# SpecFlow+ LivingDoc

**Sep 24, 2020**

# Installation on Azure DevOps

SpecFlow+LivingDoc is a set of tools that allows you to share and collaborate on Gherkin Feature Files with stakeholders that are not familiar with developer tools (such as Visual Studio).

- *SpecFlow+LivingDoc for Azure DevOps*: an Azure DevOps extension for browsing Gherkin Feature Files in a convenient web interface
- *SpecFlow+LivingDoc Generator*: a SpecFlow plugin and command line tool to generate a self-hosted HTML documentation of Gherkin Feature Files

The Gherkin files are formatted for readability, including Gherkin syntax highlighting, colour-coding and tables for example data.

Sample output:

**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 |
| To Whom it May Concern - How to Write Irate Complaintes | Lester van Bueren | Trashcorp PLC |

When I enter the search term "blue screen"

Then the search results should contain

| Title | Author | Publisher |
|---|---|---|
| Blue Screens for Numbskulls | Ben Dover | Compuscreen Publishing Ltd |

CHAPTER 1

# Installation of the Azure DevOps extension

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 the Microsoft documentation for more information:Installing Azure DevOps ExtensionsInstalling Azure DevOps Server Extensions
3. You should now see the “SpecFlow+” menu item under “Overview” in each of your projects.

CHAPTER 2

# Generating Documentation

In order to generate your living documentation from your feature files, you need to:

1. Define a build step that references your project.
2. Build the documentation.

You can then *view* your documentation by selecting **Overview | SpecFlow+** from the menu in DevOps.

CHAPTER 3

# 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.

A default build step for is included when installing the extension. **This build step only generates the documentation; it does not execute any tests or build your solution.**

For more information on configuring 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

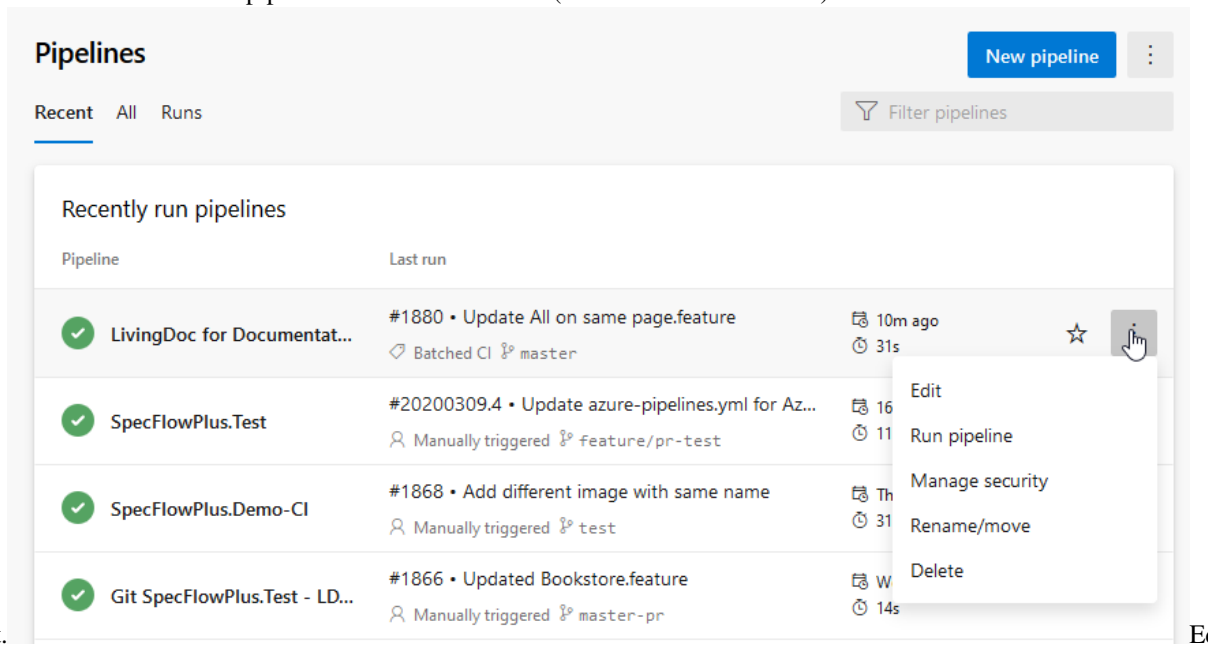
**Note:** You do not need to use DevOps/VSTS/TFS to actually build your application. You can simply add a build definition that acquires the sources and generates the documentation.

CHAPTER 4

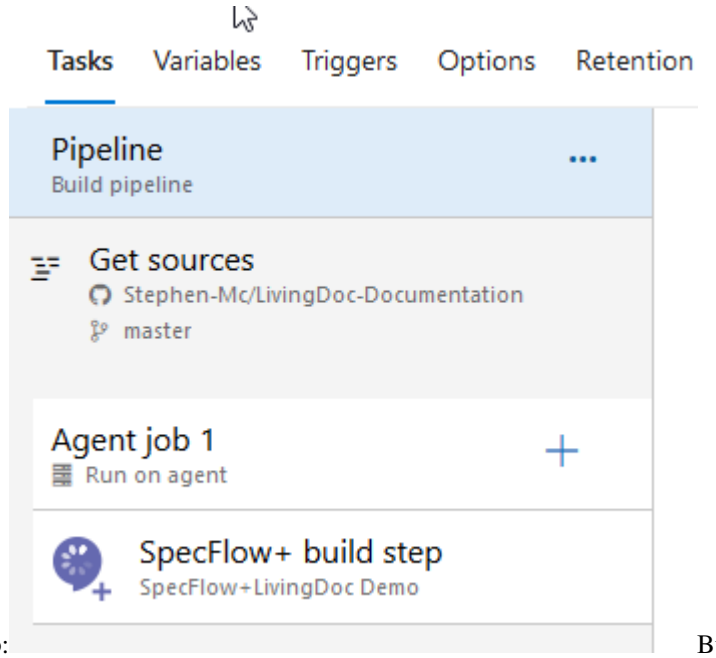
# Configuring the Build Step in DevOps

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** to edit it.




