

# Cukeddoctor Documentation

Version 2.1-SNAPSHOT

# Table of Contents

<b>1. Introduction</b>	1
<b>2. Features</b>	2
<b>2.1. Cukedoctor Converter</b>	2
2.1.1. Convert features test output into documentation	2
<b>2.2. Ordering</b>	4
2.2.1. Default ordering	5
2.2.2. Custom ordering with tags	7
<b>2.3. Enrich features</b>	9
2.3.1. DocString enrichment activated by the content type	9
2.3.2. DocString enrichment activated by a feature tag	11
2.3.3. DocString enrichment activated by a scenario tag	13
<b>2.4. Documentation introduction chapter</b>	17
2.4.1. Introduction chapter in classpath	17
<b>2.5. Tag rendering</b>	20
2.5.1. Render feature tags in that feature's scenarios	20
2.5.2. Ignore cukedoctor tags in resulting documentation	22

# Chapter 1. Introduction

Cukedoctor is a **Living documentation** tool which integrates Cucumber and AsciiDoctor in order to convert your *BDD* tests results into an awesome documentation.

Here are some design principles:

- Living documentation should be readable and highlight your software features;
  - Most bdd tools generate reports and not a truly documentation.
- Cukedoctor **do not** introduce a new API that you need to learn, instead it operates on top of [cucumber json output](#) files;
  - In the 'worst case' to [enhance](#) your documentation you will need to know a bit of [asciidoc markup](#).

In the subsequent chapters you will see a documentation which is generated by the output of [Cukedoctor's BDD tests](#), **a real bdd living documentation**.

# Chapter 2. Features

## 2.1. Cukedoctor Converter

In order to have awesome *living documentation*

As a bdd developer

I want to use **Cukedoctor** to convert my cucumber test results into **readable** living documentation.

### 2.1.1. Convert features test output into documentation

## Given

The following two features: 🍌 (000ms)

Feature: Feature1

Scenario: Scenario feature 1

Given scenario step

Feature: Feature2

Scenario: Scenario feature 2

Given scenario step

## When

I convert their json test output using cukedoctoer converter 🍌 (013ms)

To generate cucumber .json output files just execute your *BDD* tests with **json** formatter, example:



```
@RunWith(Cucumber.class)
@cucumberOptions(plugin = {"json:target/cucumber.json"})
```



**plugin** option replaced **format** option which was deprecated in newer cucumber versions.

## Then

I should have awesome living documentation 🍌 (001ms)

# Documentation

## Summary

Scenarios			Steps							Features: 2	
Passed	Failed	Total	Passed	Failed	Skipped	Pending	Undefined	Missing	Total	Duration	Status
Feature1											
1	0	1	1	0	0	0	0	0	1	647ms	passed
Feature2											
1	0	1	1	0	0	0	0	0	1	000ms	passed
Totals											
2	0	2	2	0	0	0	0	0	2	647ms	

## Features

### Feature1

#### Scenario: Scenario feature 1

##### Given

scenario step 🏠 (647ms)

### Feature2

#### Scenario: Scenario feature 2

##### Given

scenario step 🏠 (000ms)

## 2.2. Ordering

In order to have features ordered in living documentation  
As a bdd developer  
I want to control the order of features in my documentation

### **2.2.1. Default ordering**

## Given

The following two features: 🍌 (001ms)

Feature: Feature1

Scenario: Scenario feature 1

Given scenario step

Feature: Feature2

Scenario: Scenario feature 2

Given scenario step

## When

I convert them using default order 🍌 (008ms)

## Then

Features should be ordered by name in resulting documentation 🍌 (001ms)

# Features

## Feature1

### Scenario: Scenario feature 1

#### Given

scenario step 🍌 (647ms)

## Feature2

### Scenario: Scenario feature 2

#### Given

scenario step 🍌 (000ms)



### 2.2.2. Custom ordering with tags



Ordering is done using feature tag **@order-**

## Given

The following two features: 🍌 (000ms)

@order-2

Feature: Feature1

Scenario: Scenario feature 1

Given scenario step

@order-1

Feature: Feature2

Scenario: Scenario feature 2

Given scenario step

## When

I convert them using tag order 🍌 (006ms)

## Then

Features should be ordered respecting order tag 🍌 (000ms)

# Features

## Feature2

**Scenario: Scenario feature 2**

### Given

scenario step 🍌 (000ms)

## Feature1

**Scenario: Scenario feature 1**

### Given

scenario step 🍌 (001ms)

## 2.3. Enrich features

In order to have awesome *living documentation*  
As a bdd developer  
I want to render asciidoc markup inside my features.

Asciidoc markup can be used in feature **DocStrings**. To do so you can enable it by using **@asciidoc** tag at **feature** or **scenario** level.

Adding @asciidoc tag at **feature level** will make cukedocter interpret **all features docstrings** as Asciidoc markup.



Adding @asciidoc at **scenario level** will make cukedocter interpret **all steps docstrings** as asciidoc markup.



To enable asciidoc markup in a **single step** you can use **asciidoc** as **docstring content type**.



