

# Cukeddoctor Documentation

Version 3.9.1-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
2.3.4. Whitespace in descriptions	15
<b>2.4. Documentation introduction chapter</b>	19
2.4.1. Introduction chapter in classpath	19
<b>2.5. Tag rendering</b>	22
2.5.1. Render feature tags in that feature's scenarios	22
2.5.2. Ignore cukedoctor tags in resulting documentation	24
<b>2.6. Attachments</b>	26
2.6.1. Logging a string in Cucumber-JVM 6.7.0	26
2.6.2. Attaching plain text as a string with name in Cucumber-JVM 6.7.0	27
2.6.3. Attaching plain text as a byte array with name in Cucumber-JVM 6.7.0	28
2.6.4. Attaching a string CucumberJS 6.0.5	29
2.6.5. Attaching a plain text string CucumberJS 6.0.5	30
2.6.6. Attaching a plain text buffer CucumberJS 6.0.5	31
2.6.7. Logged text should appear before attachments	32
2.6.8. Multiple attachments	33
2.6.9. Do not render attachments that are not plain text	34

# 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: 🍇

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 🍇

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 🍇

# 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: 🍌

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 🍌

## Then

Features should be ordered by name in resulting documentation 🍌

# 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: 🍌

@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 🍌

## Then

Features should be ordered respecting order tag 🍌

# 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: 🍌

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 cukedocto converter 🍌

## Then

DocString asciidoc output must be rendered in my documentation 🍌

# 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

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

AsciiDoc 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: 👍

@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 cukedocter converter 👍

## Then

DocString asciidoc output must be rendered in my documentation 👍

# 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

AsciiDoc 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: 🍌

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 cukedocter converter 🍌

## Then

DocString asciidoc output must be rendered in my documentation 🍌



# 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

### 2.3.4. Whitespace in descriptions

Features and Scenarios can have multi-line descriptions. In a feature file, these may be indented. Cukedoctor uses the indentation of the first line non-blank line of the description to determine the difference between the indentation *of* the description in a feature file and your desired indentation *within* the the description itself.

## Given

the feature: 🍌

Feature: Feature One

This is the description for Feature One. The first non-blank line of this description in the feature file began with four whitespace characters.

Therefore, cukedoctoer will ignore up to the first four  
whitespace characters  
in all other lines in the same description,  
if any are present.

This includes  
further lines  
in a different  
paragraph  
in the same description.

Scenario: Scenario One

This is the description for Scenario One. The first non-blank line of this description in the feature file began with four whitespace characters.

Therefore, cukedoctoer will ignore up to the first four  
whitespace characters  
in all other lines in the same description,  
if any are present.

This includes  
further lines  
in a different  
paragraph  
in the same description.

Scenario: Scenario Two

This scenario has no indentation. You don't have to use it, after all.  
Indentation in subsequent lines is therefore fully preserved.

## When

I convert it 🍌

**Then**

it should be rendered in AsciiDoc as 

# Features

## Feature One

This is the description for Feature One. The first non-blank line of this description in the feature file began with four whitespace characters.

```
Therefore, cukedocter will ignore up to the first four
```

whitespace characters

in all other lines in the same description,

if any are present.

This includes

further lines

in a different

```
paragraph
```

```
in the same description.
```

## Scenario: Scenario One

This is the description for Scenario One. The first non-blank line of this description in the feature file began with four whitespace characters.

```
Therefore, cukedocter will ignore up to the first four
```

whitespace characters

in all other lines in the same description,

if any are present.

This includes

further lines

in a different

paragraph

in the same description.

### Scenario: Scenario Two

This scenario has no indentation. You don't have to use it, after all.

Indentation in subsequent lines is therefore fully preserved.

## 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 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/folder");
```

### Given

The following two features: 🍌

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 🍌

## 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](#).

### When

Bdd tests results are converted into documentation by Cukedoctor 🍌

### Then

Resulting documentation should have the provided introduction chapter 🍌

# 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;
  - In the 'worst case' to [enhance](#) your documentation you will need to know a bit of [asciidoc markup](#).

## 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)

## 2.5. Tag rendering

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



## Given

The following two features: 🍌

@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 🍌

## Then

the tags displayed under each scenario should not have duplicates 🍌

# 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 cukedocto tags in resulting documentation

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

## Given

The following two features: 🍌

```
@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 🍌

## Then

Cukedoctor tags should not be rendered in documentation 🍌

# Features

## Feature1

### Scenario: Scenario feature 1

tags: @someTag,@otherTag

#### Given

scenario step 🍌 (001ms)

### Scenario: Scenario feature 2

tags: @someTag,@otherTag

#### Given

scenario step 🍌 (000ms)

## 2.6. Attachments

In order to capture dynamically-generated content from my tests  
As a bdd developer  
I want to render attachments from my Cucumber tests in my living  
documentation

