JGiven Report

Table of Contents

All Scenarios

Attachments Example

☑ 7 Successful, ❗ 0 Failed, ⃠ 0 Pending, 7 Total (0s 849ms)

Attachments can be added to steps

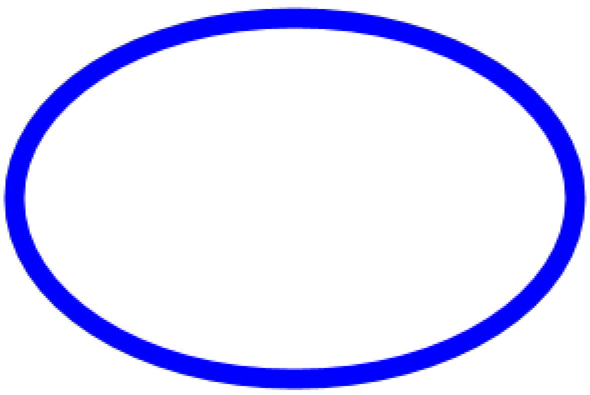
☑ (0s 8ms)

Given some text content *"Hello World"*

Then it can be added as an attachment to the step with title *Hi* 🗋

Attachments can be directly shown

☑ (0s 81ms)


Given an oval circle

Attachments work with data tables

☑ (0s 58ms)

Given some text content <**content**>

Then it can be added as an attachment to the step with title <**title**> 🗋, 🗋, 🗋

Table 1. Cases

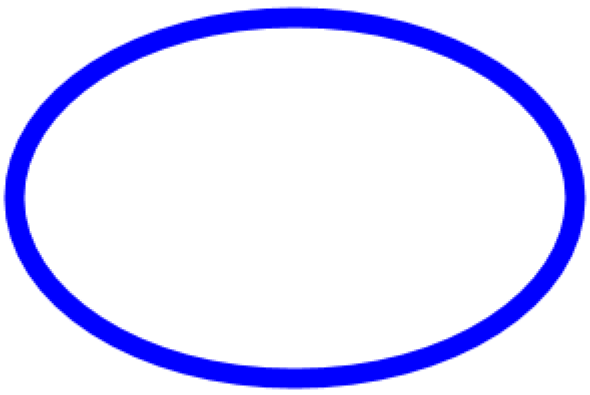
#	content	title	Status
1	"Hello World"	English	☑
2	"Hallo Welt"	German	☑
3	"□□□□"	Chinese	☑

Inline attachments can be used when having multiple cases

☑ (0s 111ms)