Feature and scenario descriptions are automatically interpreted as Asciidoc markup without the need for adding the feature tag.

### 2.3.1. DocString enrichment activated by the content type

Asciidoc markup can be used in feature **DocStrings**. To do so you can enable it by using the content type **[asciidoc]** in the DocString.

## Given

The following two features: 🍷 (000ms)

### Feature: Discrete class feature

Scenario: Render source code

Given the following source code in docstrings

```
"""asciidoc
```

```
[source, java]
-----
public int sum(int x, int y){
  int result = x + y;
  return result; (1)
}
-----
```

<1> We can have callouts in living documentation

```
"""
```

Scenario: Render table

Given the following table

```
"""asciidoc
```

```
|===
| Cell in column 1, row 1 | Cell in column 2, row 1
| Cell in column 1, row 2 | Cell in column 2, row 2
| Cell in column 1, row 3 | Cell in column 2, row 3
|===
```

```
"""
```

## When

I convert enriched docstring with asciidoc content type using cukedoctoer converter 🍌 (014ms)

## Then

DocString asciidoc output must be rendered in my documentation 🍌 (001ms)

# Features

## Discrete class feature

### Scenario: Render source code

#### Given

the following source code in docstrings 🍌 (002ms)

```
public int sum(int x, int y){  
    int result = x + y;  
    return result; (1)  
}
```

① We can have callouts in living documentation

### Scenario: Render table

#### Given

the following table 🍌 (000ms)

Cell in column 1, row 1	Cell in columnn 2, row 1
Cell in column 1, row 2	Cell in columnn 2, row 2
Cell in column 1, row 3	Cell in column 2, row 3

### 2.3.2. DocString enrichment activated by a feature tag

Asciiidoc markup can be used in feature **DocStrings**. You can enable this by applying the tag `[@asciidoc]` to the feature. Note this enables the enrichment for all DocStrings within the feature.

## Given

The following two features: 🍷 (000ms)

@asciidoc

Feature: Discrete class feature

Scenario: Render source code

Given the following source code in docstrings

"""

```
[source, java]
-----
public int sum(int x, int y){
    int result = x + y;
    return result; (1)
}
-----
```

<1> We can have callouts in living documentation

"""

Scenario: Render table

Given the following table

"""

```
|==
| Cell in column 1, row 1 | Cell in column 2, row 1
| Cell in column 1, row 2 | Cell in column 2, row 2
| Cell in column 1, row 3 | Cell in column 2, row 3
|==
```

"""

## When

I convert enriched docstring with asciidoc feature tag using cukedoctoer converter 🍌 (011ms)

## Then

DocString asciidoc output must be rendered in my documentation 🍌 (001ms)

# Features

## Discrete class feature

### Scenario: Render source code

#### Given

the following source code in docstrings 🍌 (011ms)

```
public int sum(int x, int y){  
    int result = x + y;  
    return result; (1)  
}
```

① We can have callouts in living documentation

### Scenario: Render table

#### Given

the following table 🍌 (000ms)

Cell in column 1, row 1	Cell in column 2, row 1
Cell in column 1, row 2	Cell in column 2, row 2
Cell in column 1, row 3	Cell in column 2, row 3

### 2.3.3. DocString enrichment activated by a scenario tag

Asciiidoc markup can be used in feature **DocStrings**. You can enable this by applying the tag `[@asciidoc]` to the scenario. Note this enables the enrichment for all DocStrings within the scenario.

**Given**

The following two features: 👍 (000ms)



## Feature: Discrete class feature

```
@asciidoc
```

Scenario: Render source code

Given the following source code in docstrings

```
"""
```

```
[source, java]
-----
public int sum(int x, int y){
    int result = x + y;
    return result; (1)
}
-----
```

<1> We can have callouts in living documentation

```
"""
```

```
@asciidoc
```

Scenario: Render table

Given the following table

```
"""
```

```
|===
| Cell in column 1, row 1 | Cell in column 2, row 1
| Cell in column 1, row 2 | Cell in column 2, row 2
| Cell in column 1, row 3 | Cell in column 2, row 3
|===
```

```
"""
```

## When

I convert enriched docstring with asciidoc scenario tag using cukedoctor converter 🍌 (008ms)

## Then

DocString asciidoc output must be rendered in my documentation 🍌 (000ms)

# Features

## Discrete class feature

### Scenario: Render source code

#### Given

the following source code in docstrings 🍌 (002ms)

```
public int sum(int x, int y){  
    int result = x + y;  
    return result; (1)  
}
```

① We can have callouts in living documentation

### Scenario: Render table

#### Given

the following table 🍌 (000ms)

Cell in column 1, row 1	Cell in column 2, row 1
Cell in column 1, row 2	Cell in column 2, row 2
Cell in column 1, row 3	Cell in column 2, row 3

## 2.4. Documentation introduction chapter

In order to have an introduction chapter in my documentation  
As a bdd developer  
I want to be able to provide an asciidoc based document which  
introduces my software.