Edit build




3. The current tasks are displayed on the left in the **Tasks** tab: Build Steps
4. Click on the plus icon next to your agent to add a new build step.
5. Add the SpecFlow+ LivingDoc build step to your build to generate the living documentation.
6. Enter the path to your feature files in the **Feature folder** field in the **SpecFlow+ build step**. All feature files in the selected folder and all its sub-folders are included in your living documentation.
7. 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 instead.
8. Select the language used by your Gherkin files under **Project Language**. (If you have referenced a .csproj file, this information is read from the app.config file.)
9. If you have added *links* to Azure DevOps work items to 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.
10. Make sure the step is enabled under **Control Options**.
11. If you want to include Gherkin files from multiple projects, add a separate build step for each of your projects.

Once you have defined your build, you are ready to build your documentation by queuing the build.

CHAPTER 5

# Configuring the Build Step in YAML

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.

SpecFlow+LivingDoc custom build step YAML example:

```
 steps:
-task: techtalk.techtalk-specflow-plus.specflow-plus.SpecFlowPlus@0
  displayName: 'SpecFlow+ build step SpecFlow.Plus.Runner.Specs'
  inputs:
    projectFilePath: Tests/SpecFlow.Plus.Runner.Specs/Features
    projectName: testName
    projectLanguage: en
    workItemPrefix: testPrefix
  enabled: false
  continueOnError: true
  condition: always()
  timeoutInMinutes: 10
```

- **task:** techtalk.techtalk-specflow-plus.specflow-plus.SpecFlowPlus@0
  - *@value is the task version number*
- **displayName:** (required)

## 5.1 LivingDoc specific parameters (inputs):

- **projectFilePath:** The folder containing the feature files or the path of a .NET Full FrameWork project file containing the feature files (required)
- **projectName:** The name of the project visible in the viewer
- **projectLanguage:** Gherkin languages

- **workItemPrefix:** The special prefix for tags used to add a *link* to a work item in Azure DevOps. For example, setting this to "DEVOPS_WI:" will allow you to tag a feature with `@DEVOPS_WI:1234`, and this tag will be converted to a link to work item 1234 when viewing the feature in LivingDoc.

## 5.2 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)

CHAPTER 6

# Building the Documentation

To build the documentation:

1. Configure your pipeline, see Configuring the Build Step in DevOps or Configuring the Build Step in YAML.
2. Queue the build.
3. Once the build has completed, select **Overview | SpecFlow+** from the menu to view your living documentation.

CHAPTER 7

# Markdown and Embedding Images

You can include markdown code in your feature files, and use markdown to embed images in your feature files. These images will then be displayed when viewing the living documentation in Azure DevOps. **Note: Markdown is only supported in Feature and Scenario descriptions.** This means you can only place markdown between the "Feature:" row and the first scenario, or between the "Scenario:" row and the first step definition. If you place markdown elsewhere, you will receive errors when building the documentation.

## 7.1 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.

### 7.1.1 Embedding an image in the same directory as the feature file

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

### 7.1.2 Embedding an image in a sub-directory

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

### 7.1.3 Embedding an image with an absolute reference

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

### 7.1.4 Embedding an image relative to the parent directory

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

### 7.1.5 Embedding an external image

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

### 7.1.6 Example

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

```
Feature: Everybody on the same page
![Swing](https://specflow.org/wp-content/uploads/2017/09/swing-sm.png)
    In order to ensure our software is fit-for-purpose
    As a conscientious software developer
    I want to share specifications with all stakeholders

@mytag
Scenario: Embed image
    Given I have added an embedded image to my feature file using markdown
    When I view the resulting living documentation SpecFlow+LivingDoc
    Then I see the image embedded inline
```

This is the resulting output in SpecFlow+ LivingDoc:

