Welcome to the Step-By-Step Getting Started Guide!

Feb 08, 2022

Microsoft Visual Studio

1	Install Visual Studio extension	1
2	Create calculator project	5
3	Create SpecFlow project	11
4	Create SpecFlow project - Continue	17
5	Bind the first step	21
6	Bind remaining steps	27
7	Fix implementation	33
8	Add Living Documentation	37
9	Final	43
10	Install JetBrains Rider Plugin	45
11	Create calculator project	49
12	Create SpecFlow project	55
13	Create SpecFlow project - Continue	59
14	Bind the first step	63
15	Bind remaining steps	69
16	Fix implementation	75
17	Add Living Documentation	79
18	Final	83
19	Exercise	85
20	Exercise-solution	87

21 Additional resources

CHAPTER 1

Install Visual Studio extension

10 minutes

In this step you'll learn how to install the Visual Studio extension for SpecFlow.

SpecFlow's Visual Studio extension not only enables the functionalities needed for testing automation, but is also bundled with several helpful features to make the journey more intuitive.

SpecFlow's Visual Studio extension works on Visual Studio 2017 & 2019. If you are using an older version of Visual Studio, please upgrade to the latest version.

Installation of the extension is simple:

1- Open Visual Studio. *We use Visual Studio 2019 in this guide

2- Navigate to "Extensions Manage Extensions Online" and search for "SpecFlow" in the search bar.

3- Hit Download to begin the installation. You will need to restart Visual Studio for the installation to complete:

Manage Extensions				?	×
▶ Installed	Sort by:	Relevance 🔹	pecflow		× -
 Online Visual Studio Marketplace Search Results 		SpecFlow for Visual Studio 2019 SpecFlow integration for Visual Studio 2019.	Created By: SpecFlow Team Version: 2019.0.95.11469		
 Controls Templates Tools 	۲.	Deveroom for SpecFlow (Visual Studio 2019) Visual Studio extension for working with SpecFlow projects and Gherkin feature files.	Pricing Category: Free Rating: ★★★★ (28 Votes) More Information Report Extension to Microsoft		
 Updates Roaming Extension Manager 	spex	Spex [Preview] SpecFlow (Features) sync to Azure DevOps (TestCases)			
	Ø	Azure DevOps Test Connector A Visual Studio extension allowing users to link Test Classes and SpecFlow Feature files with Azure DevOps Test Plans/Suites and Cases			
		Cholula C# Asp.Net Core 3.1 Solution builder. Containing a WebApi project			
		s	Scheduled For Install: None		
		S	Scheduled For Update: None Scheduled For Uninstall:		
		1	None		
Change your settings for Extensions					
				<u>C</u> lo	se

Once the extension is successfully installed, you can see it in the list of "Installed" extensions in the "Extensions Manage Extensions" dialog of Visual Studio.



In the next steps you'll create a simple application that will be used throughout this guide.

CHAPTER 2

Create calculator project

10 minutes

In this step you'll create the application that will be tested, also called System Under Test (SUT). The application will be a simple calculator in a C# class library.

1- Open Visual Studio and create a new C# class library by selecting "Create a new project" from the Visual Studio startup dialog:



2- Search for "Class library core" and select the "C# Class Library (.NET Core)" project template and click



Next.

3- Enter the project name as "SpecFlowCalculator", choose a location to save the project and hit *Create*. In this scenario the solution will be saved to C:\work.

Configure your new project		
Class Library (.NET Core) C# Linux macOS Windows Library		
Project name		
SpecFlowCalculator		
Location		
C:\work		
SpecFlowCalculator		
Place solution and project in the same directory		
	Back	Create

> *Note: Do NOT use any special characters in your project name e.g. (parenthesis).* This will result in build errors from the code generated by SpecFlow.

*Solution name automatically updates to project name, leave it as is.

4- Rename Class1.cs to Calculator.cs and overwrite the content with the following code :

```
using System;
namespace SpecFlowCalculator
{
    public class Calculator
    {
        public int FirstNumber { get; set; }
        public int SecondNumber { get; set; }
        public int Add()
        {
            throw new NotImplementedException();
        }
    }
}
```

K File Edit View Project Build Debug Test Analyze Tools Extensions Window Help	🔎 Specator – 🗆 🗙
🖹 🎦 🍹 😋 - 〇 稔 - 🏠 🔛 📽 🏷 - 个 - 🛛 Debug - Any CPU ト SpecFlowCal	lculator + 🎜 🖾 🚽 🖄 Live Share 🔗
Calculator.cs * * Calculator.cs * * SpecFlowCalculator * SpecFlowCalculator.Calculator 1 using System; 2 3 namespace SpecFlowCalculator 4 6 7 9 0references 8 9 0references 10 0references 11 0references 12 0 0 13 14 15	Solution Explorer
100 % - ON ho issues found V - Ln: 15 Ch: 2 SPC CRLF	
Output 	
	🛧 Add to Source Control 🔺 🚑 💡

5- Now build the solution by navigating to "Build Build Solution" You will see a "Build : 1 Succeeded" message in the output window:



The calculator application is now built. In the next step you'll learn how to create a SpecFlow project.

chapter $\mathbf{3}$

Create SpecFlow project

5 minutes

In this step you'll create a SpecFlow project and add it to the existing calculator solution:

1- Right-click the solution item "Solution 'SpecFlowCalculator' (1 of 1 project)" under the Solution Explorer and select the "Add New Project..." menu item.

	File Edit	View	Project Bu	uild De	ebug Tes PP ウィ	t Analyze C - Debu	Tools g - Ai	Extensions ny CPU	Window	v Help SpecFlowC	alculator -	Specator	_ لک Live Sł	nare 🕅	
Server Explorer Toolbox	Calculator.cs +> X Calculator.cs +> X Calculator.cs +> X Calculator Calcul								Ě E	Build So Rebuild Clean So Analyze Batch Bi Configu Manage Manage New Sol Add	Solution Search St Jution Solution and Code C uild wration Mana NuGet Pack NuGet Pack	Explorer Carlow Control Contr	• \$ 하[(Ctrl+;) <u>Calculator' (</u> Ct	I of 1 project rl+Shift+B	
	15 100 % • Output ::::::::::::::::::::::::::::::::::	No iss tfrom: Bu Build st pwCalcula == Build:	iild iarted: Pro itor -> C:\ 1 succeed	ject ₩ork work ed, 0 fi	Existing I Existing V New Iten Existing I New Solu Installatio New Edit ailed, Ø	Project Web Site h tem ution Folder on Configurati orConfig up-to-date	on File , 0 skip	Ctrl+Shift+ Shift+Alt+ oped ======	-A A C	 Add Sol Paste Rename Copy Fu Open Fo Save As Hide Un Properti 	ution to Sou III Path Older in File E Solution Filt Iloaded Proje	rce Control ixplorer er ects e of the solution	Ct F2 Alt	rl+V t+Enter	
2 -	Ready Search	n fo	or "Sp	pecFlo	w",	select	the	"Spec	cFlow	Pro	oject"	↑ Add to Sour template	e ce Control	l clic	ck



Next.

3- Enter the project name "SpecFlowCalculator.Specs". Keep the suggested location (the solution folder) and click

Configure your per	wproject		
configure your ne	w project		
SpecFlow Project C# Linux made	COS Windows Test		
Project name			
SpecFlowCalculator.Specs			
Location			
C:\work\SpecFlowCalculator		·	
	6		

Create.

> *Note: Do NOT use any special characters in your project name e.g. (parenthesis).* This will result in build errors from the code generated by SpecFlow.

4- On this next screen you can configure the Test Framework (Runner) you want to use. We are using xUnit in this tutorial, but you may choose a different unit test runner if you a have particular preference. Hit *Create* once you have made your selection.

				_	
Traata a naw Snace	low project				
cleate a new specr	low project				
ramework					
.NET 6.0		•			
st Framework					
xUnit		•			
] Add FluentAssertions library					
			Back	Crea	te

5- Visual Studio will now create the new SpecFlow project and resolve the NuGet packages in the background. You should see the new SpecFlow project in the Solution Explorer as per below:



In the next step you will learn how to add a project reference and how to use the test explorer.

CHAPTER 4

Create SpecFlow project - Continue

5 minutes

You will now add a project reference to the "SpecFlowCalculator" class library in the newly created SpecFlow project. This is necessary because we want to test the "Calculator" class implemented in the class library in the "SpecFlow-Calculator.Specs" project. To do this, follow the below steps:

1- Expand the project node "SpecFlowCalculator.Specs" in the Solution Explorer, right-click the "Dependencies" node and select the "Add Project Reference..." menu item.



2- In the "Reference Manager" dialog check the "SpecFlowCalculator" class library and click OK.

Reference Manager - SpecFlo	wCalculator.Specs					?	×
Projects					Search (Ctrl+E)		۶-
Solution	Name SpecFlowCalculator		Path C\work\SpecFlowCalculator\SpecFlowCalcul	ator\S	Name:		
Shared Projects		2		acor (Sin	specificwcalculator		
▶ COM		-0					
▶ Browse							
							_
				Browse	ОК	Cance	el

Now the solution is set up with a class library containing the implementation of the calculator and a SpecFlow project that contains the specification and tests of the calculator.