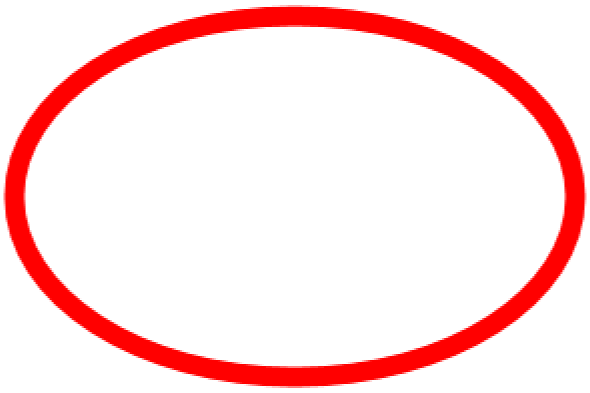
Case 1

```
color = blue
```


Given a *blue* oval circle

Case 2

```
color = red
```

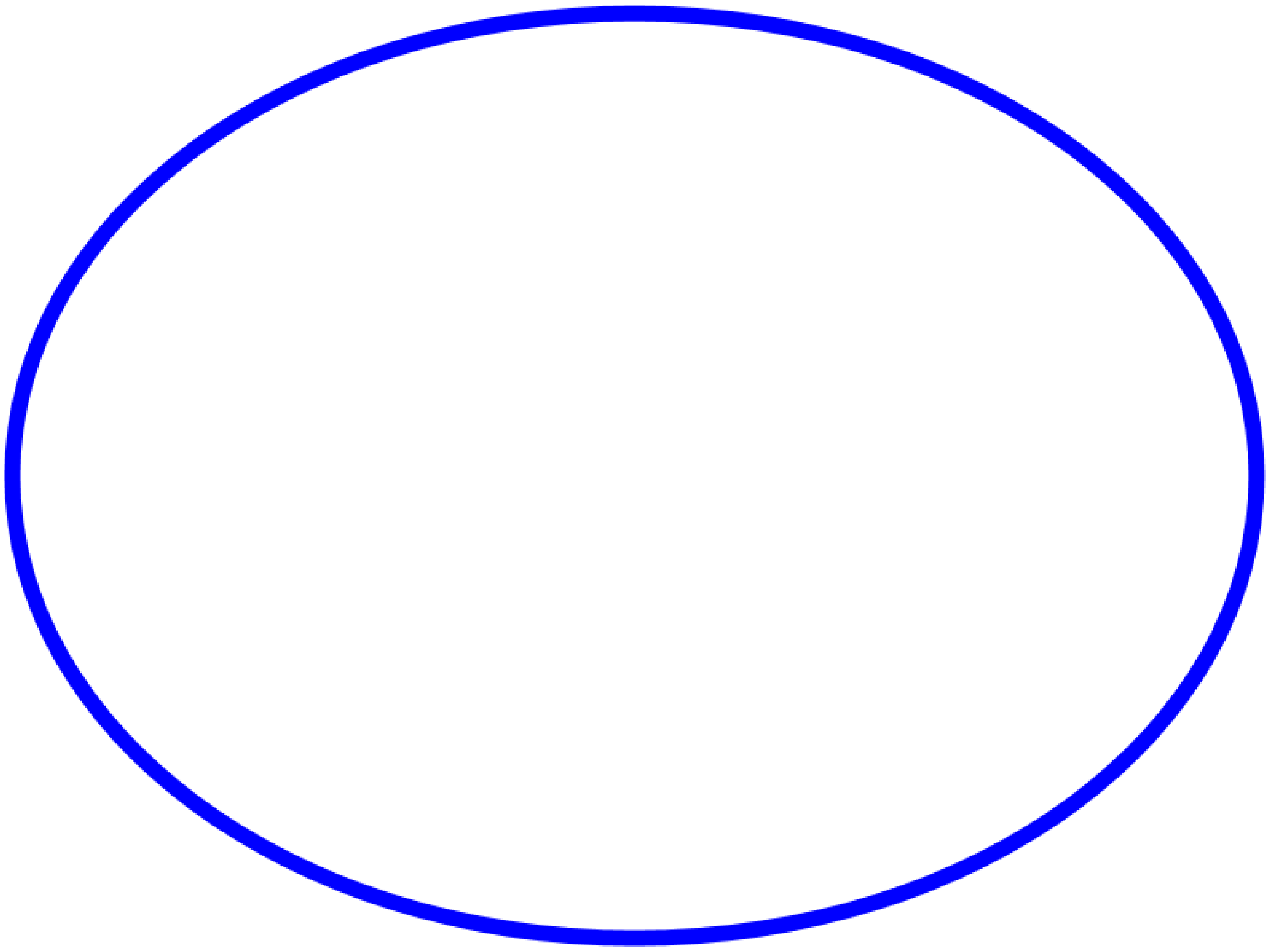


Given a *red* oval circle

Large attachments can be zoomed

☑ (0s 177ms)

Given a large oval circle



Steps can have multiple attachments

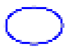
☑ (0s 7ms)

Given some text content *"Hi There"*

Then it can be added as an attachment multiple times to the step 🗋 🗋

Thumbnails are shown when not drawn

☑ (0s 404ms)

Given an oval circle as thumbnail 

Nested Steps

☑ 1 Successful, ❗ 1 Failed, 🛇 0 Pending, 2 Total (0s 7ms)

A scenario with a failing nested step on purpose

❗ (0s 1ms)

Tags: *FailingOnPurpose*

Given I fill out the registration form with invalid values ❗ (0s 1ms)

  I enter a name *Franky* ☑ (0s 0ms)

    I think a name *Franky* ☑ (0s 0ms)

    and I write the name *Franky* ☑ (0s 0ms)

  and I enter a email address *franky@acme.com* ☑ (0s 0ms)

  and something fails for demonstration purposes ❗ (0s 0ms)

When I submit the form 🛇 (0s 0ms)

Then the password matches 🛇 (0s 0ms)

A scenario with nested steps

☑ (0s 5ms)

This scenario contains nested steps.