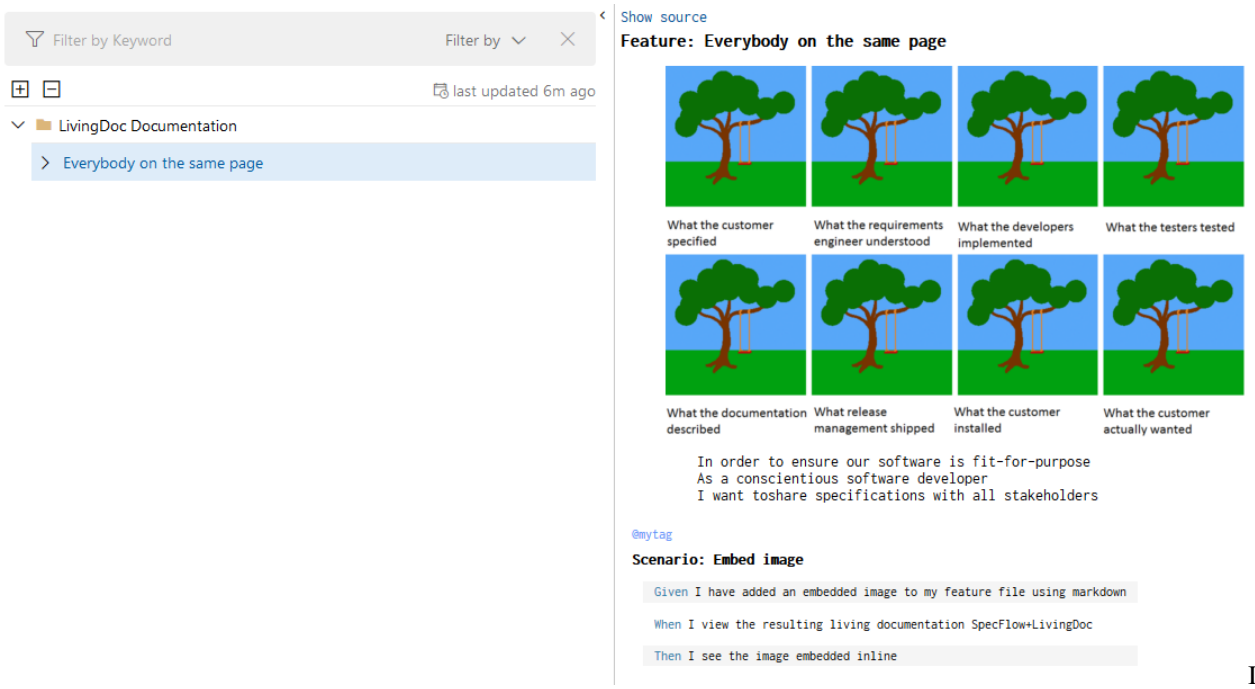



Image in Gherkin

## 7.2 Internal linking between features and scenarios

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

### 7.2.1 Linking a feature in the root directory

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

### 7.2.2 Linking a feature in a sub-directory

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

### 7.2.3 Linking a feature without link text

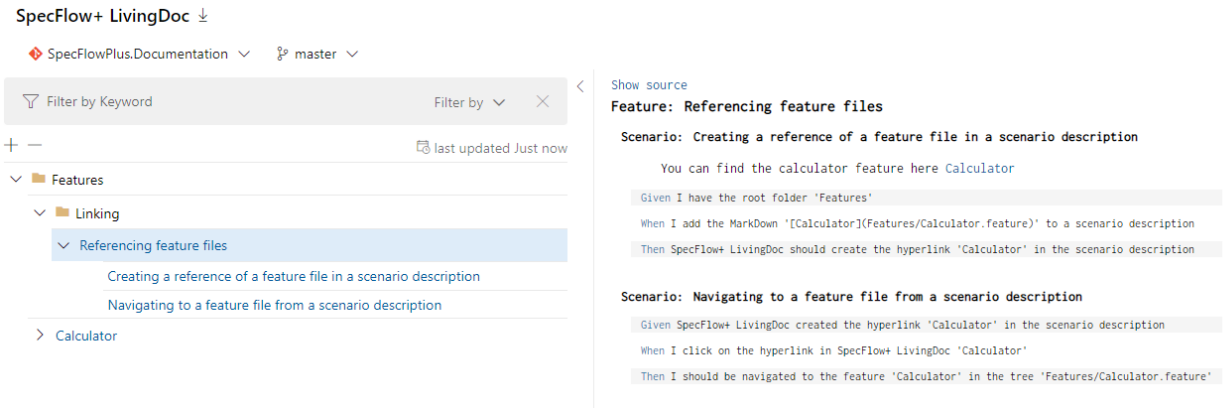
`[](root/Feature.feature)` **Note**: If the text of the link is not specified, the name of the feature will be displayed by default.

### 7.2.4 Linking Example

The following code contains a link to a feature in the Scenario description:

```
Feature: Referencing feature files

Scenario: Creating a reference of a feature file in a scenario description
You can find the calculator feature here [Calculator](Features/Calculator.feature)

    Given I have the root folder 'Features'
    When I add the MarkDown '[Calculator](Features/Calculator.feature)' to a scenario␣
↪description
    Then SpecFlow+ LivingDoc should create the hyperlink 'Calculator' in the scenario␣
↪description

Scenario: Navigating to a feature file from a scenario description

    Given SpecFlow+ LivingDoc created the hyperlink 'Calculator' in the scenario␣
↪description
    When I click on the hyperlink in SpecFlow+ LivingDoc 'Calculator'
    Then I should be navigated to the feature 'Calculator' in the tree 'Features/
↪Calculator.feature'
```

This is the resulting output in SpecFlow+ LivingDoc:




Internal link in Gherkin

CHAPTER 8

# 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.

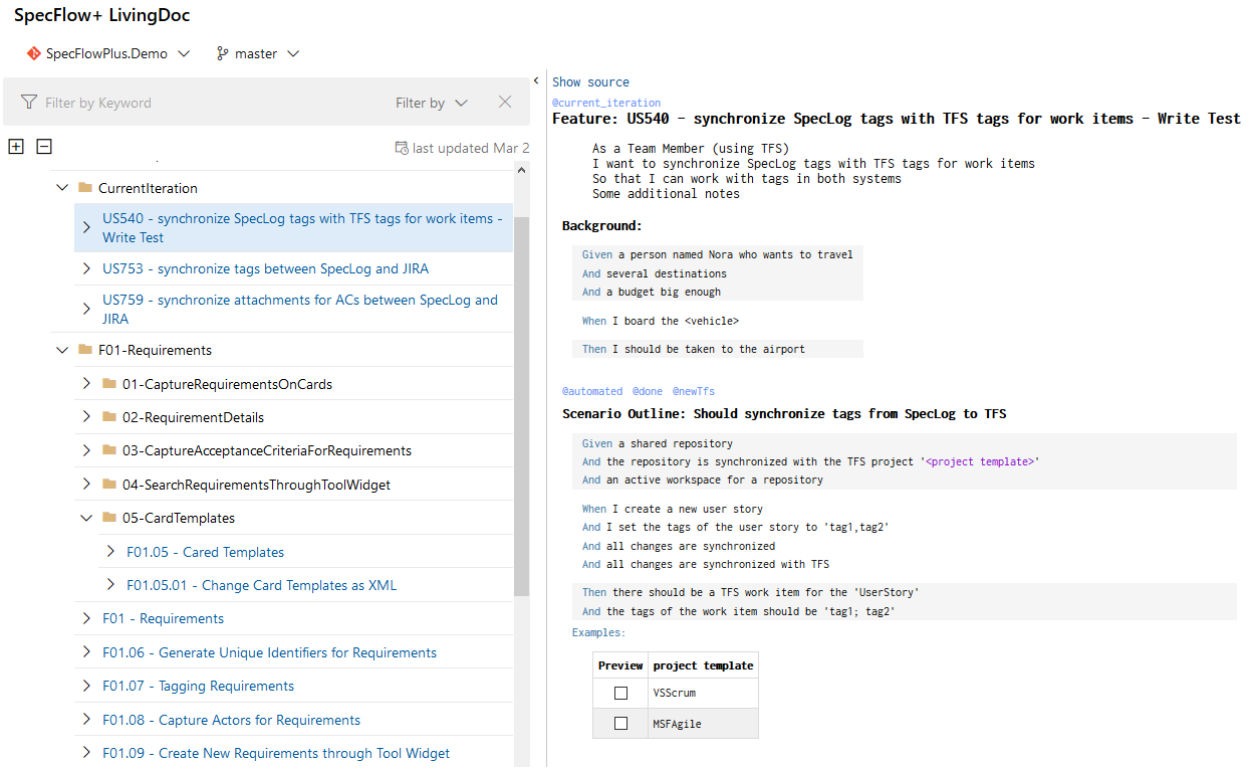
The language used to parse the feature files is determined in the following order:

- 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 9

# Viewing the Living Documentation

Once you have *generated your documentation* using the *SpecFlow+ build step*, select **Overview |SpecFlow+** from the menu to view the documentation.