3- Now build the solution. You should see the "Build : 1 succeeded" message in the output window. **Refer to page 2 step 6 of this guide if you cannot recall how to build the solution.*

4- Open the test explorer dialog from the menu "Tests Test Explorer".



5- You should see a test already added to the SpecFlow project by the project template. Run the test using the "Run All Tests in View" icon. Note that the outcome/status of the test remains "Not Run" as the test has not executed yet.

Test Explorer		
🕨 🕨 - 🧭 🔯 🖾 1 🖉 0 😒 0 🕒 1 📓 - [[= 🗃 4	🗘 - Search Te	st Explorer 🔎 -
T Run All Tests In View (Ctrl+R, V) Duration Trait	s Error Message	Group Summary
Image: SpecFlowCalculator.Specs (1)		SpecFlowCalculator.
SpecFlowCalculator.Specs (1)		Tests in group: 1
▲ () Calculator (1)		Outcomes
Add two numbers in Calculator myta	g	🌗 1 Not Run

In the next step you will learn how to automate your first scenario.

CHAPTER 5

Bind the first step

10 minutes

In this step you'll bind your first step (automate your first scenario step with SpecFlow).

*If you the page make sure skipped previous you execute the tests with your would preferred runner. The test explorer look like below (see the duration in milliseconds), but it does not do much yet and shows the "Skipped" sta-

1 - 0	🎽 🙆 🖾 1 🕑 0	😢 0 🔔 1 🛛 📓 👻 [[=	a 🌣 -		Search Test Exp	olorer	p-
Test		Duration	Traits	Error Message	e	Group Sum	mary
▶ 1 SpecFl	owCalculator.Specs (1) 167 ms				SpecFlowC Tests i © Tota Outcomes 1 Sk	alculator. n group: 1 al Duration
Open ution	the Ca Explorer	lculator.featu (SpecFlowCalculat	re fi	le by	double-clicking Features	it in Calculator f	the



The purpose of this feature file is to document the expected behavior of the calculator in a way that it is both humanreadable and suitable for test automation. SpecFlow uses the Gherkin language where you can phrase the scenarios using *Given/When/Then* steps. Currently there is a single scenario (automatically added by the SpecFlow project template) that describes how adding two numbers should work with the calculator.

Here is a closer look at the Gherkin scenario used in this template:

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

Based on the scenario text, SpecFlow generates an automated test that executes the scenario. However, it is **not yet defined** what the steps of the scenario should actually "**do**".

2- Right-click the first Given step "Given the first number is 50" and select either the "Go To Definition" or the "Go To

Server Expl	He Edit View Git Project Build Debug Test Anal 는 1월 7월 및 1월 7월 7월 ₋ 월 O - O 월 - C 월 월 월 9 Calculator.feature -> × 1 Feature: Calculator 2 ![Calculator](<u>https://specflow.org/wp-co</u>	yze lools Extensions (· · ← · Debug · Any pontent/uploads/2020/0;	Window Help Search (C CPU → SpecFlow(→ @ 9/calculator.png)	Irl,⊅ Specator — Calculator → ♬ ; B Live St Solution Explorer ○ ○ ☆ ♬ ⊙ → ♀ ♬ 『
orer Toolbox	<pre>3 In order to avoid silly mistakes 4 As a math novice 5 I *want* to be told the **sum** of * 6 7 Link to a feature: [Calculator](SpecFlow 8 ***Further read***: **[Learn more about 9 10 @mytag 11 = Scenario: Add two numbers 12 Given the first number is 50</pre>	***two*** numbers vCalculator.Specs/Feat how to generate Livin	tures/Calculator.fe ng Documentation](<u>h</u>	Search Solution 'SpecFlowCalculator Solution 'SpecFlowCalculator Calculator.cs Calculator.cs Calculator.cs Calculator.cs Calculator.specs Calculator.
	And the second number is 70 When the two numbers are added Then the result should be 120	Image: Constraint of the second se	Ctrl+R, Ctrl+R F12 Ctrl+F10 Ctrl+X Ctrl+C Ctrl+V	Hooks Hooks SpecFlowPlusRunner Steps
nd 🗖	110% Voissues found Ready	▶ Ln: 12 Ch:	30 Col: 33 TABS CRLF	↑ Add to Source Control →

Step Definition" command.

Visual Studio locates the step definition (binding) that belongs to this step. In this example, it opens the CalculatorStepDefinitions class and jumps to the GivenTheFirstNumberIs method.



*The step definition is located based on the [Binding] attribute on the class and the [Given] attribute on the method. The regular expression of the Given attribute matches the text of the scenario step.

3- Add the below field to the class to instantiate the calculator that we want to test and created in *Step 2* of this guide (SUT).

```
private readonly Calculator _calculator = new Calculator();
```

4- Replace the implementation of the first step definition method with the below code which sets the first number of the calculator.

```
[Given("the first number is (.*)")]
public void GivenTheFirstNumberIs(int number)
{
    __calculator.FirstNumber = number;
}
```

Welcome to the Step-By-Step Getting Started Guide!

🛛 🔀 File	e Edit	View	Project	Build	Debug	Test	Analyze	Tools	Extensions	Window	Help	Searc	h (Ctrl+Q)	P	Specat	or —		\times
) 🖕 🖷	12 2		위 위 위	, 8 ©	- O t	3 - 🖆	e e	୬ - ୧	- Debug	+ Any (CPU	* -				🖻 Liv	ve Share	ጽ
S Calc	CalculatorStepDefinitions.cs + × SpecFlowCalculator.Specs.csproj Calculator.cs Calculator.feature 🛎 X 🕶 🕸													× - 🌣				
🚊 🖾 s	🖞 🖾 SpecFlowCalculator.Specs 🔹 🔹 🔩 SpecFlowCalculator.Specs.Steps.CalculatorStepDefiniti 🔹 🗣 _calculator 🔹 👘														- 1			
xplo	8		<pre>// For additional details on SpecFlow step definitions see <u>https://go.specflow.org/doc-stepdef</u></pre>															
rer	9																	
l d'	10		private readonly ScenarioContext _scenarioContext;															
lbox	12 2	n F	private readenly (algulater - new (algulater())															
	12 🖉		private readonly calculator _calculator = new calculator();															
	13 0 references													- 11				
	14 Depublic CalculatorStepDefinitions(ScenarioContext scenarioContext)														- 11			
	15		{		C													1.16
	10		, _ ⁵⁰	cenar10	contex	t = s(cenar10	Lontex	τ;									- 11
	18	- F	1							_								
	19		[Given	("the f	First n	umber	is (.*)")]										
			0 referenc	es														
	20	Ē	public	void (iventh	eFirst	tNumber.	ls(int	number)									
	21		1	alculat	tor.Fir	stNum	her = n	mber:										
	23		}					,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,										
	24	-	-															
	25		[Given	("the s	second i	numbei	r is (.	*)")]										
	26		0 reference	es woid 6	ivonTh		ndNumbou		t number)									
	20	Ī	{	VOIU (irvenin	eseco	nunumber	13(11)	(indiaber)									
	28		1/	FODO: i	impleme	nt arı	range (precon	dition) l	ogic								
	29		11	For st	toring	and re	etrievi	ng scei	nario-spe	cific d	ata see	e <u>http</u>	s://go.s	pecflo	w.org/do	<u>c-shari</u>	ngdata	
	30		- 11	To use	e the m	ultil:	ine text	t or tl	he table	argumen	t of th	ne sce	nario,					
	31		11	additi	ional s [.]	tring,	/Table	parame	ters can	be defi	ned on	the s	tep defi	inition	l.			-
115 9	6 +	No	issues foun	d											► Ln:	12 Ch: 68	SPC	CRLF
C Ready	,														↑ Add to S	ource Contr	ol 🔺 🚽	

5- Execute the test in the Test Explorer and click "Open additional output for this result" from the right



In the detailed output you can see that the first step "Given the first number is 50" has been matched to the step definition method "CalculatorStepDefinitions.GivenTheFirstNumberIs" as expected, and it has been called with the argument 50. The done status means that the step executed successfully with no errors:

Given the first number is 50 -> done: CalculatorStepDefinitions.GivenTheFirstNumberls(50) (0.0s)

The next step in the scenario is pending, as we have not yet implemented it:

And the second number is 70 -> pending: CalculatorStepDefinitions.GivenTheSecondNumberls(70)

In the next step you will bind the rest of the scenario steps.

CHAPTER 6

Bind remaining steps

10 minutes

In this step you'll bind the remaining steps of the scenario.

1- Similar to the previous page, navigate to the bindings from the feature file by right-clicking the second Given step "And the second number is 70" and select either the "Go To Definition" or the "Go To Step Definition" command. Alternatively you can open the CalculatorStepDefinitions.cs directly.

2- Implement the binding of the second step "And the second number is 70" by replacing the code of the GivenTheSecondNumberIs method with the below:

```
[Given("the second number is (.*)")]
public void GivenTheSecondNumberIs(int number)
{
    __calculator.SecondNumber = number;
}
```



*We use the "And" keyword in the Gherkin scenario for better readability. The "And" keyword will be interpreted as "Given", "When" or "Then" depending on the previous step(s) in the scenario. In this example the "And the second number is 70" is interpreted as a "Given" step because the previous step is a "Given" step.

