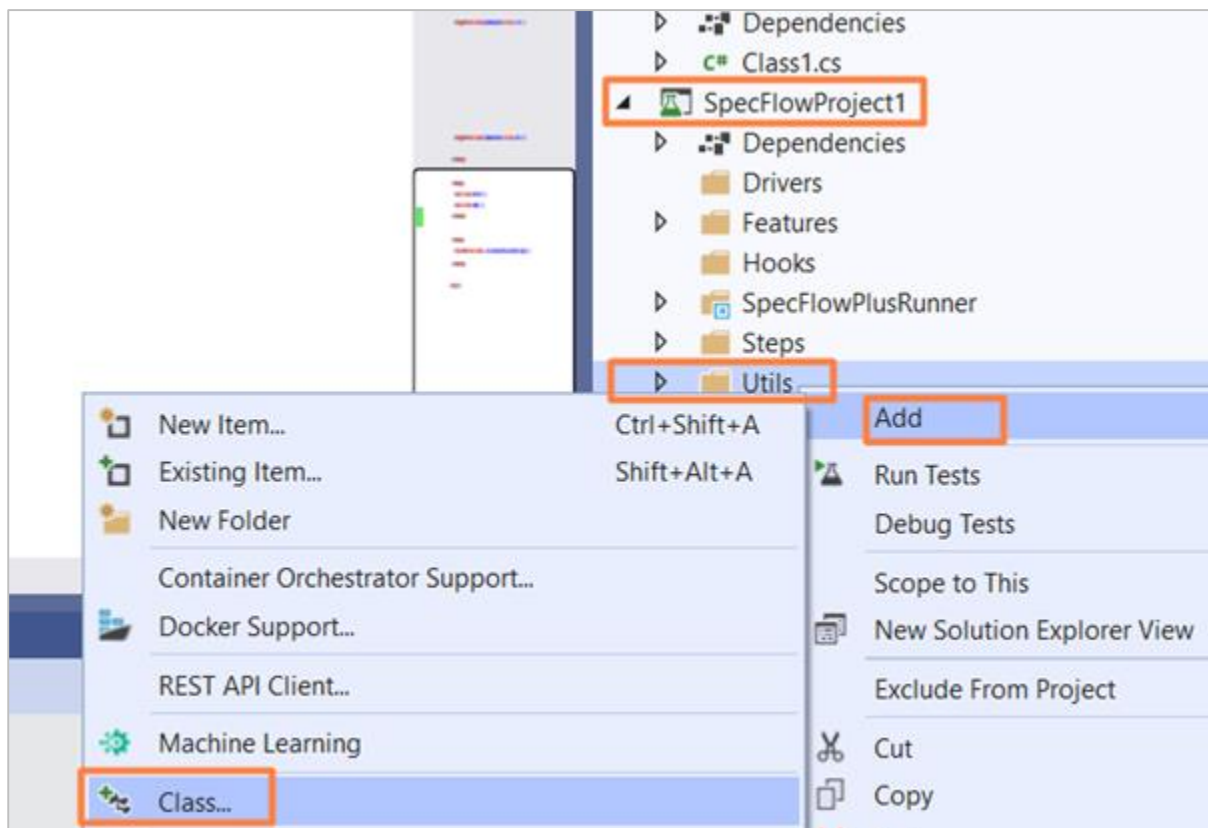
We must convert a Table to a Dictionary via **System.Collections.Generic** package. We shall create a new folder within the project and have a C# file in it. Right-click on the **SpecFlow Project**, then click on **Add**.

Select the option **New Folder**.

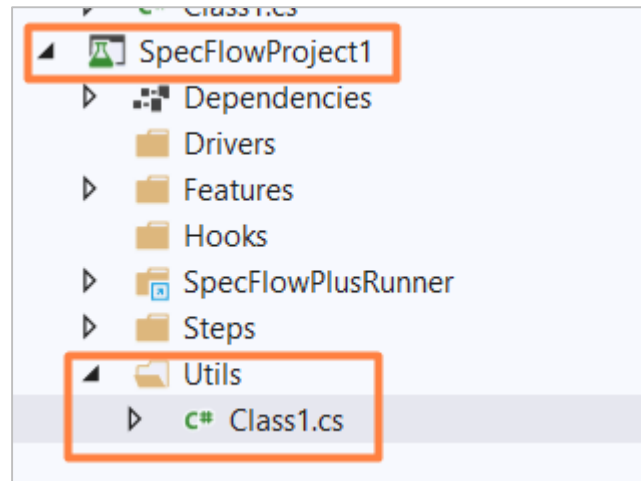


Give the name **Utils** to the folder.

