# **Cukedoctor Documentation**

Version 1.1.4-SNAPSHOT

## **Table of Contents**

1. Introduction	1
2. Features	2
2.1. Cukedoctor Converter	2
2.1.1. Convert features test output into documentation 🖷	2
2.2. Ordering	5
2.2.1. Default ordering 🖷	5
2.2.2. Custom ordering 📭	8
2.3. Enrich features	11
2.3.1. DocSting enrichment 🖣	11
2.4. Documentation introduction chapter	17
2.4.1. Introduction chapter in classpath 🖣	17

## **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 📭

```
The following two features: ▲ (340ms)
```

```
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 cukedoctor converter **★** (905ms)

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 • (013ms)

## **Documentation**

## **Summary**

Scenarios			Steps							Features: 2	
Passe d	Faile d	Total	Passe d	Faile d	Skipp ed	Pendi ng	Undef ined	Missi ng	Total	Durat ion	Statu s
&a	<pre>&amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;</pre>										
1	0	1	1	0	0	0	0	0	1	647ms	passe d
&a	<pre>&amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;</pre>										
1	0	1	1	0	0	0	0	0	1	000ms	passe d
	Totals										
2	0	2	2	0	0	0	0	0	2	64	7ms

### **Features**

### Feature1

Scenario: Scenario feature 1

### Feature2

Scenario: Scenario feature 2

Given
scenario step ♣ (000ms)

org.junit.ComparisonFailure: expected:<...ure1\*

```
scenario step 🔞 (647ms)
=== Feature2
==== Scenario:] Scenario feature 2
...> but was:<...ure1*
==== Scenario[ feature 1
```

```
Given
  scenario step 🐽 (647ms)
  === Feature2
   ====] Scenario feature 2
     =====
```

### 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 📭

The following two features: 🏚 (000ms)

Feature: Feature1

Scenario: Scenario feature 1

Given scenario step

Feature: Feature2

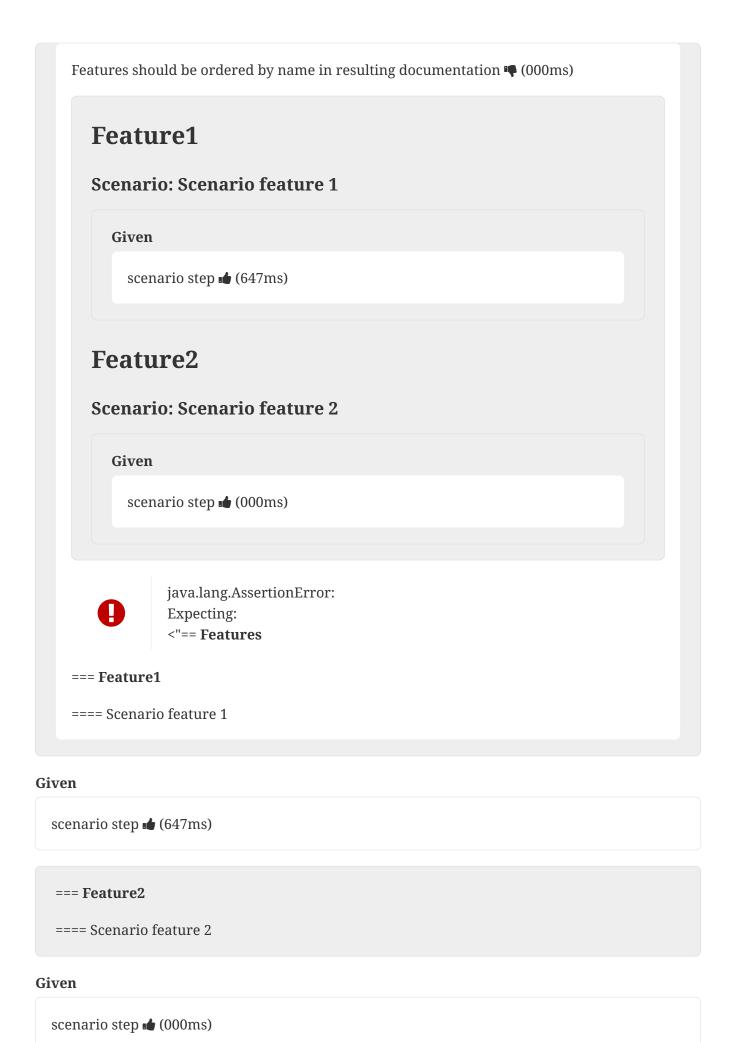
Scenario: Scenario feature 2

Given scenario step

#### When

I convert them using default order 🄞 (033ms)

#### Then