### 2.6.1. Logging a string in Cucumber-JVM 6.7.0

**Given**

a Step has logged a string in Cucumber-JVM 6.7.0 🍊

**And**

I am hiding step timings 🍊

**And**

all Cukedoctor extensions are disabled 🍊

**When**

I convert the Feature 🍊

**Then**

it will be rendered as 🍊

# Features

## Attachments

### Scenario: Cucumber JVM 6.7.0 scenario.log(String)

**Given**

a Step that performs scenario.log(String) 🍊

She sells sea shells on the sea shore

#### 2.6.2. Attaching plain text as a string with name in Cucumber-JVM 6.7.0

**Given**

a Step has attached plain text as a string with a title in Cucumber-JVM 6.7.0 🍌

**And**

I am hiding step timings 🍌

**And**

all Cukedoctor extensions are disabled 🍌

**When**

I convert the Feature 🍌

**Then**

it will be rendered as 🍌

# Features

## Attachments

### Scenario: Cucumber JVM 6.7.0

#### `scenario.attach(String, String, String)`

**Given**

a Step that performs `scenario.attach(String, String, String)` 🍌

*String plain text*

She sells sea shells on the sea shore

### 2.6.3. Attaching plain text as a byte array with name in Cucumber-JVM 6.7.0

**Given**

a Step has attached plain text as a byte array with a title in Cucumber-JVM 6.7.0 🍌

**And**

I am hiding step timings 🍌

**And**

all Cukedoctor extensions are disabled 🍌

**When**

I convert the Feature 🍌

**Then**

it will be rendered as 🍌

# Features

## Attachments

### Scenario: Cucumber JVM 6.7.0

**scenario.attach(ByteArray, String, String)**

**Given**

a Step that performs scenario.attach(ByteArray, String, String) 🍌

*Byte[] plain text*

She sells sea shells on the sea shore

#### 2.6.4. Attaching a string CucumberJS 6.0.5

**Given**

a Step has attached a string in CucumberJS 6.0.5 🍊

**And**

I am hiding step timings 🍊

**And**

all Cukedoctor extensions are disabled 🍊

**When**

I convert the Feature 🍊

**Then**

it will be rendered as 🍊

# Features

## Attachments

### Scenario: Cucumber JS 6.0.5 attach String

**Given**

a Step that performs attach String 🍊

*Attachment 1*

She sells sea shells on the sea shore

#### 2.6.5. Attaching a plain text string CucumberJS 6.0.5



**Given**

a Step has attached plain text as a string in CucumberJS 6.0.5 👍

**And**

I am hiding step timings 👍

**And**

all Cukedoctor extensions are disabled 👍

**When**

I convert the Feature 👍

**Then**

it will be rendered as 👍

# Features

## Attachments

### Scenario: Cucumber JS 6.0.5 attach String, String

**Given**

a Step that performs attach String, String 👍

*Attachment 1*

She sells sea shells on the sea shore

#### 2.6.6. Attaching a plain text buffer CucumberJS 6.0.5

**Given**

a Step has attached plain text as a buffer in CucumberJS 6.0.5 🍊

**And**

I am hiding step timings 🍊

**And**

all Cukedoctor extensions are disabled 🍊

**When**

I convert the Feature 🍊

**Then**

it will be rendered as 🍊

# Features

## Attachments

### Scenario: Cucumber JS 6.0.5 attach Buffer, String

**Given**

a Step that performs attach Buffer, String 🍊

*Attachment 1*

She sells sea shells on the sea shore

#### 2.6.7. Logged text should appear before attachments

**Given**

a Step has logged a string and attached a plain text string with a title 🍌

**And**

I am hiding step timings 🍌

**And**

all Cukedoctor extensions are disabled 🍌

**When**

I convert the Feature 🍌

**Then**

it will be rendered as 🍌

# Features

## Attachments

### Scenario: Log and attach

**Given**

a Step that logs and attaches 🍌

Peter Piper picked a peck of pickled peppers

*String plain text*

She sells sea shells on the sea shore

#### 2.6.8. Multiple attachments

**Given**

a Step has three plain text attachments, two without a title 🍌

**And**

I am hiding step timings 🍌

**And**

all Cukedoctor extensions are disabled 🍌

**When**

I convert the Feature 🍌

**Then**

it will be rendered as 🍌

# Features

## Attachments

### Scenario: Multiple attachments

**Given**

a Step that has multiple attachments 🍌

*Attachment 1*

First attachment

*Second*

Second Attachment

*Attachment 2*

Third attachment

#### 2.6.9. Do not render attachments that are not plain text

**Given**

a Step has logged an image/png attachment 🍇

**And**

I am hiding step timings 🍇

**And**

all Cukedoctor extensions are disabled 🍇

**When**

I convert the Feature 🍇

**Then**

it will be rendered as 🍇

# Features

## Attachments

### Scenario: Attaching an image

**Given**

a Step that attaches an image 🍇