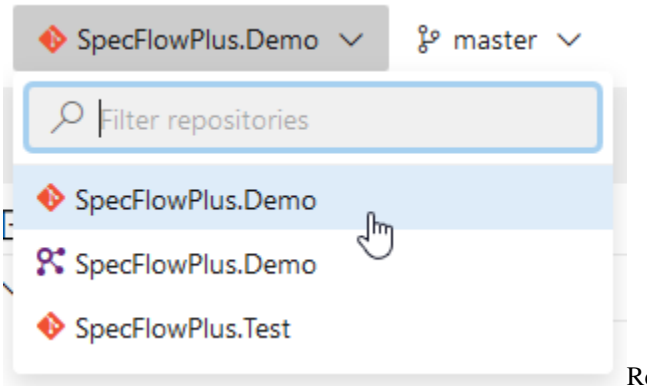

Sample Output

## 9.1 Layout and Navigation

### 9.1.1 Choosing Your Repository and Branch

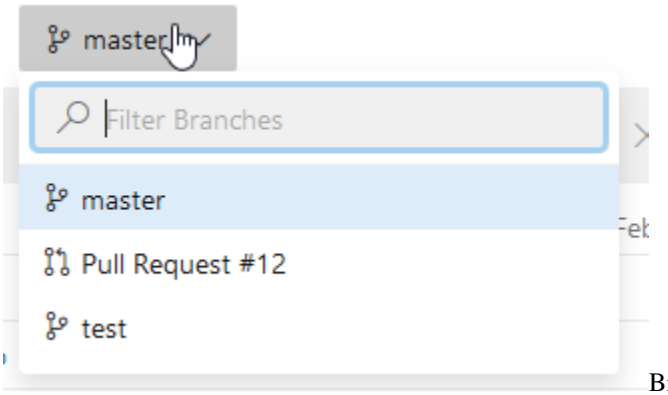
Use the drop-downs at the top of the page to select your repository and branch:

#### Repository selector




Repository selector

#### Branch selector




Branch selector

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.

### 9.1.2 Navigating the Living Documentation

The feature explorer on the left depicts the structure of your specifications. Blue entries represent features and scenarios. Black 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 in the area on the right.

### 9.1.3 Switching to Related Feature Files

If your source code (feature files) is stored in a Azure DevOps or GitHub repository, you can switch directly from the living documentation to the source feature files. You can then edit the feature files directly in Azure DevOps or on GitHub.

**Note:** If your feature files are stored in a source control provider that is not supported, this link is not displayed.

To switch to a feature file:

1. Open the corresponding page in the living documentation.
2. Click on **Show source** at the top of the page.**Note:** If your feature files are stored in a source other than Azure DevOps or GitHub, this option is not available.
3. The corresponding source feature file in your code repository is opened. You can edit the file directly in Azure DevOps or GitHub.**Note:** You may need to merge any conflicts that arise from your edits.

# CHAPTER 10

## Previewing Scenarios with Data Values

Gherkin scenarios often use tables to store a series of test values that are referenced using placeholders in the Gherkin steps. You can see the placeholder (“<preview>”) and a table containing the possible values of the placeholder in the following screenshot:

**Feature: Preview Examples**

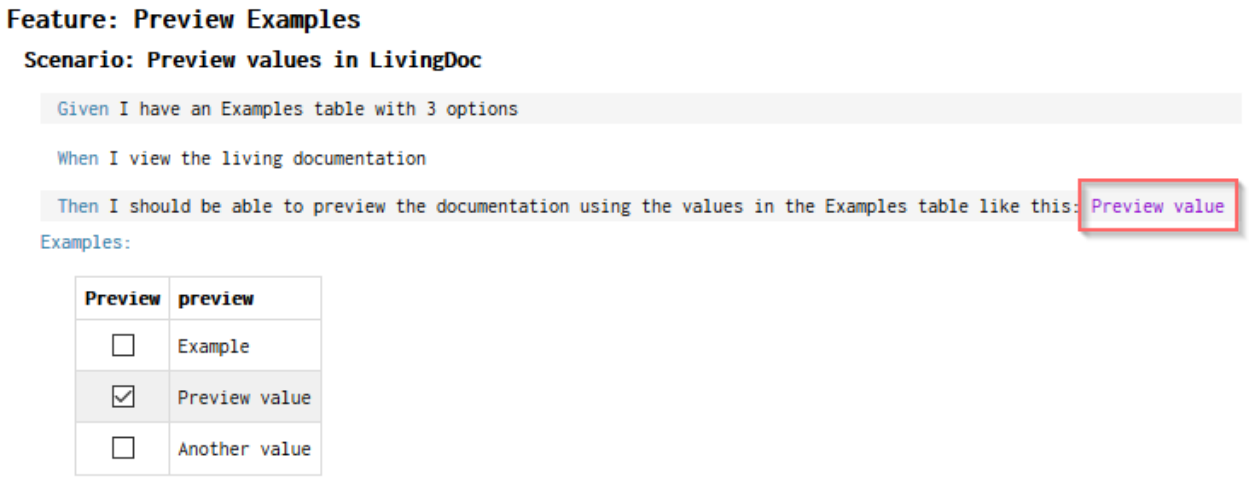
**Scenario: Preview values in LivingDoc**

```
Given I have an Examples table with 3 options
When I view the living documentation
Then I should be able to preview the documentation using the values in the Examples table like this: <preview>
Examples:
```

| Preview | preview |
|---|---|
| ☐ | Example |
| ☐ | Preview value |
| ☐ | Another value |

Placeholder

Use the preview check box in the first column to substitute the placeholders with the values in the corresponding row in the table:




Placeholder with value

CHAPTER 11

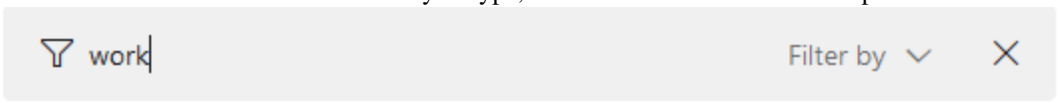
# Searching

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

## 11.1 Searching Within the Entire Documentation

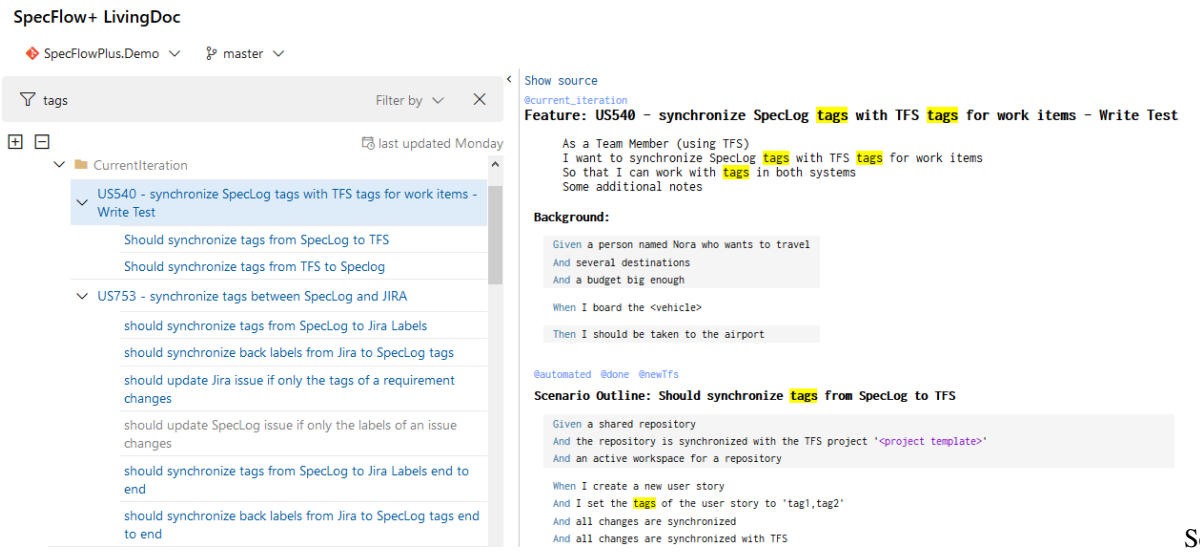
To search within your entire documentation:

1. Enter your search term in the **Filter** field. As you type, the results of the search are updated in the feature explorer.




search term

2. Folders, features and scenarios containing the search term in the titles, tags, descriptions or steps are displayed in blue. Node titles displayed in grey do not contain the search term, but at least one child or parent item is containing the search term. 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 search term is highlighted in the content.

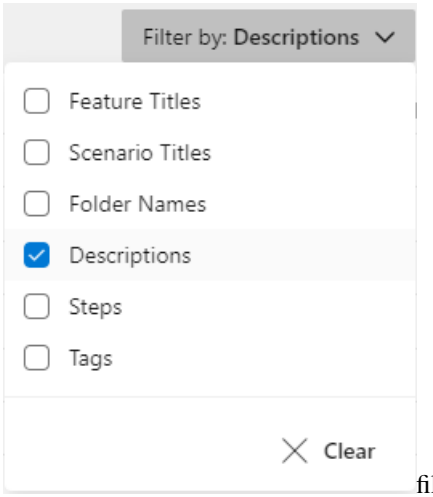



Search

## 11.2 Searching Within Specific Elements

You can restrict the search so that your search terms is only searched for in folder names, titles, descriptions, steps or tags.

To change your search filter:



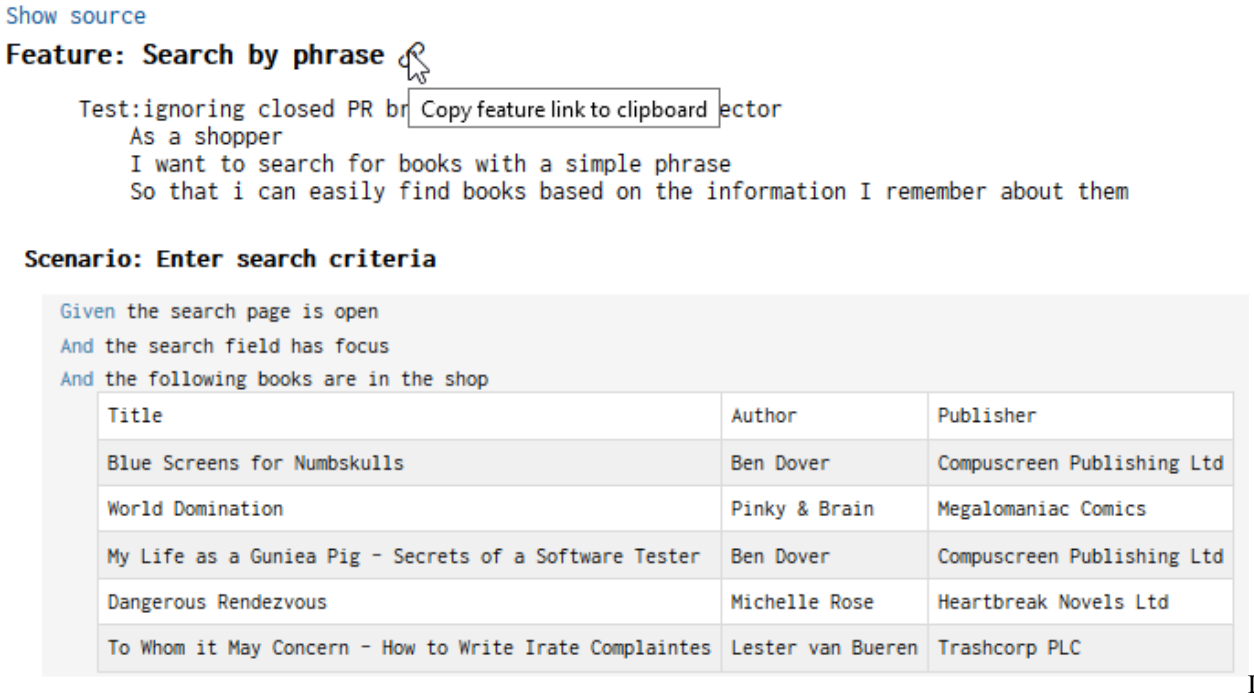

1. Open the **Filter by** drop-down. filter by
2. Select those elements you want to search in from the list.
3. Enter your search term in the **Filter** field. As you type, the results of the search are updated in the tree below.