Right-click on the new Folder created, then select the option **Add**. Click on **Class**.



Project Folder Structure



C# Class Implementation

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Data;
using TechTalk.SpecFlow;
namespace SpecFlowProject1.Utils
{
    class Class1
    {
        public static Dictionary<string, string> ToDT(Table t)
        {
            var dT = new Dictionary<string, string>();
            // iterating through rows
            foreach (var r in t.Rows)
            {
                dT.Add(r[0], r[1]);
            }
            return dT;
        }
    }
}
```

Step 3: Create a Step Definition File

The details of how to create a Step Definition File is discussed in detail in the Chapter – Step Definition File.

```

using System;
using TechTalk.SpecFlow;
namespace SpecFlowProject1.Features
{
    [Binding]
    public class UserCredentialSteps
    {
        [When(@"User types details")]
        public void WhenUserTypesDetails(Table t)
        {
            //Accessing C# class method from Step Definition
            var dict = Utils.Class1.ToDT(t);
            Console.WriteLine(dict["username"]);
            Console.WriteLine(dict["password"]);
        }

        [Then(@"user should be able to login")]
        public void ThenUserShouldBeAbleToLogin()
        {
            Console.WriteLine("User should be able to login");
        }
    }
}

```

Step 4: Execution & Results

Select **User credential(1)** Feature, then click on **Run All Tests in View**.

Select **Login Module Scenario**, then click on **Open additional output for this result link**.

Calculator (1)	135 ms	Duration: 32 ms
✓ User credential (1)	32 ms	Open additional output for this result
✓ Login module	32 ms	

Test Name: Login module

Test Outcome:  Passed

Standard Output

When User types details

```

--- table step argument ---
| KY   | Val      |
| username | tutorialspoint |
| password | pwd1        |
-> tutorialspoint
-> pwd1
-> done: UserCredentialSteps.WhenUserTypesDetails(<table>) (0.0s)

```

Then user should be able to login

