

# Living Documentation

# Table of Contents

1. <b>Introduction</b> .....	1
2. <b>Summary</b> .....	2
3. <b>Features</b> .....	3
3.1. <b>Cukedoctor Converter</b> .....	3
3.1.1. Scenario: Convert features output into documentation .....	3
3.2. <b>Ordering</b> .....	4
3.2.1. Scenario: Default ordering .....	5
3.2.2. Scenario: Custom ordering .....	6
3.3. <b>Enrich features</b> .....	8
3.3.1. Scenario: DocSting enrichment .....	8
3.3.2. Scenario: Comments enrichment .....	9

# 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. Summary

Scenarios			Steps							Features: 3	
Passed	Failed	Total	Passed	Failed	Skipped	Pending	Undefined	Missing	Total	Duration	Status
Cukedoctor Converter											
1	0	1	3	0	0	0	0	0	3	01s 949ms	passed
Ordering											
2	0	2	6	0	0	0	0	0	6	063ms	passed
Enrich features											
2	0	2	6	0	0	0	0	0	6	068ms	passed
Totals											
5	0	5	15	0	0	0	0	0	15	02s 081ms	

# Chapter 3. Features

## 3.1. Cukedoctor Converter

In order to have awesome *living documentation*  
As a bdd developer  
I want to use **Cukedoctor** to handle my cucumber reports

### 3.1.1. Scenario: Convert features output into documentation

#### Given

The following two features: 🍌 (709ms)

Feature: Feature1

Scenario: Scenario feature 1

Given scenario step

Feature: Feature2

Scenario: Scenario feature 2

Given scenario step

#### When

I convert their json output report using cukedoctor converter 🍌 (01s 238ms)

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)

## 3.2. 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

### **3.2.1. Scenario: Default ordering**

### 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 them using default order 🇺🇦 (042ms)

### Then

Features should be ordered by name in resulting documentation 🇺🇦 (000ms)

## Feature1

### Scenario: Scenario feature 1

Given

scenario step 🇺🇦 (647ms)

## Feature2

### Scenario: Scenario feature 2

Given

scenario step 🇺🇦 (000ms)

### 3.2.2. Scenario: Custom ordering



### 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



Ordering is done using feature comment '**order:**'

### When

I convert them using comment order 🇯🇵 (020ms)

### Then

Features should be ordered respecting order comment 🇯🇵 (000ms)

## Feature2

### Scenario: Scenario feature 2

Given

scenario step 🇯🇵 (000ms)

## Feature1

### Scenario: Scenario feature 1

Given

scenario step 🇯🇵 (313ms)

## 3.3. Enrich features

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

### 3.3.1. Scenario: DocSting enrichment

Asciidoc markup can be used in feature **DocStrings**. To do so you need to enable it by using **cukector-discrete** comment on the feature.

*Given*

The following two features: 🍌 (000ms)

Feature: Enrich feature

Scenario: Render source code

```
# cukedocter-discrete
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

```
# cukedocter-discrete
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 docstring enriched json output using cukedoctoer converter 🍌 (030ms)

Then

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

## Discrete class feature

### Scenario: Render source code

Given

the following source code 🍌 (267ms)

```
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

### 3.3.2. Scenario: Comments enrichment

Asciidoc markup can be used in feature comments. To do so you need to surround asciidoc markup by **curly brackets**;

Given

The following feature with asciidoc markup in comments: 🍌 (000ms)

Feature: Calculator

Scenario: Adding numbers

You can *\*asciidoc markup\** in *\_feature\_ #description#*.

NOTE: This is a very important feature!

*# {IMPORTANT: AsciiDoc markup inside *\*steps\** must be surrounded by *\*curly brackets\**.}*

Given I have numbers 1 and 2

*# {NOTE: Steps comments are placed *\*before\** each steps so this comment is for the *\*WHEN\** step.}*

When I sum the numbers

*# {\* this is a list of itens inside a feature step}*

*# {\* there is no multiline comment in gherkin}*

*# {\*\* second level list item}*

Then I should have 3 as result

*When*

I convert enriched feature json output using cukedoctoer 🍌 (037ms)

*Then*

AsciiDoc markup on comments must be rendered in my documentation 🍌 (000ms)

# Calculator

## Scenario: Adding numbers

You can use **asciidoc markup** in *feature* description.



This is a very important feature!

*Given*

I have numbers 1 and 2 👍 (114ms)



AsciiDoc markup inside **steps** must be surrounded by **curly brackets**.

*When*

I sum the numbers 👍 (000ms)



Steps comments are placed **before** each steps so this comment is for the **WHEN** step.

*Then*

I should have 3 as result 👍 (001ms)

this is a list of itens inside a feature step

there is no multiline comment in gherkin

second level list item