SpecFlow+ LivingDoc

Feb 08, 2022

Step-by-step guides

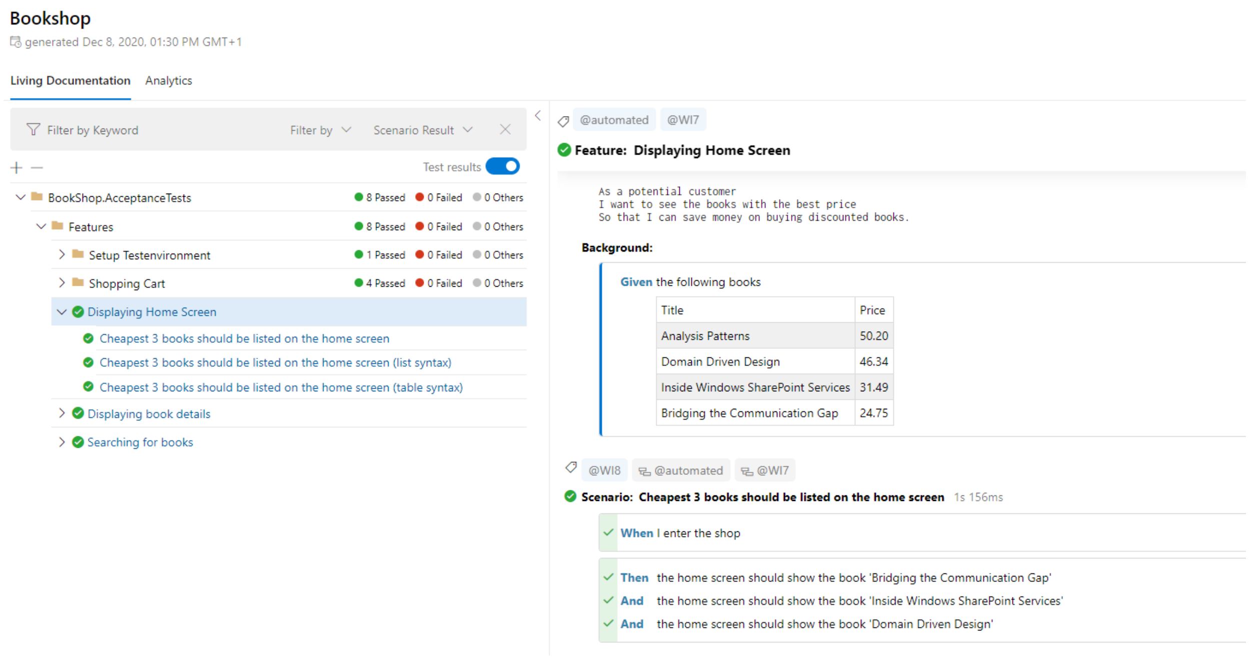
SpecFlow+LivingDoc is a set of tools that allows you to share and collaborate on Gherkin Feature Files with stakeholders who may not be familiar with developer tools.

SpecFlow+LivingDoc gives you the option to generate & share living documentation in two configurable ways:

- *SpecFlow+LivingDoc for Azure DevOps*: If your team uses Azure DevOps then we suggest installing our dedicated extension to help you generate and share LivingDoc within the familiar Azure DevOps interface.
- *SpecFlow+LivingDoc Generator*: If you want to generate a self-hosted HTML documentation with no external dependencies so you have the freedom to share it as you wish, then we suggest the SpecFlow plugin and command-line tool.

Note: You will find all the information related to LivingDoc on this page. If you are new to LivingDoc, check out the **Step-by-step guides** section to get started.

Sample output:


Bookshop
generated Dec 8, 2020, 01:30 PM GMT+1
Living Documentation
Analytics
Filter by Keyword
Filter by
Scenario Result
Test results
BookShop.AcceptanceTests 8 Passed 0 Failed 0 Others
Features 8 Passed 0 Failed 0 Others
Setup Testenvironment 1 Passed 0 Failed 0 Others
Shopping Cart 4 Passed 0 Failed 0 Others
Displaying Home Screen
Cheapest 3 books should be listed on the home screen
Cheapest 3 books should be listed on the home screen (list syntax)
Cheapest 3 books should be listed on the home screen (table syntax)
Displaying book details
Searching for books
@automated
@WI7

Feature: Displaying Home Screen

```
As a potential customer
I want to see the books with the best price
So that I can save money on buying discounted books.
```

Background:

Given the following books

Title	Price
Analysis Patterns	50.20
Domain Driven Design	46.34
Inside Windows SharePoint Services	31.49
Bridging the Communication Gap	24.75

Scenario: Cheapest 3 books should be listed on the home screen 1s 156ms

When I enter the shop

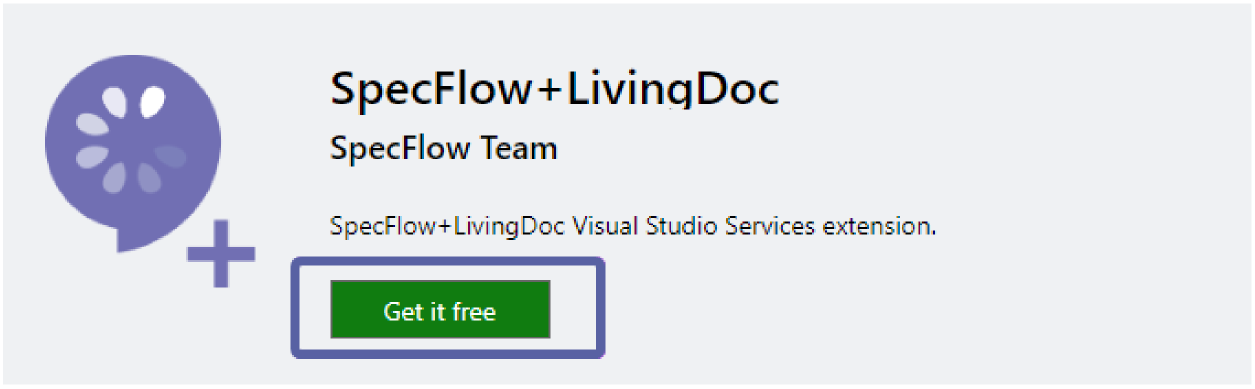
Then the home screen should show the book 'Bridging the Communication Gap'
And the home screen should show the book 'Inside Windows SharePoint Services'
And the home screen should show the book 'Domain Driven Design'

CHAPTER 1

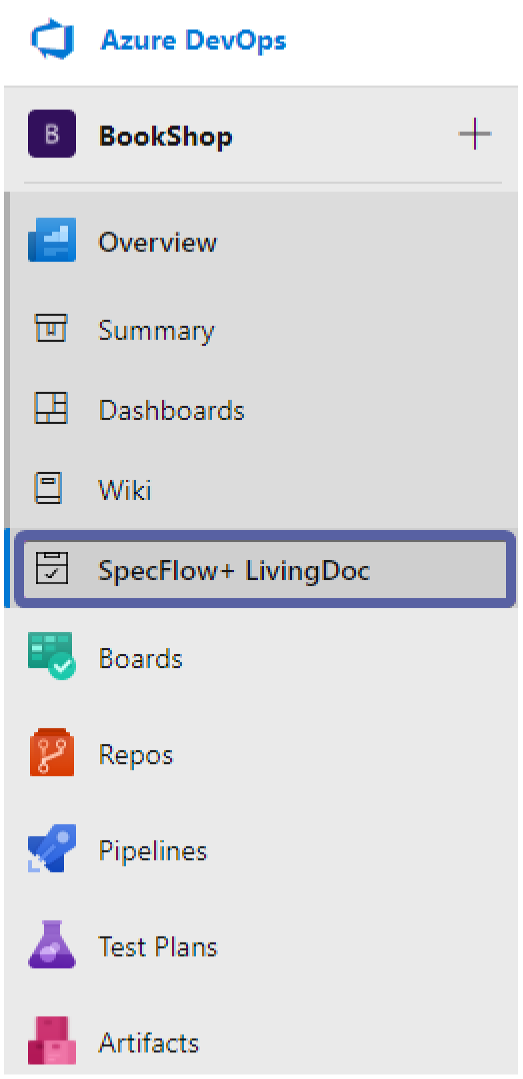
Step-by-step guide for LivingDoc Azure DevOps

1.1 Step 1 - Installation

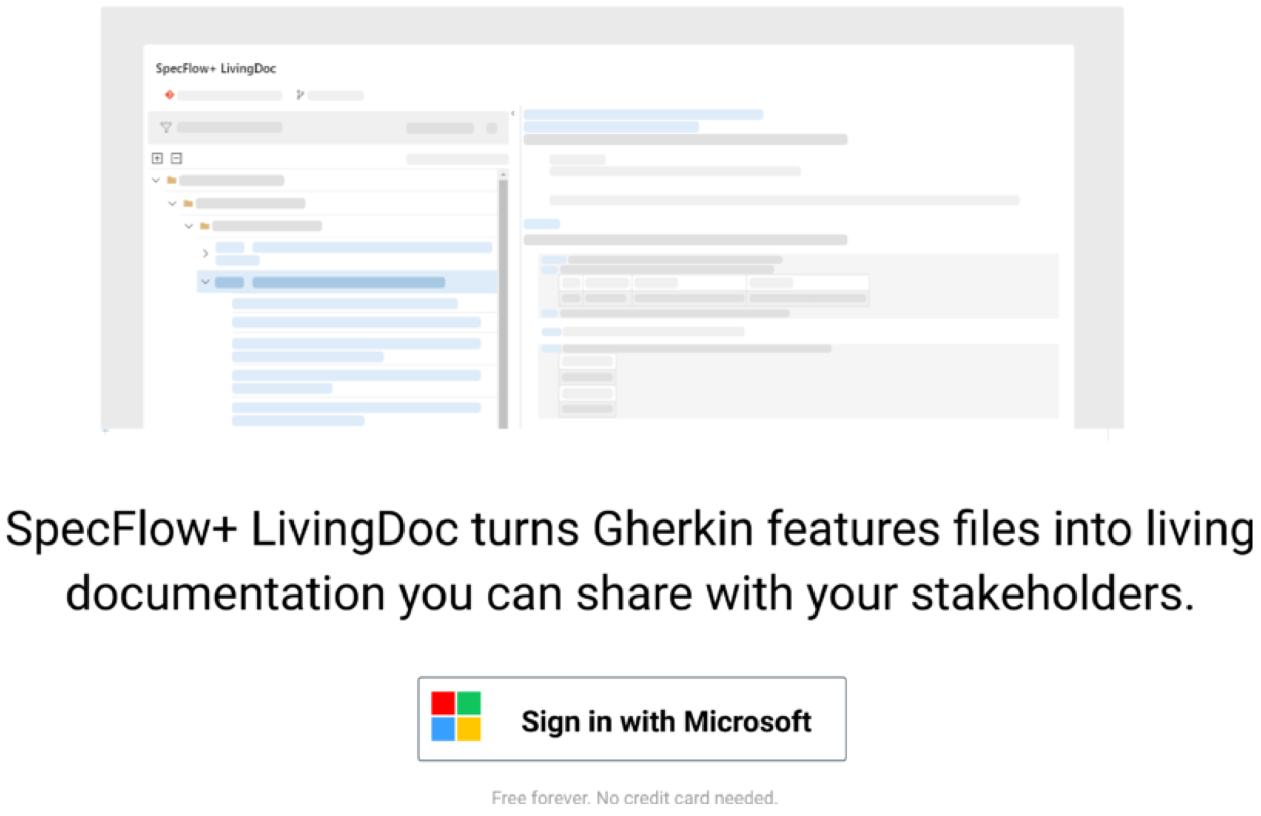
1- Go to the SpecFlow+LivingDoc page on the Visual Studio marketplace and install the **free** extension. Since this extension is tied to Microsoft Azure DevOps, you will need to sign in with your Microsoft account.



Once the installation finishes, you will see the **SpecFlow+ LivingDoc** menu item show up under the **overview** section of Azure DevOps:



2- The first time you click on SpecFlow+ LivingDoc from the overview pane, you will also be asked to sign up for a SpecFlow account. Click on **Sign in with Microsoft** and follow the instruction to create a **free** SpecFlow account.



SpecFlow+ LivingDoc turns Gherkin features files into living documentation you can share with your stakeholders.

Free forever. No credit card needed.

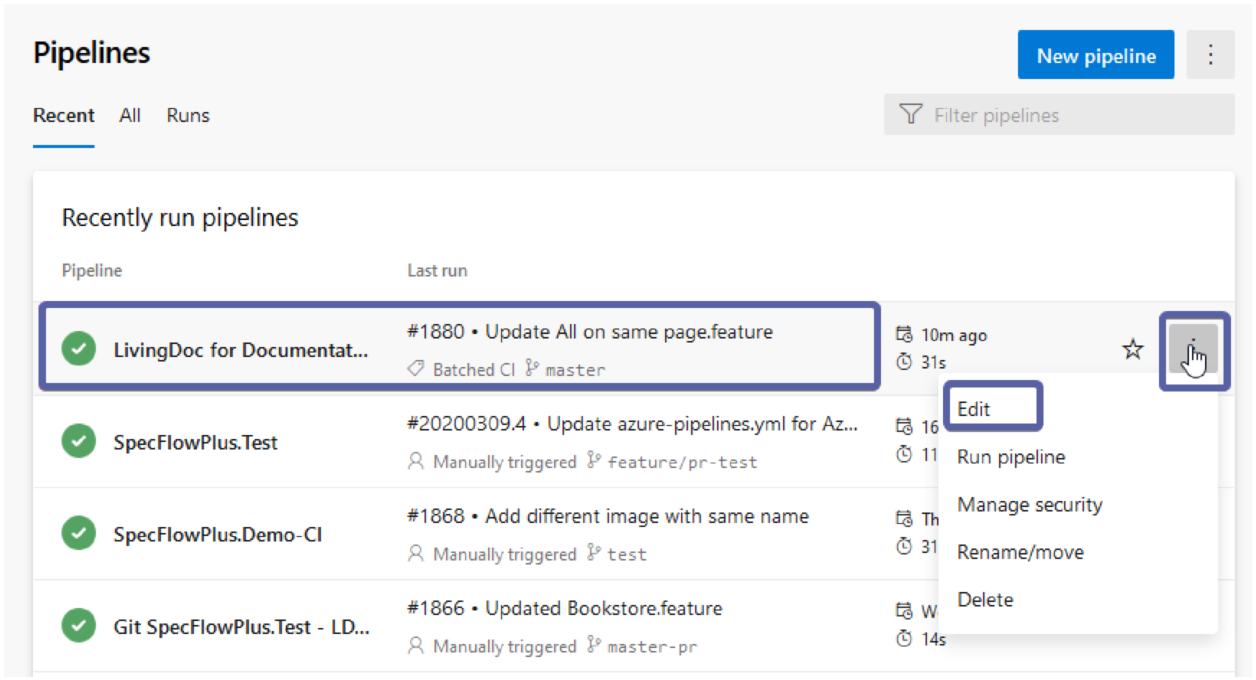
3- If you want to generate LivingDoc **with** test execution results you also need to install the SpecFlow.Plus.LivingDoc Plugin for your SpecFlow project. The plugin will generate the necessary JSON file that will carry the test results data.

1.2 Step 2 - Build step

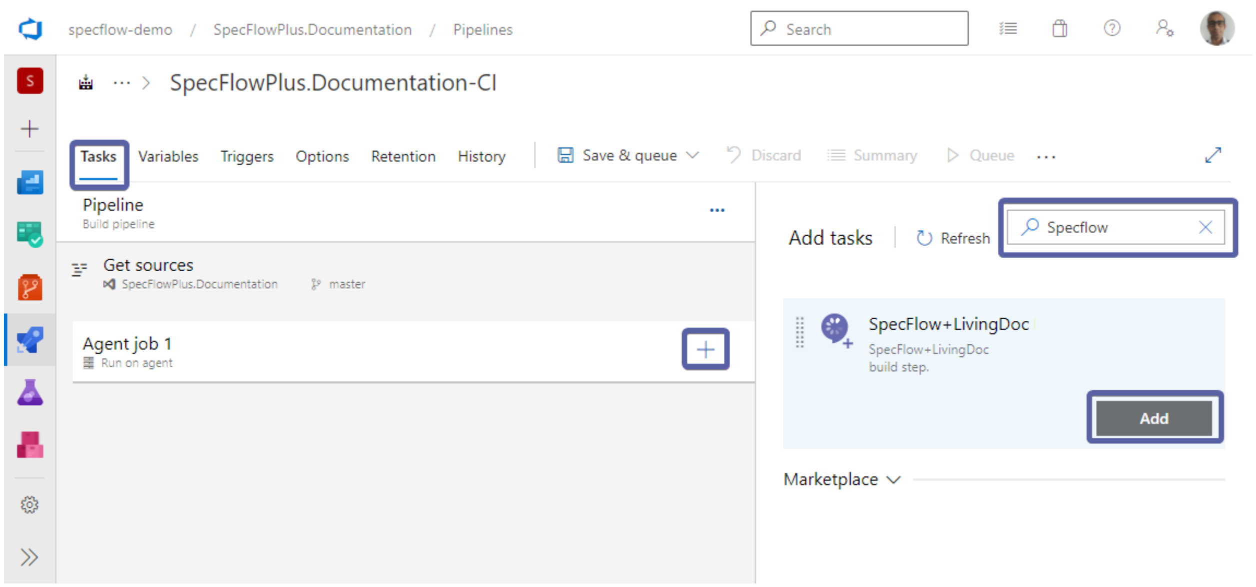
The next step is to add the build step to your pipelines. You have two options here, you can either use a YAML file to configure the build step or use the Azure DevOps interface. For the purpose of this guide, we will use the Azure DevOps interface as it is easier to use. If you like to use a YAML file you can find all the details *here*.

Before we move on, it is also important to note that the build step only generates the documentation; it does not execute any tests or build your solution.

1- Navigate to the desired pipeline from the list (or add a new one) and select Edit.


Pipelines
New pipeline
Recent
All
Runs
Filter pipelines
Recently run pipelines
Pipeline
Last run
LivingDoc for Documentat...
#1880 • Update All on same page.feature
Batched CI master
10m ago
31s
SpecFlowPlus.Test
#20200309.4 • Update azure-pipelines.yml for Az...
Manually triggered feature/pr-test
SpecFlowPlus.Demo-CI
#1868 • Add different image with same name
Manually triggered test
Git SpecFlowPlus.Test - LD...
#1866 • Updated Bookstore.feature
Manually triggered master-pr
14s
Edit
Run pipeline
Manage security
Rename/move
Delete

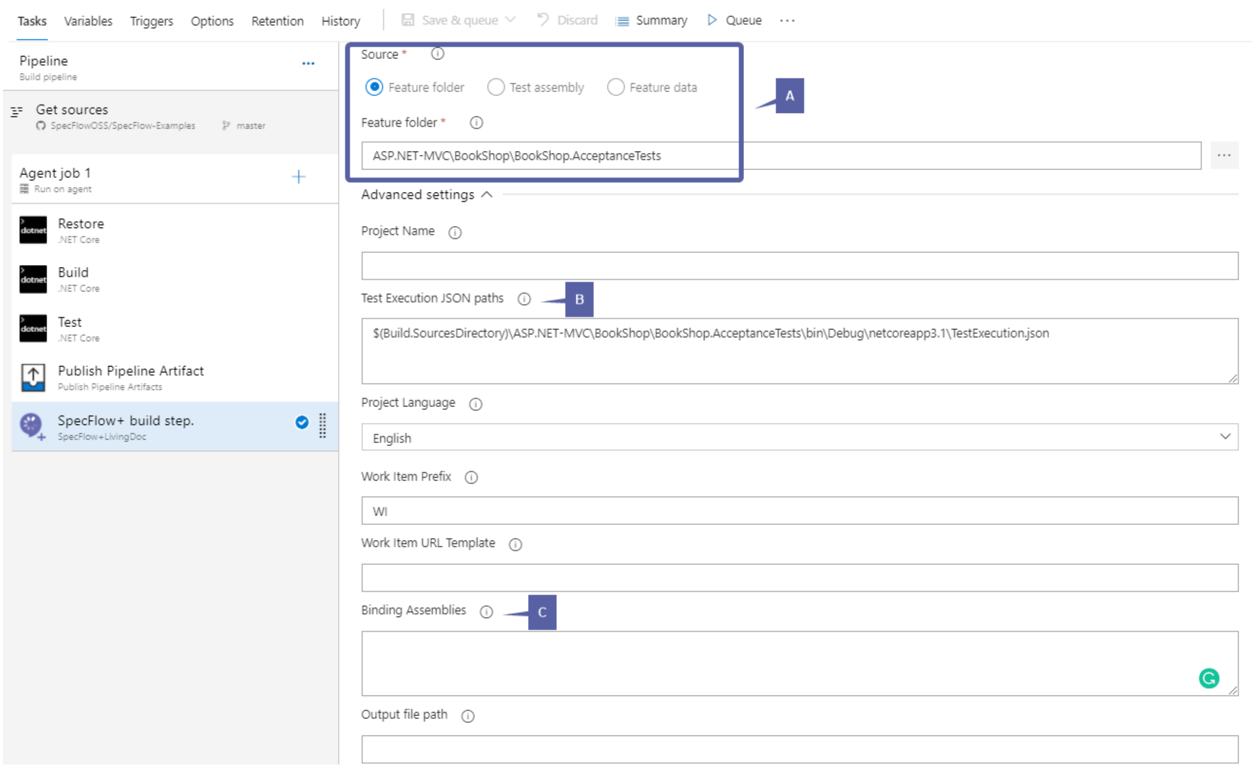
2- Add the SpecFlow+ build step as per below:



1.3 Step 3 - Configuration

The main point to keep in mind during the build step configuration is whether you want to generate LivingDoc **with** or **without** test execution results. This depends on what stage of development journey you are in. Remember you can always hide your test results using the toggle, see point 4 *here* for more info.

Let's take a quick look at the main configs:



Source * [✓ Feature folder] Test assembly Feature data

Feature folder * ASP.NET-MVC\BookShop\BookShop.AcceptanceTests

Advanced settings

Project Name

Test Execution JSON paths $(Build.SourcesDirectory)\ASP.NET-MVC\BookShop\BookShop.AcceptanceTests\bin\Debug\netcoreapp3.1\TestExecution.json

Project Language English

Work Item Prefix WI

Work Item URL Template

Binding Assemblies

Output file path

A - There are 3 sources to pick from to parse the documentation data:

Feature folder - The folder containing the feature files. Use the test project **root folder** if you want to **include** test execution results.

Test Assembly - Relative or absolute path to the test assembly. Glob patterns are supported e.g.: MyProject**\MyProject.dll

Feature Data - Relative or absolute path to the feature data JSON file. Glob patterns are supported e.g.: MyProject**\FeatureData.json

B - Enter the path to the generated JSON file by the SpecFlow.Plus.LivingDoc Plugin. This file carries all the necessary data needed for SpecFlow+LivingDoc to display your test results and is generated when you run your tests. Therefore, if you want to generate LivingDoc **with** test results, this step is mandatory.

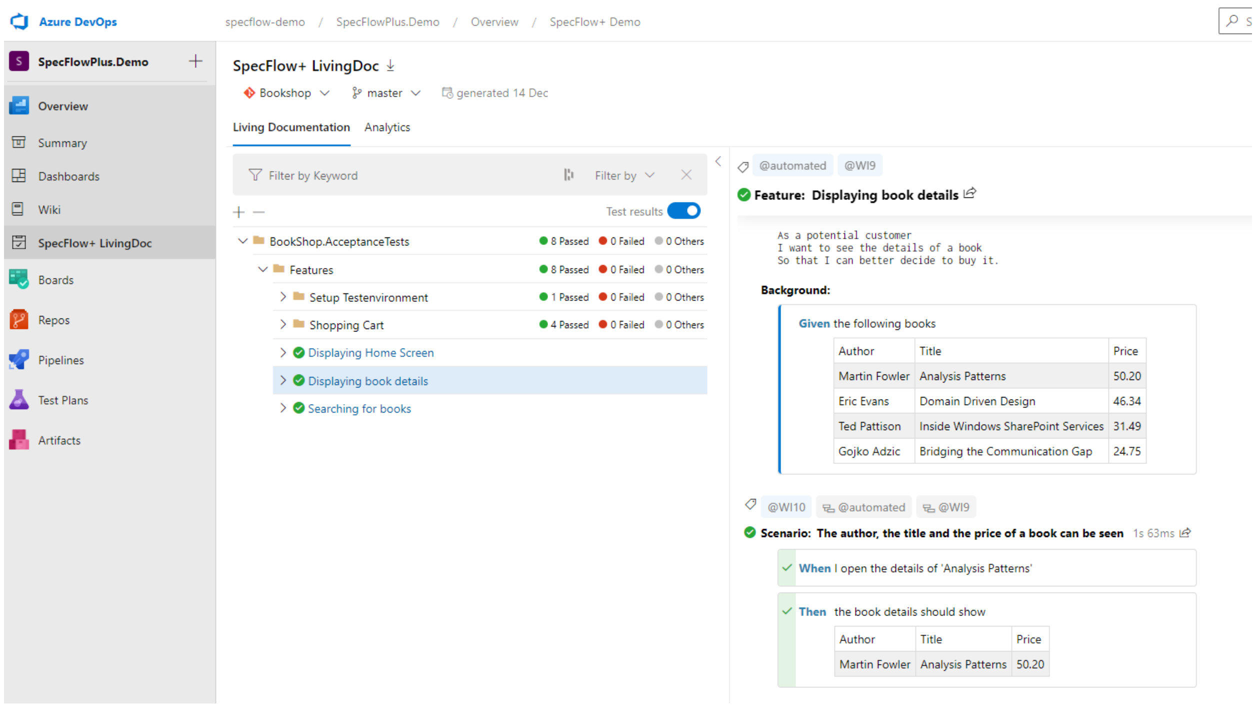
C - If you are interested to see the *Unused Step Definitions* in the analytics tab, enter your Binding Assembly paths (newline delimited) here. If your bindings are in the same assembly as your feature files you don't have to specify this field.