Given I fill out the registration form with valid values

  I enter a name *Franky*

    I think a name *Franky*

    and I write the name *Franky*

  and I enter a email address *franky@acme.com*

  and I enter a password *password1234*

  and I enter a repeated password *password1234*

When I submit the form

Then the password matches

Pending Example

☑ 0 Successful, ❗ 0 Failed, ⊘ 3 Pending, 3 Total (0s 5ms)

As a good BDD practitioner,<br>I want to write my scenarios before I start coding<br>In order to discuss them with business stakeholders

Multiple cases can be pending

⊘ (0s 3ms)

Tags: *Pending Annotation*

Given some state ⊘ (0s 0ms)

When a <**actionCount**> action ⊘ (0s 0ms)

Then some result ⊘ (0s 0ms)

Table 2. Cases

#	actionCount	Status
1	1st	⊘
2	2nd	⊘

Scenarios that are pending can be annotated with the Pending annotation

⊘ (0s 0ms)

Tags: *Pending Annotation*

Given some state ⊘ (0s 0ms)

When some action ⊘ (0s 0ms)

Then some result ⊘ (0s 0ms)

Single steps can be annotated with Pending

⊘ (0s 1ms)

Tags: *Pending Annotation*

Given some state ☑ (0s 0ms)

When some pending action ⊘ (0s 0ms)

Then some result ☑ (0s 0ms)

Serve Coffee

☑ 11 Successful, ❗ 4 Failed, ⊘ 0 Pending, 15 Total (1s 161ms)

```
In order to refresh myself</br>as a customer</br>I want coffee to be served
```

A failing scenario for demonstration purposes

❗ (0s 3ms)

Tags: *FailingOnPurpose*

Given a coffee machine ☑ (0s 0ms)
An empty coffee machine that is already turned on.<br>The coffee price is set to 2 EUR.

and there are no more coffees left ☑ (0s 0ms)

When I press the coffee button ☑ (0s 0ms)

Then I should be served a coffee ❗ (0s 0ms)

and steps following a failed step should be skipped 🚫 (0s 0ms)
This step is still visible in the report, but was actually not executed. It is marked as skipped in the report.

A failing scenario for demonstration purposes

A scenario with a failing test case for demonstration purposes

❗ (0s 10ms)

Tags: *FailingOnPurpose*

Case 1

```
withCoffees = true
```

Given a coffee machine
An empty coffee machine that is already turned on.<br>The coffee price is set to 2 EUR.

and there are *2* coffees left in the machine
The number of coffees in the machine is set to the given value.

When I insert *2* one euro coins

and I press the coffee button

Then I should be served a coffee

Case 2

```
withCoffees = false
```

Given a coffee machine ☑ (0s 0ms)
An empty coffee machine that is already turned on.<br>The coffee price is set to 2 EUR.

When I insert *2* one euro coins ☑ (0s 0ms)

and I press the coffee button ☑ (0s 0ms)

Then I should be served a coffee ❗ (0s 4ms)

A turned off coffee machine cannot serve coffee

☑ (0s 4ms)

Given a coffee machine
An empty coffee machine that is already turned on.<br>The coffee price is set to 2 EUR.

and the machine is turned off

When I press the coffee button

Then no coffee should be served

An empty coffee machine cannot serve any coffee

☑ (0s 3ms)

Tags: *Order*

Given an empty coffee machine

When I insert *5* one euro coins

and I press the coffee button

Then an error should be shown

and no coffee should be served

Buy a coffee

☑ (0s 22ms)

Tags: *TagsWithCustomStyle*

Given a coffee machine
An empty coffee machine that is already turned on.<br>The coffee price is set to 2 EUR.

and there are <**coffees**> coffees left in the machine
The number of coffees in the machine is set to the given value.

and the machine is <**onOrOff**>

and the coffee costs *2* euros

When I insert <**dollars**> one euro coins

and I press the coffee button

Then I <**shouldOrShouldNot**> be served a coffee

Table 3. Cases

#	coffees	onOrOff	dollars	shouldOrShouldNot	Status
1	1	on	1	should not	✓
2	1	on	2	should	✓
3	0	on	2	should not	✓
4	1	off	2	should not	✓

Coffee making gets better

✓ (0s 6ms)

Given a coffee machine
An empty coffee machine that is already turned on.<br>The coffee price is set to 2 EUR.