3- Next, implement the binding of the third step, "When the two numbers are added", by replacing the code of the WhenTheTwoNumbersAreAdded method with the below. The method must have a When attribute, as it belongs to the "When" step in the scenario.

```
private int _result;
[When("the two numbers are added")]
public void WhenTheTwoNumbersAreAdded()
{
    _result = _calculator.Add();
}
```



This implementation calls the Add method of the calculator. Note that the result of the addition is not stored by the calculator in a property/field but it is returned to the caller. It's a good idea to store the returned value in a field so that we can work with the result afterwards.

4- Implement the binding of the last step, "Then the result should be 120", by replacing the code of the ThenTheResultShouldBe method. The method must have a Then attribute, as it belongs to a "Then" step in the scenario.

Add a namespace using for FluentAssertions at the top of the file:

```
using FluentAssertions;
```

Use the below code for implementation of the "Then" step which validates if the result of the addition matches the expected value (using the FluentAssertions library).

```
[Then("the result should be (.*)")]
public void ThenTheResultShouldBe(int result)
{
    _result.Should().Be(result);
}
```



After implementing all step definitions and cleaning up the file you should have the following code:

```
using FluentAssertions;
using TechTalk.SpecFlow;
namespace SpecFlowCalculator.Specs.Steps
{
    [Binding]
   public sealed class CalculatorStepDefinitions
    {
        // For additional details on SpecFlow step definitions see https://go.
→ specflow.org/doc-stepdef
        private readonly ScenarioContext _scenarioContext;
        private readonly Calculator _calculator = new Calculator();
        private int _result;
        public CalculatorStepDefinitions(ScenarioContext scenarioContext)
        {
            _scenarioContext = scenarioContext;
        }
        [Given("the first number is (.*)")]
        public void GivenTheFirstNumberIs(int number)
        {
            _calculator.FirstNumber = number;
        }
```

(continues on next page)

(continued from previous page)

```
[Given("the second number is (.*)")]
       public void GivenTheSecondNumberIs(int number)
        {
            _calculator.SecondNumber = number;
        }
        [When("the two numbers are added")]
       public void WhenTheTwoNumbersAreAdded()
        {
            _result = _calculator.Add();
        }
        [Then("the result should be (.*)")]
       public void ThenTheResultShouldBe(int result)
        {
            _result.Should().Be(result);
       }
   }
}
```

5- Build the solution. The build should succeed.

6- Run the test again.

The test should execute and fail, this is expected. In the Test Detail Summary pane of Test Explorer you can see that a NotImplementedException has been thrown in the Add method of the

🕩 🕨 - 🕑 🔞 🖾 1 💟 0 🕴 1	🗐 - [E 🖻 🌣	-	Search Test Explorer	، م
Test	Duration	Traits	Fest Detail Summary	
▲ SpecFlowCalculator.Specs (1)	140 ms		😢 Add two numbers in Calculator	
🔺 😢 SpecFlowCalculator.Specs (1)	140 ms		Source: <u>Calculator.feature</u> line	11
🔺 😢 Calculator (1)	140 ms		Test has multiple result outcomes	
😣 Add two numbers in Calculator	140 ms	mytag	🔀 3 Failed	
			 1) Add two numbers Duration: 118 ms Message: The method or operation is no Stack Trace: System.NotImplementedException Calculator.Add() line 12 CalculatorStepDefinitions.Whe BindingInvoker.InvokeBinding(TestExecutionEngine.ExecuteSt 	t imp n: Th <u>nTheT</u> <u>IBind</u> enMat

7- Click on the "Open additional output for this result" below the stack trace to see a more detailed log of the