We have covered the basics here to get you going and generating your documentation. There are a number of other configurations here, you can find further details on them *here*.

1.4 Step 4 - Viewing LivingDoc

1- Once the configurations are done and the pipeline run is finished, simply navigate to the overview section of Azure DevOps as per step 1 and click on SpecFlow+LivingDoc.

2- You will find your LivingDoc here. Make sure you check out the *Viewing LivingDoc* & *Test results* sections to read about all the features and options you have here. The *Enhancing LivingDoc* section in docs also provides additional information on functionalities such as *Markdown and Embedding images*.



BookShop.AcceptanceTests ● 8 Passed ● 0 Failed ● 0 Others
Features ● 8 Passed ● 0 Failed ● 0 Others
Setup Testenvironment ● 1 Passed ● 0 Failed ● 0 Others
Shopping Cart ● 4 Passed ● 0 Failed ● 0 Others
Displaying Home Screen
Displaying book details
Searching for books

@automated @WI9

Feature: Displaying book details

```
As a potential customer
I want to see the details of a book
So that I can better decide to buy it.
```

Background:

Given the following books

Author	Title	Price
Martin Fowler	Analysis Patterns	50.20
Eric Evans	Domain Driven Design	46.34
Ted Pattison	Inside Windows SharePoint Services	31.49
Gojko Adzic	Bridging the Communication Gap	24.75

@WI10 @automated @WI9

Scenario: The author, the title and the price of a book can be seen 1s 63ms

When I open the details of 'Analysis Patterns'

Then the book details should show

Author	Title	Price
Martin Fowler	Analysis Patterns	50.20

CHAPTER 2

Step-by-step guide for LivingDoc Generator

The LivingDoc Generator enables you to generate living documentation in HTML format with no external dependencies.

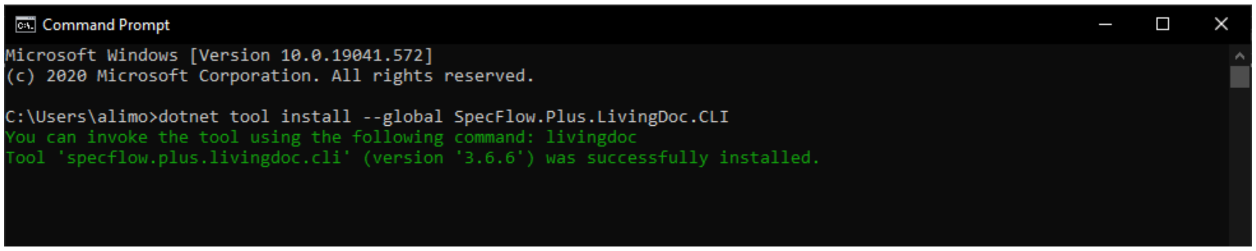
2.1 Step 1 - Installation

1- Setup the SpecFlow.Plus.LivingDoc Plugin for your SpecFlow project (if you have created the project using the SpecFlow Visual Studio project template, this dependency should already be there).

> Note: This plugin is only required if you want to generate living documentation ***with*** *test results, otherwise you can skip this plugin installation and only install the command line tool below. In this example we will be generating LivingDoc* ***with*** *test results.*

2- Next, open a command prompt and run the following command to install the CLI tool, this is a mandatory installation:

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

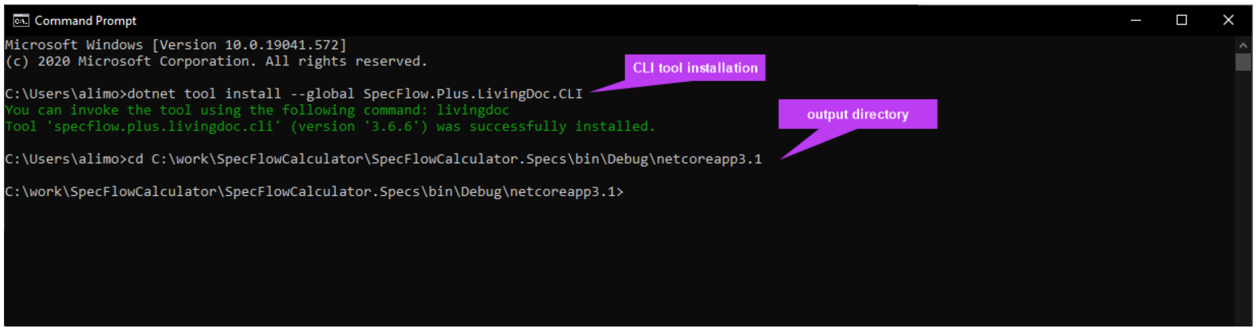


```
Microsoft Windows [Version 10.0.19041.572]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\alimo>dotnet tool install --global SpecFlow.Plus.LivingDoc.CLI
You can invoke the tool using the following command: livingdoc
Tool 'specflow.plus.livingdoc.cli' (version '3.6.6') was successfully installed.
```

2.2 Step 2 - Generate LivingDoc

1- Navigate to the path where your **SpecFlow project** is located. In this example, the solution was set up in the `C:\work` folder:

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

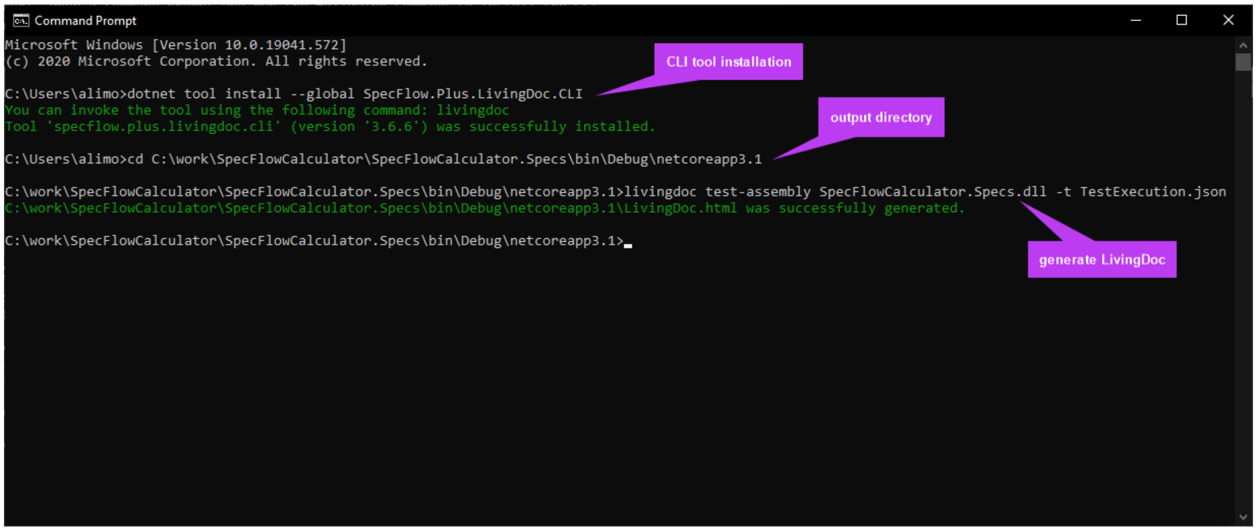


```
Microsoft Windows [Version 10.0.19041.572]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\alimo>dotnet tool install --global SpecFlow.Plus.LivingDoc.CLI
You can invoke the tool using the following command: livingdoc
Tool 'specflow.plus.livingdoc.cli' (version '3.6.6') was successfully installed.

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

C:\work\SpecFlowCalculator\SpecFlowCalculator.Specs\bin\Debug\netcoreapp3.1>
```

2- Now you can run the LivingDoc CLI by using the below command to generate the report.

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

> **Note:** As mentioned in step 1 the `TestExecution.json` file is generated by installing the SpecFlow.Plus.LivingDoc Plugin for your SpecFlow project and holds the tests results. By default, `TestExecution.json` will be generated in the test assembly folder of your project when you execute your tests e.g.: *SpecFlowCalculator.Specs\bin\Debug\netcoreapp3.1\TestExecution.json*.



```
Microsoft Windows [Version 10.0.19041.572]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\alimo>dotnet tool install --global SpecFlow.Plus.LivingDoc.CLI
You can invoke the tool using the following command: livingdoc
Tool 'specflow.plus.livingdoc.cli' (version '3.6.6') was successfully installed.

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

C:\work\SpecFlowCalculator\SpecFlowCalculator.Specs\bin\Debug\netcoreapp3.1>livingdoc test-assembly SpecFlowCalculator.Specs.dll -t TestExecution.json
C:\work\SpecFlowCalculator\SpecFlowCalculator.Specs\bin\Debug\netcoreapp3.1\LivingDoc.html was successfully generated.

C:\work\SpecFlowCalculator\SpecFlowCalculator.Specs\bin\Debug\netcoreapp3.1>
```

> **Note:** We are generating LivingDoc **with** test results here, hence why we are using the *test-assembly* command. Removing `-t TestExecution.json` part of the command will allow you to generate LivingDoc **without** test results. Check *feature-data* and *feature-folder* for alternative commands.

2.3 Step 3 - Viewing LivingDoc

1- The command-line tool will generate an HTML file titled `LivingDoc.html` in the same folder as the **output directory of the SpecFlow project**. You can manually navigate to this folder and open this file in your favorite browser or use the command-line tool to do it:

```
  C:\work\SpecFlowCalculator\SpecFlowCalculator.Specs\bin\Debug\netcoreapp3.
↪1\LivingDoc.html
```

> **Note:** Check here for more info on how to change the output directory of LivingDoc CLI tool.

Make sure you check out the *Viewing LivingDoc* & *Test results* sections to read about all the features and options you have here.

Here is sample LivingDoc file:

2.3.1 Sharing LivingDoc

One of the key usage scenarios of LivingDoc is to help teams collaborate and form a shared understanding in their development journey. Therefore, sharing the generated LivingDoc file plays an important role. The generated LivingDoc can easily be shared in numerous ways depending on your team's current infrastructure setup and needs.

Please visit the *Sharing & Publishing* page for some ideas and options on how to share LivingDoc.

CHAPTER 3

Generating LivingDoc for Azure DevOps

In order to generate living documentation from your feature files using Specflow in Azure DevOps, you need to:

1 - Install the Azure DevOps Extension.

2 - Define a build step that references your project.

3 - Queue the build.

4 - *View* your documentation.

The configurations in each of these steps will help you truly customize Livingdoc to bring real value to you and your team.

CHAPTER 4

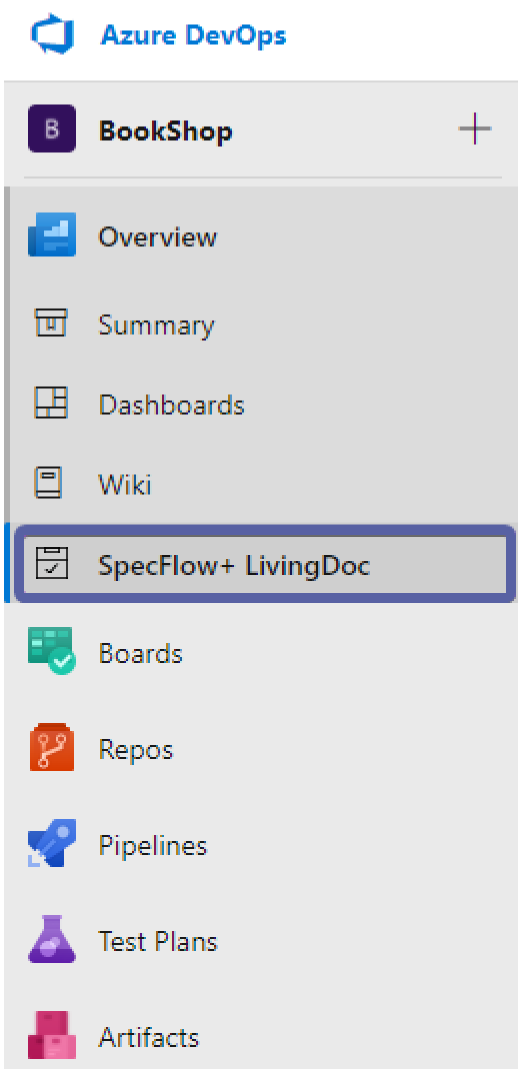
Azure DevOps extension installation

1- Visit the SpecFlow+ LivingDoc page on the Visual Studio marketplace.

2- Click on the **Get it free** button and proceed with the installation as you would with any other extension.

*Refer to Microsoft documentation if you are having issues installing an extension:*Installing Azure DevOps ExtensionsInstalling Azure DevOps Server Extensions

3- You should now see the "SpecFlow+ LivingDoc" menu item under "Overview" in each of your projects:


Azure DevOps
BookShop
Overview
Summary
Dashboards
Wiki
SpecFlow+ LivingDoc
Boards
Repos
Pipelines
Test Plans
Artifacts

CHAPTER 5

Adding a Build Step

Generating living documentation from your Gherkin files with SpecFlow+ LivingDoc requires you to add the SpecFlow+ build step to your build process. This build step parses the Gherkin files in your solution and formats them for display in DevOps/VSTS/TFS.

The build step requires .NET Core 3.1 installed on the build agent. If you use Microsoft hosted agents it is automatically available. Alternatively you can use the Use .NET Core task to install the .NET Core version.

Please note the build step only generates the documentation; it does not execute any tests or build your solution.

There are two ways to configure the build step, see the appropriate chapter depending on the type of build:

- Build step configured in DevOps/TFS/VSTS: See Configuring-the-Build-Step-in-DevOps
- YAML build step: See Configuring the Build Step in YAML

Note: You do not need to use DevOps/VSTS/TFS to actually build your application. You can simply add a build definition that acquires the sources and generates the documentation.

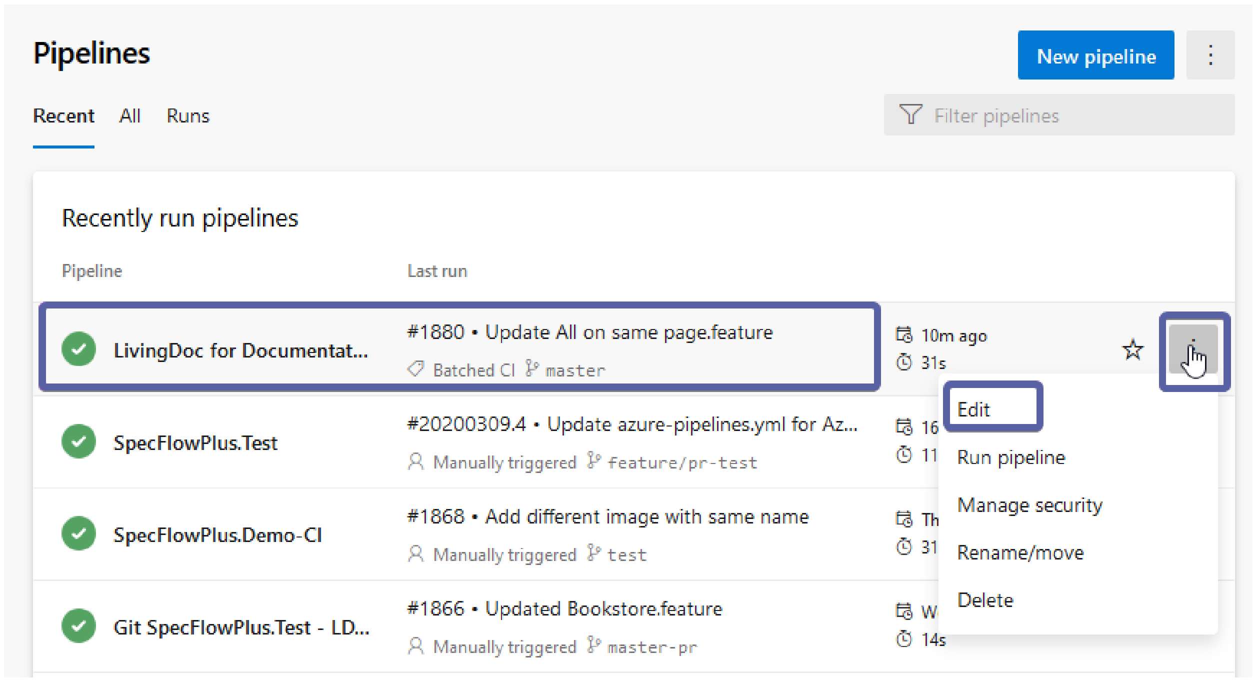
CHAPTER 6

Configuring the Build Step in DevOps

To add the build step in Azure DevOps:

1 - Select **Pipelines | Pipelines** from the menu in Azure DevOps.

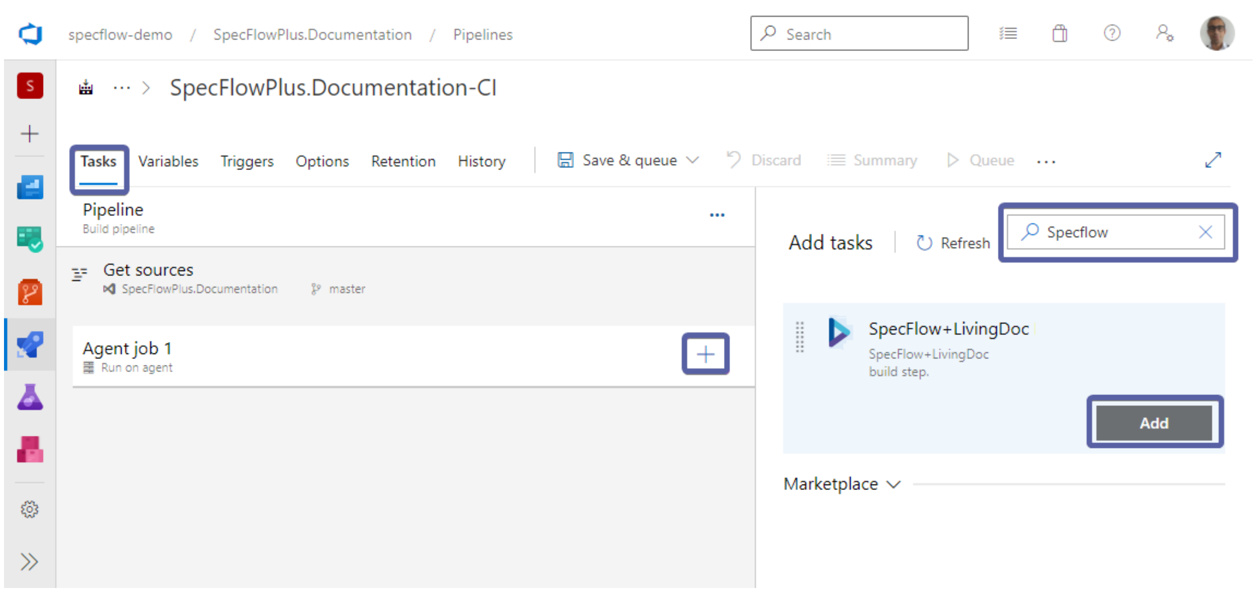
2 - Locate the desired pipeline from the list (or add a new one) and select **Edit**.


Pipelines
New pipeline
Recent
All
Runs
Filter pipelines
Recently run pipelines
Pipeline
Last run
LivingDoc for Documentat...
#1880 • Update All on same page.feature
Batched CI master
10m ago
31s
SpecFlowPlus.Test
#20200309.4 • Update azure-pipelines.yml for Az...
Manually triggered feature/pr-test
SpecFlowPlus.Demo-CI
#1868 • Add different image with same name
Manually triggered test
Git SpecFlowPlus.Test - LD...
#1866 • Updated Bookstore.feature
Manually triggered master-pr
14s
Edit
Run pipeline
Manage security
Rename/move
Delete

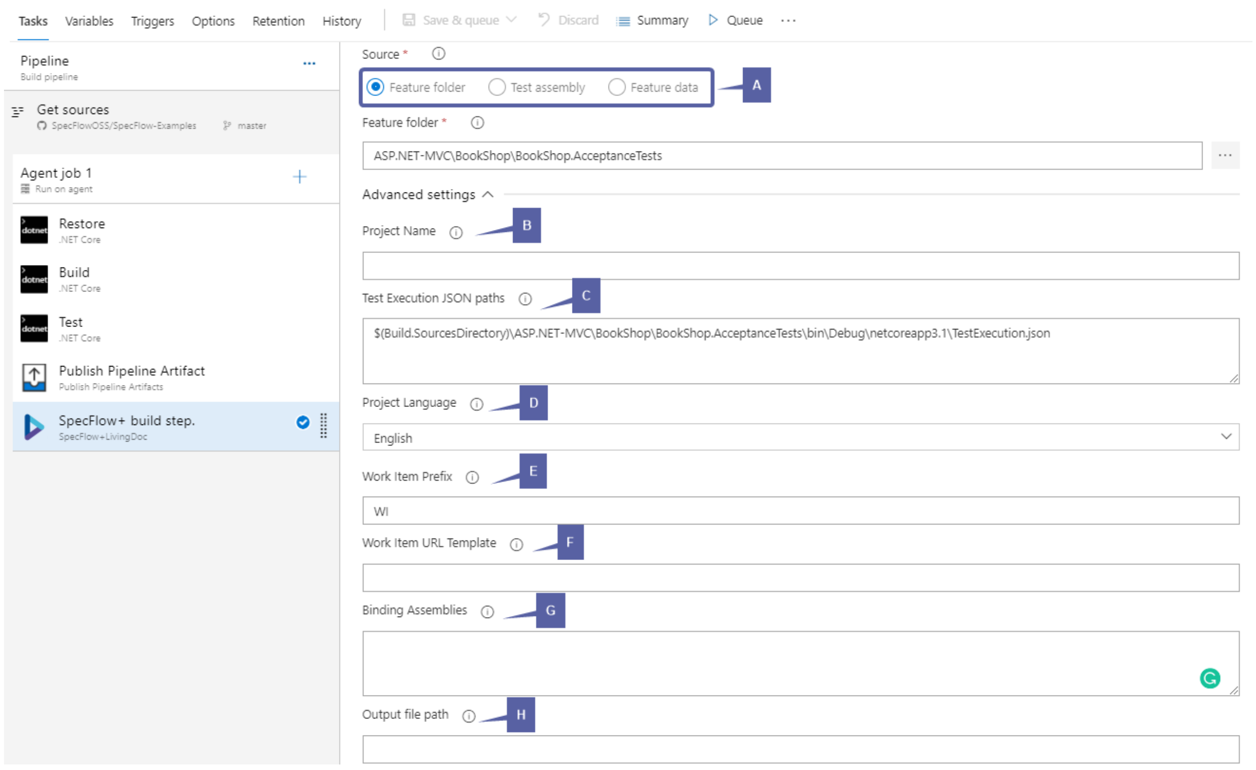
3 - The current tasks are displayed under the **Tasks** tab

4 - Click on the plus icon next to your agent to add a new build step.

5 - Look for "Specflow" in the search bar and add the SpecFlow+ LivingDoc build step to your build:



Once you add the SpecFlow + LivingDoc build step you will be prompted to setup the build configuration:



Source * [✓ Feature folder] Test assembly Feature data

Feature folder *: ASP.NET-MVC\BookShop\BookShop.AcceptanceTests

Advanced settings

Project Name:

Test Execution JSON paths: $(Build.SourcesDirectory)\ASP.NET-MVC\BookShop\BookShop.AcceptanceTests\bin\Debug\netcoreapp3.1\TestExecution.json

Project Language: English

Work Item Prefix: WI

Work Item URL Template:

Binding Assemblies:

Output file path:

A - There are 3 sources to pick from to parse the documentation data:

Feature folder - The folder containing the feature files. Use the test project **root folder** if you want to **include** test execution results.