```

-> User should be able to login
-> done: UserCredentialSteps.ThenUserShouldBeAbleToLogin() (0.0s)

```

The Scenario got executed with data passed from a Table (converted to a Dictionary) in the Feature File within the When step.

23. SpecFlow — Table with CreateInstance

CreateInstance<T> is an extension of the Table method. It transforms the data in the Table to an object. It is one of the popular techniques to have parameterization of data in a **vertical alignment**.

The **SpecFlow Assist Helpers** package is used to work on tables. Also, we have to add namespace **TechTalk.SpecFlow.Assist** to our code.

The Table **headers** in the Feature File can be of any name, for example: KEY, VALUE. However, the first column should point to the name of the property and the second column should point to its corresponding value.

Step 1: Create a Feature File

The details of how to create a Feature File is discussed in detail in the Chapter – Feature File.

Feature: User credential

Scenario: Login module

When User types details

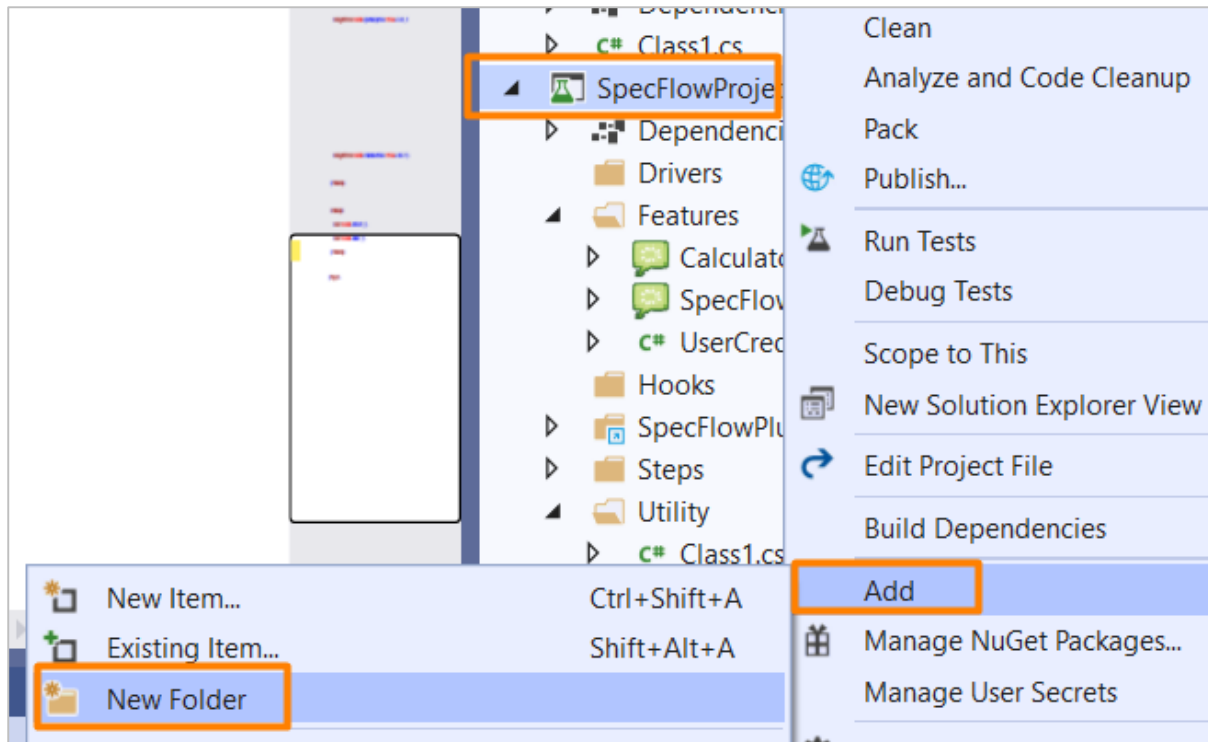
KEY	VALUE
username	tutorialspoint
password	pwd1

Then user should be able to login

Step 2: Create C# File to access String Objects

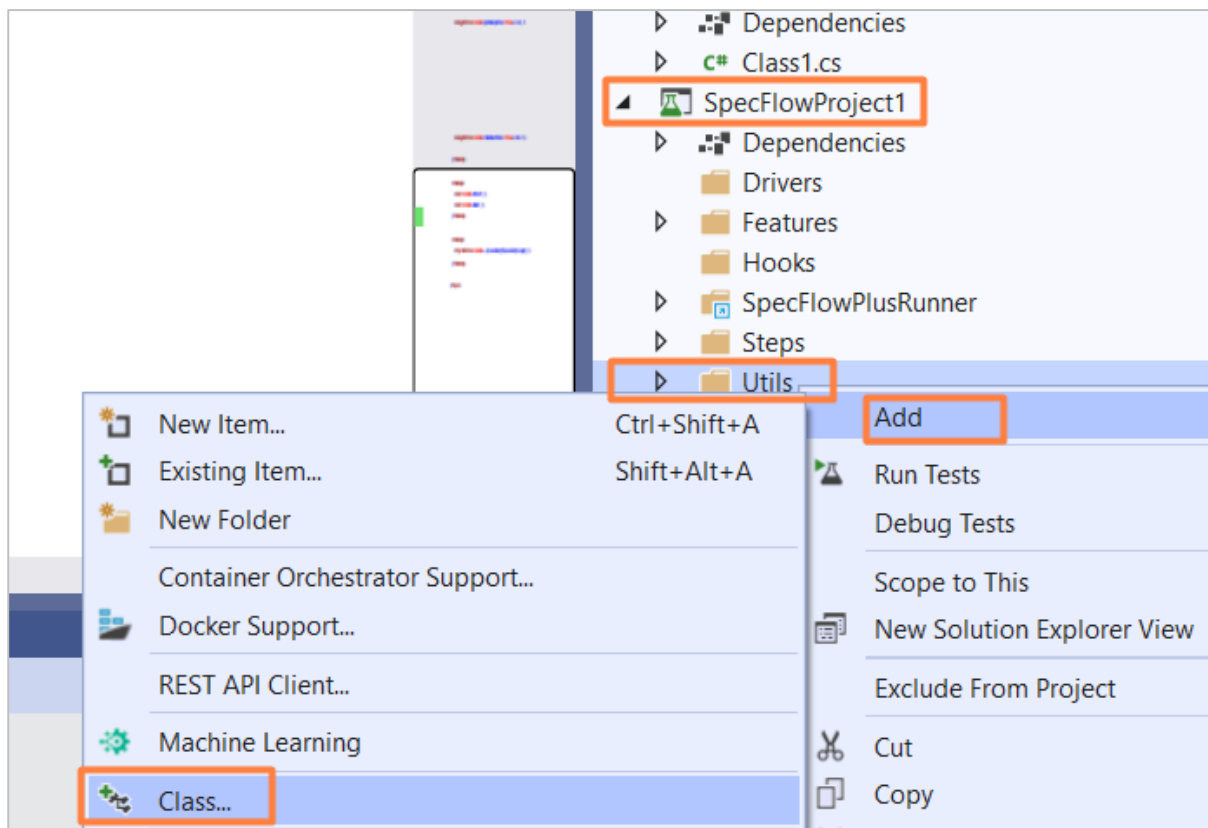
We shall create a new folder within the project and have a C# file in it. Right-click on the **SpecFlow Project**, then click on **Add**.

Select the option **New Folder**.

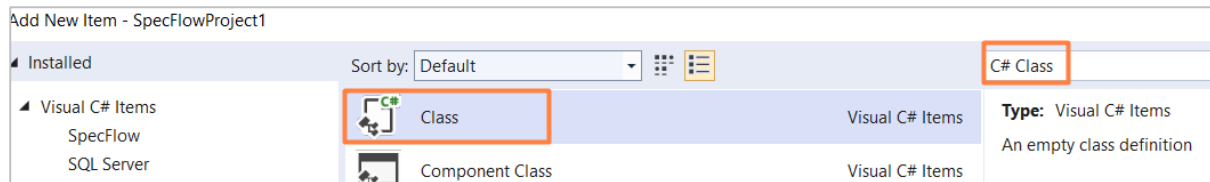


Give the name **Utils** to the folder.

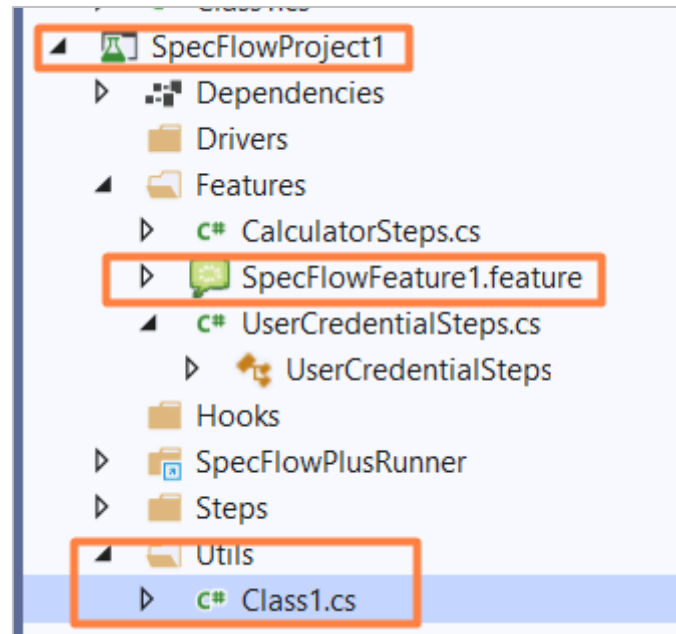
Right-click on the new Folder created, then select the option **Add**. Click on **Class**.



Type **C# Class** in the search box and search. Select the option **Class** from the search result and then click on **Add** to proceed.



Project Folder Structure



C# Class Implementation