### 2.4.1. Introduction chapter in classpath

## Given

The following two features: 🍌 (000ms)

Feature: Feature1

Scenario: Scenario feature 1

Given scenario step

Feature: Feature2

Scenario: Scenario feature 2

Given scenario step

## And

The following asciidoc document is on your application classpath 🍌 (010ms)

= \*Introduction\*

Cukedoctor is a *\*Living documentation\** tool which integrates Cucumber and Asciidoctor in order to convert your `_BDD_` tests results into an awesome documentation.

Here are some design principles:

\* Living documentation should be readable and highlight your software features;

\*\* Most bdd tools generate reports and not a truly documentation.

\* Cukedoctor *\*do not\** introduce a new API that you need to learn, instead it operates on top of

[http://www.relishapp.com/cucumber/cucumber/docs/formatters/json-output-formatter\[cucumber json output^\]](http://www.relishapp.com/cucumber/cucumber/docs/formatters/json-output-formatter[cucumber json output^]) files;

\*\* In the 'worst case' to `<<Enrich-features,enhance>>` your documentation you will need to know a bit of [http://asciidoctor.org/docs/what-is-asciidoc/\[asciidoc markup^\]](http://asciidoctor.org/docs/what-is-asciidoc/[asciidoc markup^]).

## When

Bdd tests results are converted into documentation by Cukedoctor 🍌 (000ms)

## Then

Resulting documentation should have the provided introduction chapter 🍌 (000ms)

= \*Documentation\*

== \*Introduction\*

Cukedoctor is a *\*Living documentation\** tool which integrates Cucumber and AsciiDoctor in order to convert your `_BDD_` tests results into an awesome documentation.

Here are some design principles:

- \* Living documentation should be readable and highlight your software features;
- \*\* Most bdd tools generate reports and not a truly documentation.
- \* Cukedoctor *\*do not\** introduce a new API that you need to learn, instead it operates on top of [http://www.relishapp.com/cucumber/cucumber/docs/formatters/json-output-formatter\[cucumber json output^\]](http://www.relishapp.com/cucumber/cucumber/docs/formatters/json-output-formatter[cucumber json output^]) files;
- \*\* In the 'worst case' to `<<Enrich-features,enhance>>` your documentation you will need to know a bit of [http://asciidoctor.org/docs/what-is-asciidoc/\[asciidoc markup^\]](http://asciidoctor.org/docs/what-is-asciidoc/[asciidoc markup^]).

== *\*Summary\**

[cols="12\*^m", options="header,footer"]

|===

3+|Scenarios 7+|Steps 2+|Features: 2

|[green]##Passed\*#

|[red]##Failed\*#

|Total

|[green]##Passed\*#

|[red]##Failed\*#

|[purple]##Skipped\*#

|[maroon]##Pending\*#

|[yellow]##Undefined\*#

|[blue]##Missing\*#

|Total

|Duration

|Status

12+^|\*<<Feature1>>\*

|1

|0

|1

|1

|0

|0

|0

|0

|0

|1

|647ms

|[green]##passed\*#

12+^|\*<<Feature2>>\*

|1

|0

```

|1
|1
|0
|0
|0
|0
|0
|1
|000ms
|[green]#*passed*#
12+^|*Totals*
|2|0|2|2|0|0|0|0|2 2+|647ms
|===

== *Features*

[[Feature1, Feature1]]
=== *Feature1*

==== Scenario: Scenario feature 1

```

## Given

scenario step 👍 (647ms)

=== **Feature2**

==== Scenario: Scenario feature 2

## Given

scenario step 👍 (000ms)

## 2.5. Tag rendering

### 2.5.1. Render feature tags in that feature's scenarios

### Given

The following two features: 🍌 (000ms)

@someTag

Feature: Feature1

@otherTag

Scenario: Scenario feature 1

Given scenario step

@someTag @otherTag

Scenario: Scenario feature 2

Given scenario step

### When

I render the feature 🍌 (006ms)

### Then

the tags displayed under each scenario should not have duplicates 🍌 (000ms)

# Features

## Feature1

### Scenario: Scenario feature 1

tags: @someTag,@otherTag

#### Given

scenario step 🍷 (001ms)

### Scenario: Scenario feature 2

tags: @someTag,@otherTag

#### Given

scenario step 🍷 (000ms)

## 2.5.2. Ignore cukedocter tags in resulting documentation

Cukedocter specific tags like `@asciidoc` and `@order` **should not** be rendered in resulting documentation.



## Given

The following two features: 🍌 (000ms)

@someTag @asciidoc @order-99

Feature: Feature1

@otherTag @asciidoc

Scenario: Scenario feature 1

Given scenario step

@someTag @otherTag

Scenario: Scenario feature 2

Given scenario step

## When

I render the feature 🍌 (005ms)

## Then

Cukedoctor tags should not be rendered in documentation 🍌 (001ms)

# Features

## Feature1

### Scenario: Scenario feature 1

tags: @someTag,@otherTag

#### Given

scenario step 👍 (001ms)

### Scenario: Scenario feature 2

tags: @someTag,@otherTag

#### Given

scenario step 👍 (000ms)