> ***Note:*** *If you pick "Feature Folder" as your source, you will be able to edit feature files* directly within AzDo.

Test Assembly - Relative or absolute path to the test assembly. Glob patterns are supported e.g.: `MyProject/**/MyProject.dll`

Feature Data - Relative or absolute path to the feature data JSON file. Glob patterns are supported e.g.: `MyProject/`

`**/FeatureData.json`

B - Enter a **Project Name**. This name is used by the root node in the tree. If you do not enter a name here, the name of the Visual Studio project is used by default.

C - If you want to include test execution results in the report, you must specify the **Test Execution JSON paths**. It has to be an absolute or relative path to the Test Execution JSON files generated by the SpecFlow.Plus.LivingDocPlugin. **Glob patterns are only supported when you use a relative path** e.g.: `MyProject/**/TestExecution.json`

Important: If you pick **Feature folder** as your source and want to display the execution results, you **MUST** provide the **Test Project Root folder**.

> Note: If you have *multiple JSON files*, i.e. when you run tests in parallel, you may add them all here. To do so, add the path to each JSON file in a separate line (new line delimited).

D - Select the language used by your Gherkin files under **Project Language**. This is optional because in most cases the language can be auto- detected. You can read more about language options here.

E - If you have added *links* to Azure DevOps work items in your feature files using tags, enter the prefix used to identify the work items here.

For example, if you enter "DEVOPS_WI:" as the work item prefix and define the tag "@DEVOPS_WI:1234" in your feature file, the tag will link to work item #1234 when displayed in LivingDoc.

F - Enter the URL to the specified work item here. Example : https://dev.azure.com/fabrikam/FabrikamProj/_workitems/edit/{id}

G - If you are interested to see the *Unused Step Definitions* in the analytics tab, enter your Binding Assembly paths (newline delimited) here. If your bindings are in the same assembly as your feature files you don't have to specify this field.

Glob patters are supported e.g.: `MyProject/**/MyBindings.dll`.

H - You can configure the output path of LivingDoc here.

Once you have defined all the required fields in your build step, you can queue the build.

> *Note: If you want to include Gherkin files from multiple projects, add a separate build step for each of your projects.*

CHAPTER 7

Configuring the Build Step in YAML

Learn how to configure the build step in a YAML file.

Before you proceed, note that YAML is whitespace sensitive. Please copy and paste the example in a text editor with syntax highlighting (e.g. Notepad++).

For information on the YAML schema, see the Microsoft reference guide for Azure Pipelines YAML schema.

7.1 SpecFlow+LivingDoc custom build step YAML snippet

```yaml
- task: SpecFlowPlus@0
  inputs:
    #generatorSource: 'FeatureFolder' # Required. Options: FeatureFolder, ␣
↪TestAssembly, FeatureData
    #projectFilePath:  # Required when generatorSource == FeatureFolder
    #testAssemblyFilePath:  # Required when generatorSource == TestAssembly
    #featureDataJsonFilePath:  # Required when generatorSource == FeatureData
    #projectName: # Optional
    #testExecutionJson: # Optional
    #projectLanguage: 'en' # Optional
    #workItemPrefix: # Optional
    #workItemUrlTemplate: # Optional
    #bindingAssemblies: # Optional
    #output: # Optional
```

7.2 Arguments

7.3 Non-LivingDoc specific parameters

- **enabled:** boolean (not needed when true)

- **continueOnError:** boolean (not needed when false)
- **condition:**
- **timeoutInMinutes:** Specifies the maximum time, in minutes, that a task is allowed to execute before being canceled by server (zero value indicates an infinite timeout)

7.4 Examples

7.4.1 FeatureFolder example

```
- task: SpecFlowPlus@0
  displayName: 'LivingDoc with FeatureFolder generatorSource'
  inputs:
    generatorSource: 'FeatureFolder'
    projectFilePath: 'BookShop.AcceptanceTests'
    projectName: 'testName'
    testExecutionJson: 'BookShop.AcceptanceTests/**/TestExecution.json'
    projectLanguage: 'en'
    workItemPrefix: 'WI'
    workItemUrlTemplate: 'https://dev.azure.com/specflow/BookShop/_workitems/edit/{id}
↪'
    bindingAssemblies: |
        BookShop.AcceptanceTests/**/BookShop.AcceptanceTests.dll
        BookShop.AcceptanceTests/**/MyBindings.dll
  enabled: false
  continueOnError: true
  condition: always()
  timeoutInMinutes: 10
```

7.4.2 TestAssembly example

```
- task: SpecFlowPlus@0
  displayName: 'LivingDoc with TestAssembly generatorSource'
  inputs:
    generatorSource: 'TestAssembly'
    testAssemblyFilePath: 'BookShop.AcceptanceTests/bin/**/BookShop.AcceptanceTests.
↪dll'
    projectName: 'testName'
    testExecutionJson: 'BookShop.AcceptanceTests/**/TestExecution.json'
    projectLanguage: 'en'
    workItemPrefix: 'WI'
    workItemUrlTemplate: 'https://dev.azure.com/specflow/BookShop/_workitems/edit/{id}
↪'
    bindingAssemblies: 'BookShop.AcceptanceTests/**/MyBindings.dll'
  enabled: false
  continueOnError: true
  condition: always()
  timeoutInMinutes: 10
```

7.4.3 FeatureData example

```
- task: SpecFlowPlus@0
  displayName: 'LivingDoc with FeatureData generatorSource'
  inputs:
    generatorSource: 'FeatureData'
    featureDataJsonFilePath: './**/FeatureData.json'
    testExecutionJson: 'BookShop.AcceptanceTests/**/TestExecution.json'
    workItemPrefix: 'WI'
    workItemUrlTemplate: 'https://dev.azure.com/specflow/BookShop/_workitems/edit/{id}
↪'
    bindingAssemblies: |
        BookShop.AcceptanceTests/**/BookShop.AcceptanceTests.dll
        BookShop.AcceptanceTests/**/MyBindings.dll
  enabled: false
  continueOnError: true
  condition: always()
  timeoutInMinutes: 10
```

7.4.4 Example of `output`

Specify output file name as `MyFeatureData.json` and place it in the `LivingDoc` directory. If the directory is not exists, the tool will create it.

```
- task: SpecFlowPlus@0
  displayName: 'LivingDoc with custom output'
  inputs:
    generatorSource: 'FeatureFolder'
    projectFilePath: 'BookShop.AcceptanceTests'
    projectName: 'testName'
    testExecutionJson: 'BookShop.AcceptanceTests/**/TestExecution.json'
    projectLanguage: 'en'
    workItemPrefix: 'WI'
    workItemUrlTemplate: 'https://dev.azure.com/specflow/BookShop/_workitems/edit/{id}
↪'
    bindingAssemblies: |
        BookShop.AcceptanceTests/**/BookShop.AcceptanceTests.dll
        BookShop.AcceptanceTests/**/MyBindings.dll
    output: 'LivingDoc/MyFeatureData.json'
  enabled: false
  continueOnError: true
  condition: always()
  timeoutInMinutes: 10
```

CHAPTER 8

Generating LivingDoc using CLI

SpecFlow+LivingDoc Generator allows you to generate a local or self-hosted HTML of your Gherkin Feature Files without the need of Azure DevOps. You also have the option to generate documentation **with or without test results**.

8.1 Generation excluding test results

In order to generate your living documentation without test execution results, you need to:

1. Install the command line tool
2. Execute the command line tool to generate the documentation.

8.2 Generation including test results

In order to generate your living documentation with test execution results, you need to:

1. Setup the SpecFlow.Plus.LivingDocPlugin for your SpecFlow project (if you have created the project using the SpecFlow Visual Studio project template, this dependency should already be there).
2. Build your project.
3. Run your tests. This will generate a **TestExecution.json** next to your TestAssembly which is needed in the next step.
4. Install the command line tool
5. Execute the command line tool and make sure to pass the test execution JSON files to the CLI.

CHAPTER 9

LivingDoc Plugin Setup

If you want to generate LicingDoc **with** test results you must install this plugin, otherwise, you may skip this step to generate LivingDoc **without** test results.

9.1 Installation

Add the SpecFlow.Plus.LivingDocPlugin NuGet package to your SpecFlow project.

9.2 Configuration

LivingDoc Generator configuration options have a default setting. Simple SpecFlow projects may not require any further configuration.

9.3 `livingDocGenerator`

Use this section to extend your `specflow.json` with LivingDoc Generator configuration.

9.3.1 Placeholder support in `filePath`

The following placeholders are supported inside the `filePath`:

- `{CurrentDirectory}` the current working directory. **Note:** Add this placeholder if the `TestExecution.json` file is not generated to the output directory. This is required in case of xUnit targeting full framework.
- `{ProcessId}` the Id of the current process
- `{ThreadId}` the Id of the current thread

- `{Now}` the timestamp with the current date and time. The default format is `yyyyMMddhhmmss`, but it can be overwritten with format strings. e.g: `{Now:yyyy-MM-dd}` For an overview of the available format strings, see Standard Date and Time Format Strings and Custom Date and Time Format Strings in the official Microsoft documentation.

These placeholders can be used to make the generated file name unique when the tests are running in parallel and the test runner supports process level isolation like the SpecFlow+Runner. All placeholders are case insensitive, so they can be used with different casing. e.g: `{ThreadId}` or `{threadid}`.

9.3.2 Examples

simple specflow.json example:

```
{
  "livingDocGenerator": {
    "enabled": true,
    "filePath": "TestExecution.json"
  }
}
```

specflow.json example with CurrentDirectory placeholder:

```
{
  "livingDocGenerator": {
    "enabled": true,
    "filePath": "{CurrentDirectory}\\TestExecution.json"
  }
}
```

specflow.json example for SpecFlow+Runner using Process isolation:

```
{
  "livingDocGenerator": {
    "enabled": true,
    "filePath": "TestExecution_{ProcessId}_{ThreadId}_{Now}.json"
  }
}
```

CHAPTER 10

Installing the CLI tool

10.1 Prerequisites

SpecFlow.Plus.LivingDoc.CLI requires the .NET Core SDK 3.1 or higher to be installed. Information on setting up the .NET Core SDK can be found in the official Microsoft guide.

10.2 Installing SpecFlow.Plus.LivingDoc.CLI

To install SpecFlow.Plus.LivingDoc.CLI:

1. Open a command prompt
2. Run the following command:

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

10.3 Updating SpecFlow.Plus.LivingDoc.CLI

To update SpecFlow.Plus.LivingDoc.CLI:

1. Open a command prompt.
2. Run the following command:

```
dotnet tool update --global SpecFlow.Plus.LivingDoc.CLI
```

10.4 Uninstalling SpecFlow.Plus.LivingDoc.CLI

To uninstall SpecFlow.Plus.LivingDoc.CLI:

1. Open a command prompt.
2. Run the following command:

```
dotnet tool uninstall --global SpecFlow.Plus.LivingDoc.CLI
```

Once you have installed the command line tool, please read the Using the command line tool page to learn how to use it.

CHAPTER 11

Using the CLI tool

There are three ways to execute and configure LivingDoc commands depending on where you have stored your feature file data:

11.1 Commands

- Feature-folder : Generates living documentation from feature files from the file system.
- Test-assembly : Generates living documentation from a compiled SpecFlow test assembly.
- Feature-data : Generates living documentation from pre-parsed features stored in a feature data JSON file.

11.2 Synopsis

```
livingdoc -h|--help
livingdoc --version

livingdoc <COMMAND> [-h|--help] [command-options] [arguments]
```

11.3 Generating with test results

*Note : You must have the LivingDoc Plugin Setup to generate LivingDoc **with** test results.*

Here are the list of commands to use to quickly generate LivingDoc **with test results** from different sources which you have stored your feature file data:

Generate the Living Documentation from SpecFlow test assembly:

```
livingdoc test-assembly C:\Work\MyProject.Specs\bin\Debug\netcoreapp3.1\MyProject.
↪Specs.dll -t C:\Work\MyProject.Specs\bin\debug\netcoreapp3.1\TestExecution.json
```

Generate the Living Documentation from feature files:

```
livingdoc feature-folder C:\Work\MyProject.Specs -t C:\Work\MyProject.
↪Specs\bin\debug\netcoreapp3.1\TestExecution.json
```

11.4 Generating without test results

Here are the list of commands to use to quickly generate LivingDoc **without test results** from different sources which you have stored your feature file data:

Generate the Living Documentation from SpecFlow test assembly:

```
livingdoc test-assembly C:\Work\MyProject.Specs\bin\Debug\netcoreapp3.1\MyProject.
↪Specs.dll
```

Generate the Living Documentation from feature files:

```
livingdoc feature-folder C:\Work\MyProject.Specs
```

11.5 Simple Options

There are a number of options with SpecFlow+LivingDoc command-line tool depending on where you have stored your feature file data. Please check the relevant page for each method for more information.

Here are a few simple and handy ones:

Generate the Living Documentation with a custom title/header:

```
livingdoc feature-folder C:\Work\MyProject.Specs --title "BookShop"
```

Generate the Living Documentation for a specific output path:

```
livingdoc feature-folder C:\Work\MyProject.Specs --output C:\Temp\MyReport.html
```

> ***NOTE:*** If the directory given in the "–output" path does not already exist, LivingDoc CLI will automatically create the output directories in the given output path.

Generate the Living Documentation with work item prefix and work item URL template:

```
livingdoc feature-folder C:\Work\MyProject.Specs --work-item-prefix WI --work-item-
↪url-template https://dev.azure.com/specflow/BookShop/_backlogs/backlog/BookShop
↪%20Team/Stories/?workitem={id}
```

CHAPTER 12

livingdoc feature-folder

`livingdoc feature-folder` - Generates living documentation from feature files from the file system.

12.1 Synopsis

```
livingdoc feature-folder
    [--project-language <projectLanguage>]
    [--test-execution-json <testExecutionJson>]
    [--title <title>]
    [--project-name <project-name>]
    [--work-item-url-template <workItemUrlTemplate>]
    [--work-item-prefix <workItemPrefix>]
    [--binding-assemblies <bindingAssembly1 [bindingAssembly2 ...]>]
    [--output <output>]
    [--output-type <HTML|JSON>]
  <featureFolder>
```

12.2 Arguments

`<featureFolder>`

Relative (from the working directory) or absolute path of the root folder containing the feature files. **If the path to the file contains a space, make sure to enclose it in quotes.**

*Note: If you also want to see test results in your living documentation, you **MUST** provide the Test Project Root folder and not a subfolder.*

12.3 Options

- `--project-language <projectLanguage>`

The language used in your feature files. See Gherkin languages.

- `-t|--test-execution-json <testExecutionJson>`

  Relative (from the working directory) or absolute path of the test execution JSON files generated by the SpecFlow.Plus.LivingDocPlugin during test execution.

  NOTE: If you used the SpecFlow+ Runner with its parallel execution features you may end up with multiple JSON files.You can include multiple JSON files separated by a space in the command-line tool as per below

- `-t|--test-execution-json <testExecutionJson_1 testExecutionJson_2 ... >`

  OR

  Use a wild card (*) to include them all:

- `-t|--test-execution-json <testExecutionJson*>`

  Check this guide for further info on this subject.

- `--binding-assemblies <bindingAssembly1 [bindingAssembly2 ...]>`

  Relative (from the working directory) or absolute path of the SpecFlow binding assemblies separated by space. The <*TestAssembly*> does not have to be specified as a binding assembly because it is always scanned for SpecFlow bindings.

- `--output <output>`

  Relative (from the working directory) or absolute path to the generated output file. Default value: `LivingDoc.html`

  NOTE: If the directory given in the "–output" path does not already exist, LivingDoc CLI will automatically create the output directories in the given output path.

  The last element in a path ending in a slash or backslash is treated as a directory (not a file name),if a directory with the same name does not exist.For example, **report** in –output ./out/out2/**report/** is treated as a directory if it does not exist yet. However, if a directory named **report** does exist in the path, then **report** is treated as a directory regardless of the path ending in a slash/backslash or not.

- `--output-type <HTML|JSON>`

  The type of output file to be generated. The `HTML` output is the living documentation report. The `JSON` output contains the pre-parsed feature data optionally extended with further information (test execution results, binding information, etc.). Default value: `HTML`

- `--work-item-prefix <workItemPrefix>`

  The name of the special tag you mark the scenarios with to link them to the corresponding work items.

- `--work-item-url-template <workItemTemplate>`

  The URL template to use to generate the external links. e.g: `https://dev.azure.com/fabrikam/FabrikamProj/_workitems/edit/{id}`

- `--title <title>`

  The title/header of the generated document. Default value: The root folder node's name.

- `--project-name <project-name>`

  This name is used by the root node in the tree. Default value: The name of the folder containing the feature files.

CHAPTER 13

livingdoc test-assembly

`livingdoc test-assembly` - Generates living documentation from a compiled SpecFlow test assembly.

13.1 Synopsis

```
livingdoc test-assembly
    [--project-language <projectLanguage>]
    [-t|--test-execution-json <testExecutionJson>]
    [--title <title>]
    [--project-name <project-name>]
    [--work-item-url-template <workItemUrlTemplate>]
    [--work-item-prefix <workItemPrefix>]
    [--binding-assemblies <bindingAssembly1 [bindingAssembly2 ...]>]
    [-o|--output <output>]
    [--output-type <HTML|JSON>]
  <testAssembly>
```

13.2 Arguments

`<testAssembly>`

Relative (from the working directory) or absolute path of the SpecFlow test assembly.

***If the path to the file contains a space, make sure to enclose it in quotes.**

13.3 Options

- `--project-language <projectLanguage>`

  The language used in your feature files. See Gherkin languages.

- `-t|--test-execution-json <testExecutionJson>`

  Relative (from the working directory) or absolute path of the test execution JSON files generated by the SpecFlow.Plus.LivingDocPlugin during test execution.

  NOTE: If you used the SpecFlow+ Runner with its parallel execution features you may end up with multiple JSON files.You can include multiple JSON files separated by a space in the command-line tool as per below

- `-t|--test-execution-json <testExecutionJson_1 testExecutionJson_2 ... >`

  OR

  Use a wild card (*) to include them all:

- `-t|--test-execution-json <testExecutionJson*>`

  Check this guide for further info on this subject.

- `--binding-assemblies <bindingAssembly1 [bindingAssembly2 ...]>`

  Relative (from the working directory) or absolute path of the SpecFlow binding assemblies separated by space. The <*TestAssembly*> does not have to be specified as a binding assembly because it is always scanned for SpecFlow bindings.

- `-o|--output <output>`

  Relative (from the working directory) or absolute path to the generated output file. Default value: `LivingDoc.html`

  NOTE: If the directory given in the "–output" path does not already exist, LivingDoc CLI will automatically create the output directories in the given output path.

  The last element in a path ending in a slash or backslash is treated as a directory (not a file name),if a directory with the same name does not exist.For example, **report** in –output ./out/out2/**report/** is treated as a directory if it does not exist yet. However, if a directory named **report** does exist in the path, then **report** is treated as a directory regardless of the path ending in a slash/backslash or not.

- `--output-type <HTML|JSON>`

  The type of output file to be generated. The `HTML` output is the living documentation report. The `JSON` output contains the pre-parsed feature data optionally extended with further information (test execution results, binding information, etc.). Default value: `HTML`

- `--work-item-prefix <workItemPrefix>`

  The name of the special tag you mark the scenarios with to link them to the corresponding work items.

- `--work-item-url-template <workItemTemplate>`

  The URL template to use to generate the external links. e.g: `https://dev.azure.com/fabrikam/FabrikamProj/_workitems/edit/{id}`

- `--title <title>`

  The title/header of the generated document. Default value: The root folder node's name.

- `--project-name <project-name>`

  This name is used by the root node in the tree. Default value: The name of the SpecFlow test assembly.

CHAPTER 14

livingdoc feature-data

`livingdoc feature-data` - Generates living documentation from pre-parsed features stored in a Feature Data JSON file.

14.1 Synopsis

```
livingdoc feature-data
    [-t|--test-execution-json <testExecutionJson>]
    [--title <title>]
    [--work-item-url-template <workItemUrlTemplate>]
    [--work-item-prefix <workItemPrefix>]
    [--binding-assemblies <bindingAssembly1 [bindingAssembly2 ...]>]
    [-o|--output <output>]
    [--output-type <HTML|JSON>]
  <featureDataJson>
```

14.2 Arguments

`<featureDataJson>`

Relative (from the working directory) or absolute path of the Feature Data JSON. **If the path to the file contains a space, make sure to enclose it in quotes.**

The Feature Data JSON is **different** from the TestExecution.json generated by the SpecFlow.Plus.LivingDocPlugin. You can create a Feature Data JSON with running the LivingDoc CLI and using the `--output-type JSON` option. This can be helpful if you have a multiple step pipeline where you want to first generate the the Feature Data JSON and in a later step further enhance it with a TestExecution.json.

14.3 Options

- `-t|--test-execution-json <testExecutionJson>`

  Relative (from the working directory) or absolute path of the test execution JSON files generated by the SpecFlow.Plus.LivingDocPlugin during test execution.

  NOTE: If you used the SpecFlow+ Runner with its parallel execution features you may end up with multiple JSON files.You can include multiple JSON files separated by a space in the command-line tool as per below:

- `-t|--test-execution-json <testExecutionJson_1 testExecutionJson_2 ... >`

  OR

  Use a wild card (*) to include them all:

- `-t|--test-execution-json <testExecutionJson*>`

  Check this guide for further info on this subject.

- `--binding-assemblies <bindingAssembly1 [bindingAssembly2 ...]>`

  Relative (from the working directory) or absolute path of the SpecFlow binding assemblies separated by space. The <*TestAssembly*> does not have to be specified as a binding assembly because it is always scanned for SpecFlow bindings.

- `-o|--output <output>`

  Relative (from the working directory) or absolute path to the generated output file. Default value: `LivingDoc.html`

  NOTE: If the directory given in the "–output" path does not already exist, LivingDoc CLI will automatically create the output directories in the given output path.

  The last element in a path ending in a slash or backslash is treated as a directory (not a file name),if a directory with the same name does not exist.For example, **report** in –output ./out/out2/**report/** is treated as a directory if it does not exist yet. However, if a directory named **report** does exist in the path, then **report** is treated as a directory regardless of the path ending in a slash/backslash or not.

- `--output-type <HTML|JSON>`

  The type of output file to be generated. The `HTML` output is the living documentation report. The `JSON` output contains the pre-parsed feature data optionally extended with further information (test execution results, binding information, etc.). Default value: `HTML`

- `--work-item-prefix <workItemPrefix>`

  The name of the special tag you mark the scenarios with to link them to the corresponding work items.

- `--work-item-url-template <workItemTemplate>`

  The URL template to use to generate the external links. e.g: `https://dev.azure.com/fabrikam/FabrikamProj/_workitems/edit/{id}`

- `--title <title>`

  The title/header of the generated document. Default value: The root folder node's name.

CHAPTER 15

Azure DevOps Extension

Once you have *generated your documentation* using the *SpecFlow+ build step*

1 - Select **Overview |SpecFlow+** from the Azure DevOps menu to view the documentation.

2 - Use the drop-down at the top of the page to select your repository.

3 - Use the drop-down next to it to select the branch. You can also choose pull requests from the branch selector. Once you have selected a build, the date of the build is displayed at the top.

Note: If the date shown here is older than your last build, this may indicate that the SpecFlow+ build step failed. This can happen if the build task fails to update the cache. If this occurs, manually queue a new build, which should refresh the cache.

4 - The "Test results" toggle displayed here gives you the option to hide or show the test execution results in LivingDoc.

5 - The feature explorer depicts the structure of your specifications. Blue text entries represent features and scenarios while black text entries are based on the file structure of your project.

6 - A summary of the execution results are display for every folder in the feature tree. There are three different statuses defined here; Passed,Failed, and Other, read more about them *here*.

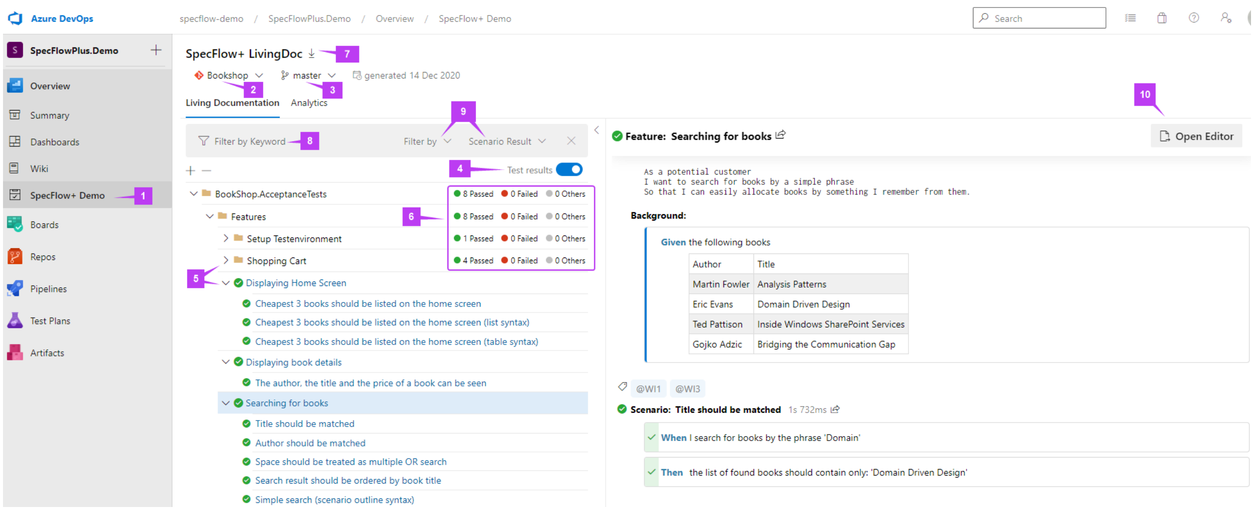
7 - You can download your Living Documentation from Azure DevOps as a stand-alone HTML file to share it with team members who do not use AzureDevOps.