```
using System;
using System.Collections.Generic;
using System.Text;
namespace SpecFlowProject1.Utils
{
    class Class1
    {
        public class Input
        {
            //Declaring string objects
            public string Input1 { get; set; }
            public string Input2 { get; set; }
        }
    }
}
```


Step 3: Create a Step Definition File

The details of how to create a Step Definition File is discussed in detail in the Chapter – Step Definition File.

```
using System;
using TechTalk.SpecFlow;
using TechTalk.SpecFlow.Assist;
namespace SpecFlowProject1.Features
{
    [Binding]
    public class UserCredentialSteps
    {
        [When(@"User types details")]
        public void WhenUserTypesDetails(Table t)
        {
            //access data with CreateInstance method using C# class
method
            var i = t.CreateInstance<Utils.Class1.Input>();
            Console.WriteLine(i.Input1);
            Console.WriteLine(i.Input2);
        }
        [Then(@"user should be able to login")]
        public void ThenUserShouldBeAbleToLogin()
        {
            Console.WriteLine("User should be able to login");
        }
    }
}
```

Step 4: Execution & Results

Select **User credential(1)** Feature, then click on **Run All Tests in View**.

Select **Login Module Scenario**, then click on **Open additional output for this result** link.

Test Name: Login module

Test Outcome: Passed

Standard Output

```
-> -> Loading plugin C:\Users\... \Desktop\DebomitaJava\
-> -> Loading plugin C:\Users\... \Desktop\DebomitaJava\
-> -> Using default config
```

```
When User types details
--- table step argument ---
| KEY   | VALUE      |
| username | tutorialspoint |
| password | pwd1       |
-> done: UserCredentialSteps.WhenUserTypesDetails(<table>) (0.0s)
```

```
Then user should be able to login
-> User should be able to login
-> done: UserCredentialSteps.ThenUserShouldBeAbleToLogin() (0.0s)
```

The Scenario got executed with data passed from a Table in the feature file within the When step using CreateInstance method.

24. SpecFlow — Table with CreateSet

CreateSet<T> is an extension of the Table method. It transforms the data in the Table to a group of objects. It is one of the popular techniques to have parameterization of data in a **horizontal** alignment.

We can handle one or many rows of data with this method. The **SpecFlow Assist Helpers** package is used to work on tables. Also, we have to add namespace **TechTalk.SpecFlow.Assist** to our code.

The **CreateSet<T>** method obtains an **IEnumerable<T>** depending on the matched data in the Table. It has values for all the objects. It makes sure to have the correct type conversions from string to a linked property.

Step 1: Create a Feature File

The details of how to create a Feature File is discussed in detail in the Chapter – Feature File.

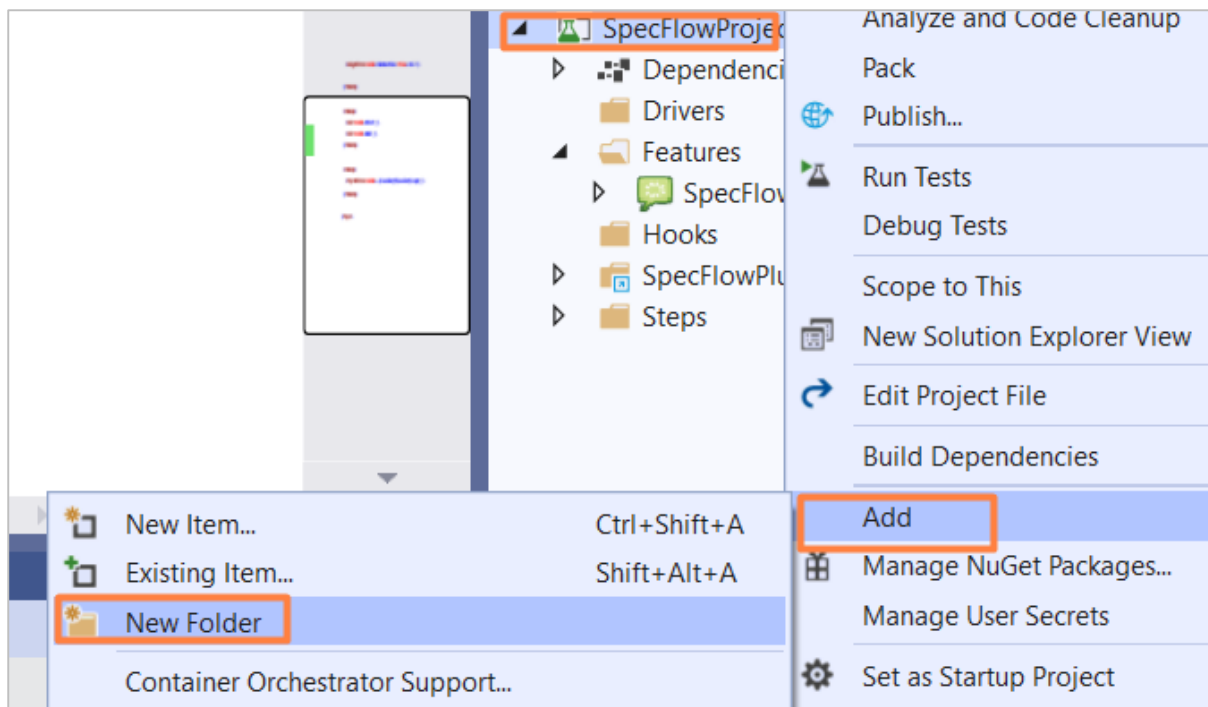
```
Feature: User credential