lest Outcome	: 💟 Failed
Message:	The method or operation is not implemented.
- Standard C	Dutput
-> -> Loa -> -> Loa -> -> Usir	ding plugin C:\work\SpecFlowCalculator\SpecFlowCalculator.Specs\bin\Debug\netcoreapp3.1\LivingDoc.SpecFlowPlugin.dll ding plugin C:\work\SpecFlowCalculator\SpecFlowCalculator.Specs\bin\Debug\netcoreapp3.1\SpecRun.Runtime.SpecFlowPlug ig default config
Given the -> done: (first number is 50 CalculatorStepDefinitions.GivenTheFirstNumberIs(50) (0,1s)
And the se -> done: (cond number is 70 CalculatorStepDefinitions.GivenTheSecondNumberIs(70) (0,0s)
When the -> error: T	two numbers are added he method or operation is not implemented.
Then the r -> skippe	esult should be 120 d because of previous errors
	•
Standard E	
Standard E	rror
Standard E The metho at SpecF	rror od or operation is not implemented.System.NotImplementedException: The method or operation is not implemented. lowCalculator.Calculator.Add() in C:\work\SpecFlowCalculator\SpecFlowCalculator\Calculator.cs:line 12
Standard E The metho at SpecF at SpecF	rror
Standard E The metho at SpecF at SpecFlow at TechT	rror
Standard E The metho at Specf \SpecFlow at TechT. ITestTrace	rror
Standard E The metho at Specf SpecFlow at TechT ITestTrace at TechT	rror
Standard E The metho at Specf \SpecFlow at TechT ITestTrace at TechT \TechTalk.	rror do or operation is not implemented.System.NotImplementedException: The method or operation is not implemented. lowCalculator.Calculator.Add() in C:\work\SpecFlowCalculator\SpecFlowCalculator\Calculator.Calculator.cs:line 12 lowCalculator.Specs.Steps.CalculatorStepDefinitions.WhenTheTwoNumbersAreAdded() in C:\work\SpecFlowCalculator Calculator.Specs\Steps\CalculatorStepDefinitions.cs:line 39 alk.SpecFlow.Bindings.BindingInvoker.InvokeBinding(IBinding binding, IContextManager contextManager, Object[] arguments, r testTracer, TimeSpan& duration) in D:\a\1\s\TechTalk.SpecFlow\BindingsBindingInvoker.cs:line 69 alk.SpecFlow.Infrastructure.TestExecutionEngine.ExecuteStepMatch(BindingMatch match, Object[] arguments) in D:\a\1\s SpecFlow\Infrastructure\TestExecutionEngine.cs:line 514
Standard E The methu at SpecF specFlow at TechT ITestTrace at TechT \TechTalk. at TechT	rror do or operation is not implemented.System.NotImplementedException: The method or operation is not implemented. lowCalculator.Calculator.Add() in C:\work\SpecFlowCalculator\SpecFlowCalculator.Calculator.cs:line 12 lowCalculator.Specs.Steps.CalculatorStepDefinitions.WhenTheTwoNumbersAreAdded() in C:\work\SpecFlowCalculator Calculator.Specs\Steps\CalculatorStepDefinitions.cs:line 39 alk.SpecFlow.Bindings.BindingInvoker.InvokeBinding(IBinding binding, IContextManager contextManager, Object[] arguments r testTracer, TimeSpan& duration) in D:\a\1\s\TechTalk.SpecFlow\Bindings\BindingInvoker.cs:line 69 alk.SpecFlow.Infrastructure.TestExecutionEngine.ExecuteStepMatch(BindingMatch match, Object[] arguments SpecFlow.Infrastructure\TestExecutionEngine.cs:line 514 alk.SpecFlowPlugin.Runtime.RunnerTestExecutionEngine.ExecuteStepMatch(BindingMatch match, Object[] arguments
Standard E The methic at SpecF SpecFlow at TechT ITestTracet at TechT \TechTalk. at TechT at TechT	rror do or operation is not implemented.System.NotImplementedException: The method or operation is not implemented. lowCalculator.Calculator.Add() in C:\work\SpecFlowCalculator\SpecFlowCalculator\Calculator.Calculator.cs:line 12 lowCalculator.Specs.Steps.CalculatorStepDefinitions.WhenTheTwoNumbersAreAdded() in C:\work\SpecFlowCalculator Calculator.Specs\Steps\CalculatorStepDefinitions.cs:line 39 alk.SpecFlow.Bindings.BindingInvoker.InvokeBinding(IBinding binding, IContextManager contextManager, Object[] arguments r testTracer, TimeSpan& duration) in D:\a\1\s\TechTalk.SpecFlow\Bindings\BindingInvoker.cs:line 69 alk.SpecFlow.Infrastructure.TestExecutionEngine.ExecuteStepMatch(BindingMatch match, Object[] arguments specFlow\Infrastructure\TestExecutionEngine.ExecuteStep[(ContextManager contextManager, StepInstance stepInstance) in bCarscFlow.Infrastructure.TestExecutionEngine.ExecuteStep[(ContextManager contextManager, StepInstance) in bCarscFlow.Infrastructure.TestExecutionEngine.ExecuteStep[(ContextManager contextManager, StepInstance) in bCarscFlow.Infrastructure.TestExecutionEngine.ExecuteStep[(ContextManager contextManager, StepInstance) in bCarscFlow.Infrastructure.TestExecutionEngine.ExecuteStep[(ContextManager contextManager, StepInstance) in bCarscFlow.Infrastructure.TestExecutionEngine.ExecuteStep
Standard E The methi at SpecF SpecFlow at TechT ITestTrace at TechT \TechTalk. at TechT at TechTal at TechTal	
Standard E The methi at Specf at SpecFlow at TechT ITestTracei at TechT (TechTalk: at TechT at TechT (\s\TechTal at TechT	fror dot operation is not implemented.System.NotImplementedException: The method or operation is not implemented. lowCalculator.Calculator.Add() in C:\work\SpecFlowCalculator\SpecFlowCalculator\Calculator.cs:line 12 lowCalculator.Specs.Steps.CalculatorStepDefinitions.WhenTheTwoNumbersAreAdded() in C:\work\SpecFlowCalculator Calculator.Specs\Steps\CalculatorStepDefinitions.cs:line 39 alk.SpecFlow.Bindings.BindingInvoker.InvokeBinding(IBinding binding, IContextManager contextManager, Object[] arguments r testTracer, TimeSpan& duration) in D:\a\1\s\TechTalk.SpecFlow\Bindings\BindingInvoker.cs:line 69 alk.SpecFlow.Infrastructure.TestExecutionEngine.ExecuteStepMatch(BindingMatch match, Object[] arguments specFlow\Infrastructure\TestExecutionEngine.ExecuteStep[(ContextManager contextManager, StepInstance stepInstance) in k.SpecFlow.Infrastructure.TestExecutionEngine.cs:line 435 alk.SpecFlow.Infrastructure.TestExecutionEngine.cs:line 435 alk.SpecFlow.Infrastructure.TestExecutionEngine.OnAfterLastStep() in D:\a\1\s\TechTalk.SpecFlow\Infrastructure testInstructure testInstructure.260
Standard E The methic at SpecF SpecFlow at TechT ITestTracei at TechT \TechTalk. at TechT at TechTal \s\TechTal at TechTal at TechTal at TechTal at TechTal at TechTal at TechTal at TechTal	rror do or operation is not implemented.System.NotImplementedException: The method or operation is not implemented. lowCalculator.Calculator.Add() in C:\work\SpecFlowCalculator\SpecFlowCalculator.Calculator.cs:line 12 lowCalculator.Specs.Steps.CalculatorStepDefinitions.WhenTheTwoNumbersAreAdded() in C:\work\SpecFlowCalculator Calculator.Specs.Steps.CalculatorStepDefinitions.cs:line 39 alk.SpecFlow.Bindings.BindingInvoker.InvokeBinding(IBinding binding, IContextManager contextManager, Object[] arguments, r testTracer, TimeSpan& duration) in D:\a\1\s\TechTalk.SpecFlow\Bindings\BindingInvoker.cs:line 69 alk.SpecFlow.Infrastructure.TestExecutionEngine.EsecuteStepMatch(BindingMatch match, Object[] arguments SpecFlow/Infrastructure\TestExecutionEngine.estine 514 alk.SpecFlow.Infrastructure.TestExecutionEngine.ExecuteStep[(IContextManager contextManager, StepInstance stepInstance) in k.SpecFlow\Infrastructure.TestExecutionEngine.estine 435 alk.SpecFlow.Infrastructure.TestExecutionEngine.onAfterLastStep() in D:\a\1\s\TechTalk.SpecFlow\Infrastructure tionEngine.cs:line 260 alk.SpecFlow.Infrastructure.TestExecutionEngine.RunnerTestExecutionEngine.OnAfterLastStep()
Standard E The methic at SpecF SpecFlow at TechT ITestTracei at TechT \TechTalk. at TechT at TechT \s\TechTal at TechT \s\TechTal at TechT at TechT at TechT at TechT at TechT at TechT at TechT at TechT at TechT at TechT	rror do or operation is not implemented.System.NotImplementedException: The method or operation is not implemented. lowCalculator.Calculator.Add() in C:\work\SpecFlowCalculator\SpecFlowCalculator.Calculator.cs:line 12 lowCalculator.Specs.Steps.CalculatorStepDefinitions.WhenTheTwoNumbersAreAdded() in C:\work\SpecFlowCalculator Calculator.Specs\Steps\CalculatorStepDefinitions.cs:line 39 alk.SpecFlow.Bindings.BindingInvoker.InvokeBinding(IBinding binding, IContextManager contextManager, Object[] arguments, r testTracer, TimeSpan& duration) in D:\a\1\s\TechTalk.SpecFlow\Bindings\BindingInvoker.cs:line 69 alk.SpecFlow.Infrastructure.TestExecutionEngine.ExecuteStepMatch(BindingMatch match, Object[] arguments SpecFlow/Infrastructure\TestExecutionEngine.es:line 514 alk.SpecFlow.Infrastructure.TestExecutionEngine.ExecuteStep[(ContextManager contextManager, StepInstance stepInstance) in k.SpecFlow\Infrastructure.TestExecutionEngine.es:line 435 alk.SpecFlow.Infrastructure.TestExecutionEngine.cs:line 435 alk.SpecFlow.Infrastructure.TestExecutionEngine.OnAfterLastStep() in D:\a\1\s\TechTalk.SpecFlow\Infrastructure tionEngine.cs:line 260 alk.SpecFlow.TestRunner.CollectScenarioErrors() in D:\a\1\s\TechTalk.SpecFlow\TestRunner.cs:line 60
Standard E The methic at SpecF (SpecFlow at TechT (TestTrace) at TechT (TechTalk.) at TechT at TechT (S\TechTal) at TechT (S\TechTal) at TechT (S\TechTal) at TechT (S\TechTal) at TechT (S) (TestExect) at TechT at TechT at TechT at TechT at TechT at TechT	rror do or operation is not implemented.System.NotImplementedException: The method or operation is not implemented. lowCalculator.Calculator.Add() in C:\work\SpecFlowCalculator\SpecFlowCalculator.Calculator.cs:line 12 lowCalculator.Specs.Steps.CalculatorStepDefinitions.WhenTheTwoNumbersAreAdded() in C:\work\SpecFlowCalculator Calculator.Specs\Steps\CalculatorStepDefinitions.cs:line 39 alk.SpecFlow.Bindings.BindingInvoker.InvokeBinding(IBinding binding, IContextManager contextManager, Object[] arguments, r testTracer, TimeSpan& duration) in D:\a\1\s\TechTalk.SpecFlow\Bindings\BindingInvoker.cs:line 69 alk.SpecFlow.Infrastructure.TestExecutionEngine.ExecuteStepMatch(BindingMatch match, Object[] arguments) in D:\a\1\s SpecFlow\Infrastructure\TestExecutionEngine.es:line 514 alk.SpecFlow.Infrastructure.TestExecutionEngine.ExecuteStep(IContextManager contextManager, StepInstance stepInstance) in k.SpecFlow\Infrastructure.TestExecutionEngine.ExecuteStep(IContextManager contextManager, StepInstance stepInstance) in k.SpecFlow\Infrastructure.TestExecutionEngine.Cs:line 435 alk.SpecFlow\Infrastructure.TestExecutionEngine.OnAfterLastStep() in D:\a\1\s\TechTalk.SpecFlow\Infrastructure tionEngine.cs:line 260 alk.SpecFlow.TestRunner.CollectScenarioErrors() in D:\a\1\s\TechTalk.SpecFlow\TestRunner.cs:line 60 lowCalculator.Specs.Features.CalculatorFeature.ScenarioCleanup()

You can see that the first two "Given" steps executed successfully and the "When the two numbers are added" step failed with an error. This is because the addition method of the calculator is not implemented yet.

In the next step you'll fix the implementation of the calculator to fix this error.
Fix implementation

3 minutes

In this step you'll fix the implementation error of the calculator in the previous page.

 $\label{eq:calculator.cs} 1- Open \verb|Calculator.cs| in the \verb|SpecFlowCalculator.class| library and replace the implementation of the \verb|Add| method| with the below code:$

```
public int Add()
{
    return FirstNumber + SecondNumber;
}
```

🗙 File Edit View Project Build Debug Test Analyze Tools Extensions Window Search... 🔎 Spec...ator × Help * ¹¹ 🖁 🔚 🌾 👅 📜 🍿 🦄 🦏 🚽 😳 😋 🔹 📸 🕶 🍟 🤔 🔊 - 🤍 - 🛛 Debug 🕞 Any CPU 🖄 Live Share Ŕ Server Explorer CalculatorStepDefinitions.cs Calculator.cs 🛥 🗙 SpecFlowCalculator.Specs.csproj • # × C# SpecFlowCalculator 👻 🔩 SpecFlowCalculator.Calculator 👻 🎤 SecondNumber - ÷ © © ∯ 🛗 - Čo - ≒ 🗗 🕲 s 1 using System; ρ-Search Solution Explorer (Ctrl+;) 2 Solution 'SpecFlowCalculator' (2 of 2 projects) 3 namespace SpecFlowCalculator Toolbox C# SpecFlowCalculator 4 { ... Dependencies 2 references Calculator. 5 public class Calculator SpecFlowCalculator.Specs 6 { Dependencies 2 references Drivers 7 public int FirstNumber { get; set; } Features 2 references Calculator.feature public int SecondNumber { get; set; } 8 Hooks 9 Þ 💼 SpecFlowPlusRunner 💼 Steps public int Add() 10 11 12 return FirstNumber + SecondNumber; 13 7 14 15 115 % 👻 🖉 No issues found 🛛 🚿 🔹 🕨 🕨 Ln: 15 Ch: 2 SPC CRLF ↑ Add to Source Control ▲ 🗇 Ready -0

2- Build the solution. The build should succeed.