Note: The downloaded Living Documentation will not have a Repository selector, nor a Branch selector. It will contain the data from your selected Repository and Branch.

8 - This is the keyword lookup function, which allows you to search for keywords in LivingDoc, for more details *check out the dedicated page* for this function.

9 - Filters to add to the keyword lookup function.

10 - This open editor button allows you directly edit your feature files from within the AzureDevOps envirnoment.



BookShop.AcceptanceTests	8 Passed	0 Failed	0 Others
Features	8 Passed	0 Failed	0 Others
Setup Testenvironment	1 Passed	0 Failed	0 Others
Shopping Cart	4 Passed	0 Failed	0 Others

- Displaying Home Screen
  - Cheapest 3 books should be listed on the home screen
  - Cheapest 3 books should be listed on the home screen (list syntax)
  - Cheapest 3 books should be listed on the home screen (table syntax)
- Displaying book details
  - The author, the title and the price of a book can be seen
- Searching for books
  - Title should be matched
  - Author should be matched
  - Space should be treated as multiple OR search
  - Search result should be ordered by book title
  - Simple search (scenario outline syntax)

Feature: Searching for books

```
As a potential customer
I want to search for books by a simple phrase
So that I can easily allocate books by something I remember from them.
```

Background:

Given the following books

Author	Title
Martin Fowler	Analysis Patterns
Eric Evans	Domain Driven Design
Ted Pattison	Inside Windows SharePoint Services
Gojko Adzic	Bridging the Communication Gap

@WI1 @WI3

Scenario: Title should be matched 1s 732ms

✓ **When** I search for books by the phrase 'Domain'

✓ **Then** the list of found books should contain only: 'Domain Driven Design'

> *Note: Code comments are intentionally not displayed in LivingDoc to avoid polluting the document.*

Click on a scenario or feature (blue text) in the feature explorer to display the documentation generated for that scenario or feature on the right column, this is the test execution results page, *read more about it here*.

CHAPTER 16

Generator - CLI Tool

Once you have *generated your living documentation* file using SpecFlow+LivingDoc command-line tool, you will see a new file labelled **LivingDoc.html** in your chosen output directory.

Open the HTML file using your favorite browser:

1 - This is the keyword lookup function which allows you to search for keywords in LivingDoc, for more details *check out the dedicated page* for this function.

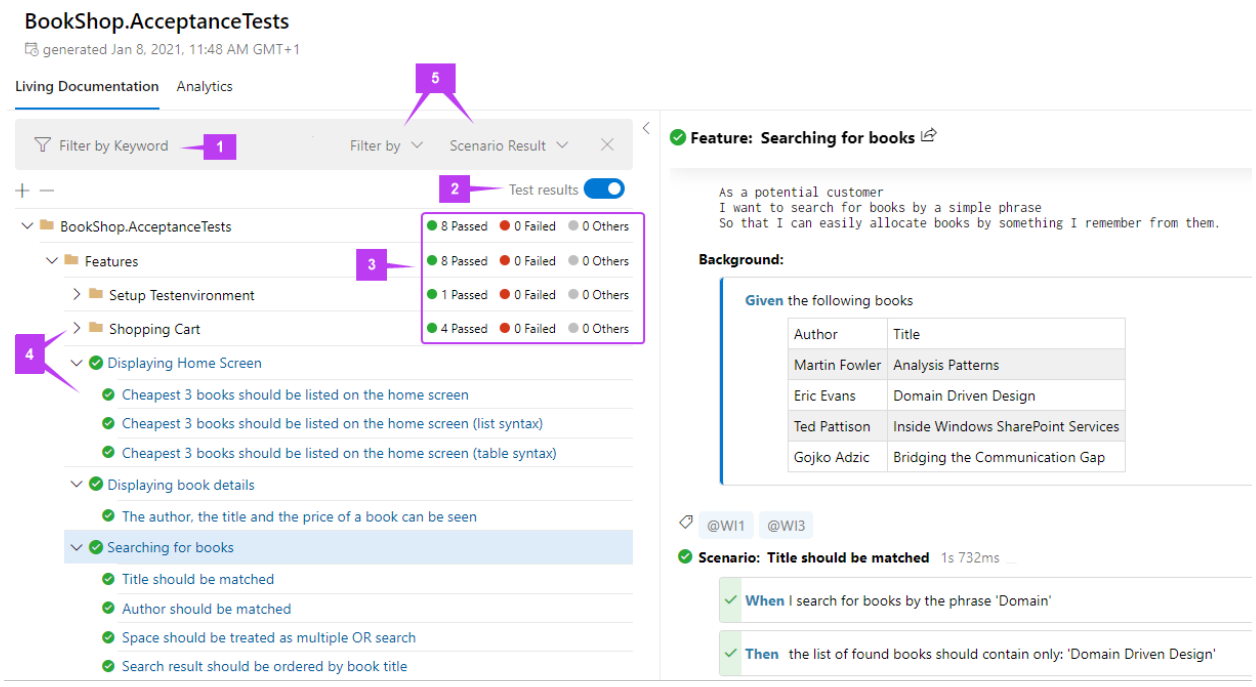
2 - The "Test results" toggle displayed here gives you the option to hide or show the test execution results in LivingDoc.

3 - A summary of the execution results are display for every folder in the feature tree. There are three different statuses defined here; Passed,Failed, and Other, read more about them *here*.

4 - The feature explorer depicts the structure of your specifications. Blue text entries represent features and scenarios while black text entries are based on the file structure of your project.

Click on a scenario or feature (blue text) in the feature explorer to display the documentation generated for that scenario or feature on the right column, this is the test execution page, *read more about it here*.

5 - These are filter options to add to the keyword lookup function.



BookShop.AcceptanceTests
generated Jan 8, 2021, 11:48 AM GMT+1

BookShop.AcceptanceTests	● 8 Passed	● 0 Failed	● 0 Others
Features	● 8 Passed	● 0 Failed	● 0 Others
Setup Testenvironment	● 1 Passed	● 0 Failed	● 0 Others
Shopping Cart	● 4 Passed	● 0 Failed	● 0 Others

- ✓ Displaying Home Screen
  - ✓ Cheapest 3 books should be listed on the home screen
  - ✓ Cheapest 3 books should be listed on the home screen (list syntax)
  - ✓ Cheapest 3 books should be listed on the home screen (table syntax)
- ✓ Displaying book details
  - ✓ The author, the title and the price of a book can be seen
- ✓ Searching for books
  - ✓ Title should be matched
  - ✓ Author should be matched
  - ✓ Space should be treated as multiple OR search
  - ✓ Search result should be ordered by book title

✓ **Feature: Searching for books**

```
As a potential customer
I want to search for books by a simple phrase
So that I can easily allocate books by something I remember from them.
```

Background:

Given the following books

Author	Title
Martin Fowler	Analysis Patterns
Eric Evans	Domain Driven Design
Ted Pattison	Inside Windows SharePoint Services
Gojko Adzic	Bridging the Communication Gap

@WI1 @WI3

✓ **Scenario: Title should be matched** 1s 732ms

✓ **When** I search for books by the phrase 'Domain'

✓ **Then** the list of found books should contain only: 'Domain Driven Design'

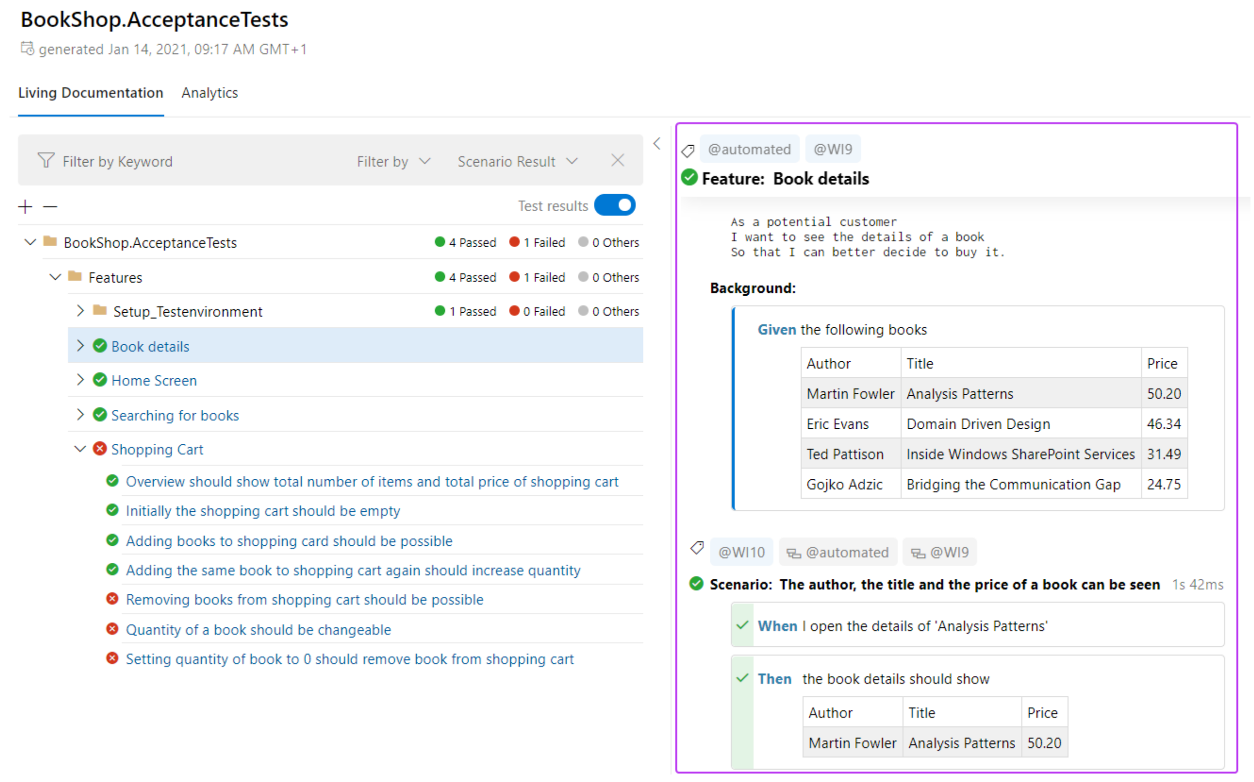
> *Note: Code comments are intentionally not displayed in LivingDoc to avoid polluting the document.*

Click on a scenario or feature (blue text) in the feature explorer to display the documentation generated for that scenario or feature on the right column, this is the test execution results page, *read more about it here*.

CHAPTER 17

Test results

By clicking on a scenario or feature (blue text) in the feature explorer, the right column of LivingDoc updates with the test execution results of that scenario or feature.



BookShop.AcceptanceTests

generated Jan 14, 2021, 09:17 AM GMT+1

- BookShop.AcceptanceTests ● 4 Passed ● 1 Failed ● 0 Others
  - Features ● 4 Passed ● 1 Failed ● 0 Others
    - Setup_Testenvironment ● 1 Passed ● 0 Failed ● 0 Others
    - ✓ Book details
    - ✓ Home Screen
    - ✓ Searching for books
    - ✗ Shopping Cart
      - ✓ Overview should show total number of items and total price of shopping cart
      - ✓ Initially the shopping cart should be empty
      - ✓ Adding books to shopping card should be possible
      - ✓ Adding the same book to shopping cart again should increase quantity
      - ✗ Removing books from shopping cart should be possible
      - ✗ Quantity of a book should be changeable
      - ✗ Setting quantity of book to 0 should remove book from shopping cart

@automated @WI9

✓ **Feature: Book details**

```
As a potential customer
I want to see the details of a book
So that I can better decide to buy it.
```

Background:

Given the following books

Author	Title	Price
Martin Fowler	Analysis Patterns	50.20
Eric Evans	Domain Driven Design	46.34
Ted Pattison	Inside Windows SharePoint Services	31.49
Gojko Adzic	Bridging the Communication Gap	24.75

@WI10 @automated @WI9

✓ **Scenario: The author, the title and the price of a book can be seen** 1s 42ms

✓ **When** I open the details of 'Analysis Patterns'

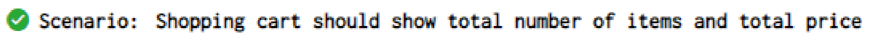
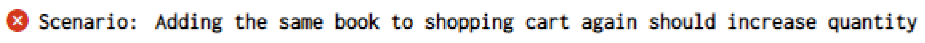
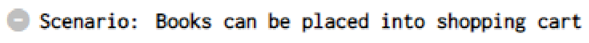
✓ **Then** the book details should show

Author	Title	Price
Martin Fowler	Analysis Patterns	50.20

Here is a detailed overview of all the features here:

17.1 Test execution status

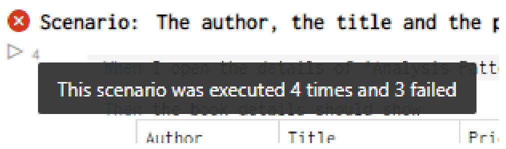
The state of test execution is defined as per following:

- **Passed** 
  Scenario: Shopping cart should show total number of items and total price
- **Failed** 
  Scenario: Adding the same book to shopping cart again should increase quantity
- **Other** 
  Scenario: Books can be placed into shopping cart
  - Not executed
  - Skipped
  - A step binding is missing
  - A step was marked as Pending

17.2 Viewing multiple execution results

When a scenario or a scenario outline example row has been executed multiple times during a test execution the following indicator is shown:

The details of the executions can be seen by hovering over the triangle icon:

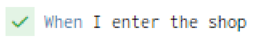
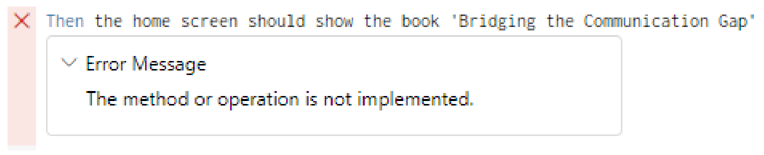
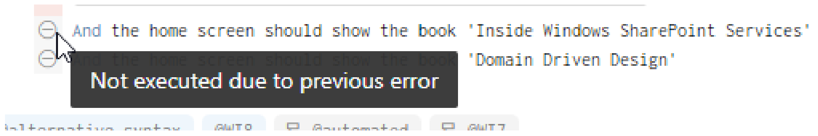


Scenario: The author, the title and the

This scenario was executed 4 times and 3 failed

17.3 Step execution status

The state of step execution can be the following:

- Passed 
  When I enter the shop
- Failed 
  Then the home screen should show the book 'Bridging the Communication Gap'
  Error Message
  The method or operation is not implemented.
- Other 
  And the home screen should show the book 'Inside Windows SharePoint Services'
  'Domain Driven Design'
  Not executed due to previous error
  - Not executed (the Scenario was not selected for execution)
  - Not executed due to previous error
  - No step definition found

  - Pending step definition
  - Step level execution result is not available (an old version (<v3.4.133) of the LivingDocPlugin is used for the generation)

17.4 Test result aggregation

Individual test execution results are available on different levels. This can be Steps, Scenarios, examples of Scenario Outlines and Features. The aggregation to each higher level is done by the following logic:

- Passed: all executions have at least one "Passed" or were "Others"
- Failed: at least one execution has "Failed"
- Others: all executions have the state "Others"

17.5 Previewing Scenarios Outlines with data values

Gherkin scenarios often use tables to store a series of test values that are referenced using placeholders in the Gherkin steps.

LivingDoc allows you to easily toggle between these values to easily view the results and puts them in an easy to understand table format:

Note the the two placeholders in the below example are < books > and < search phrase >

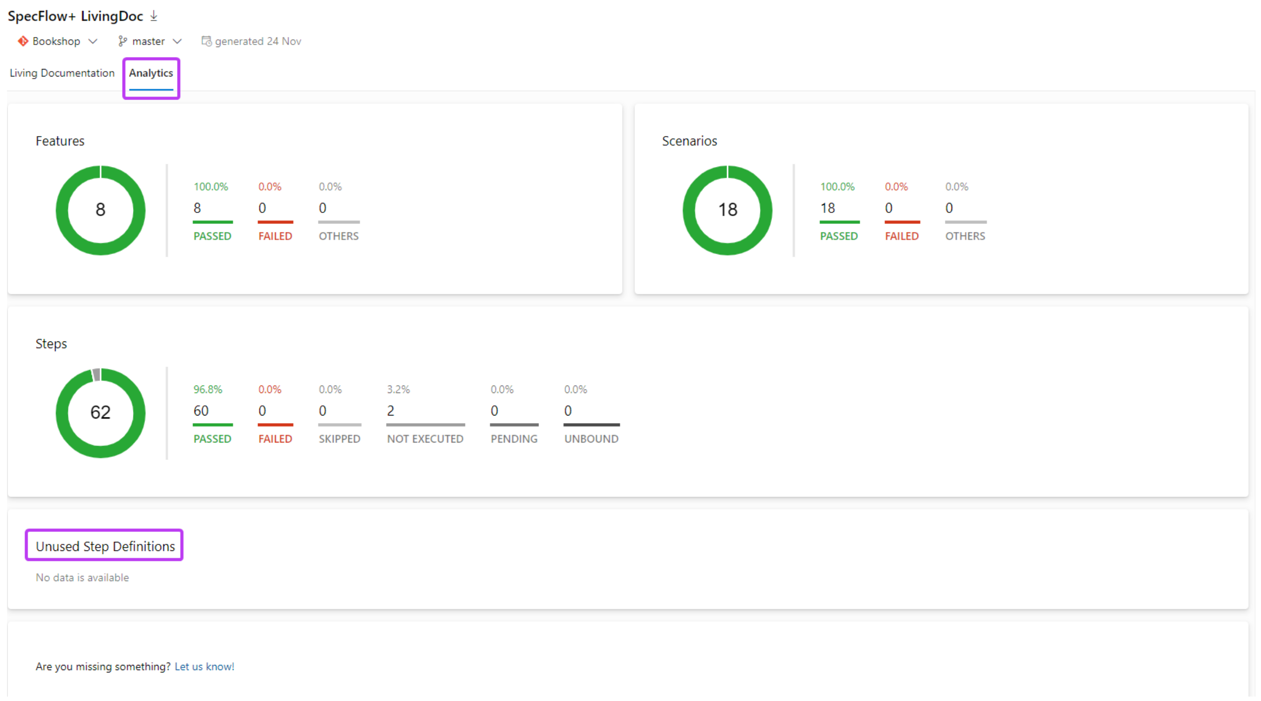
17.6 Other tools and options

Now that you have understood how to read your test-execution results, make sure to check the **Enhancement in LivingDoc** section for a complete list of features and options in LivingDoc.

CHAPTER 18

Analytics

The second tab on SpecFlow's LivingDoc gives you an overall overview of all execution results in Features, Scenarios and steps. This is available both in the LivingDoc Azure DevOps extension and the LivingDoc Generator:



This tab also shows you any **Unused Step Definitions**, please check the *relevant documentaion* for this section.

CHAPTER 19

Unused Step Definitions

Use the CLI tool to generate LivingDoc report which also includes Unused step Definition report under the analytics tab. Scope handling included.

You can use this report to find unused code in the automation layer, as it will list the unused bindings from your test assembly/binding assemblies.

Steps appearing under the Unused Step Definitions section are steps that exist in the automation layer but are not used in any feature files.

19.1 Usage

Add the optional `--binding-assemblies` option to the command line tool to specify the assemblies you want to scan. Glob patters are supported e.g.: `MyProject/**/MyBindings.dll`.

If you have both the feature files and step definitions in the same test assembly, you don't have to explicitly specify the `--binding-assemblies` option. See an *example* below.

If a Step Definition appears under the Unused Step Definitions, you should check if all of the Step Definition Attributes are used. Any unused Step Definition Attribute will be reported as Unused Step Definition. Example:

```
Feature: Calculator

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
```

```
[When("the two numbers are added")]
[When("we sum the two numbers")]
public void WhenTheTwoNumbersAreAdded()
{
```

(continues on next page)

(continued from previous page)

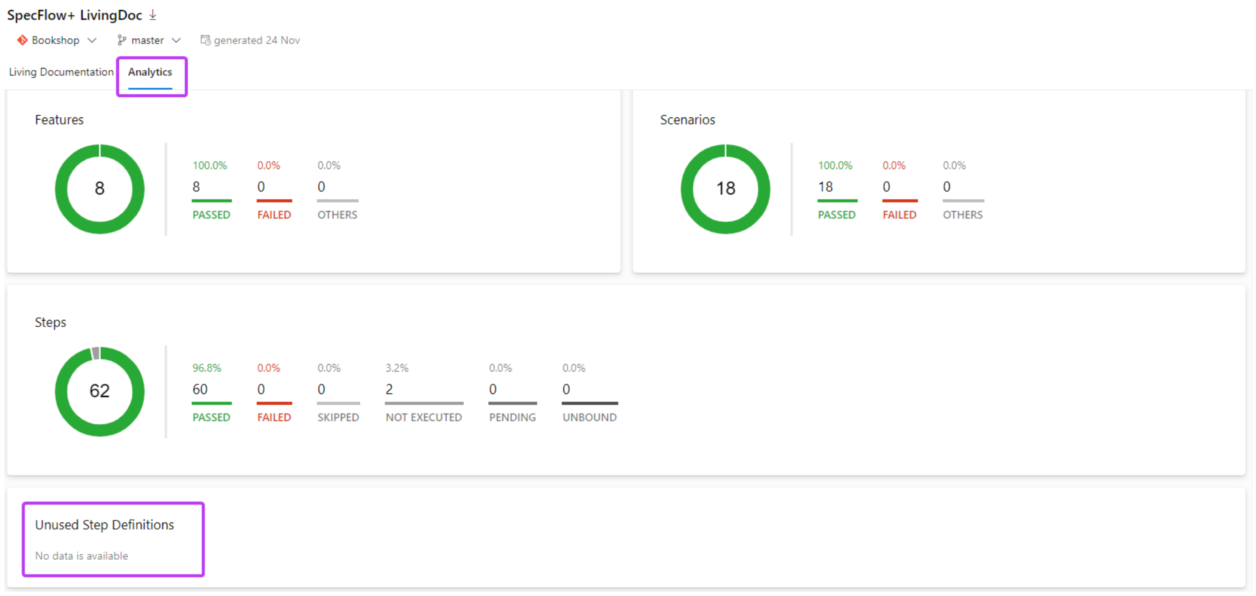
```
    // ...
}
```

The Step Definition Attribute with Regex `we sum the two numbers` will be reported as Unused Step Definition, because it is not used in any of the Feature files.

19.2 Viewing Unused Step Definition Report

You can find it on the Analytics tab of the Living Documentation under the Unused Step Definitions section.