```
">
to contain:
<"
=== Feature1
==== Scenario: Scenario feature 1...
```

### 2.2.2. Custom ordering 📭

The following two features: ๗ (000ms)

#order: 2

Feature: Feature1

Scenario: Scenario feature 1

Given scenario step

#order: 1

Feature: Feature2

Scenario: Scenario feature 2

Given scenario step

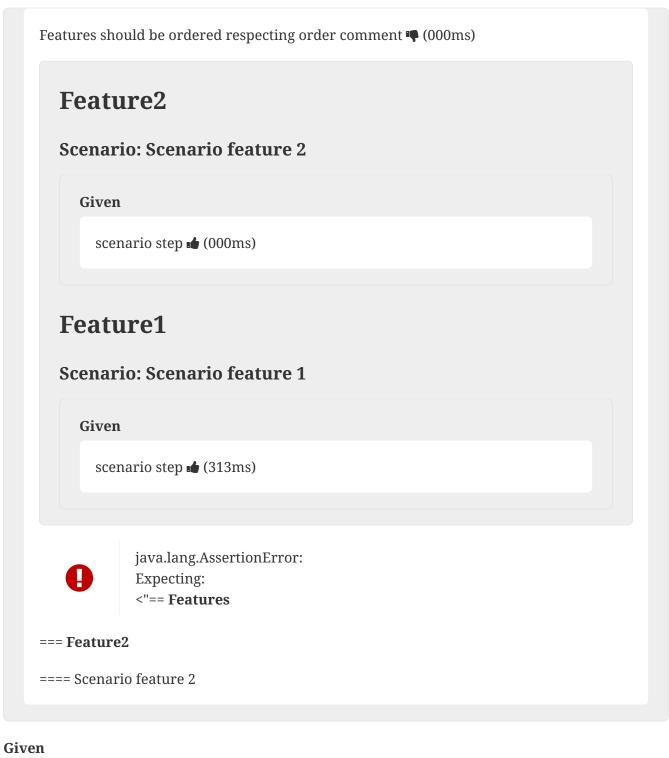


Ordering is done using feature comment '#order:'

#### When

I convert them using comment order **■** (019ms)

#### Then



```
scenario step •• (000ms)

=== Feature1

==== Scenario feature 1
```

scenario step 🕩 (313ms)

```
">
to contain:
<"
=== Feature2
==== Scenario: Scenario feature 2...
```

### 2.3. Enrich features

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

### 2.3.1. DocSting enrichment 📭

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

The following two features: **★** (000ms)

```
Feature: Enrich feature
  Scenario: Render source code
    # cukedoctor-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
    # cukedoctor-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 cukedoctor converter docstring enriched json output using cukedoctor converter (046ms)

#### Then

DocString asciidoc output must be rendered in my documentation ♥ (003ms)

### Discrete class feature

#### Scenario: Render source code

#### Given

the following source code de (267ms)

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

#### 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



java.lang.AssertionError:

Expecting:

<"== Features

=== Discrete class feature

```
public int sum(int x, int y){
   int result = x + y;
   return result; ①
}
① We can have callouts in living documentation>
```

```
==== Render table
```

Give...

==== 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: d (000ms)

```
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 cukedoctor **d** (041ms)

#### 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 id (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



java.lang.AssertionError: Expecting:

<"==Features

#### ===Calculator

====Addingnumbers

Youcanuse\*asciidocmarkup\*in\_feature\_#description#.

NOTE: This is a very important feature!

Ihavenumbers1and2 **★**(114ms)

IMPORTANT: Asciidocmarkupinside\*steps\*mustbesurroundedby\*curlybrackets\*.

#### When

Isumthenumbers (000ms)

NOTE:Stepscommentsarepl...

## 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 📭

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 d (053ms)

## 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;



The introduction file must be named **intro-chapter.adoc** and can be in any package of your application,

By default Cukedoctor will look into application folders but you can make Cukedoctor look into external folder by setting the following system property:



System.setProperty("INTRO\_CHAPTER\_DIR","/home/some/external/f
older");

#### When

Bdd tests results are converted into documentation by Cukedoctor documentation documentation by Cukedoctor documentation documen

#### 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:

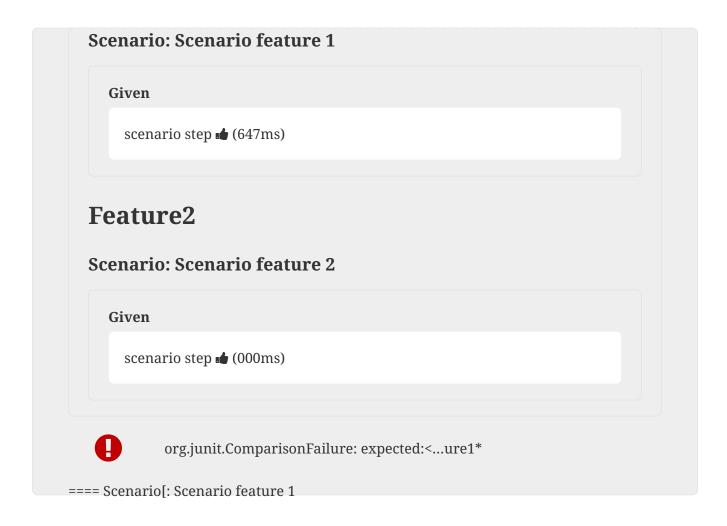
- Living documentation should be readable and highlight your software features;
  - Most bdd tools generate reports and not a truly documentation.
- Cukedoctor **do not** introduce a new API that you need to learn, instead it operates on top of cucumber json output files;

## Summary

Scenar	rios	Steps						Features: 2			
Passe d	Faile d	Total	Passe d	Faile d	Skip ped	Pend ing	Unde fined		Total	Durat ion	Statu s
&amj	< <feature1&< th=""></feature1&<>										
1	0	1	1	0	0	0	0	0	1	647m s	passe d
&amj	< <feature2&< td=""><td>np;am</td></feature2&<>								np;am		
1	0	1	1	0	0	0	0	0	1	000m s	passe d
Totals											
2	0	2	2	0	0	0	0	0	2	647ms	

### **Features**

### Feature1



```
scenario step de (647ms)

=== Feature2

==== Scenario:] Scenario feature 2
...> but was:<...ure1*

==== Scenario[ feature 1
```

#### Given

```
scenario step 🔞 (647ms)
```

=== Feature2		
====] Scenario feature 2		
====		