CHAPTER 12

# Linking

## 12.1 Copying Links to Feature Files

When you hover over a feature or scenario title in the document display area, a link icon appears. Clicking on the icon copies the URL of the selected feature or scenario to the clipboard.



## 12.2 Adding Links to Work Items in Azure DevOps

You can define a prefix for work items in your build task. If you do this, you can create tags that link directly to related work items in Azure DevOps.

For example, if you define a work item prefix as "DEVOPS_WI:" in your *build step*, then you can add tags to your feature files including the work item ID. These tags will be converted to links when parsed by LivingDoc.

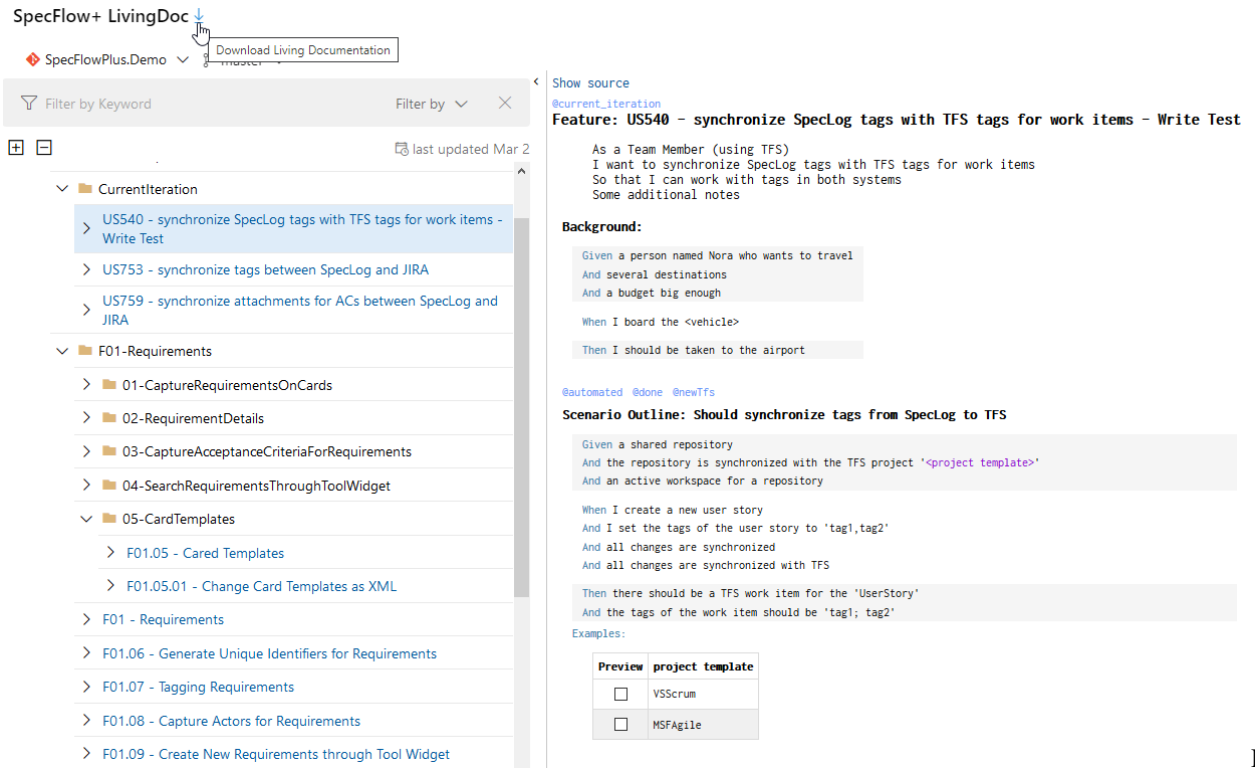
For example `@DEVOPS_WI:1234` will create a link to work item 1234 in LivingDoc.

CHAPTER 13

# Downloading the Living Documentation

You can download your Living Documentation from Azure DevOps.

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.




Download

CHAPTER 14

# Generating Documentation

SpecFlow+LivingDoc Generator allows you to generate a local or self-hosted HTML of your Gherkin Feature Files without the need of Azure DevOps.

In order to generate your living documentation, you need to:

1. Install 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 FeatureData.json next to your TestAssembly which is needed in the next step.
4. Setup and execute the command line tool to create the documentation.

You can then view your documentation by opening the generated LivingDoc.html.

CHAPTER 15

# Setup the LivingDocPlugin

## 15.1 Installation

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

## 15.2 Configuration

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

## 15.3 `livingDocGenerator`

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

**specflow.json example:**

```
{
  "livingDocGenerator": {
    "enabled": true,
    "filepath": "FeatureData.json"
  }
}
```

CHAPTER 16

# Installing the command line tool

## 16.1 Setup

### 16.1.1 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.

## 16.2 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`

## 16.3 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`

## 16.4 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`

## 16.5 Usage of `livingdoc`

Using the command line tool

CHAPTER 17

# Using the command line tool

## 17.1 Name

livingdoc - The generic driver for the LivingDoc Generator CLI.

## 17.2 Arguments

`JSONLocation`

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

## 17.3 Options

- `--output <path>`

  Relative (from the working directory) or abosulte path to the generated HTML. Default value: `LivingDoc.html`