- No data found for the report



Features

8

100.0%	0.0%	0.0%
8	0	0
PASSED	FAILED	OTHERS

Scenarios

18

100.0%	0.0%	0.0%
18	0	0
PASSED	FAILED	OTHERS

Steps

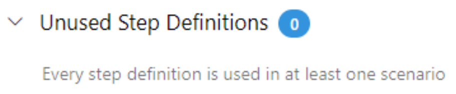
62

96.8%	0.0%	0.0%	3.2%	0.0%	0.0%
60	0	0	2	0	0
PASSED	FAILED	SKIPPED	NOT EXECUTED	PENDING	UNBOUND

Unused Step Definitions

No data is available

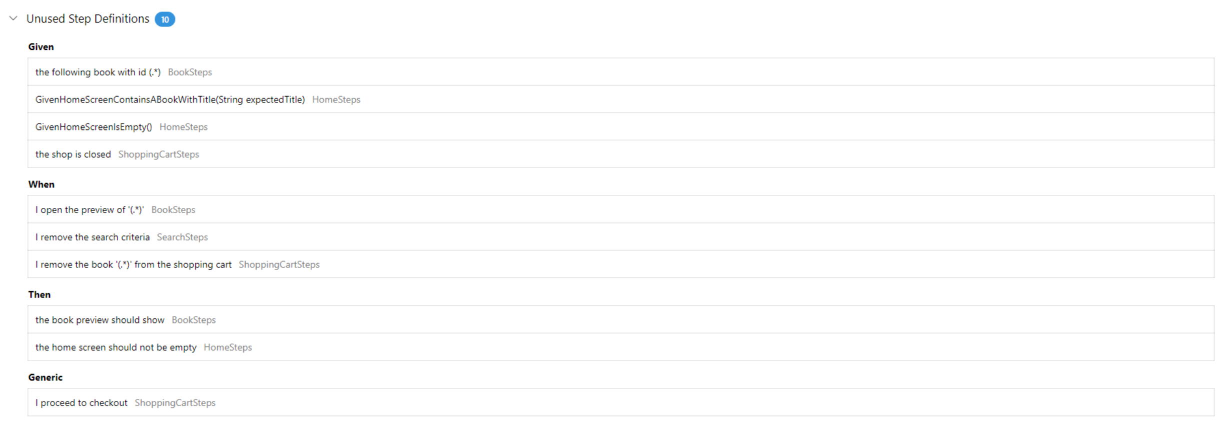
- No unused step definitions - All step definitions are used



Unused Step Definitions 0

Every step definition is used in at least one scenario

- Some step definitions are not used



Unused Step Definitions 10

Given

the following book with id (.*) BookSteps

GivenHomeScreenContainsABookWithTitle(String expectedTitle) HomeSteps

GivenHomeScreenIsEmpty() HomeSteps

the shop is closed ShoppingCartSteps

When

I open the preview of '(.*)' BookSteps

I remove the search criteria SearchSteps

I remove the book '(.*)' from the shopping cart ShoppingCartSteps

Then

the book preview should show BookSteps

the home screen should not be empty HomeSteps

Generic

I proceed to checkout ShoppingCartSteps

19.3 Examples

19.3.1 feature-folder command

- Collect Unused Step Definitions from a binding assembly using `feature-folder` command:

```
livingdoc feature-folder C:/Work/MyProject/Features --binding-assemblies "C:/Work/
↪MyProject/bin/Debug/MyBindings.dll"
```

- Collect Unused Step Definitions from **multiple** binding assemblies using `feature-folder` command:

```
livingdoc feature-folder C:/Work/MyProject/Features --binding-assemblies "C:/Work/
↪MyProject/bin/Debug/MyBindings.dll" "C:/Work/MyProject/bin/Debug/
↪MyStepDefinitions.dll"
```

19.3.2 feature-data command

- Collect Unused Step Definitions from a binding assembly using `feature-data` command:

```
livingdoc feature-data C:/Work/FeatureData.json --binding-assemblies "C:/Work/
↪MyProject/bin/Debug/MyBindings.dll"
```

- Collect Unused Step Definitions from **multiple** binding assemblies using `feature-data` command:

```
livingdoc feature-data C:/Work/FeatureData.json --binding-assemblies "C:/Work/
↪MyProject/bin/Debug/MyBindings.dll" "C:/Work/MyProject/bin/Debug/
↪MyStepDefinitions.dll"
```

19.3.3 test-assembly command

- Collect Unused Step Definitions from the test assembly using the `test-assembly` command:

```
livingdoc test-assembly C:/Work/MyProject/MyAssembly.dll
```

In this case the `MyAssembly.dll` will be scanned **automatically** to find Unused Step Definitions. In other words, you don't need to specify `--binding-assemblies "C:/Work/MyProject/MyAssembly.dll"` explicitly.

- Collect Unused Step Definitions from the test assembly and from a binding assembly using the `test-assembly` command:

```
livingdoc test-assembly C:/Work/MyProject/MyAssembly.dll --binding-assemblies "C:/
↪Work/MyProject/bin/Debug/MyBindings.dll"
```

- Collect Unused Step Definitions from the test assembly and from binding assemblies using the `test-assembly` command:

```
livingdoc test-assembly C:/Work/MyProject/MyAssembly.dll --binding-assemblies "C:/
↪Work/MyProject/bin/Debug/MyBindings.dll" "C:/Work/MyProject/bin/Debug/
↪MyStepDefinitions.dll"
```

In this case 3 assemblies will be scanned:

  - `MyAssembly.dll`
  - `MyBindings.dll`
  - `MyStepDefinitions.dll`

19.4 Limitations

Currently only Regular expressions in attributes is supported. Learn more about Step Matching Styles.

CHAPTER 20

Finding keywords in LivingDoc

The number of scenarios in a project can grow very quickly, so finding the data you are looking for is important. SpecFlow+ LivingDoc allows you to find keywords in folder names, tags, titles (features, scenarios, scenario outlines), descriptions and steps.

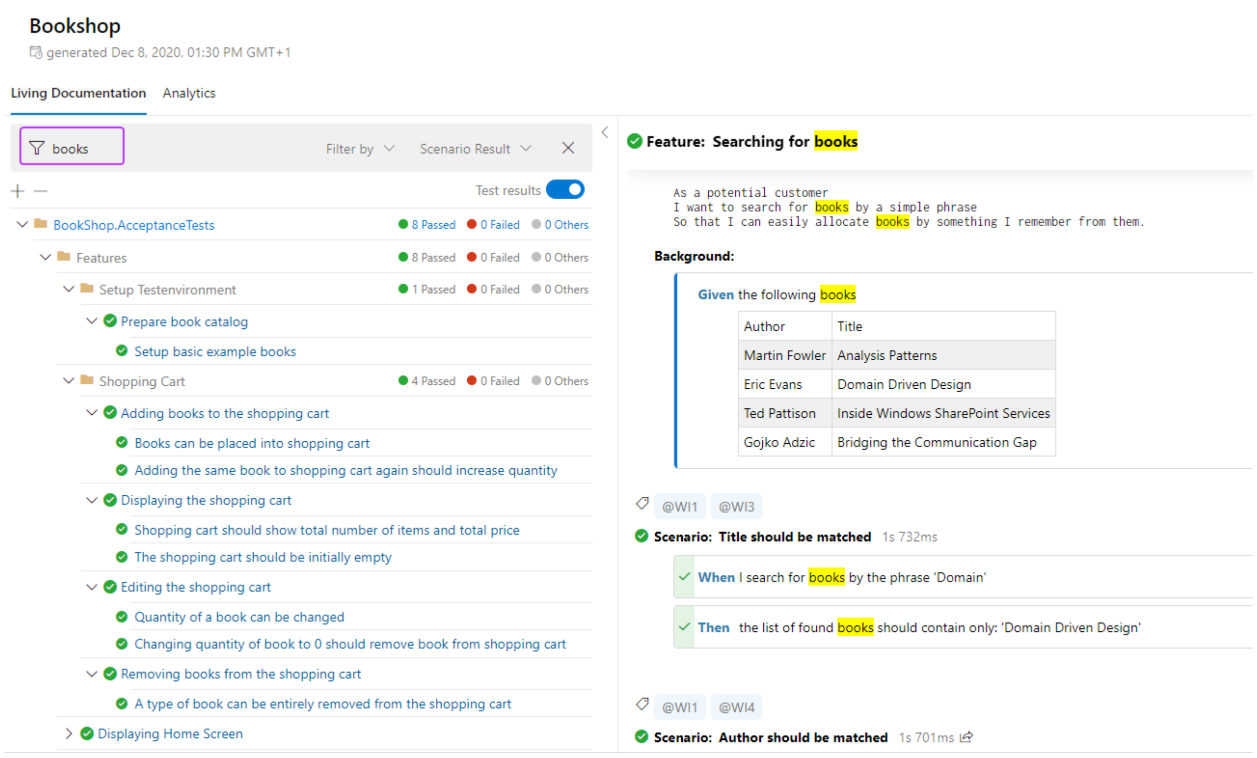
20.1 Finding a keyword in the entire documentation

To find a keyword within your entire documentation:

1 - Enter your keyword in the **Filter** field. As you type, the results are updated in the feature explorer.

2 - Folders, features and scenarios containing the keyword in the titles, tags, descriptions or steps are displayed in blue. Node titles displayed in grey do not contain the keyword, but at least one child or parent item is containing the keyword. All other nodes (i.e. with no hits) are hidden in the tree.

3 - Click on an entry in the tree to display that entry. The keyword is highlighted yellow in the content.



Bookshop

generated Dec 8, 2020, 01:30 PM GMT+1

Living Documentation   Analytics

books   Filter by   Scenario Result

Test results

- BookShop.AcceptanceTests — ● 8 Passed ● 0 Failed ● 0 Others
  - Features — ● 8 Passed ● 0 Failed ● 0 Others
    - Setup Testenvironment — ● 1 Passed ● 0 Failed ● 0 Others
      - ✓ Prepare book catalog
        - ✓ Setup basic example books
    - Shopping Cart — ● 4 Passed ● 0 Failed ● 0 Others
      - ✓ Adding books to the shopping cart
        - ✓ Books can be placed into shopping cart
        - ✓ Adding the same book to shopping cart again should increase quantity
      - ✓ Displaying the shopping cart
        - ✓ Shopping cart should show total number of items and total price
        - ✓ The shopping cart should be initially empty
      - ✓ Editing the shopping cart
        - ✓ Quantity of a book can be changed
        - ✓ Changing quantity of book to 0 should remove book from shopping cart
      - ✓ Removing books from the shopping cart
        - ✓ A type of book can be entirely removed from the shopping cart
    - ✓ Displaying Home Screen

✓ **Feature: Searching for books**

```
As a potential customer
I want to search for books by a simple phrase
So that I can easily allocate books by something I remember from them.
```

Background:

Given the following books

Author	Title
Martin Fowler	Analysis Patterns
Eric Evans	Domain Driven Design
Ted Pattison	Inside Windows SharePoint Services
Gojko Adzic	Bridging the Communication Gap

@WI1 @WI3

✓ **Scenario: Title should be matched** 1s 732ms

✓ **When** I search for books by the phrase 'Domain'

✓ **Then** the list of found books should contain only: 'Domain Driven Design'

@WI1 @WI4

✓ **Scenario: Author should be matched** 1s 701ms

20.2 Using filters

20.2.1 Keyword Filter

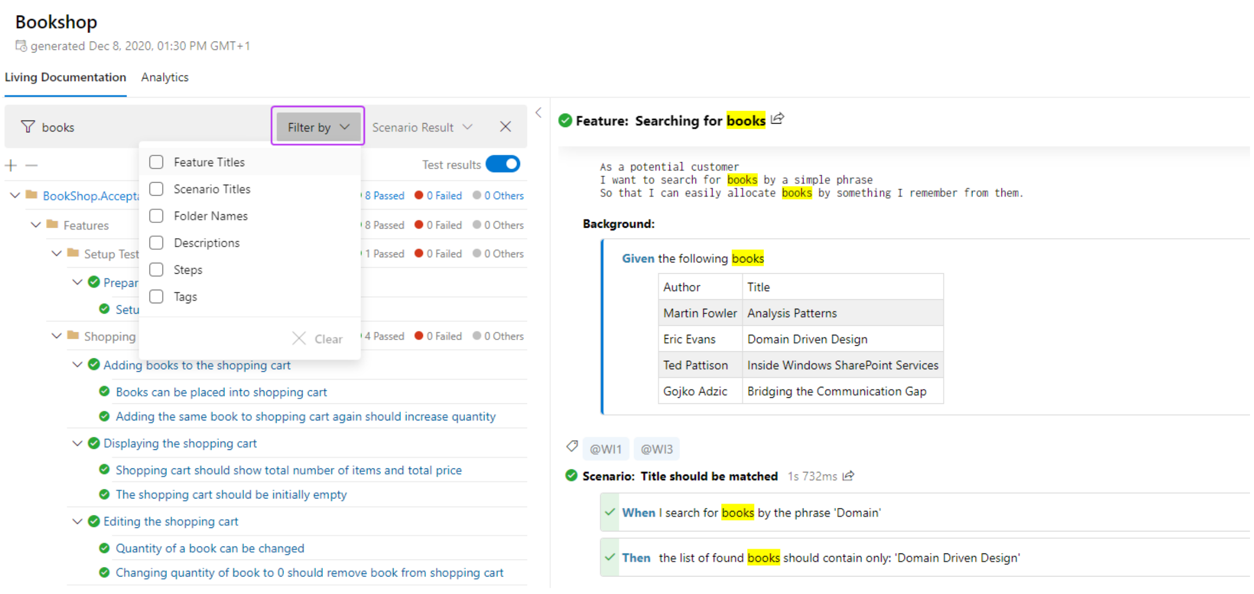
You can add a filter so that your keywords is only looked for in specific folder names, titles, descriptions, steps or tags.

To change the filter:

1 - Open the **Filter by** drop-down.

2 - Select the filters you want to apply from the list. You can select multiple filters.

3 - Enter your keyword.


Bookshop
generated Dec 8, 2020, 01:30 PM GMT+1
Living Documentation Analytics

books | Filter by | Scenario Result

- Feature Titles
- Scenario Titles
- Folder Names
- Descriptions
- Steps
- Tags

Clear

Test results

- BookShop.Accept… 8 Passed 0 Failed 0 Others
  - Features 8 Passed 0 Failed 0 Others
    - Setup Test… 1 Passed 0 Failed 0 Others
      - Prepar…
        - Setu…
    - Shopping 4 Passed 0 Failed 0 Others
      - Adding books to the shopping cart
        - Books can be placed into shopping cart
        - Adding the same book to shopping cart again should increase quantity
      - Displaying the shopping cart
        - Shopping cart should show total number of items and total price
        - The shopping cart should be initially empty
      - Editing the shopping cart
        - Quantity of a book can be changed
        - Changing quantity of book to 0 should remove book from shopping cart

Feature: Searching for books

```
As a potential customer
I want to search for books by a simple phrase
So that I can easily allocate books by something I remember from them.
```

Background:

Given the following books

Author	Title
Martin Fowler	Analysis Patterns
Eric Evans	Domain Driven Design
Ted Pattison	Inside Windows SharePoint Services
Gojko Adzic	Bridging the Communication Gap

@WI1 @WI3

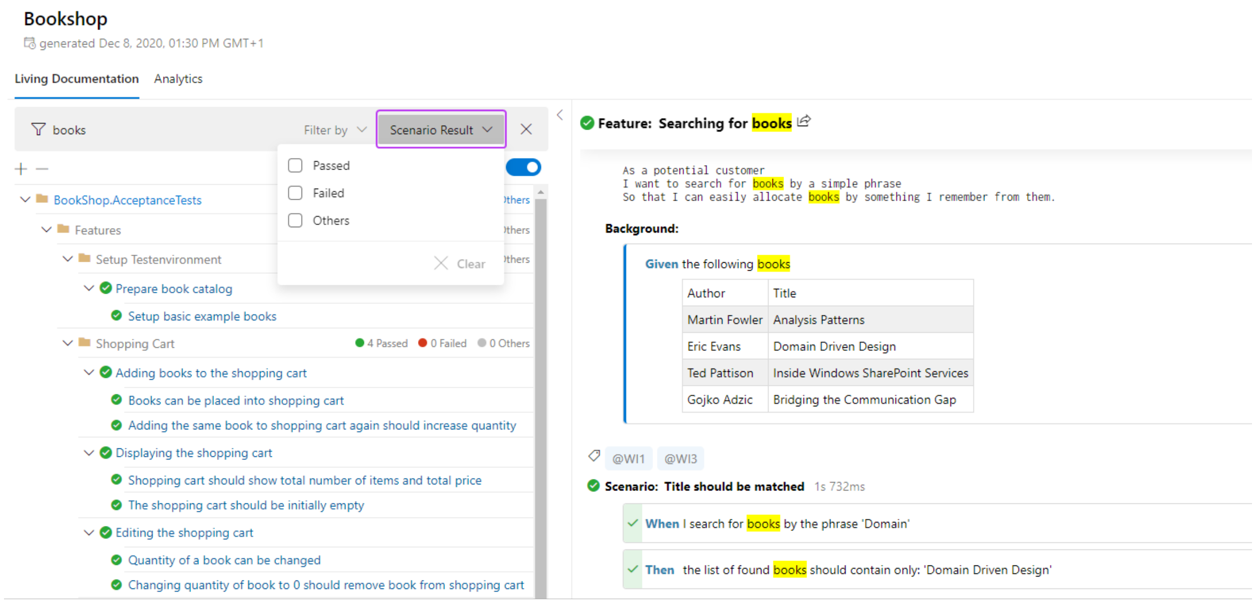
Scenario: Title should be matched 1s 732ms

When I search for books by the phrase 'Domain'

Then the list of found books should contain only: 'Domain Driven Design'

20.2.2 Scenario status filter

You also have the option to filter your results based on the status of the test, e.g passed,failed, or others.


Bookshop
generated Dec 8, 2020, 01:30 PM GMT+1
Living Documentation Analytics

books | Filter by | Scenario Result

- Passed
- Failed
- Others

Clear

- BookShop.AcceptanceTests …thers
  - Features …thers
    - Setup Testenvironment …thers
      - Prepare book catalog
        - Setup basic example books
    - Shopping Cart 4 Passed 0 Failed 0 Others
      - Adding books to the shopping cart
        - Books can be placed into shopping cart
        - Adding the same book to shopping cart again should increase quantity
      - Displaying the shopping cart
        - Shopping cart should show total number of items and total price
        - The shopping cart should be initially empty
      - Editing the shopping cart
        - Quantity of a book can be changed
        - Changing quantity of book to 0 should remove book from shopping cart

Feature: Searching for books

```
As a potential customer
I want to search for books by a simple phrase
So that I can easily allocate books by something I remember from them.
```

Background:

Given the following books

Author	Title
Martin Fowler	Analysis Patterns
Eric Evans	Domain Driven Design
Ted Pattison	Inside Windows SharePoint Services
Gojko Adzic	Bridging the Communication Gap

@WI1 @WI3

Scenario: Title should be matched 1s 732ms

When I search for books by the phrase 'Domain'

Then the list of found books should contain only: 'Domain Driven Design'

CHAPTER 21

Embedding Images & Markdown

You can include markdown code in your feature files with the standard Markdown features such as bold,italic, lists etc.

>*Note:* *SpecFlow+LivingDoc supports the mark down syntax by showdown.js.*

You can also use markdown to embed images in your feature files. These images will then be displayed when viewing LivingDoc

21.1 Embedding Images

When embedding images, the path to the image can be specified as a relative or absolute path. You can also embed images stored externally, such as on a website. Paths are relative to the location of the feature file.

Here are the possible ways to embed to images in feature files:

21.1.1 Embedding an image in the same directory as the feature file

```
![Alt text](image.png)
```

21.1.2 Embedding an image in a sub-directory

```
![Alt text](folder/image.png)
```

21.1.3 Embedding an image with an absolute reference

```
![Alt text](/folder/image.png)
```

21.1.4 Embedding an image relative to the parent directory

```
![Alt text](../image.png)
```

21.1.5 Embedding an external image

```
![Alt text](HTTPS://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcQyBVE0UKugTT3yaJZ7fpr1nVI
```

21.2 Example

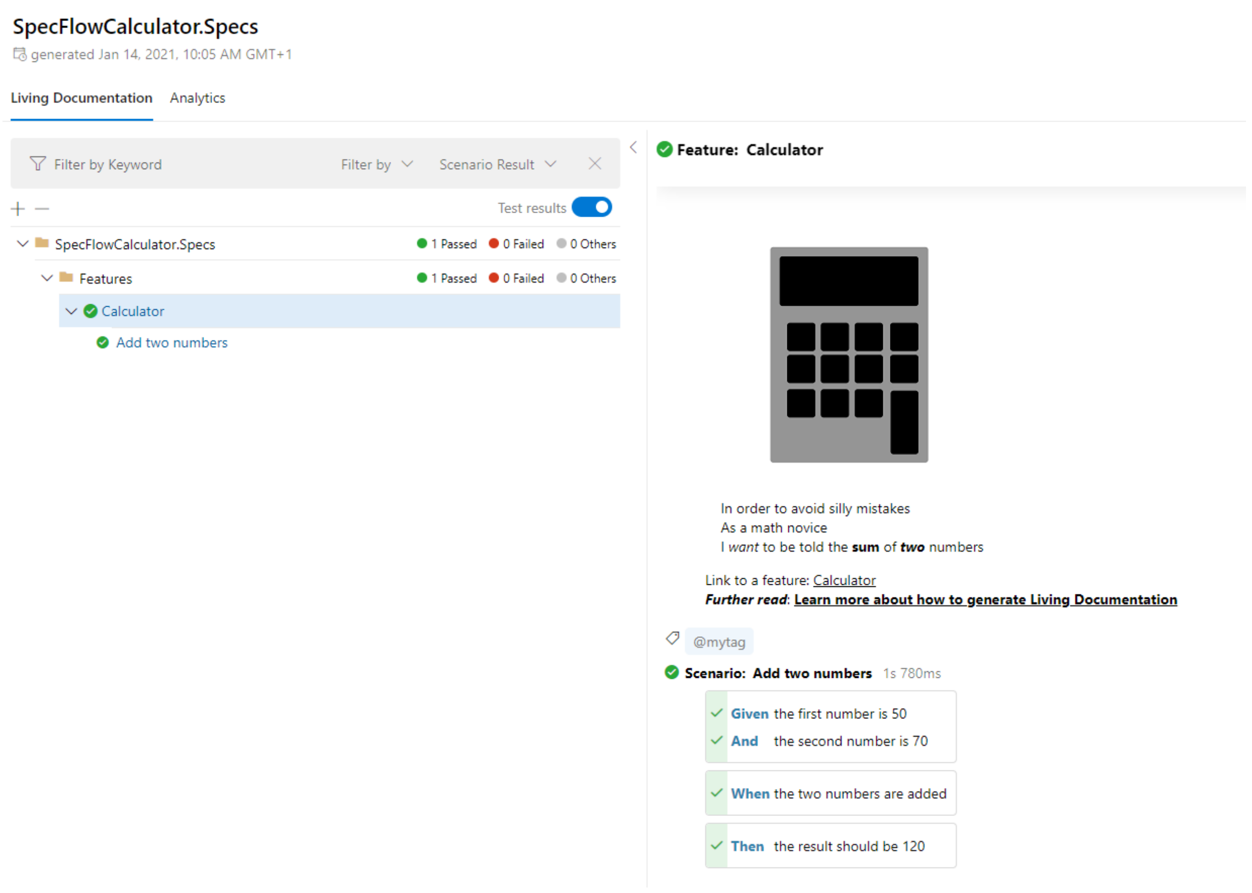
21.2.1 Links

The following code contains a link to an image in the Feature description:

*Note the link **must** be between the "Feature:" row and the first scenario.*

```
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
@mytag
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
```

This is the resulting output in SpecFlow+ LivingDoc:


SpecFlowCalculator.Specs
generated Jan 14, 2021, 10:05 AM GMT+1

SpecFlowCalculator.Specs ● 1 Passed ● 0 Failed ● 0 Others
Features ● 1 Passed ● 0 Failed ● 0 Others
Calculator
Add two numbers

Feature: Calculator

In order to avoid silly mistakes
As a math novice
I *want* to be told the **sum** of ***two*** numbers

Link to a feature: Calculator
Further read: **Learn more about how to generate Living Documentation**

@mytag
Scenario: Add two numbers 1s 780ms
✓ Given the first number is 50
✓ And the second number is 70
✓ When the two numbers are added
✓ Then the result should be 120

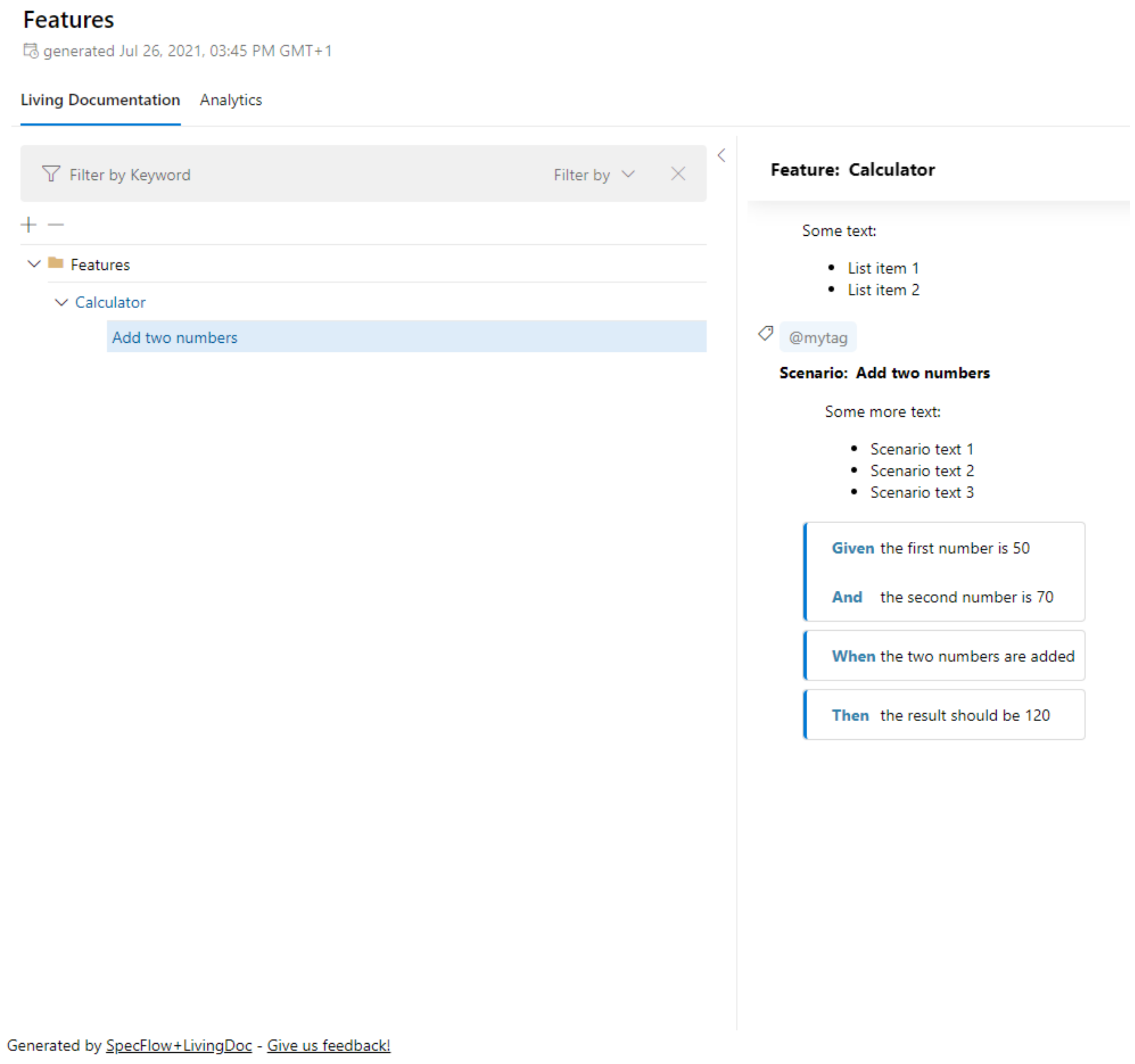
21.2.2 Text

The following code demonstrates the use of Markdown syntax for arbitrary text within a feature file.