3- Run the test. The test should now execute and succeed with the green tick marks indicating no errors:



4- Click on the "Open additional output for this result" to see a more detailed log of the

Test Name: Add two numbers							
Standard Output							
-> -> Loading plugin C:\work\SpecFlowCalcul -> -> Loading plugin C:\work\SpecFlowCalcul -> -> Using default config	lator\SpecFlowCalculator.Specs\bin\De lator\SpecFlowCalculator.Specs\bin\De	ebug\netcoreapp3.1\Liv bug\netcoreapp3.1\Sp	vingDoc.SpecFlowPlugi becRun.Runtime.SpecFl	in.dll owPlugin.dll			
Given the first number is 50 -> done: CalculatorStepDefinitions.GivenTheFi	irstNumberls(50) (0,0s)						
And the second number is 70 -> done: CalculatorStepDefinitions.GivenTheSt	And the second number is 70 -> done: CalculatorStepDefinitions.GivenTheSecondNumberIs(70) (0,0s)						
When the two numbers are added -> done: CalculatorStepDefinitions.WhenTheT	When the two numbers are added -> done: CalculatorStepDefinitions.WhenTheTwoNumbersAreAdded() (0,0s)						
Then the result should be 120 -> done: CalculatorStepDefinitions.ThenTheRe	esultShouldBe(120) (0,0s)						

You can see that each step executed successfully and the test is passed.

In the next step you'll learn how to generate living documentation.

Add Living Documentation

5 minutes

In this step you'll learn how to generate a living documentation from your test execution results so you can easily share them with your team.

> Note: If you have skipped the previous steps make sure your project tests have been executed before continuing with this step.

	2	All	Apps	Document	s Web	More	•								ন্দ
/ie	sual St Cod	Best ma	atch												
	e		Commar App	nd Prompt											
ar I	ndow Cente	Apps De 20	eveloper 19	Command	Prompt fo	or VS	>				Comm	and P	rompt		
		Mi v2	icrosoft A .9	Azure Comr	mand Pror	npt -	>	ď	Open						
		Setting	s					5	Run as	adminis	trator				
		□ Re W	place Co indows P	ommand Pro PowerShell i	ompt with in the Win	+ X	>	₽ -¤	Open f Pin to s	ile locat Start	ion				
		∦ Tu	rn high c	contrast on	or off		>	ᅻᅿ	Pin to t	taskbar					
		≣ M	anage ap	p execution	n aliases		>								
		යු Ch	nange Na	irrator's key	/board lay	out	>								
		💷 Us tin	e Sticky I ne for ke	Keys to pre yboard sho	ss one key ortcuts	at a	>								
		යු Ch	noose if t	he Narrator	r key is loc	ked	>								
		Search	the web												
		ρ _{co}	mmand -	- See web res	sults		>								
1- Open a command prompt.		<i>р</i> со	mmand					0	Ъ	9	F	:		×	×

2- Install the LivingDoc CLI as a global dotnet tool.

dotnet tool install --global SpecFlow.Plus.LivingDoc.CLI



3- Navigate to the **output directory of the SpecFlow project**. In this example the solution was setup in the C:\work folder.

cd C:\work\SpecFlowCalculator\SpecFlowCalculator.Specs\bin\Debug\netcoreapp3.1

4- Run the LivingDoc CLI by using the below command to generate the HTML report.

livingdoc test-assembly SpecFlowCalculator.Specs.dll -t TestExecution.json

Command Prompt		_		×
C:\work\SpecFlowCalculator\SpecFlowCalculator.Specs\bin\Debug\netcoreapp3.1>livingdoc test-assembly SpecFlowCalculator.Specs.dll	-t Tes	stExecut	tion.js	son ^
C:\work\SpecFlowCalculator\SpecFlowCalculator.Specs\bin\Debug\netcoreapp3.1\LivingDoc.html was successfully generated.				
C:\work\SpecFlowCalculator\SpecFlowCalculator.Specs\bin\Debug\netcoreapp3.1>_				
				~

5- Open the generated HTML with your favorite browser. The HTML file is stored in the same folder as the **output directory of the SpecFlow project**.

*Note: if you run into issues here, e.g your JSON file name is FeatureData.JSON instead of TestExecution.JSON, this indicates you have an older version of the CLI tool. Please check our migration guide here to upgrade to the latest

version.

Review the living documentation of the calculator features that you have implemented. Select the "Calculator" feature in the tree. On the right pane check the detailed description of the feature and the scenarios. You can also see the "green" test execution result of the scenarios and steps.

S LivingDoc.html#/document/Stan: × +		>
\leftarrow \rightarrow C () File C:/work/SpecFlowCalculator/Sp	pecFlowCalculator.Specs/bin/Debug/	netcoreapp3.1/LivingDoc.html#/document/Standalone/fea 🛧 😑 🌵 🌲 🔥
SpecFlowCalculator.Specs		
∑ Filter by Keyword	Filter by $ \sim imes $	Seature: Calculator
+ -	Test results 🦲	
✓ ■ SpecFlowCalculator.Specs	● 1 Passed ● 0 Failed ● 0 Others	
V 🖿 Features	● 1 Passed ● 0 Failed ● 0 Others	
✓ Scalculator		
		In order to avoid silly mistakes As a math idiot
		I want to be told the sum of two numbers
		Link to a feature: <u>Calculator</u> Further read: <u>Learn more about how to generate Living Documentation</u>
		@mytag
		Scenario: Add two numbers 2s 122ms
		 Given the first number is 50 And the second number is 70
		V When the two numbers are added
		✓ Then the result should be 120

Check the test result summary by clicking on the "Analytics" tab:



SpecFlow+LivingDoc is packed with great features that truly bring your documentation to life!

To read more about SpecFlow+LivingDoc and its features, please visit our dedicated LivingDoc documentation page.

Final

CONGRATULATIONS!



You have now successfully created and tested your first SpecFlow project.

We have put together a little exercise for you to test your newly acquired skills, *check it out here*.

Check out our examples page if you are looking for additional sample projects. We have also put together a sample project using Selenium for UI automation, you can find it here.

To keep up to date with the latest on SpecFlow Join the SpecFlow Community.

Install JetBrains Rider Plugin

10 minutes

In this step you'll learn how to install the SpecFlow for Rider plugin. SpecFlow's JetBrains Rider plugin not only enables the functionalities needed for testing automation, but is also bundled with several helpful features to make the journey more intuitive.

The plugin can be found either at the JetBrains marketplace or directly from within the Rider IDE.

To install the plugin directly from JetBrains Rider:

1- Open JetBrains Rider

2- Navigate to File Settings Plugins (Ctrl+Alt+S) and search for "SpecFlow" in the search bar:



3- Hit Install and then Accept when prompted with the privacy note. You can find our privacy policy here

P Settings			×
Q	Plugins	Marketplac	e Installed 🌣
 > Appearance & Behavior Keymap > Editor Plugins () Environment 	Q- SpecFlow Search Results (1) Sor SpecFlow for Rider ± 2,6K	× t By: Relevance - Install	SpecFlow for Rider Install ± 2,6K SpecFlow Testing Frameworks Support Language 1.4.0
 > Version Control > Build, Execution, Deployment > Languages & Frameworks > Tools 			Plugin homepage * Adds support for syntax highlighting, smart navigation, and auto-completion to help you write better Gherkin feature files and automation code.
? Manage Layers			Save

4- You are then required to restart the Rider IDE, hit *Restart*:

P Settings				×
٩	Plugins	Marketplace Instal	lled 🌣	Reset
> Appearance & Behavior	Q∗ SpecFlow	×	SpecFlow for Rider	Restart IDE
Editor Plugins Environment Version Control	Search Kesults (1) SpecFlow for Rider ± 2,6K	Sort By: Relevance • Restart IDE Plu	± 2,6K SpecFlow Testing Frameworks Support	Language 1.4.0 Mar 25
 > Build, Execution, Deployment > Languages & Frameworks > Tools 		Add aut aut Size	ds support for syntax highlighting, smart n io-completion to help you write better Ghe omation code. Change Notes e: 1,85 MB	avigation, and rkin feature files and
? Manage Layers				Save Cancel

The installation is now finished. In the next step you'll create a simple application that will be used throughout this guide.

Create calculator project

10 minutes

In this step you'll create the application that will be tested, also called System Under Test (SUT). The application will be a simple calculator in a C# class library.

1- Open JetBrains Rider and create a new C# class library by selecting "New Solution" from the startup dialog:

Welcome to JetBrains Rider		- 🗆 X
JetBrains Rider 2020.3.3	Q Search projects	New Solution Open Get from VCS
Projects	Solution2 ~\RiderProjects\Solution2\Solution2.sln	
Tutorials		
Configure 🗸 Help 🗸		Q

2- Select "Class library" and use the below configurations and click Create.