Scenario: Login module
    When User types details
        | Name      | Password |
        | t1        | pwd      |
        | t2        | pwd1     |
    Then user should be able to login
```

Step 2: Create C# File to access String Objects

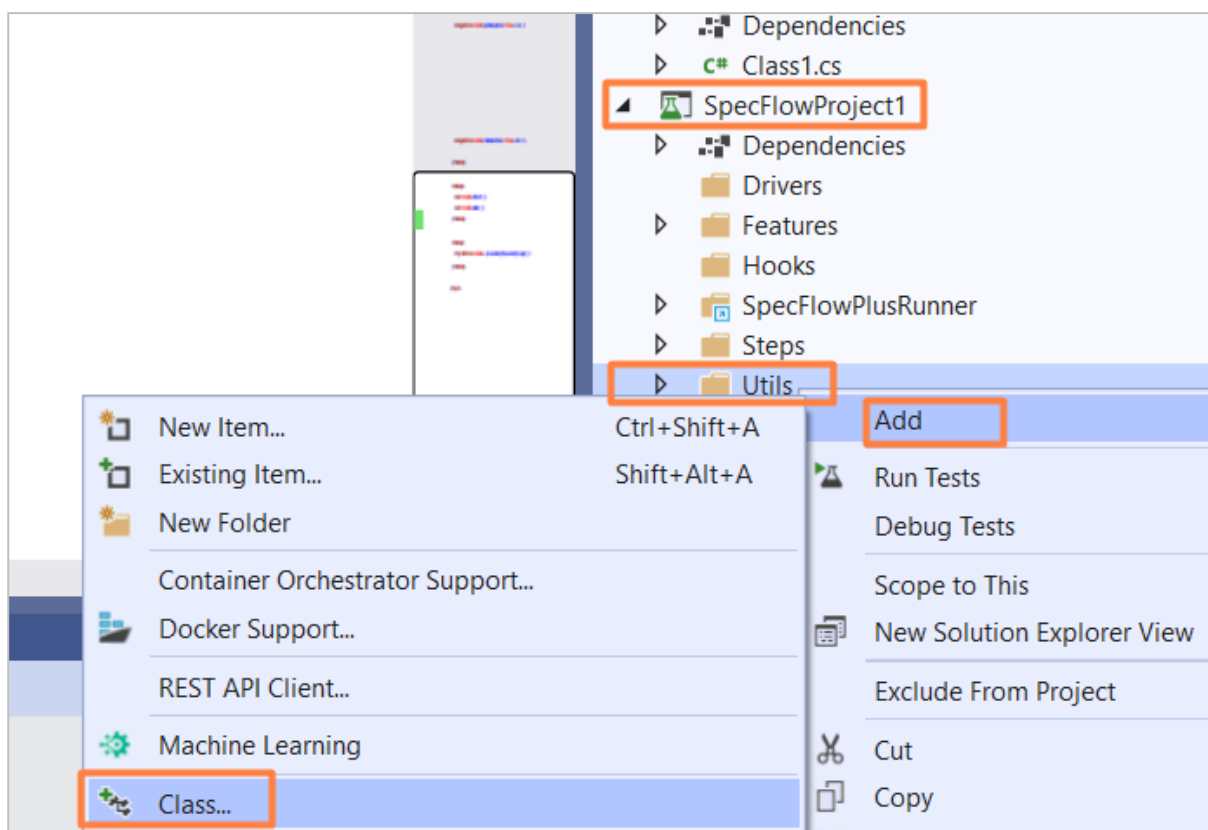
We shall create a new folder within the project and have a C# file in it. Right-click on the **SpecFlow Project**, then click on **Add**.

Select the option **New Folder**.

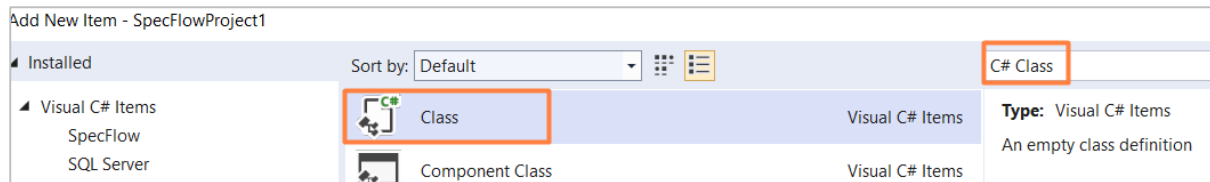


Give the name **Utils** to the folder.

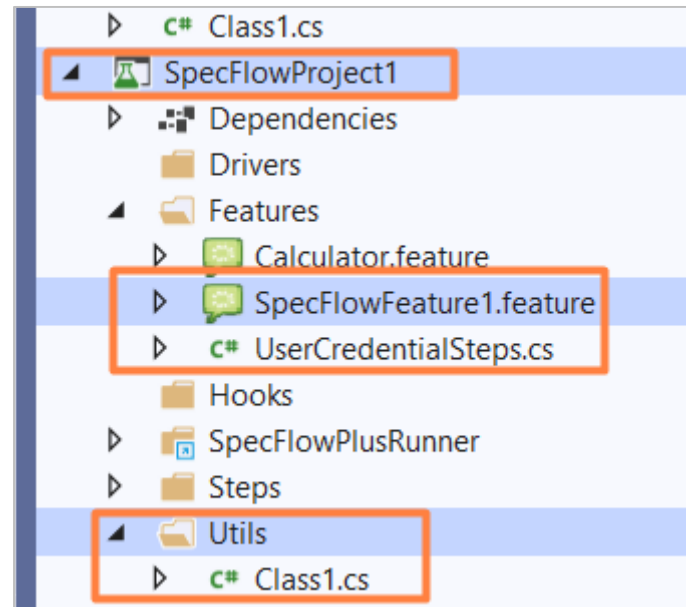
Right-click on the new Folder created, then select the option **Add**. Click on **Class**.



Type **C# Class** in the search box and search. Select the option **Class** from the search result and then click on **Add** to proceed.



Project Folder Structure



C# Class Implementation