When I make coffee for the <**runNr**> time

Then the result is <**result**>

Table 4. Cases

#	Description	runNr	result	Status
1	On the first run	1	quite ok	✓
2	And on the second run	2	well-done	✓

Coffee is not served

✓ (0s 15ms)

Given a coffee machine
An empty coffee machine that is already turned on.<br>The coffee price is set to 2 EUR.

and the coffee costs *2* euros

and there are <**coffees**> coffees left in the machine
The number of coffees in the machine is set to the given value.

When I insert <**euros**> one euro coins

and I press the coffee button

Then I should not be served a coffee

Table 5. Cases

#	coffees	euros	Status
1	1	1	✓
2	0	2	✓
3	1	0	✓

Correct messages are shown

☑ (0s 31ms)

Tags: *Data Tables*

Given a coffee machine
An empty coffee machine that is already turned on.<br>The coffee price is set to 2 EUR.

and there are <**coffees left**> coffees left in the machine
The number of coffees in the machine is set to the given value.

When I insert <**number of coins**> one euro coins

and I press the coffee button

Then the message <**message**> is shown

Table 6. Cases

#	**coffees left**	**number of coins**	**message**	**Status**
1	0	0	Error: No coffees left	☑
2	0	1	Error: No coffees left	☑
3	1	0	Error: Insufficient money	☑
4	0	5	Error: No coffees left	☑
5	1	5	Enjoy your coffee!	☑

Intro words are not required

☑ (0s 2ms)

Given a coffee machine
An empty coffee machine that is already turned on.<br>The coffee price is set to 2 EUR.

the coffee costs *5* euros

there are *3* coffees left in the machine
The number of coffees in the machine is set to the given value.

When I press the coffee button

Then an error should be shown

no coffee should be served

Long error messages should wrapped

❗ (0s 1ms)

Tags: *FailingOnPurpose*

Given an exception with a very long message ❗ (0s 0ms)

No coffee left error is shown when there is no coffee left

☑ (0s 8ms)

Tags: *Order*

Given an empty coffee machine

When I insert *5* one euro coins

and I press the coffee button

Then the message *Error: No coffees left* is shown

Not enough money message is shown when insufficient money was given

☑ (0s 1ms)

Tags: *Order*

Given a coffee machine
An empty coffee machine that is already turned on.<br>The coffee price is set to 2 EUR.

and there are *2* coffees left in the machine
The number of coffees in the machine is set to the given value.

When I insert *1* one euro coins

and I press the coffee button

Then the message *Error: Insufficient money* is shown

Serving a coffee reduces the number of available coffees by one

☑ (0s 21ms)

Tags: *Data Tables*

Given a coffee machine
An empty coffee machine that is already turned on.<br>The coffee price is set to 2 EUR.

and there are <**initial coffees**> coffees left in the machine
The number of coffees in the machine is set to the given value.

When I insert *2* one euro coins

and I press the coffee button

Then a coffee should be served

and there are <**coffees left**> coffees left in the machine <**coffees left**>

Table 7. Cases

#	initial coffees	coffees left	Status
1	1	0	✓
2	3	2	✓
3	10	9	✓

Should fail with unexpected runtime exception

❗ (1s 4ms)

Tags: *FailingOnPurpose*

Then *should throw a runtime exception* ❗ (0s 999ms)

Turned off machines should not serve coffee

✓ (0s 22ms)

Tags: *Case Diffs*

Case 1:

```
onOrOff = true
```

Given a coffee machine
An empty coffee machine that is already turned on.<br>The coffee price is set to 2 EUR.