- Solution & Project name: SpecFlowCalculator
- Solution directory: *choose a location to save the project in this example the solution is saved to C:\projects
- Language: C#
- Framework: netcoreapp3.1

P New Solution			×
٩	.NET / .NET Core > 0	Class Library	
Empty Solution More Templates NET / .NET Core Class Library Console Application Desktop Application Unit Test Project ASP.NET Core Web Applic Worker Service .NET Framework Class Library	Template author: <u>S</u> olution name: <u>P</u> roject name: Solution <u>d</u> irectory: SDK: Language: Framework:	Microsoft SpecFlowCalculator SpecFlowCalculator C:\projects Put solution and project in the same directory Create Git repository 5.0.103 Change C# netcoreapp3.1	1
Desktop Application Unit Test Project ASP.NET Web Application Shared Project Unity Class Library Xamarin Library Application Unity Let Project	Resulting project st	ructure Project template info pwCalculator\SpecFlowCalculator.sln pwCalculator\SpecFlowCalculator\ <project files=""></project>	
Or Unit Test Project		Create	el

3- Rename Class1.cs to Calculator.cs and overwrite the content with the following code :

```
using System;
namespace SpecFlowCalculator
{
    public class Calculator
    {
        public int FirstNumber { get; set; }
        public int SecondNumber { get; set; }
        public int Add()
        {
            throw new NotImplementedException();
        }
    }
}
```



4- Now build the solution by navigating to "Build Build Solution". You should see a "Build Succeeded" message in the output window:

E <u>File Edit View N</u> avigate <u>Code R</u> efactor	Build Run Iests Tools VCS Window Help SpecFlowCalculator -		×			
SpecFlowCalculator) Tc# SpecFlowCalculator)	K Build Solution Ctrl+Shift+B ▼ Add Configuration ►	i 🔲 🖿	Q			
ছ & Solution ▼ 😌 🗄 🛣 🗰 — 🗰 🗸	Rebuild Solution		A			
Image: Constraint of the second se	Build Current Project: SpecFlowCalculator Ctrl+Shift+F7 Build Startup Project Advanced Build Actions Change Solution Configuration Change Device % Cancel Ctrl+F9 ; } public int SecondNumber { get; set; }	•	Errors In Solution 🔹 IL Viewer			
10 11 12 13 14 15	<pre>public int Add() { throw new NotImplementedException(); } </pre>		())) Database			
Struc	() SpecFlowCalculator					
Build: Build succeeded		\$ -				
Build with surface heuristics started at 15:10:05 Build completed in 00:00:00.006 Build succeeded at 15:10:05						
Build whole solution	SpecFlowCalculator 15:2 CRLF UTF-8 4 spaces	Event Log	▼			

The calculator application is now built. In the next step you'll learn how to create a SpecFlow project.

Create SpecFlow project

5 minutes

In this step you'll create a SpecFlow project and add it to the existing calculator solution.

1- Right-click the solution item "'*SpecFlowCalculator*'(*1 of 1 project*)" under the Solution Explorer and select the "Add New Project" menu item.

RD	<u>F</u> ile <u>E</u> dit <u>V</u> iew <u>N</u> avigate <u>C</u> ode <u>R</u> efactor <u>B</u> uild R <u>u</u> n <u>T</u> ests To	oo <u>l</u> s VCS <u>W</u> indow <u>H</u> elp SpecFlowCalculate	or — 🗆	×
	SpecFlowCalculator.sln	i Debug Any CPU 👻	Add Configuration 🕨 🐞 🔳	F Q
Explorer	▲ Solution → ③ 王 ☆ ← C# Calculator.cs × G 구 구 Q 1 Using System SpecFlowCalculator 1 pro Add →	m; New Project Ctrl+Shift+N		Errors In Solu
	 > & Dependencies C# Calculator.cs Scratches and Consoles Manage NuGet Packages Unload Projects Reload Projects Suild Solution Run Unit Tests Ctrl+U, R Publish 	New Solution Folder Add Existing Project Attach Existing Folder New Scratch File Ctrl+Alt+Shift+Insert Data Source		tion 🔹 IL Viewer
	Advanced Build Actions Show Local <u>H</u> istory Refactor This Ctrl+Shift+R ▶ Inspect Code Code Cleanup Ctrl+E.C	DDL Data Source Data Source from URL Data Source from Path Driver and Data Source Driver	0;	()))) Database
orites 🔚 Structure	Build: Build succeeded Build with surfac Properties Build completed in Alt+Enter	culator 10:05		-
Fav	Build succeeded at 15:10:05 ⇒ TODO ® NuGet © Unit Tests © Dynamic Program Analysis Create new project	Terminal SpecFlowCalculator 1	Event Lo S:2 CRLF UTF-8 4 spaces 1 @ @g	g V

2- Click on *SpecFlow Project Template* under the *Other* category, enter the project name as **"SpecFlowCalcula-tor.Specs"**, keep the suggested location (the solution folder), pick *xUnit* as the Test Framework and hit *Create*:

Note: If you cannot see SpecFlow Project Template, ensure you have SpecFlow for Rider Plugin 1.6.0 or higher installed. (Only compatible with Rider 2021.1 or higher)

Note: Currently running the tests from the feature files is only possible with xUnit and NUnit.*

P New Project		×	
Q.	Other > SpecFlow P	Project	
More Templates .NET / .NET Core .NET / .NET Core Class Library Console Application Desktop Application Unit Test Project ASP.NET Core Web Applicati Worker Service	Template author: <u>P</u> roject name: Project <u>d</u> irectory: Language: Framework: Test Framework: FluentAssertions: Resulting project st	SpecFlow Team SpecFlowCalculator.Specs C:\projects\SpecFlowCalculator C# Include Tucture Project template info	
 Desktop Application Unit Test Project ASP.NET Web Application Shared Project Unity Class Library Xamarin Library Application Unit Test Project Other 	ktop Application t Test Project .NET Web Application red Project ty Class Library Xamarin ary vlication t Test Project Other	ator.Specs\ <project files=""></project>	
SpecFlow Project		Create Cancel	

3- JetBrains Rider will now create the new project, you should see the new SpecFlow project in the Solution Explorer as per below:



In the next step you will learn how to add a project reference and how to use the test explorer.

Create SpecFlow project - Continue

5 minutes

You will now add a project reference to the "SpecFlowCalculator" class library in the newly created SpecFlow project. This is necessary because we want to test the "Calculator" class implemented in the class library in the "SpecFlow-Calculator.Specs" project. To do this, follow the below steps:

1- Expand the project node "SpecFlowCalculator.Specs" in the Solution Explorer, right-click the "Dependencies" node and select the "Add Reference..." menu item.

2- In the "Add Reference" dialog check the "SpecFlowCalculator" class library and click Add.

🖺 <u>F</u> ile <u>E</u> dit <u>V</u> iew <u>N</u> avigate <u>C</u> ode <u>R</u> efactor <u>B</u> uild R <u>u</u> n <u>T</u> ests Too <u>l</u> s VCS <u>W</u> indow <u>H</u> elp SpecFlowCalculator — 🗆	×
🔽 SpecFlowCalculator > T ^C # SpecFlowCalculator.Specs.csproj 🔨 🔽 Debug Any CPU 🔻 Add Configuration 🕨 🔅 🔳	ΡQ
▲ Solution ▼ ⊕ Ξ ÷ ♥ − ♠ Solution ▼ ⊕ Ξ ÷ ♥ − ♥ □ ⊕ SpecFlowCalculator · 2 projects ♥ □ ⊕ SpecFlowCalculator.Specs Search Everyw ▶ ⊕ Dependencies ▲ dd Reference ▶ ⊕ Projects ▶ ⊕ Drivers ▲ dd Reference ▶ ⊕ Features ▶ ⊕ Hooks ▶ ⊕ Steps Find Dependent Code	Errors In Solution
Scratches and Consoles Navigation Bar Drop files here I reference is selected Add From Add Cancel	og 👽

Now the solution is set up with a class library containing the implementation of the calculator and a SpecFlow project that contains the specification and tests of the calculator.

3- Now build the solution. You should see the "Build Succeeded" message in the output window.

4- Run all the tests by navigating to "Tests Run All Tests from Solution":

> Note: The red underline applied to the project name and feature file in the explorer pane is a known xUnit bug in Rider and does **NOT** indicate an error.

Eile Edit View Navigate Code Refactor	<u>B</u> uild R <u>u</u> n	<u>T</u> ests T	oo <u>l</u> s VCS <u>W</u> indov	w <u>H</u> elp SpecF	lowCalculator	-		2	×
culator $\ensuremath{ar{r}}\xspace$ SpecFlowCalculator.Specs $>$ Steps $>$ C# C	alculatorStep	Unit	Testing Quick List	Alt+Shift+U	All tests from Solution 🔻		ĕ.∎		Q
🖞 📣 Solution 👻 😳 🚊 🛧 🗢 —	C# Calcula	🕨 Run	Unit Tests	Ctrl+U, R					д
e F T Q	15	道 Debu	ug Unit Tests	Ctrl+U, D				~	Erro
SpecElowCalculator 2 projects	16	🕨 Run	Current Session	Ctrl+U, Y			nl sro		
✓ C ² SpecFlowCalculator	17	🕨 Run	All Tests from Solution	on Ctrl+U, L	umber is (.*)")]				Solu
> & Dependencies		🕈 Repe	at Previous Run	Ctrl+U, U				- 1	- tion
C# Calculator.cs	18	🔎 Reru	n Failed Tests	Ctrl+U, F	stNumberIs(int number)				1
✓ IC ^a SpecFlowCalculator.Specs	19	Run	Tests Until Failure	Ctrl+U, W					\sim
A Dependencies	20	Debu	ug Tests Until Failure		rrange (precondition) l	ogic			1
> 🐰 .NETCoreApp 3.1	21	Stop	Execution	Ctrl+U, S	retrieving scenario-spe	cifi	c dat	l see	ewe
> 🖸 Imports	22	+ App	end Tests to Session	Ctrl+U, A	tine text on the table	araw	ment N	iff th	e 7
Drivers		Great Creat	te New Session	Ctrl+U, N				~ _	()))
Air tests rom solution X		Appe	end Contained Tests					4. —	Data
▶ ▼ Q Ξ ≚	^ ↓ ‡	Rem	ove Selected Test						base
			Clear Test Results		xUnit.SpecFlowPlugin.XUnitPendir	igStep	Exceptio	n: Test	
			is Unit Test Explorer	Ctrl+Alt+U	s.CalculatorFeature.AddTwoNumbers				
을 😋 🔍 🗢 CalculatorFeature (1 test) Fai	led: One or r	Focu	s Unit Test Session	Ctrl+Alt+T					
물 🗢 🗢 Add two numbers Failed:	🗢 🗣 🗢 Add two numbers Failed: TechTa k.Sp				Plugin.XUnitPendingStepException:				
			Tests Tool Window A	Actions 🕨 🕨	definitions are not implemented				
			on Options	►					
ites				►					
avor	TechT	TechTalk.SpecFlow.xUnit.SpecFlowPlugin.XUnitPendingStepException							
*	Test pending: One or more step definitions are not implemented yet.								
Latouratorsteoperintitions.5tventnertirstwumperiston									
	ynamic Progra	m Analysi v 12·53)	s ice rerminal	lator Specs 15.5	CRIE LITE-8 4 spaces 💁 🗥	1 erro	tin 1 fil		∇