- `--workItemPrefix <prefix>`

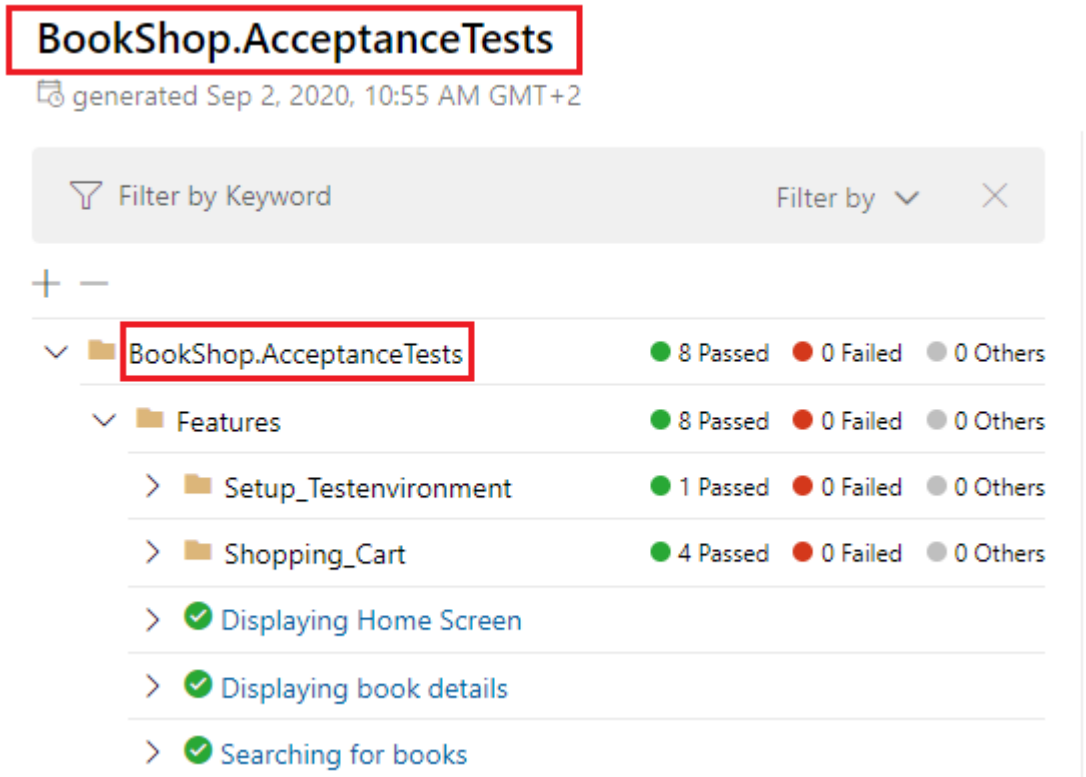
  The special tag you mark the scenarios with to link them to the respecting work items.

- `--workItemUrlTemplate <urlTemplate>`

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

- `--title <documentTitle>`

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




LivingDoc with default title

## 17.4 Examples:

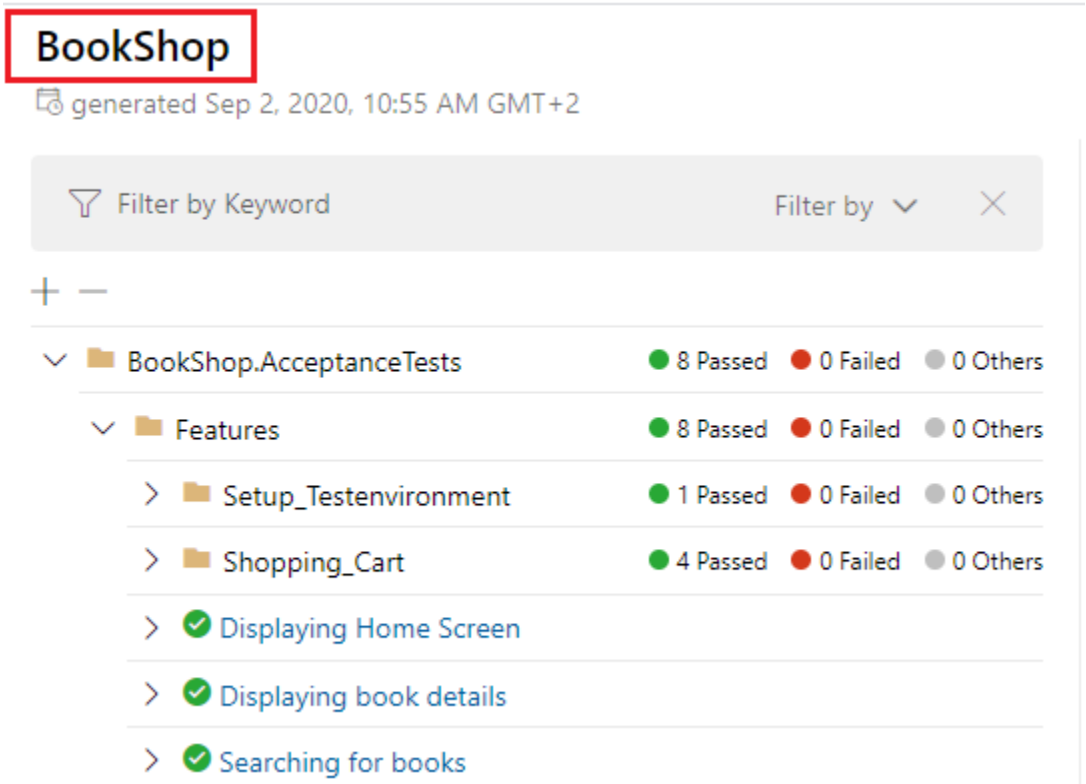
Generate the Living Documentation from the FeatureData.json:

```
livingdoc C:\Work\FeatureData.json
```

```
livingdoc FeatureData.json
```

Generate the Living Documentation with a custom title/header:

```
livingdoc C:\Work\FeatureData.json --title "BookShop"
```




LivingDoc with custom title

Generate the Living Documentation for a specific output path:

```
livingdoc C:\Work\FeatureData.json --output C:\Temp\MyReport.html
```

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

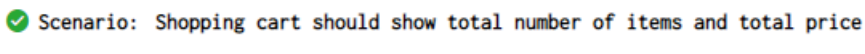
```
livingdoc C:\Work\FeatureData.json --workItemPrefix WI --workItemUrlTemplate https://
↪dev.azure.com/specflow/BookShop/_backlogs/backlog/BookShop%20Team/Stories/?workitem=
↪{id}
```

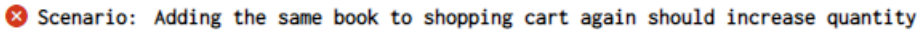
CHAPTER 18

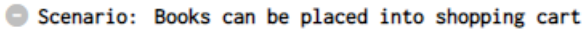
# Viewing test execution results

## 18.1 Test execution status

The state of an execution can be the following:

- Passed 

- Failed 

- Other 

  - Not executed
  - Skipped
  - A step binding is missing
  - A step was marked as Pending