```
using System;
using System.Collections.Generic;
using System.Text;
namespace SpecFlowProject1.Utils
{
    class Class1
    {
        public class Input
        {
            //two string objects declared
            public string Input1 { get; set; }
            public string Input2 { get; set; }
        }
    }
}
```

Step 3: Create a Step Definition File

The details of how to create a Step Definition File is discussed in detail in the Chapter – Step Definition File.

```
using System;
using TechTalk.SpecFlow;
using TechTalk.SpecFlow.Assist;
namespace SpecFlowProject1.Features
{
    [Binding]
    public class UserCredentialSteps
    {
        [When(@"User types details")]
        public void WhenUserTypesDetails(Table t)
        {
            //access Table data with CreateSet method
            var i = t.CreateSet<Utils.Class1.Input>();
            //iterate over rows
            foreach (var r in i)
            {
                Console.WriteLine(r.Input1);
                Console.WriteLine(r.Input2);
            }
        }
        [Then(@"user should be able to login")]
        public void ThenUserShouldBeAbleToLogin()
        {
            Console.WriteLine("User should be able to login");
        }
    }
}
```

Step 4: Execution & Results

Select User credential(1) Feature, then click on Run All Tests in View.

Test	Duration	Traits	Group Summary
SpecFlowProject1 (2)	153 ms		User credential
SpecFlowProject1 (2)	153 ms		Tests in group: 1
Calculator (1)	153 ms		Outcomes
User credential (1)			1 Not Run

Select **Login Module Scenario**, then click on **Open additional output for this result** link.

Calculator (1)	152 ms	Duration: 49 ms
User credential (1)	49 ms	Open additional output for this result
Login module	49 ms	

Test Name: Login module

Test Outcome: ✔ Passed

Standard Output

```

When User types details
  --- table step argument ---
  | Name | Password |
  | t1  | pwd    |
  | t2  | pwd1   |
-> done: UserCredentialSteps.WhenUserTypesDetails(<table>) (0.0s)

Then user should be able to login
-> User should be able to login
-> done: UserCredentialSteps.ThenUserShouldBeAbleToLogin() (0.0s)

```

The scenario got executed with data passed from a Table in the feature file within the When step using CreateSet method.