The tests would fail as expected as our step definitions are not yet implemented.

In the next step you will learn how to automate your first scenario and implement the step definitions.

Bind the first step

10 minutes

In this step you'll bind your first step (automate your first scenario step with SpecFlow).

1- Open the Calculator.feature file by double-clicking it in the Solution Explorer (SpecFlowCalculator.Specs Features Calculator.feature)



The purpose of this feature file is to document the expected behavior of the calculator in a way that it is both human-

readable and suitable for test automation. SpecFlow uses the Gherkin language where you can phrase the scenarios using *Given/When/Then* steps. Currently there is a single scenario (automatically added by the SpecFlow project template) that describes how adding two numbers should work with the calculator.

Here is a closer look at the Gherkin scenario used in this template:

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

Based on the scenario text, SpecFlow generates an automated test that executes the scenario. However, it is **not yet defined** what the steps of the scenario should actually "**do**".

2- Right-click the first *Given* step "Given the first number is 50" and select "*Go To* \rightarrow *Implementation*" or use the "*Ctrl* + *F12*" shortcut.



The SpecFlow plugin locates the step definition (binding) that belongs to this step. In this example, it opens the CalculatorStepDefinitions class and jumps to the GivenTheFirstNumberIs method.



*The step definition is located based on the [Binding] attribute on the class and the [Given] attribute on the method. The regular expression of the Given attribute matches the text of the scenario step.

3- Add the below field to the class to instantiate the calculator that we want to test and created in *Step 2* of this guide (SUT).

private readonly Calculator _calculator = new Calculator();

4- Replace the implementation of the first step definition method with the below code which sets the first number of the calculator.

```
[Given("the first number is (.*)")]
public void GivenTheFirstNumberIs(int number)
{
    __calculator.FirstNumber = number;
}
```



5- Execute the test in the Test Explorer and open the text explorer output to see the details. You can see the "*done*" status here indicating the first step "Given the first number is 50" has been matched to the step definition method as per above binding. The remaining steps are yet to be implemented and are in "*pending* and "*skipped*" status as expected.



In the next step you will bind the rest of the scenario steps.
Bind remaining steps

10 minutes

In this step you'll bind the remaining steps of the scenario.

1- Similar to the previous step, Right-click the second *Given* step "And the second number is 70" and select "*Go To* \rightarrow *Implementation*" or use the "*Ctrl* + *F12*" shortcut.

2- Implement the binding of the second step "And the second number is 70" by replacing the code of the GivenTheSecondNumberIs method with the below:

```
[Given("the second number is (.*)")]
public void GivenTheSecondNumberIs(int number)
{
    __calculator.SecondNumber = number;
}
```



> Note: We use the "And" keyword in the Gherkin scenario for better readability. The "And" keyword will be interpreted as "Given", "When" or "Then" depending on the previous step(s) in the scenario. In this example the "And the second number is 70" is interpreted as a "Given" step because the previous step is a "Given" step.

3- Next, implement the binding of the third step, "When the two numbers are added", by replacing the code of the WhenTheTwoNumbersAreAdded method with the below. The method must have a When attribute, as it belongs to the "When" step in the scenario.

private int _result;

```
[When("the two numbers are added")]
public void WhenTheTwoNumbersAreAdded()
{
    _result = _calculator.Add();
}
```



This implementation calls the Add method of the calculator. Note that the result of the addition is not stored by the calculator in a property/field but it is returned to the caller. It's a good idea to store the returned value in a field so that we can work with the result afterwards.

4- Implement the binding of the last step, "Then the result should be 120", by replacing the code of the ThenTheResultShouldBe method. The method must have a Then attribute, as it belongs to a "Then" step in the scenario.

Add a namespace using for xUnit at the top of the file:

```
using Xunit;
```

Use the below code for implementation of the "Then" step which validates if the result of the addition matches the expected value.

```
[Then("the result should be (.*)")]
public void ThenTheResultShouldBe(int result)
{
    Assert.Equal(result, _result);
}
```



After implementing all step definitions and cleaning up the file you should have the following code:

```
using TechTalk.SpecFlow;
using Xunit;
namespace SpecFlowCalculator.Specs.Steps
    [Binding]
   public sealed class CalculatorStepDefinitions
    {
        private readonly ScenarioContext _scenarioContext;
        private readonly Calculator _calculator = new Calculator();
        private int _result;
        public CalculatorStepDefinitions(ScenarioContext scenarioContext)
        {
            _scenarioContext = scenarioContext;
        }
        [Given("the first number is (.*)")]
        public void GivenTheFirstNumberIs(int number)
        {
            _calculator.FirstNumber = number;
        }
        [Given("the second number is (.*)")]
        public void GivenTheSecondNumberIs(int number)
```

(continues on next page)

(continued from previous page)

```
{
    __calculator.SecondNumber = number;
}
[When("the two numbers are added")]
public void WhenTheTwoNumbersAreAdded()
{
    __result = _calculator.Add();
}
[Then("the result should be (.*)")]
public void ThenTheResultShouldBe(int result)
{
    Assert.Equal(result, _result);
}
```

5- Build the solution. The build should succeed.

6- Run the test again.

The test should execute and fail, this is expected. In the Test Detail Summary pane of Test Explorer you can see that the first two "Given" steps executed successfully and the "When the two numbers are added" step failed with an *error*: *The method or operation is not implemented*. This is because the addition method of the calculator is not implemented yet.

RD	<u>F</u> ile <u>E</u> d	lit <u>V</u> iew <u>N</u>	<u>N</u> avigate <u>C</u> ode <u>R</u> efact	tor <u>B</u> uild R <u>u</u> n	<u>T</u> ests Too <u>l</u> s \	VCS <u>W</u> indow <u>H</u> elp	SpecFlow	Calculator		_	[×	(
	SpecFlow	Calculator)	C [®] SpecFlowCalculator	Specs $ angle$ Steps $ angle$	C# CalculatorS1	🔨 🕅 Debug Any	CPU 🔻	🕈 All tests fr	om Solution 🔻		ð,		Þ	Q
rer	😌 Calcu	lator.featur	e × C# CalculatorStep	Definitions.cs $ imes$										4
Explo	29		{								A	1 ^	\sim	Erro
	30		_calcula	tor.SecondNu	umber = numb	er;								nl s
	31		}											Solu
	32													tion
	33		[When(regex: "t	the two numb	ers are add	ed")]								
			1 usage									ĥĥĥ		<u>_</u>
		Sterior	eps 🔿 🔩 CalculatorSte	$pDefinitions \rightarrow$	GivenTheSecond	ondNumberls								- Sie
	Unit Tests	: Explore	er 🗢 All tests from S	olution $ imes$								\$	—	wer
	▶ -	Q	4	<u>∓</u> ↑ ↓ :	北区文法	oð 1 🗢 1								
	• •		cElowCalculator.Specs (i	test) Failed: 1 te	Add two num	bers [157 ms] System.N	otimplemen	itedException: T	he method or on	eratio	n is no	ot imr	lem	Dat
	ð.	✓ ○ () ≤	SpecFlowCalculator.Spec	s.Features (1 test		is cis [151 mis] systemint	o ann preimei		ine incluied of op					tabas
	¢	× 😑	CalculatorFeature (1 test) Failed: One or r	Given the	first number is	50							ö
	, - -		Add two numbers Fa	iled: System.Not	-> done: C	alculatorStepDef	inition	s.GivenTheF	irstNumberI	<mark>s(</mark> 50) (0),0s)	
E					And the se	cond number is 7	70							
ructi					-> done: C	alculatorStepDef	inition	s.GivenTheS	econdNumber	Is(7	0)			
St					(0,0s)						_			
					When the t	wo numbers are a	dded			0.0-				
ites					-> error:	ine method or op	120	is not imp	Lemented. (0,0S	, ,			
avor					-> skinned	because of prev	ious er	nons						
\star					> Skiphed	because of prev	1003 011	015						
	≡ торо) ∉Õ Dyn	namic Program Analysis	© Unit Tests	Terminal	🕲 NuGet 🛛 🔨 Build					3 E	vent L	og	
	Unit test s	ession 'All t	ests from Solution' is fin	ished	Spe	ecFlowCalculator.Specs	26:38 CF	RLF UTF-8 4 s	paces 🍙 🚱	1 erro	r in 1 f	file (Ð.	V

In the next step you'll fix the implementation of the calculator to fix this error.