Note: Asterisks ***cannot be used*** *as bullet points below the scenario declarations becuase they are a gherkin keyword (see here).*

```
Feature: Calculator

Some text:
- List item 1
- List item 2

@mytag
Scenario: Add two numbers

Some more text:
- Scenario text 1
- Scenario text 2
- Scenario text 3

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

This is the resulting output in SpecFlow+ LivingDoc:



Features

generated Jul 26, 2021, 03:45 PM GMT+1

Living Documentation Analytics

Feature: Calculator

Some text:

- List item 1
- List item 2

@mytag

Scenario: Add two numbers

Some more text:

- Scenario text 1
- Scenario text 2
- Scenario text 3

Given the first number is 50

And the second number is 70

When the two numbers are added

Then the result should be 120

Generated by SpecFlow+LivingDoc - Give us feedback!

CHAPTER 22

Linking within LivingDoc

22.1 Internal linking between features and scenarios

You can link your features by using their file names in the feature or scenario description. The location of the linked feature file is defined by your folder structure.

22.1.1 Linking a feature in the root directory

```
[Link text](root/Feature.feature)
```

22.1.2 Linking a feature in a sub-directory

```
[Link text](root/sub-directory/Feature.feature)
```

22.1.3 Linking a feature without link text

```
[](root/Feature.feature)
```

If the [Link Text] is not specified, the name of the feature will be displayed by default.

22.1.4 Linking Example

The following code contains a link to a feature:

```
Feature: Home Screen

As a potential customer
I want to search for books by a simple phrase
So that I can easily locate books by something I remember from them
```

(continues on next page)

(continued from previous page)

```

The search input is located on the [Home Screen](<BookShop.AcceptanceTests/Features/
↪Book Search.feature>).

Background:
        Given the following books
                | Author        | Title                          |
                | Martin Fowler | Analysis Patterns              |
                | Eric Evans    | Domain Driven Design           |
                | Ted Pattison  | Inside Windows SharePoint Services |
                | Gojko Adzic   | Bridging the Communication Gap     |

@WI8
Scenario: Title should match
        When I search for books by the phrase 'Domain'
        Then the list of found books should contain only 'Domain Driven Design'
```

Notes:

- Markdown is only supported in **Feature and Scenario descriptions**.If you place Markdown elsewhere, you will receive errors when building the documentation
- The markdown syntax supported here is **Showdown's Markdown syntax**. This is why angle brackets <> are used in the link above. Showdown's Markdown syntax requires angle brackets when there is space between characters, notice the blank space between "Book" and "Search" in the link.
- Notice the folder structure in the link example:

  (BookShop.AcceptanceTests/Features/Book Search.feature)

  root: BookShop.AcceptanceTest

  sub-directory: Features

  Feature.feature: Book Search.feature

This is the resulting output in SpecFlow+ LivingDoc:

CHAPTER 23

Link to ALM systems

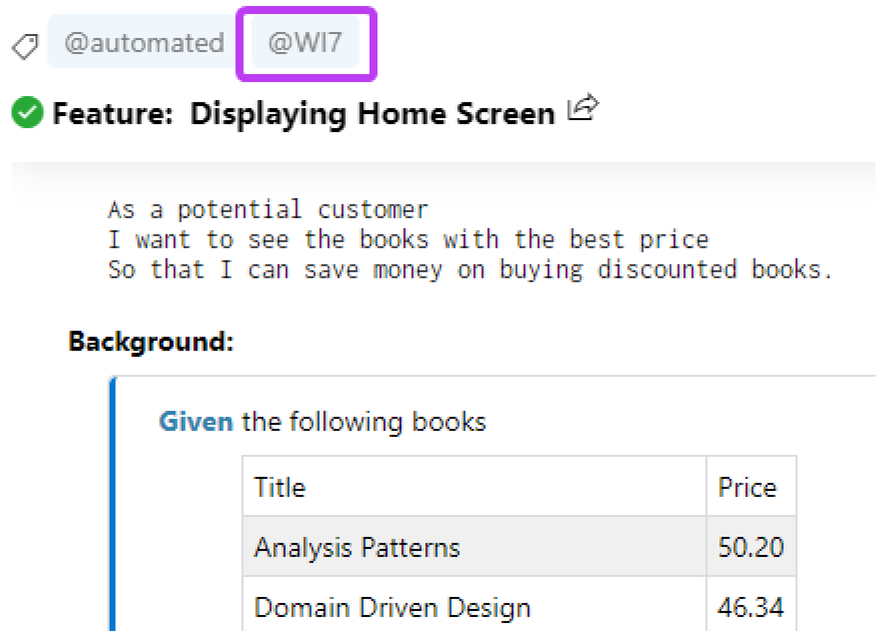
Features and Scenarios can be linked to external issues/tickets/work items/etc. by using Gherkin tags. These special tags allow quick navigation from LivingDoc to your application lifecycle management (ALM) software of choice, allowing you quick access to additional information such as status of the ticket, assigned person, and work item status.

23.1 External links in Azure DevOps

The SpecFlow+LivigDoc Azure DevOps extension makes linking to Azure DevOps work items easy.

For example, if you define a work item prefix as `WI:` in your *Azure DevOps build step*, then tags starting with this prefix will be converted to links when parsed by LivingDoc.

For instance, `@WI:7` will create a link to work item "7" in Azure DevOps:


@automated @WI7

Feature: Displaying Home Screen

```
As a potential customer
I want to see the books with the best price
So that I can save money on buying discounted books.
```

Background:

Given the following books

Title	Price
Analysis Patterns	50.20
Domain Driven Design	46.34

23.2 External Links in LivingDoc Generator

You can configure external ALM linking in the LivingDoc Generator too.

You can do this by creating external links to a specific work item in any ALM system.

For example, if you define the Gherkin tag prefix as `WI` and you want link to Azure DevOps then you can use the following command to generate the documentation:

```
livingdoc test-assembly C:\Work\MyProject.Specs\bin\Debug\netcoreapp3.1\MyProject.
↪Specs.dll --work-item-prefix WI --work-item-url-template https://dev.azure.com/
↪specflow/BookShop/_backlogs/backlog/BookShop%20Team/Stories/?workitem={id}
```

or in Atlassian Jira:

```
 --work-item-url-template https://jira.atlassian.com/browse/YOURPROJECT-{id}
```

Then tags starting with this prefix will be converted to links when parsed by LivingDoc.

> **Note:** If you are using **PowerShell** to generate LivingDoc, you must have the `--work-item-url-template` URL in single quotes to avoid getting errors:

```
livingdoc test-assembly C:\Work\MyProject.Specs\bin\Debug\netcoreapp3.1\MyProject.
↪Specs.dll --work-item-prefix WI --work-item-url-template 'https://dev.azure.com/
↪specflow/BookShop/_backlogs/backlog/BookShop%20Team/Stories/?workitem={id}'
```

CHAPTER 24

Editing Feature Files in Azure DevOps

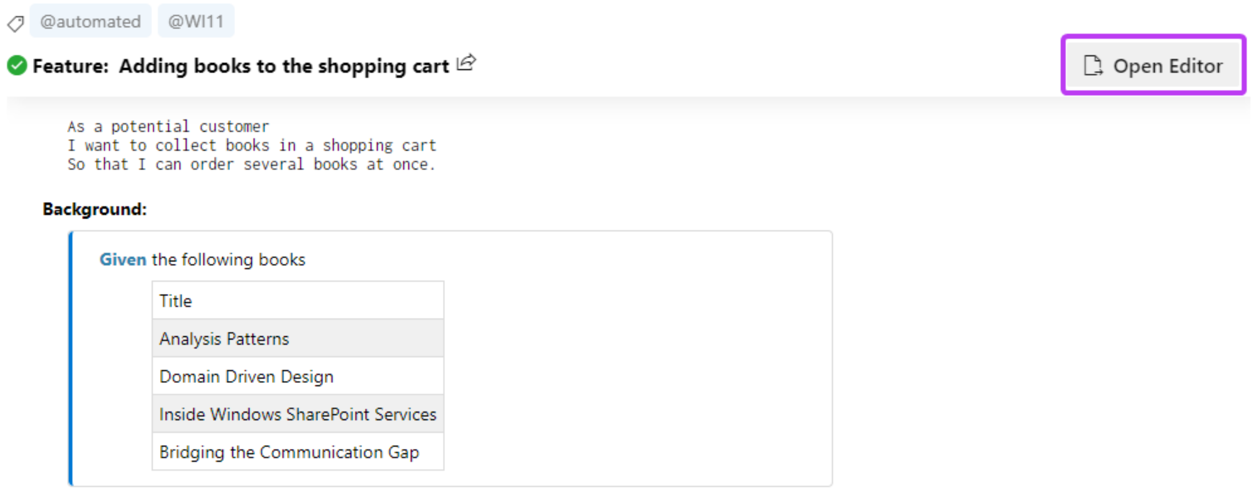
If your (feature files) are stored in Azure DevOps repositories, you can switch directly from the LivingDoc to the source feature files and edit files directly in Azure DevOps.

> Important: This feature is only available if you picked "Feature Folder" as your **source** folder when configuring the *build step in AzDo*, see section A.

To edit a feature file:

1 - Open the corresponding page in LivingDoc.

2 - Click on **Open Editor** on the top right of the page.



Feature: Adding books to the shopping cart

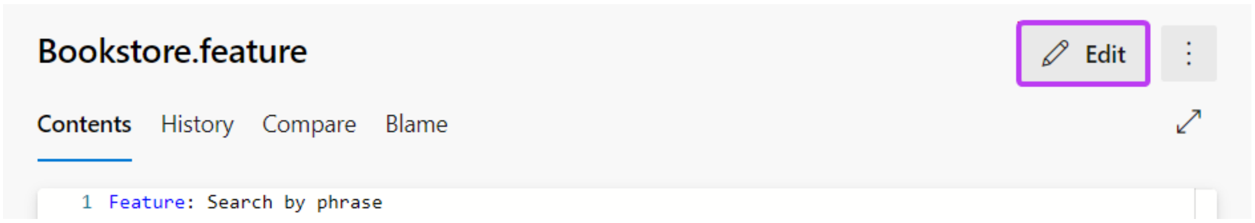
```
As a potential customer
I want to collect books in a shopping cart
So that I can order several books at once.
```

Background:

Given the following books

Title
Analysis Patterns
Domain Driven Design
Inside Windows SharePoint Services
Bridging the Communication Gap

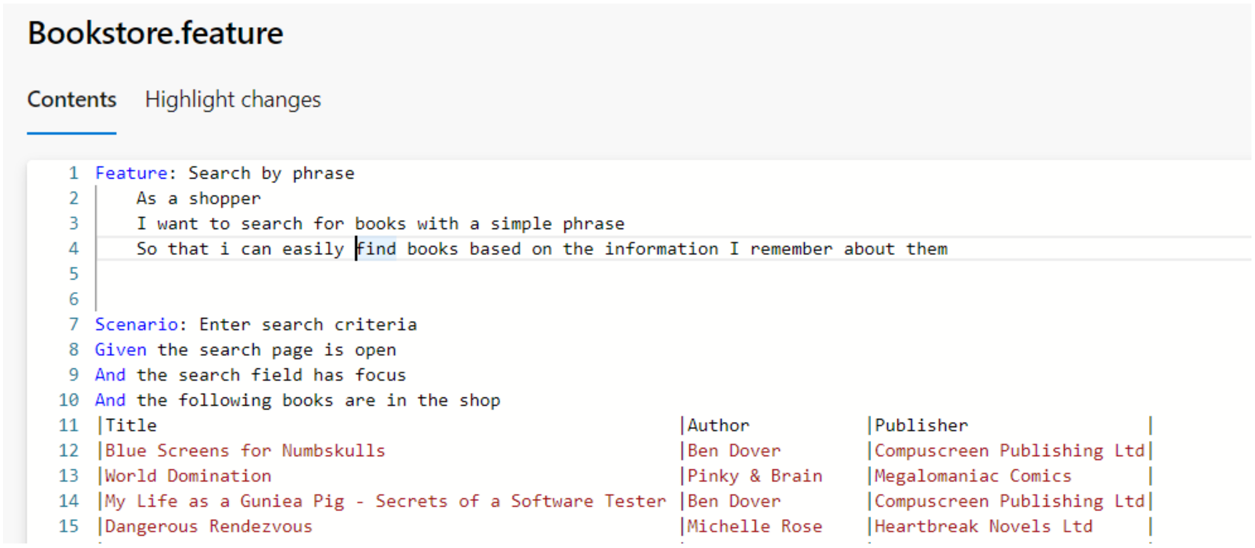
3 - The corresponding source feature file in your code repository is opened. You can **Edit** the file directly in Azure DevOps.



```
Feature: Search by phrase
```

Syntax highlighting is also supported in **Azure DevOps Repos** for the following Gherkin languages:

- English
- German



```
Feature: Search by phrase
    As a shopper
    I want to search for books with a simple phrase
    So that i can easily find books based on the information I remember about them


Scenario: Enter search criteria
Given the search page is open
And the search field has focus
And the following books are in the shop
|Title                                      |Author        |Publisher                |
|Blue Screens for Numbskulls                |Ben Dover     |Compuscreen Publishing Ltd|
|World Domination                           |Pinky & Brain |Megalomaniac Comics      |
|My Life as a Guniea Pig - Secrets of a Software Tester |Ben Dover     |Compuscreen Publishing Ltd|
|Dangerous Rendezvous                       |Michelle Rose |Heartbreak Novels Ltd    |
```

CHAPTER 25

Documentation Languages

SpecFlow+ LivingDoc parses your feature files looking for keywords, and uses these keywords to format the output. SpecFlow+ LivingDoc supports all languages supported by the official Gherkin parser.

The language used to parse the feature files is determined in the following order:

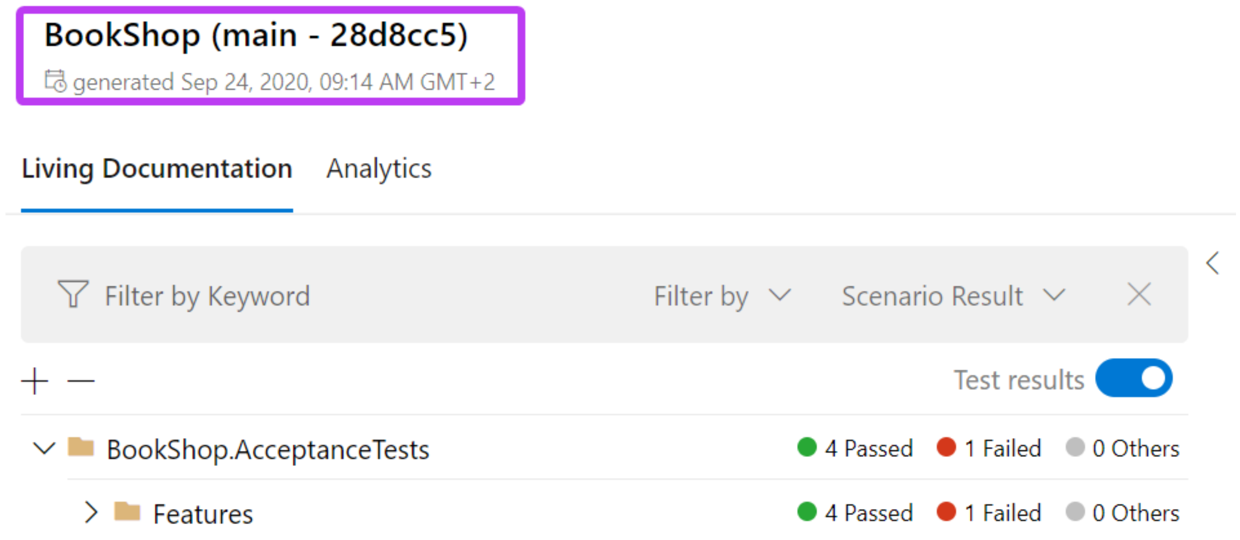
- The language specified in the Gherkin file itself using the `# language` header
- The language specified in your app.config file (deprecated in SpecFlow 3)
- The language specified in your build step
- If none of the above apply, SpecFlow+ LivingDoc defaults to en-US

CHAPTER 26

Adding branch name to LivingDoc

26.1 Branch name in LivingDoc Generator

Using the `--title` command line option the branch name (and commit information) can be included in the generated HTML.

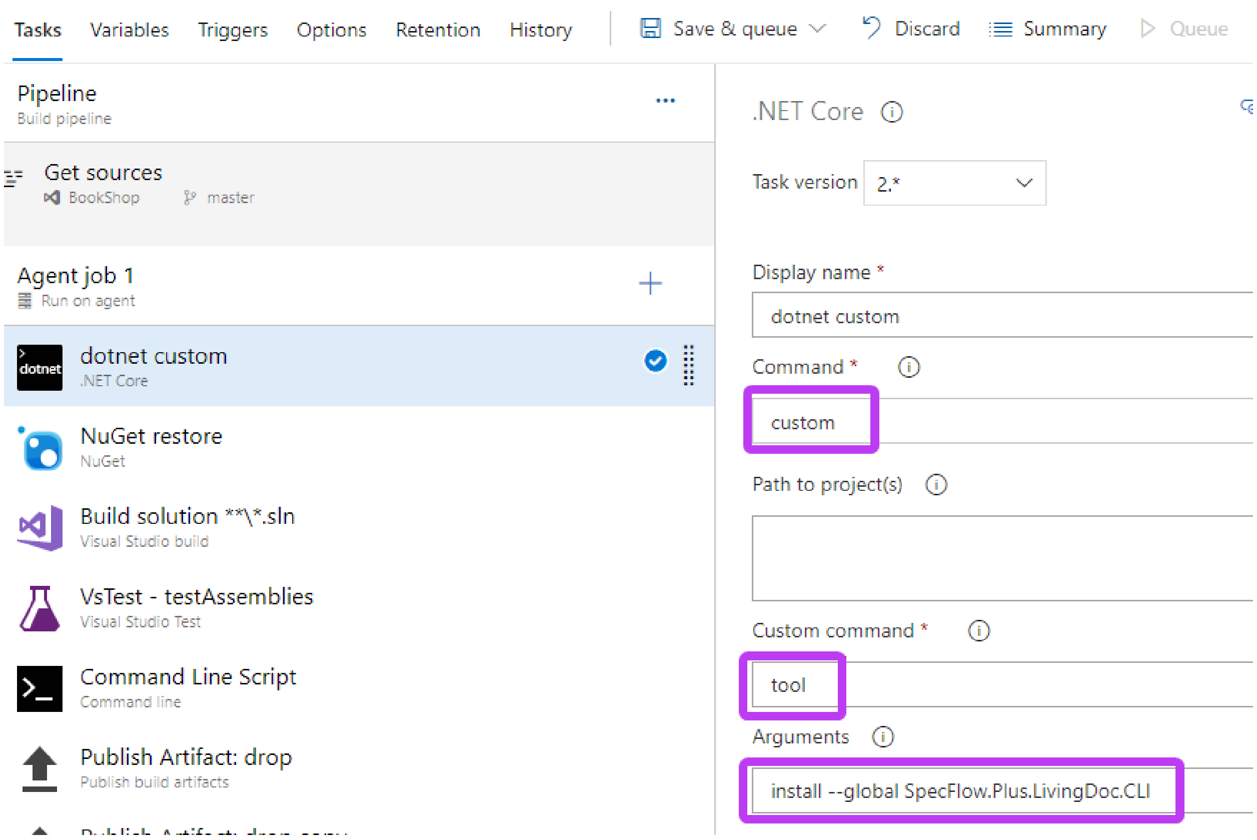


This process can be automated using a CI/CD system.

26.2 Branch name in Azure DevOps Pipeline

Azure DevOps Pipelines provides a rich set of variables like the branch name and commit message. Any of these variables can be used as part of the document title when the LivingDoc.CLI task is executed in the Pipeline.

First the LivingDoc.CLI has to be installed on the Agent, using the .NET Core build task



Task version 2.*

Display name * dotnet custom

Command * custom

Path to project(s)

Custom command * tool

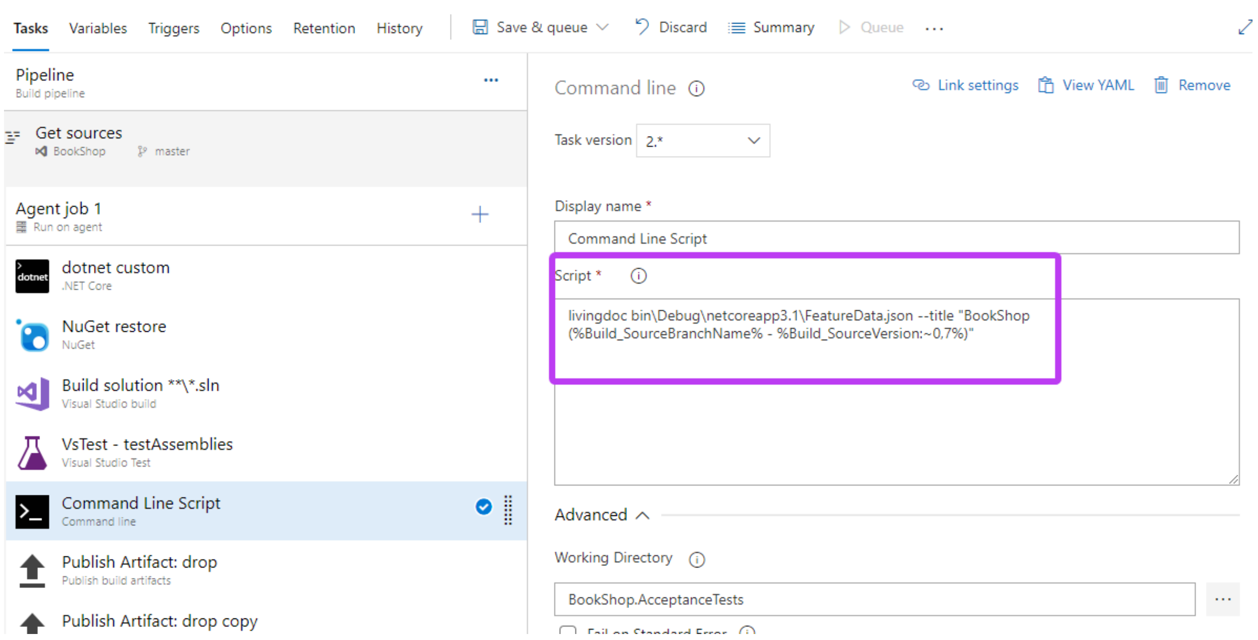
Arguments install --global SpecFlow.Plus.LivingDoc.CLI

- Command `custom`
- Custom command: `tool`
- Arguments: `install --global SpecFlow.Plus.LivingDoc.CLI`

YAML