## 18.2 Test result aggregation

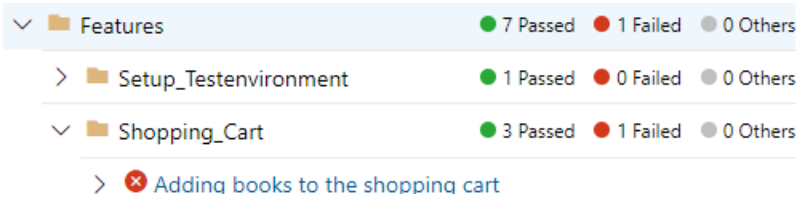
The individual test execution results are aggregated using the following method:

- Scenario
  - Passed: all executions of this scenario have at least one “Passed” or were “Others”
  - Failed: at least one execution of this scenario has “Failed”
  - Others: all executions of this scenario have the state “Others”
- Scenario outline example row
  - Passed: all executions of this scenario outline example row have at least one “Passed” or were “Others”

  - Failed: at least one execution of this scenario outline example row has “Failed”
  - Others: all executions of this scenario outline example row have the state “Others”
- Scenario outline
  - Passed: all executions of its example rows have at least one “Passed” or were “Others”
  - Failed: at least one execution of its example rows has “Failed”
  - Others: all executions of this of its example rows have the state “Others”
- Feature
  - Passed: all executions of the scenarios of this feature have at least one “Passed” or were “Others”
  - Failed: at least one scenario of this feature has “Failed”
  - Others: all executions of the scenarios of this feature have the state “Others”

## 18.3 Test result summary

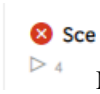
A summary of the execution results are display for every folder in the feature tree:



Summary

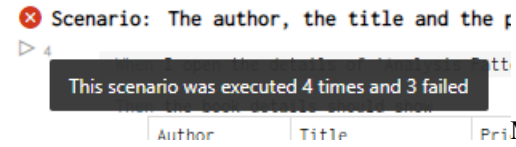
## 18.4 Viewing multiple execution results

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



Multiple Execution Indicator

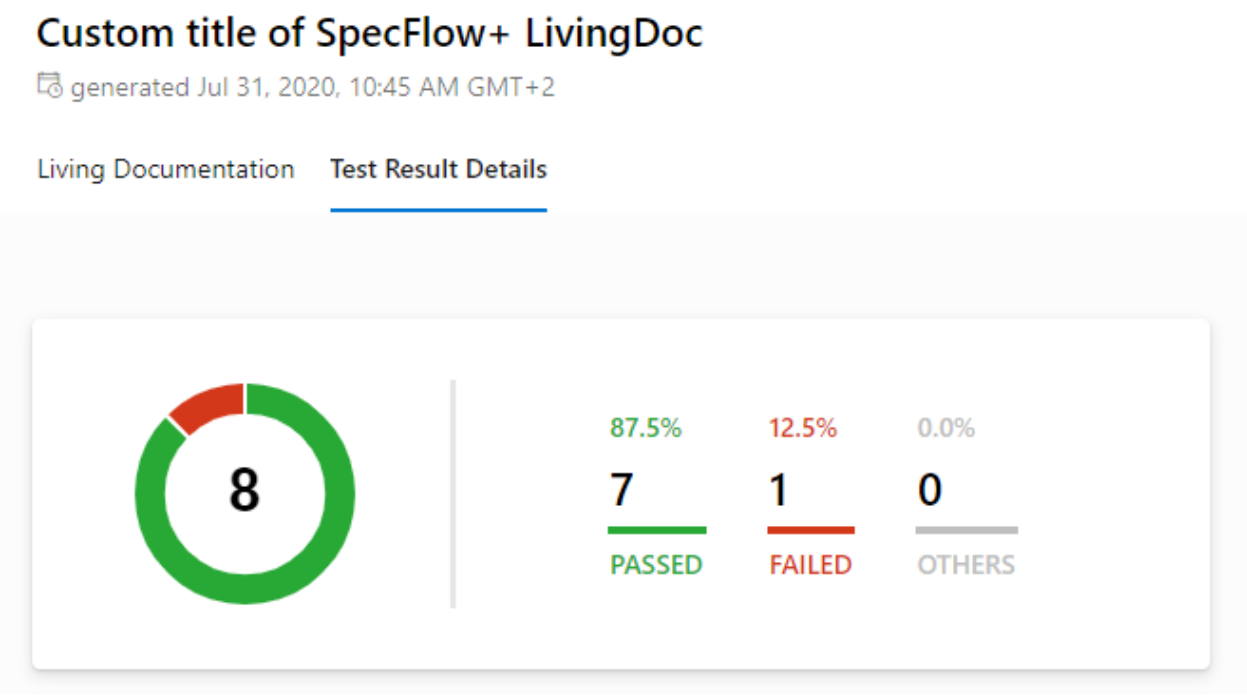
The details of the executions can be seen by hovering over the triangle icon:



Multiple Execution Tooltip

## 18.5 Test result details

If you go to the Test Result Details tab, you can see the test execution results in numbers and percentages:




Multiple Execution Tooltip

CHAPTER 19

# How to share the generated HTML with your team?

The generated HTML is a single page application with no external depedencies, so it is easy to share within your team.

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

- you can use any file sharing solution (Windows file share, Sharpoint, Dropbox etc.)
- you can host the generated HTML with any of the static HTML hosting services (Azure Blob Storage, AWS)
- you can directly store it in your Continuous Integration Server

## 19.1 Sharing with a Continuous Integration Server

### 19.1.1 TeamCity

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

### 19.1.2 Jenkins

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

### 19.1.3 GitLab

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

### 19.1.4 Azure Devops

Currently there is no support for publishing HTML reports from a Build pipeline. You can vote for this feature request: Support for generic HTML Publishing inside Build and Release Pipelines

The recommended approach for Azure Devops is to use our SpecFlow+LivingDoc for Azure Devops extension.

As alternative solutions you can publish the generated HTML as,

- a build artifact using the Publish Build Artifacts task
  - in this case the generated HTML can be downloaded from the Build Result page
- you can publish the HTML to Azure Storage following this article: Deploying a static website to Azure Storage using Azure DevOps

CHAPTER 20

# Known issues and limitations

- Folder names containing spaces are replaced with underscores in the tree view
- When using SpecFlow+Runner only the SharedAppdomain isolation mode is supported
- 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`