Fix implementation

3 minutes

In this step you'll fix the implementation error of the calculator in the previous page.

 $\label{eq:calculator.cs} 1- Open \verb|Calculator.cs| in the \verb|SpecFlowCalculator.cs| in the \verb|SpecFlowCalculator.cs| in the below code: \\$

```
public int Add()
{
    return FirstNumber + SecondNumber;
}
```



2- Build the solution. The build should succeed.

3- Run the test. The test should now execute and succeed with the green tick marks indicating no errors:



The automation phase is finished, in the next step you'll learn how to generate living documentation for reporting purposes.

Add Living Documentation

5 minutes

In this step you'll learn how to generate a living documentation from your test execution results so you can easily share them with your team.

1- Open a Command Prompt.

2- Install the LivingDoc CLI as a global dotnet tool.



3- Navigate to the **output directory of the SpecFlow project**. In this example the solution was setup in the C:\projects folder.

cd C:\projects\SpecFlowCalculator\SpecFlowCalculator.Specs\bin\Debug\netcoreapp3.1

4- Run the LivingDoc CLI by using the below command to generate the HTML report.

*Note if you run into issues here, e.g your JSON file name is FeatureData.JSON instead of TestExecution.JSON, this indicates you have an older version of the CLI tool. Please check our migration guide here to upgrade to the latest version.

5- Open the generated HTML with your favorite browser. The HTML file is stored in the same folder as the **output directory of the SpecFlow project**.

C:\projects\SpecFlowCalculator\SpecFlowCalculator.Specs\bin\Debug\netcoreapp3. \rightarrow 1\LivingDoc.html

Review the living documentation of the calculator features that you have implemented. Select the "Calculator" feature in the tree. On the right pane check the detailed description of the feature and the scenarios. You can also see the "green" test execution result of the scenarios and steps.

S LivingDoc.html#/document/Stanc × +		– – ×
igstarrow igstarro	Calculator/SpecFlowCalculator.Specs/bin/Debug/net	coreapp3.1/LivingDoc.html#/document/Standalone/feature/2f3 🏠 😁 Incognito 🔅
SpecFlowCalculator.Specs		
Living Documentation Analytics		
√ Filter by Keyword	Filter by $$	✓ Feature: Calculator Help us: Want to edit? Want to comment? ▲
+ -	Test results	
✓ ■ SpecFlowCalculator.Specs	● 1 Passed ● 0 Failed ● 0 Others	
V Features	● 1 Passed ● 0 Failed ● 0 Others	
V 🥝 Calculator		
Add two numbers		
		Simple calculator for adding two numbers
		Link to a feature: <u>Calculator</u> Further read: <u>Learn more about how to generate Living Documentation</u>
		🖉 @mytag
		Scenario: Add two numbers 10ms
		 Given the first number is 50 And the second number is 70
		✓ When the two numbers are added
Generated by SpecElow+LivingDoc - Give us fee	thackl	✓ Then the result should be 120 .

Check the test result summary by clicking on the "Analytics" tab:

Features					Scenarios					
	100.0% 1	0.0% O	0.0% 0		(1	100.0% 1	0.0% 0	0.0% 0	
U	PASSED	FAILED	OTHERS				PASSED	FAILED	OTHERS	
Steps										
	100.0%	0.0%	0.0%	0.0%	0.0%	0.0%				
4	4	0	0	0	0	0				

SpecFlow+LivingDoc is packed with great features that truly bring your documentation to life! To read more about SpecFlow+LivingDoc and its features, please visit our LivingDoc documentation page.

Final

CONGRATULATIONS!



You have now successfully created and tested your first SpecFlow project.

We have put together a little exercise for you to test your newly acquired skills, *check it out here*.

Check out our examples page if you are looking for additional sample projects. We have also put together a sample project using Selenium for UI automation, you can find it here.

To keep up to date with the latest on SpecFlow Join the SpecFlow Community.

Exercise

15 minutes

In this step it is your turn to implement the subtraction feature of the calculator.

1- Add the following scenario to the Calculator.feature file.

```
Scenario: Subtract two numbers
Given the first number is 120
And the second number is 70
When the two numbers are subtracted
Then the result should be 50
```

2- Build the solution. The build should succeed.

you tests 3-Run the tests. Notice that have 2 (corresponding to now "Skipped" your two scenarios) and the second scenario is because of missing

Test Explorer						
🕨 🕨 🕑 🔽 2 🕑 1 🔀 0 🥼	1 📓 - [[] 🗇 🌣 -	Search Test Explorer				
est	Duration Traits	Test Detail Summary				
 SpecFlowCalculator.Specs (2) 	206 ms	🔒 Subtract two numbers in Calculator				
🔺 🕑 SpecFlowCalculator.Specs (2)	206 ms	Source: <u>Calculator.feature</u> line 18				
🔺 🕑 Calculator (2)	206 ms					
🕑 Add two numbers	97 ms mytag					
🛕 Subtract two numbers	109 ms					

bindings.

4- Click on the "Open additional output for this result" and review the details of the scenario



execution.

Now it is your turn to implement the subtraction feature in three short steps:

- 1. First add the missing binding (with the minimum code structure necessary) to get a red scenario.
- 2. Next turn the scenario green by actually implementing the subtraction logic in the calculator.
- 3. Refactor your implementation if necessary (scenarios should remain green).

Did it work out?

Check the next step for a possible solution.

Exercise-solution

10 minutes

In this step you can review the solution for the challenge in the previous step (implementing the subtraction in the calculator).

To recap the 3 steps to implement the new feature:

- 1. First add the missing bindings (with the minimum code structure necessary) to get a red scenario.
- 2. Next turn the scenario green by actually implementing the logic.
- 3. Refactor the implementation if necessary (scenarios should remain green).

The first step is to add the missing binding and necessary code to have a red scenario. A quick way of generating the necessary binding methods is to you right-click the unbound step in the feature file and select "Go To

17				
18				
19 Given the first number is 120				
20 And the second number is 70				
21 When the two numbers are subtracted	1			
22 Then the result should be 50	χ	Rename	Ctrl+R, Ctrl+R	
	1	Go To Definition	F12	
		Breakpoint		•
	k	Run To Cursor	Ctrl+F10	
	Ж	Cut	Ctrl+X	
	ŋ	Сору	Ctrl+C	
	ப	Paste	Ctrl+V	
		Annotation		•
		Outlining		•
		Generate Step Definitions Go To Step Definition	Ctrl+Shift+Alt+	s
				_

Definition".

If SpecFlow does not find the corresponding binding method it offers to generate the skeleton of the binding method

Go to binding						
?	No matching step binding found for this step! Do you want to copy the step binding skeleton to the clipboard? [When(@"the two numbers are subtracted")] public void WhenTheTwoNumbersAreSubtracted() { ScenarioContext.Current.Pending(); }					
	Yes No					

into your clipboard.

Now you can easily paste the method into the binding class and change the implementation.

```
[When(@"the two numbers are subtracted")]
public void WhenTheTwoNumbersAreSubtracted()
{
    _result = _calculator.Subtract();
}
```

To be able to implement the binding in a meaningful way you have to extend the public interface of the calculator as well to support the subtraction. However, in the first step, your only goal is to get to an executable red scenario. Hence you have to add a Subtract method to the calculator, but the implementation should be skipped e.g. by throwing a NotImplementedException. Note that in this case the scenario will fail in the "When" step already due to the exception and the "Then" step will be skipped.

```
using System;
namespace SpecFlowCalculator
{
   public class Calculator
    {
        public int FirstNumber { get; set; }
        public int SecondNumber { get; set; }
        public int Add()
        {
            return FirstNumber + SecondNumber;
        }
        public int Subtract()
        {
            throw new NotImplementedException();
        }
    }
}
```

Alternatively you can return a dummy value (e.g. constant 0). In this case the scenario will also run the "Then" step and fail on the assertion. This is especially beneficial if you've just created the binding of the Then step too and you want to make sure that the binding works as expected.

```
public int Subtract()
{
    return 0;
}
```

If you build the solution and run the tests the scenario should be red and you're ready to move on to the second step.

The second step is to implement the subtraction of the calculator to get the scenario green.

```
public int Subtract()
{
    return FirstNumber - SecondNumber;
}
```

If you run the tests again the scenario should be green.

The last step is to refactor the code while keeping all scenarios green. However, in this case the implementation is so simple that we can skip the refactoring step now.

Additional resources

Want to learn more?

- Learn Gherkin
 - What is Gherkin?
 - Given-When-Then with Style Challenge
 - Gherkin Reference
- Architectures & Good Practices
 - Getting started with the BookShop example
- Automation Patterns
 - Page Object Pattern
 - Driver Pattern
- Example code
 - SpecFlow Examples
 - Given-When-Then with Style challenges
 - * Challenge 1-2
 - * Challenge 7
 - * Challenge 8
- Example Projects
 - Examples

Need Help?

- Forum
- Trainers
- Online Courses