```
steps:
- task: DotNetCoreCLI@2
  displayName: 'dotnet custom'
  inputs:
    command: custom
    custom: tool
    arguments: 'install --global SpecFlow.Plus.LivingDoc.CLI'
```

Then the CLI can be invoked after the **Visual Studio Test** task using the Command line task. In the script parameter the `%Build_SourceBranchName%` and `%Build_SourceVersion%` variables will be replaced during execution:



Task version 2.*

Display name * Command Line Script

Script *

```
livingdoc bin\Debug\netcoreapp3.1\FeatureData.json --title "BookShop
(%Build_SourceBranchName% - %Build_SourceVersion:~0,7%)"
```

Working Directory BookShop.AcceptanceTests

Fail on Standard Error

Script:

```
livingdoc test-assembly MyProject.Specs\bin\Debug\netcoreapp3.1\MyProject.Specs.dll --
↪title "BookShop (%Build_SourceBranchName% - Build_SourceVersion:~0,7%)"
```

YAML

```
    steps:
    - script: 'livingdoc test-assembly MyProject.Specs\bin\Debug\netcoreapp3.
↪1\MyProject.Specs.dll --title "BookShop (%Build_SourceBranchName% - Build_
↪SourceVersion:~0,7%)"'
      workingDirectory: BookShop.AcceptanceTests
      displayName: 'Command Line Script'
```

CHAPTER 27

Output API

The SpecFlow Output API allows you to display texts and attachments in your IDE's test explorer output window and also in SpecFlow+LivingDoc.

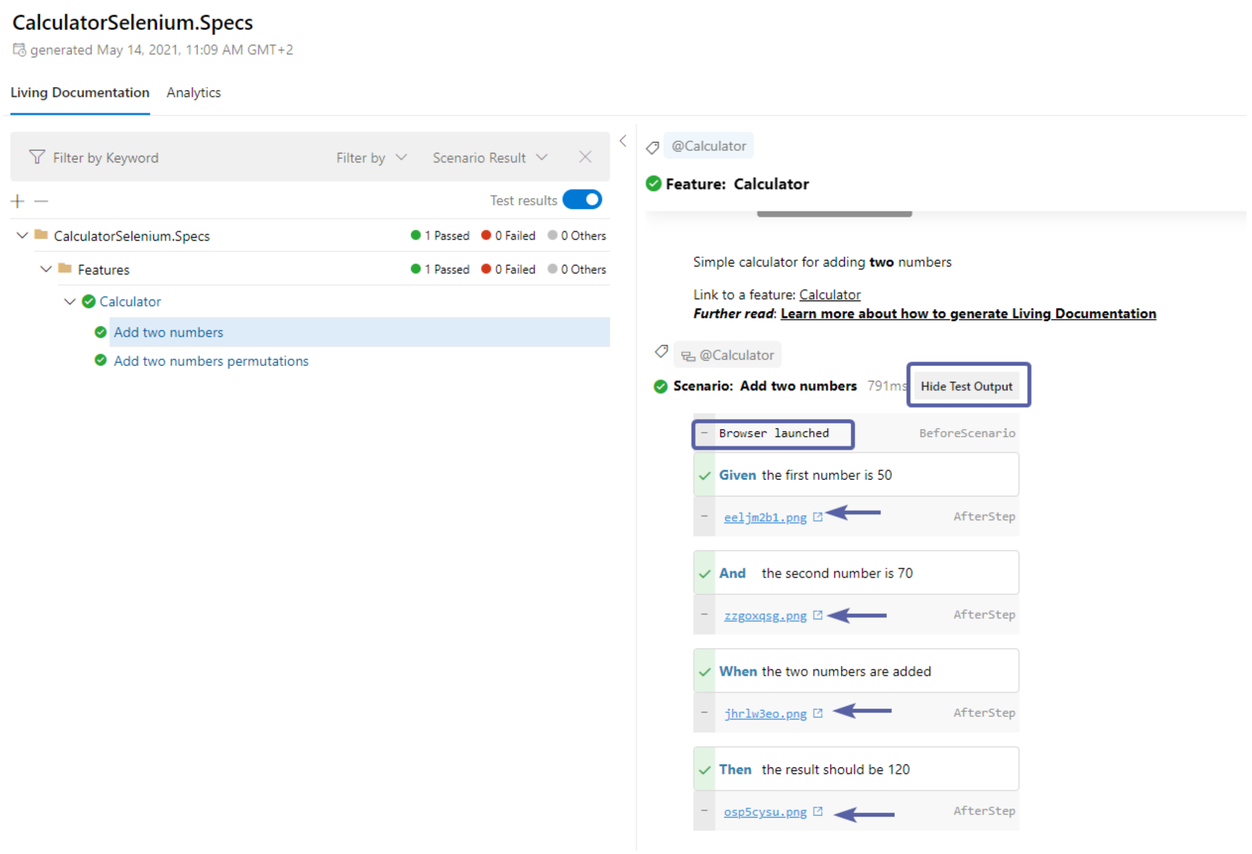
Please read the SpecFlow Output API docs page to learn how to take advantage of this feature.

27.1 Example

Check here to see the project used to generate the LivingDoc below.

Results can be viewed both in Visual Studio test explorer output window and SpecFlow+LivingDoc.

In SpecFlow+LivingDoc, no additional setup is required, simply *generate LivingDoc* as you normally do and view the output texts and attachments by toggling the *Show/Hide Test Output* button :


CalculatorSelenium.Specs
generated May 14, 2021, 11:09 AM GMT+2

Living Documentation | Analytics

Filter by Keyword | Filter by | Scenario Result

Test results

- CalculatorSelenium.Specs — ● 1 Passed ● 0 Failed ● 0 Others
  - Features — ● 1 Passed ● 0 Failed ● 0 Others
    - Calculator
      - Add two numbers
      - Add two numbers permutations

@Calculator

Feature: Calculator

Simple calculator for adding **two** numbers

Link to a feature: Calculator
Further read: **Learn more about how to generate Living Documentation**

@Calculator

Scenario: Add two numbers 791ms | Hide Test Output

```
Browser launched                BeforeScenario
```
Given the first number is 50
```
eeljm2b1.png                    AfterStep
```
And the second number is 70
```
zzgoxqsg.png                    AfterStep
```
When the two numbers are added
```
jhrlw3eo.png                    AfterStep
```
Then the result should be 120
```
osp5cysu.png                    AfterStep
```

> Note: *If the test output toggle is missing, it may be that you are on an older version of SpecFlow+LivingDoc, click here to update to the latest version.*

> Note: *The Output API in SpecFlow+LivingDoc supports the following four Hooks :*

- *BeforeScenario,*
- *AfterScenario,*
- *BeforeStep,*
- *AfterStep*

CHAPTER 28

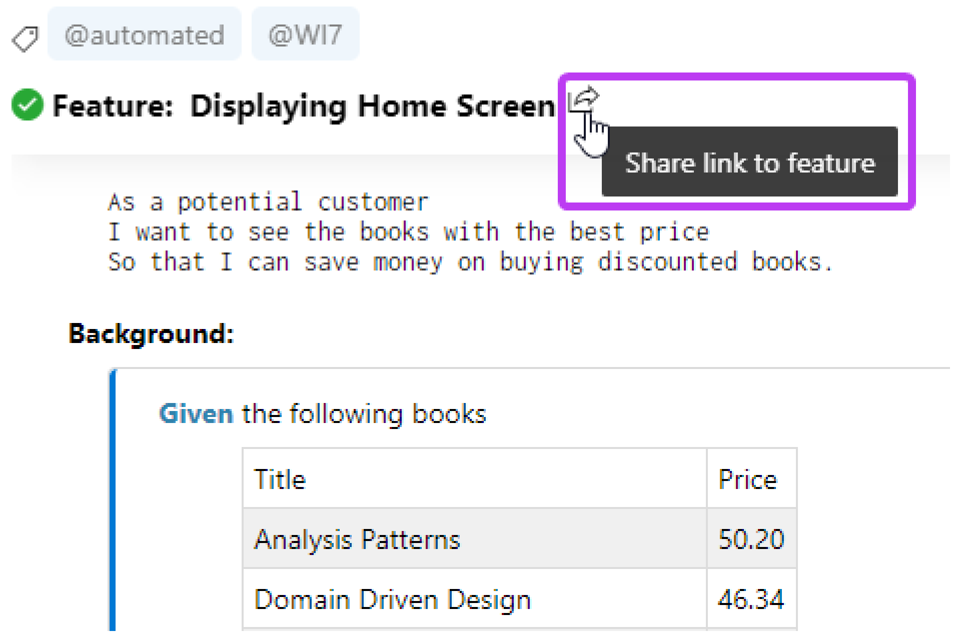
Sharing & Publishing SpecFlow+LivingDoc

28.1 Sharing links to Feature Files in Azure DevOps

Team collaboration with SpecFlow+Living Azure DevOps extension is as easy as just giving access to team members in Azure DevOps so they can view LicingDocs under the "Overview" tab.

SpecFlow+Living Azure DevOps extension also gives you the option to quickly create links to Feature Files in Azure DevOps.

You can do this by clicking on the share icon which copies the URL of the selected feature or scenario to the clipboard. You can then share this link with anyone who has access to the project in Azure DevOps.



Feature: Displaying Home Screen

```
As a potential customer
I want to see the books with the best price
So that I can save money on buying discounted books.
```

Background:

Given the following books

Title	Price
Analysis Patterns	50.20
Domain Driven Design	46.34

Note this option is not available when using LivingDoc generated from the CLI tool

28.2 Sharing SpecFlow+LivingDoc HTML file

If you used the CLI tool to generate stand alone LivingDoc then keep in mind that the generated HTML is a single page application with no external dependencies, so you can share it as you wish

The possible sharing options depend on your current infrastructure setup and needs, but here are some ideas:

- Any file sharing solution (Windows file share, Sharpoint, Dropbox etc.)
- Hosting the generated HTML with any of the static HTML hosting services such as Azure Blob Storage and AWS

28.3 Store it in your Continuous Integration Server

28.3.1 TeamCity

You can use the Thrid-Party Reports feature of TeamCity to publish the HTML to your Build Results.

28.3.2 Jenkins

You can use the HTML Publisher Plugin of Jenkis to publish the HTML.

28.3.3 GitLab

You can publish the HTML as GitLab Pages. You can follow this article which shows how to publish a HTML coverate report.

CHAPTER 29

SpecFlow+LivingDoc Generator Migration v3.4 to v3.5

With version 3.5 the LivingDoc Generator has been updated to support various document generation scenarios.

29.1 Upgrade guide

To upgrade LivingDoc Generator from v3.4 to v3.5 in your solution please follow the steps below.

1. Make sure that you upgrade both the plugin and the CLI to the same version
   - The *SpecFlow.Plus.LivingDocPlugin* has to be updated as a NuGet package in the SpecFlow project
   - The *Command Line Tool (CLI)* has to be updated as a dotnet tool:

     To update SpecFlow.Plus.LivingDoc.CLI:

     – Open a command prompt.

     – Run the following command:`dotnet tool update --global SpecFlow.Plus.LivingDoc.CLI`
2. The default file name of the JSON file generated by the LivingDocPlugin has been changed from "FeatureData.json" to "TestExecution.json". With v3.5 this JSON file contains the test execution details only and the feature files are parsed by the CLI. Make sure to look for this new file after executing your SpecFlow scenarios to be able to pass it to the CLI (see further below).

   > Please note that if you have configured a custom file path in your `specflow.json` then the generated filename will not change automatically with the upgrade. We recommend to change the filename in the configuration to be consistent with the new semantics.
3. Change your command line scripts and build scripts to call the CLI with the new parameters
   - Old command

```
livingdoc C:\Work\MyProject.Specs\bin\Debug\netcoreapp3.1\FeatureData.json
```

   - New command

```
    livingdoc test-assembly C:\Work\MyProject.Specs\bin\Debug\netcoreapp3.
↪1\MyProject.Specs.dll -t C:\Work\MyProject.Specs\bin\debug\netcoreapp3.
↪1\TestExecution.json
```

Please note the following changes in the *parameter syntax*:

1. The first parameter of the CLI is a mandatory command name.
2. The list of available options depends on the selected command (still many options are available with all commands).
3. All parameter names are in kebab format (e.g. test-assembly, feature-folder, work-item-url-template, etc.)

Please check the *Troubleshooting* page if you run into issues.

CHAPTER 30

Troubleshooting

30.1 Required command was not provided

Symptoms The CLI prints the following error:

> Required command was not provided. Unrecognized command or argument '. . . '

Resolution The upgraded CLI expects a mandatory "command" parameter. Please make sure to update your command line scripts according to the new parameter syntax. See an example in the upgrade guide above.

30.2 Unrecognized command or argument

Symptoms The CLI prints the following error:

> Unrecognized command or argument '. . . '

Resolution The upgraded CLI uses kebab casing in the parameters. Please check the casing of the provided command or argument. See the *CLI parameters*.

30.3 Cannot find the specified JSON file - FeatureData.json

Symptoms: The CLI prints the following error:

> Error: Cannot find the specified JSON file: <. . . some path..>\FeatureData.json

Resolution LivingDocPlugin v3.5 does not generate a "FeatureData.json" file anymore, the default file name has been changed to "TestExecution.json". Please make sure to update the command line parameters when calling the CLI. Also make sure that you updated the LivingDocPlugin and the CLI to the same version.

30.4 Standalone HTML does not load

Symptoms When opening the standalone LivingDoc.html only a “Loading. . . ” message is displayed, but the content is not loaded. In the browser console the following error is displayed:

> Uncaught TypeError: e.tree.Nodes[0] is undefined

Resolution This error might occur if the LivingDocPlugin has been updated to v3.5, and the generated JSON (”TestExecution.json”) is being passed to an older version of the CLI. Please update the CLI to the same version as the LivingDocPlugin.

30.5 No “TestExecution.json” generated

Symptoms When running the SpecFlow tests there is no “TestExecution.json” generated.

Resolution Please check:

1. Check the package version of the *SpecFlow.Plus.LivingDocPlugin* NuGet package in your SpecFlow project. You have to upgrade the NuGet package to at least v3.5 to have the new “TestExecution.json” output generated. Also make sure that you updated the LivingDocPlugin and the CLI to the same version.
2. Check your specflow.json configuration. You might have configured a custom file path for the LivingDocPlugin.
3. If you use the default configuration search for the file in the output directory of the SpecFlow project (in the same folder where the test assembly is located).

CHAPTER 31

Merging Multiple Test Results

31.1 Use case

If you have been running your tests in parallel using SpecFlow+Runner or any other Runner, your test results may end up in multiple JSON files. You may also have different testing stages running in parallel, which would produce multiple JSON files.

You have the option to combine these JSON files that carry your test results data into a single Living Documentation, depending on which SpecFlow LivingDoc tool you are using:

31.2 SpecFlow+LivingDoc Generator

To merge multiple JSON files in the command-line tool simply separate them with a space:

```
C:\work\BookShop\BookShop.AcceptanceTests\bin\Debug\netcoreapp3.1>livingdoc test-
↪assembly BookShop.AcceptanceTests.dll -t TestExecution_1.json TestExecution_2.json
```

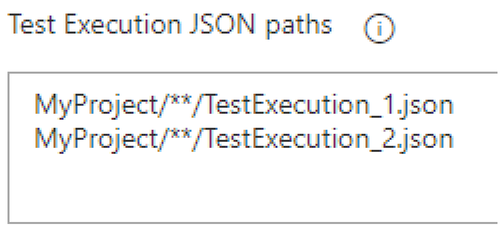
In the above example we are generating LivingDoc from the test assembly which is located in our local *C:\work\BookShop\BookShop.AcceptanceTests\bin\Debug\netcoreapp3.1* directory. We also have two JSON files, TestExecution_1 & TestExecution_2, that are generated when we executed our tests in the same folder. As you can see you can include any number of JSON files by separating them with a space.

If you choose to includes all of your JSON files you can use the wild card * to do so:

```
C:\work\BookShop\BookShop.AcceptanceTests\bin\Debug\netcoreapp3.1>livingdoc test-
↪assembly BookShop.AcceptanceTests.dll -t TestExecution*.json
```

31.3 SpecFlow+LivingDoc for Azure DevOps

To merge multiple JSON files in Azure Devops you have to include them in your build step in separate lines:



Test Execution JSON paths

```
MyProject/**/TestExecution_1.json
MyProject/**/TestExecution_2.json
```

CHAPTER 32

LivingDoc in AzDo release pipeline

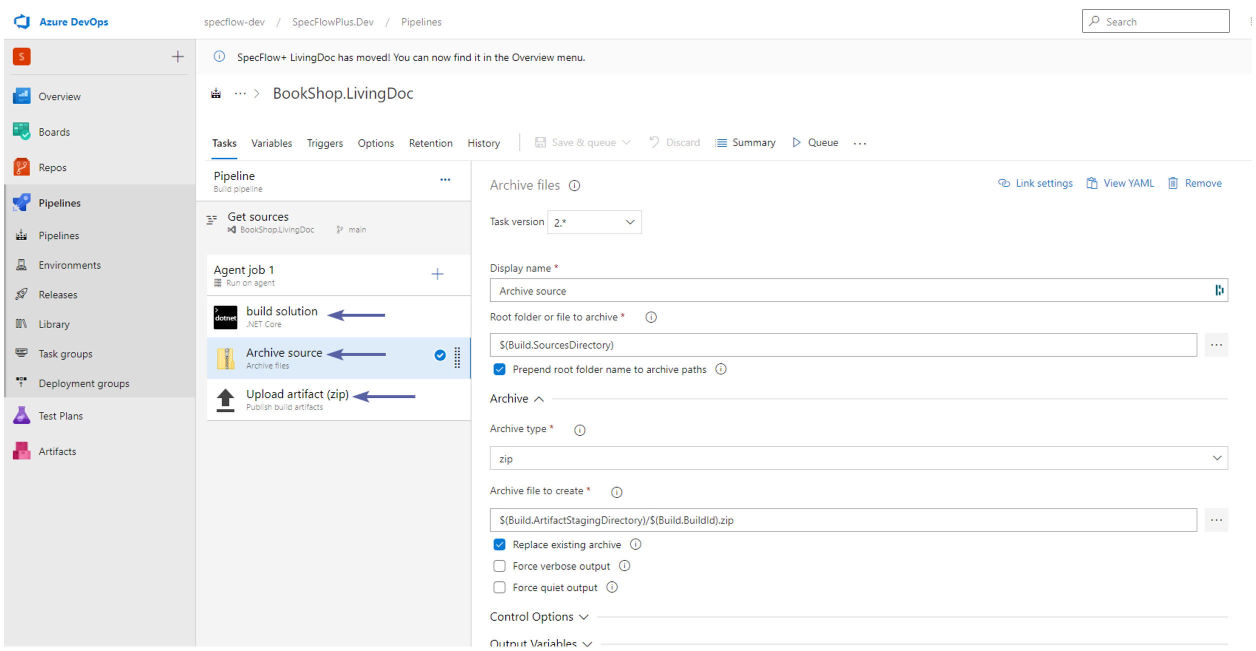
The SpecFlow+LivingDoc Azure DevOps build task is **not** limited to only the build pipeline as described *here*. The build task can also be added later in the **release pipeline**.

32.1 Use Case

There are a number of ways to build, test and generate LivingDoc in Azure DevOps. The general *configurations* remain the same with file path changes depending on your solution.

32.2 Example

In the below example, the solution is built, archived and then uploaded in the build pipeline:



SpecFlow+ LivingDoc has moved! You can now find it in the Overview menu.

BookShop.LivingDoc

Pipeline
Build pipeline

Get sources
BookShop.LivingDoc main

Agent job 1
Run on agent

build solution
.NET Core

Archive source
Archive files

Upload artifact (zip)
Publish build artifacts

Archive files

Task version: 2.*

Display name *: Archive source

Root folder or file to archive *: $(Build.SourcesDirectory)

[✓] Prepend root folder name to archive paths

Archive

Archive type *: zip

Archive file to create *: $(Build.ArtifactStagingDirectory)/$(Build.BuildId).zip

[✓] Replace existing archive

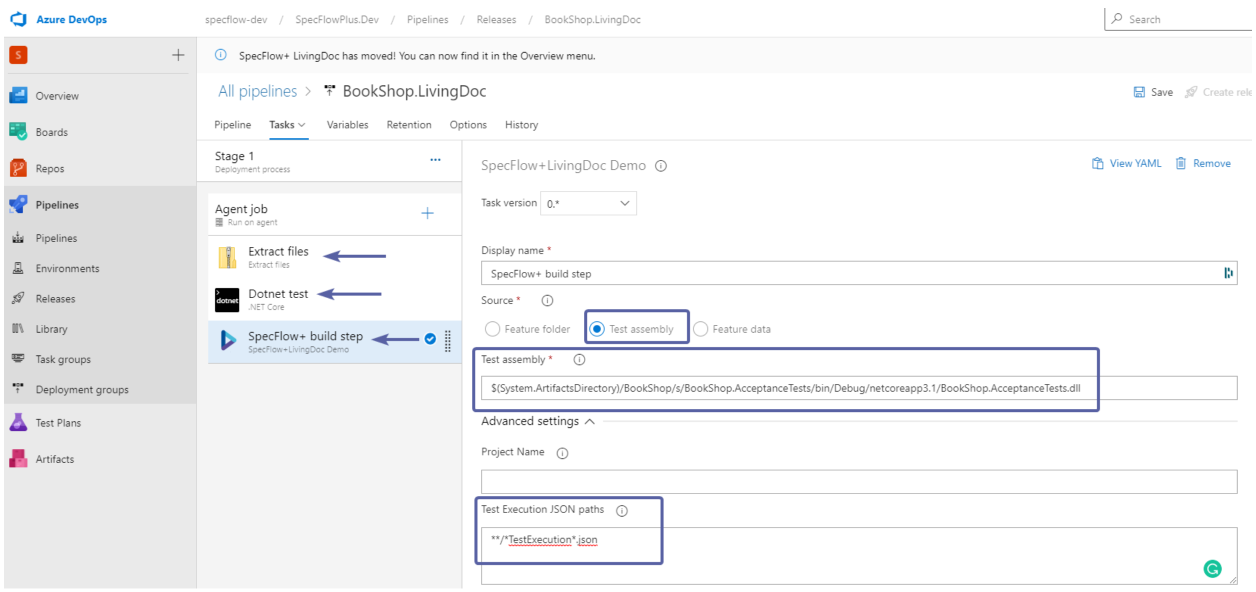
[ ] Force verbose output

[ ] Force quiet output

Control Options

Output Variables

In the release pipeline, the archived source files are extracted and the tests are executed. The SpecFlow build step is added as the last step after test execution:



SpecFlow+ LivingDoc has moved! You can now find it in the Overview menu.

BookShop.LivingDoc

Stage 1
Deployment process

Agent job
Run on agent

Extract files
Extract files

Dotnet test
.NET Core

SpecFlow+ build step
SpecFlow+LivingDoc Demo

SpecFlow+LivingDoc Demo

Task version: 0.*

Display name *: SpecFlow+ build step

Source *: Feature folder [✓ Test assembly] Feature data

Test assembly *: $(System.ArtifactsDirectory)/BookShop/s/BookShop.AcceptanceTests/bin/Debug/netcoreapp3.1/BookShop.AcceptanceTests.dll

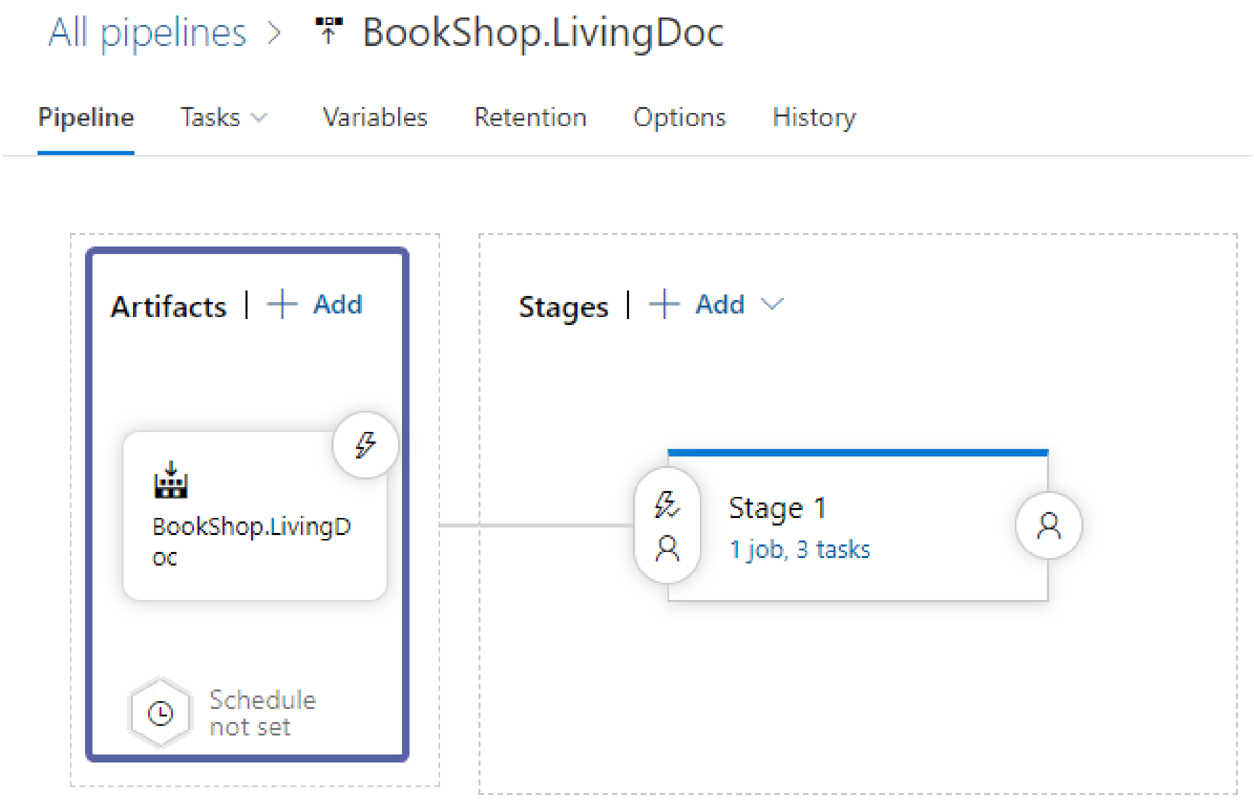
Advanced settings

Project Name:

Test Execution JSON paths: **/*TestExecution*.json

Configuring the SpecFlow build step in the release pipeline is no different to doing it in the build pipeline.

> ***Note*** *SpecFlow build step can be used in a Release pipeline* ***only if*** *there is a valid* ***'BUILD' artifact*** *reference added as the first artifact in the release definition. LivingDoc will derive the repository and branch information from the referenced build.*



There are 3 sources to pick from to parse the documentation data:

Feature folder - The folder containing the feature files.

> *Note If you pick **feature folder** as your source and would like to display the execution results, you MUST provide the **test project root folder**.*

Test Assembly - Relative or absolute path to the test assembly. Glob patterns are supported e.g.: `MyProject/**/MyProject.dll`

Feature Data - Relative or absolute path to the feature data JSON file. Glob patterns are supported e.g.: `MyProject/**/FeatureData.json`

In this example we used the *test-assembly*.

If you want to include test execution results in LivingDoc you must specify the Test Execution JSON paths. It has to be an absolute or relative path to the Test Execution JSON files generated by the SpecFlow.Plus.LivingDocPlugin. Glob patterns are supported e.g.: `MyProject/**/TestExecution.json`

If you have been running your tests in parallel using SpecFlow+Runner or any other Runner, adding a * at the end of the JSON file name will include them all in the generated LivingDoc. This is the case in the example build task shown above.

CHAPTER 33

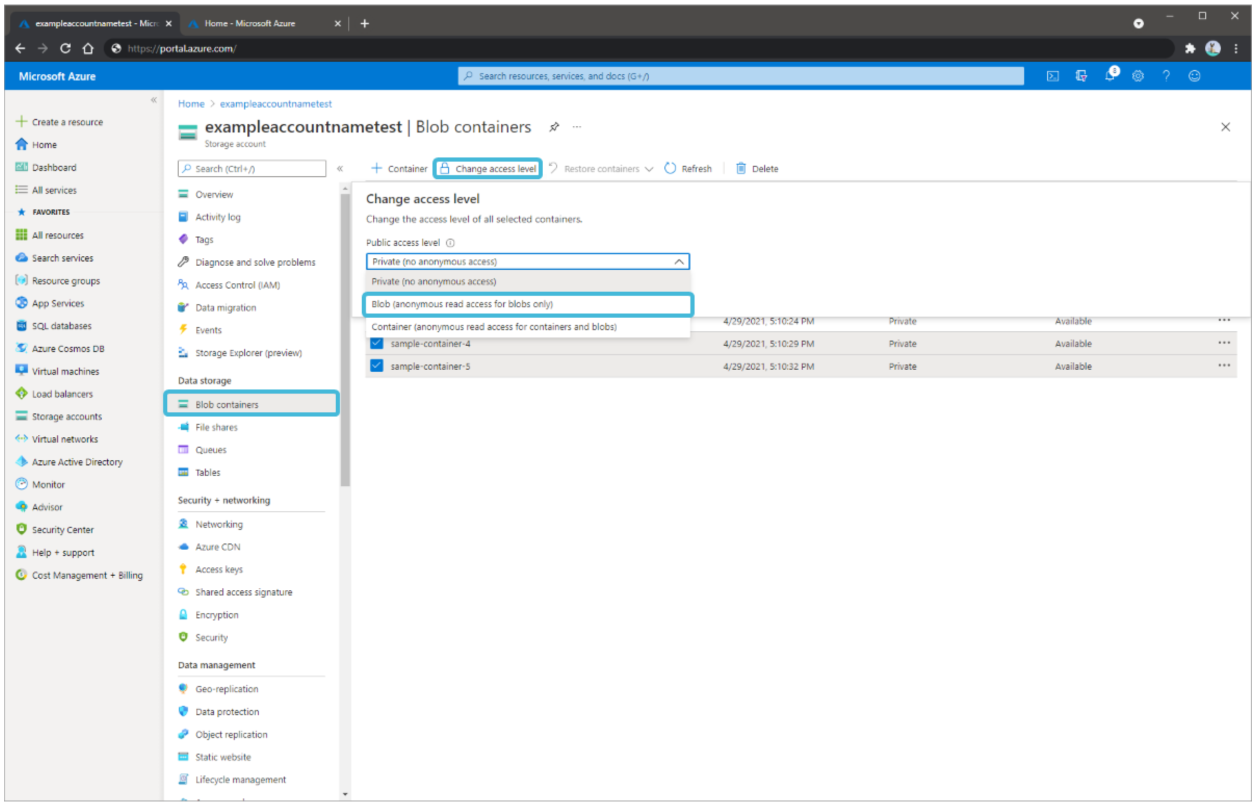
Storing Images in Azure Blob Storage

The SpecFlow Output API allows you to display texts and attachments in your IDE's test explorer output window and also in SpecFlow+LivingDoc. These images are saved and uploaded from your local machine, therefore only visible to you. If you want these images to be viewable you must host these images in another space, for instance, Azure Blob Storage.

We have put together the below guide to demonstrate a use case. Please note this is **not** the only solution and there might be other alternatives.

33.1 Prerequisites

- A storage account needs to created in Azure.
- A container needs to be created with ***Public read access for blobs only*** permission. You can read more about this on Microsoft's website



Change access level

Change the access level of all selected containers.

Public access level: Private (no anonymous access)

- Private (no anonymous access)
- Blob (anonymous read access for blobs only)
- Container (anonymous read access for containers and blobs)

	4/29/2021, 5:10:24 PM	Private	Available
sample-container-4	4/29/2021, 5:10:29 PM	Private	Available
sample-container-5	4/29/2021, 5:10:32 PM	Private	Available

- A Service Connections needs to be created for the Azure subscription in Azure DevOps with the permission ***Storage Blob Data Contributor*** configured for the service principal. You can read more about this on Microsoft's website and here.

33.2 Example

You can download this example repo from our GitHub page.

Our example project works as per following:

- Every test-run saves the images locally to a pre-defined folder e.g. *Screenshots*.
- The test code receives the URL template for the attachments from the build using an environment variable
- After creating and saving the screenshots locally, the code uses the URL template and the file name and generates the URL for the output API:

```
namespace CalculatorSelenium.Specs.Hooks
{
    [Binding]
    public sealed class LoggingHooks
    {
        private readonly ISpecFlowOutputHelper _specFlowOutputHelper;
        private readonly BrowserDriver _browserDriver;
        private string? _attachmentTemplate;

        public LoggingHooks(ISpecFlowOutputHelper specFlowOutputHelper, BrowserDriver␣
↪browserDriver)
        {
```

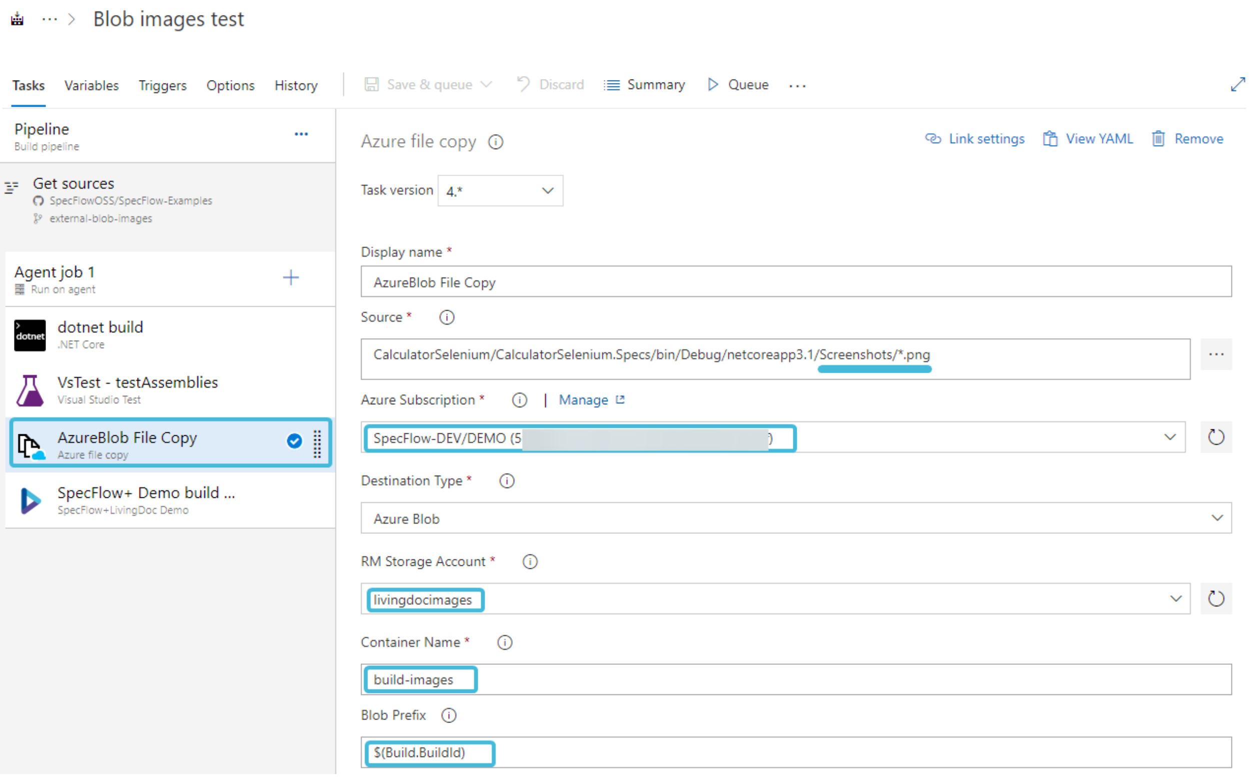
(continues on next page)

(continued from previous page)

```
            _specFlowOutputHelper = specFlowOutputHelper;
            _browserDriver = browserDriver;
            _attachmentTemplate = Environment.GetEnvironmentVariable("ATTACHMENT_
↪TEMPLATE");
        }

        [AfterStep]
        public void AfterStep()
        {
            var screenshot = _browserDriver.Current.TakeScreenshot();
            var imagePath = Path.GetFileNameWithoutExtension(Path.GetTempFileName())␣
↪+ ".png";
            Directory.CreateDirectory("Screenshots");
            screenshot.SaveAsFile(Path.Combine("Screenshots", imagePath));
            if (!string.IsNullOrEmpty(_attachmentTemplate))
                _specFlowOutputHelper.AddAttachment(string.Format(_attachmentTemplate,
↪ imagePath));
        }
    }
}
```

33.2.1 Setup in Azure DevOps

- The build contains the AzureCopy task which uploads the images from the Screenshots folder to the blob storage to a folder unique to the build In AzDo. In LivingDoc the absolute URLs can then be opened. The following settings needs to be configured for the Azure Copy task:



Task version: 4.*

Display name *: AzureBlob File Copy

Source *: CalculatorSelenium/CalculatorSelenium.Specs/bin/Debug/netcoreapp3.1/Screenshots/*.png

Azure Subscription * | Manage: SpecFlow-DEV/DEMO (5[redacted])

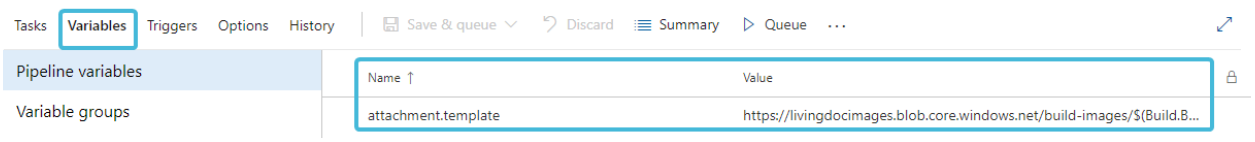
Destination Type *: Azure Blob

RM Storage Account *: livingdocimages

Container Name *: build-images

Blob Prefix: $(Build.BuildId)

- The URL template needs to set as a variable for the build in the Classis pipeline:



Name ↑	Value
attachment.template	https://livingdocimages.blob.core.windows.net/build-images/$(Build.B...

- The URL template should match the container name and the blob prefix:

```
https://livingdocimages.blob.core.windows.net/build-images/$(Build.BuildId)/{0}
```

> ***Note:*** In YAML the URL template can be passed on to the VSTEST task:

```
steps:
- task: VSTest@2
  displayName: 'VsTest - testAssemblies'
  inputs:
testAssemblyVer2: |
**\*Specs*.dll
!**\*TestAdapter.dll
!**\obj\**
searchFolder: '$(System.DefaultWorkingDirectory)\CalculatorSelenium'
env:
ATTACHMENT_TEMPLATE: 'https://livingdocimages.blob.core.windows.net/build-images/
↪$(Build.BuildId)/{0}'
```

33.2.2 Important Notes

- The container needs ***Blob access*** so anonymous users can access the files.
- The AzureCopy task can create the container but with **private** access. The user needs to access container and set the blob access permission as per prerequisites above.
- The container name has some restrictions: only lowercase letters, numbers, and dash -.
- The URL template needs to be kept sync with the other parameters.

CHAPTER 34

Azure DevOps Extension

34.1 Version 0.6.964 (10 September 2021)

- Bugfixes
  - Fixed security issue in markdown parsing

34.2 Version 0.6.961 (26 August 2021)

- Bugfixes
  - Fix scoped bindings detection

34.3 Version 0.6.946 (30 July 2021)

- Bugfixes
  - **[BuildTask]** Examples table with duplicate column headers should throw an exception Github#2471

34.4 Version 0.6.943 (26 July 2021)

- Bugfixes
  - **[BuildTask]** Improve error handling of the Build Task

34.5 Version 0.6.934 (18 June 2021)

- Update SpecFlow logo

34.6 Version 0.6.916 (11 May 2021)

- New Features
  - Display backgrounds within the scenario sections
  - Display outputs (text/attachments)
  - **[BuildTask]** Extend with option to include/exclude test outputs
- Bugfixes
  - **[BuildTask]** Fix path escaping bug in the BuildTask, related to Github#2407

34.7 Version 0.6.859 (19 April 2021)

- New Features
  - Display different icons for different types of repositories
- Bugfixes
  - **[BuildTask]** Fix validation error in on prem installation

34.8 Version 0.6.839 (26 March 2021)

- New Features
  - **[BuildTask]** Allow to choose input source (feature-folder, test-assembly, feature-data)
  - **[BuildTask]** Option to specify custom Work item URL template

34.9 Version 0.6.784 (08 March 2021)

- Technical update

34.10 Version 0.6.759 (16 February 2021)

- Technical update

34.11 Version 0.6.739 (25 January 2021)

- New Features
  - Small UI improvements

34.12 Version 0.6.724 (13 January 2021)

- New Features
  - Filter the feature tree by Scenario Result
  - Show a warning message when the TestExecution.json contains non matchable items, related to Github#2249
  - **[BuildTask]** Allow to merge multiple TestExecution.json files into one report
- Bugfixes
  - Fix the styling of Markdown tables in Feature/Scenario descriptions Github#2246

34.13 Version 0.6.672 (15 December 2020)

- New Features:
  - None
- Bugfixes:
  - Feature folder paths with trailing slash are correctly handled
  - The result of multiple tasks in the same pipeline is correctly merged

34.14 Version 0.6.669 (08 December 2020)

- New Features:
  - Display scenario step results
  - Display a summary of executed scenarios
  - Display SpecFlow test execution results
  - Display unused step definitions
  - Display summary chart of different step levels
  - Display test execution duration
  - Switch between test and non-test execution results view (toggle)

34.15 Version 0.6.465 (08 October 2020)

- New Features:
  - The "Show Source" button is renamed to "Open Editor"
  - Syntax highlighting for `.feature` files in the built in *Azure Repos editor*
  - Display inherited tags of the feature on the scenarios
- Bugfixes:
  - Fix linking to other feature files when the filename and the feature title are different
  - "Open Editor" button is working correctly when the branch name contains slashes (/)

34.16 Version 0.6.437 (24 September 2020)

- New Features:
  - The feature tree *filtering and coloring* is improved
  - New Share link icon for features and scenarios
  - **[BuildTask]** The error message is displayed when the data upload fails
- Bugfixes:
  - **[BuildTask]** The build tasks stopped updating the LivingDoc if the size of the feature files were too big

34.17 Version 0.6.418 (18 September 2020)

- New Features:
  - Add support for link to other feature files from the feature/scenario markdown description
  - *Download* the Living Documentation as a HTML
  - Small UI improvements
- Bugfixes:
  - None

34.18 Version 0.6.327 (25 August 2020)

- New Features:
  - None
- Bugfixes:
  - Fixed: The feature tree is empty if it contains a Scenario Outline without Examples

34.19 Version 0.6.323 (25 August 2020)

- New Features:
  - Small UI improvements
- Bugfixes:
  - None

34.20 Version 0.6.232 (25 July 2020)

- New Features:
  - None
- Bugfixes:
  - Error handling and logging improvements

34.21 Version 0.6.201 (07 May 2020)

- New Features:
  - None
- Bugfixes:
  - Dependency updates

34.22 Version 0.6.198 (29 April 2020)

- New Features:
  - None
- Bugfixes:
  - Error handling and logging improvements

34.23 Version 0.6.192 (21 April 2020)

- New Features:
  - The feature tree *filtering* is improved
- Bugfix
  - When all the branches with LivingDoc data are deleted the repository is still selectable, but no branch selector is displayed

34.24 Version 0.6.177 (30 March 2020)

- New Features:
  - SpecFlow Account integration
- Bugfix
  - None

34.25 Version 0.6.137 (11 March 2020)

- New Features:
  - None
- Bugfixes:
  - Error handling and logging improvements

34.26 Version 0.6.135 (10 March 2020)

- New Features:
  - None
- Bugfixes:
  - Error handling and logging improvements

34.27 Previous versions

The changes log are published in the SpecFlow BDD Blog

CHAPTER 35

LivingDoc Generator

35.1 Version 3.9.57 (10 September 2021)

- Bugfixes
  - Fixed security issue in markdown parsing

35.2 Version 3.9.50 (26 August 2021)

- Remove feedback buttons

35.3 Version 3.9.35 (30 July 2021)

- Bugfixes:
  - **[CLI]** Examples table with duplicate column headers should throw an exception Github#2471

35.4 Version 3.9.30 (26 July 2021)

- New Features
  - **[LivingDocPlugin]** Extend available TestExecution.json placeholders with timestamp

35.5 Version 3.9.5 (18 June 2021)

- Update SpecFlow logo
- Bugfixes:

- **[CLI]** Fix work item linking bug in the BuildTask, related to Github#2333

35.6 Version 3.8.35 (11 May 2021)

- New Features
  - **[LivingDocPlugin]** Collect outputs (text/attachments)
  - **[CLI]** Extend with option to include/exclude test outputs
- Bugfixes:
  - **[LivingDocPlugin]** No TestExecution.json generated for netcoreapp2.1

35.7 Version 3.7.141 (26 March 2021)

- New Features
  - **[CLI]** Keep parsed data in feature-data mode

35.8 Version 3.7.43 (08 March 2021)

- Bugfixes:
  - **[CLI]** has now both a netcoreapp3.1 and net5.0 version in the NuGet package

35.9 Version 3.7.10 (17 February 2021)

- New Features
  - **[LivingDocPlugin]** Support for SpecFlow 3.7
  - **[CLI]** Improve the handling of the feature-data input source
- Bugfixes
  - **[LivingDocPlugin]** Fix for TestExecution.json file generation with XUnit running on .net 4.8 by introducing a placeholder for the current directory

35.10 Version 3.6.6 (25 January 2021)

- New Features
  - **[LivingDocPlugin]** Support for SpecFlow 3.6

35.11 Version 3.5.286 (15 January 2021)

- New Features
  - **[LivingDocPlugin]** Remove the automatic SpecFlow+Runner isolation mode detection and add placeholder support in the generated TestExecution.json file name

35.12 Version 3.5.279 (13 January 2021)

- New Features
  - **[Generated HTML]** Filter the feature tree by Scenario Result
  - **[Generated HTML]** Show a warning message when the TestExecution.json contains non matchable items, related to Github#2249
  - **[CLI]** Allow to merge multiple TestExecution.json files into one report
  - **[CLI]** Automatically create folders for report output path
  - **[LivingDocPlugin]** Generate multiple TestExecution.json files when using with SpecFlow+Runner in Process isolation mode
- Bugfixes
  - **[Generated HTML]** Fix the styling of Markdown tables in Feature/Scenario descriptions Github#2246
  - **[Generated HTML]** Fix the navigation for features with the same feature title Github#2247
  - **[LivingDocPlugin]** Fix the duration reporting of the executed steps

35.13 Version 3.5.186 (08 December 2020)

- New features:
  - **[Generated HTML]** *Display unused step definitions*
  - **[Generated HTML]** Display summary chart of different step levels
  - **[Generated HTML]** Display test execution duration
  - **[Generated HTML]** Switch between test and non-test execution results view (toggle)
  - **[CLI]** Enable generation of documentation without test execution
  - **[CLI]** Coloring LivingDoc.Plugin CLI output for successfully generated LivingDoc (JSON/HTML)
- Bugfixes:
  - **[LivingDocPlugin]** Fix NuGet warning about version conflict when creating new SpecFlow project

35.14 Version 3.4.242 (21 October 2020)

- New Features:
  - **[Generated HTML]** Visualize step execution results
  - **[Generated HTML]** Test Result Summary tab is extended to show the Scenario execution summary
- Bugfixes:
  - **[LivingDocPlugin]** Using the Rule keyword no longer crashes the LivingDocPlugin Github#2162
  - **[CLI]** Fix HTML tags encoding in Scenario/Feature descriptions Github#2147

35.15 Version 3.4.211 (08 October 2020)

- New Features:
  - **[Generated HTML]** Display inherited tags of the feature files on the scenarios
- Bugfixes:
  - None

35.16 Version 3.4.192 (01 October 2020)

- New Features:
  - **[Generated HTML]** Small UI improvements
- Bugfixes:
  - **[Generated HTML]** Fix linking to other feature files when the filename and the feature title are different

35.17 Version 3.4.167 (24 September 2020)

- New Features:
  - **[Generated HTML]** The feature tree *filtering and coloring* is improved.
- Bugfixes:
  - None

35.18 Version 3.4.133 (17 September 2020)

- New Features:
  - **[Generated HTML]** Add support for link to other feature files from the feature/scenario markdown description.
  - **[Generated HTML]** Add a new Test Result Details tab
  - **[CLI]** Support for custom report titles
  - **[LivingDocPlugin]** Extend the generated FeatureData.json with Step level execution results
- Bugfixes:
  - None

35.19 Version 3.4.2 (19 August 2020)

- New Features:
  - **[Generated HTML]** Show the *test execution results*
  - **[LivingDocPlugin]** Extend the generated FeatureData.json with Scenario level execution results
- Bugfixes:

  - None

35.20 Version 3.3.48 (07 August 2020)

First stable version.

CHAPTER 36

Known issues and limitations

The following only applies to SpecFlow.LivingDoc Generator CLI versions lower than 3.8.35 with SpecFlow, SpecFlow.NUnit, SpecFlow.xUnit, and SpecFlow.MSTest versions lower than 3.8.7:

- Folder names containing spaces are replaced with underscores in the tree view
- Feature file names containing a dot will be treated as a folder name e.g. `Example app.config.feature` will be dispalyed as `Example app\config.feature`