and there are *2* coffees left in the machine
The number of coffees in the machine is set to the given value.

and the machine is *on*

When I insert *2* one euro coins

and I press the coffee button

Then I should be served a coffee

Case 2:

```
onOrOff = false
```

Given a coffee machine
An empty coffee machine that is already turned on.<br>The coffee price is set to 2 EUR.

and there are *2* coffees left in the machine
The number of coffees in the machine is set to the given value.

and the machine is *off*

When I insert *2* one euro coins

and I press the coffee button

Then I should not be served a coffee

and no error is shown

Failing Scenarios

A failing scenario for demonstration purposes

❗ (0s 3ms)

Tags: *FailingOnPurpose*

Given a coffee machine ☑ (0s 0ms)
An empty coffee machine that is already turned on.<br>The coffee price is set to 2 EUR.

and there are no more coffees left ☑ (0s 0ms)

When I press the coffee button ☑ (0s 0ms)

Then I should be served a coffee ❗ (0s 0ms)

and steps following a failed step should be skipped ⊘ (0s 0ms)
This step is still visible in the report, but was actually not executed. It is marked as skipped in the report.

A failing scenario for demonstration purposes

A scenario with a failing test case for demonstration purposes

❗ (0s 10ms)

Tags: *FailingOnPurpose*

Case 1

```
withCoffees = true
```

Given a coffee machine
An empty coffee machine that is already turned on.<br>The coffee price is set to 2 EUR.

and there are *2* coffees left in the machine
The number of coffees in the machine is set to the given value.

When I insert *2* one euro coins

and I press the coffee button

Then I should be served a coffee

Case 2

```
withCoffees = false
```

Given a coffee machine ☑ (0s 0ms)
An empty coffee machine that is already turned on.<br>The coffee price is set to 2 EUR.

When I insert *2* one euro coins ☑ (0s 0ms)

and I press the coffee button ☑ (0s 0ms)

Then I should be served a coffee ❗ (0s 4ms)

Long error messages should wrapped

❗ (0s 1ms)

Tags: *FailingOnPurpose*

Given an exception with a very long message ❗ (0s 0ms)

Should fail with unexpected runtime exception

❗ (1s 4ms)

Tags: *FailingOnPurpose*

Then *should throw a runtime exception* ❗ (0s 999ms)

Pending Scenarios

Multiple cases can be pending

⊘ (0s 3ms)

Tags: *Pending Annotation*

Given some state ⊘ (0s 0ms)

When a <**actionCount**> action ⊘ (0s 0ms)

Then some result ⊘ (0s 0ms)

Table 8. Cases

#	actionCount	Status
1	1st	⊘
2	2nd	⊘

Scenarios that are pending can be annotated with the Pending annotation

⊘ (0s 0ms)

Tags: *Pending Annotation*

Given some state ⊘ (0s 0ms)

When some action ⊘ (0s 0ms)

Then some result ⊘ (0s 0ms)

Single steps can be annotated with Pending

⊘ (0s 1ms)

Tags: *Pending Annotation*

Given some state ☑ (0s 0ms)

When some pending action ⊘ (0s 0ms)

Then some result ☑ (0s 0ms)

Tags

FailingOnPurpose

A scenario with a failing nested step on purpose

❗ (0s 1ms)

Tags: *FailingOnPurpose*

Given I fill out the registration form with invalid values ❗ (0s 1ms)

- I enter a name *Franky* ☑ (0s 0ms)
  - I think a name *Franky* ☑ (0s 0ms)
  - **and** I write the name *Franky* ☑ (0s 0ms)
- **and** I enter a email address *franky@acme.com* ☑ (0s 0ms)
- **and** something fails for demonstration purposes ❗ (0s 0ms)

When I submit the form ⊘ (0s 0ms)

Then the password matches ⊘ (0s 0ms)

A failing scenario for demonstration purposes

❗ (0s 3ms)

Tags: *FailingOnPurpose*

Given a coffee machine ☑ (0s 0ms)
An empty coffee machine that is already turned on.<br>The coffee price is set to 2 EUR.

and there are no more coffees left ☑ (0s 0ms)

When I press the coffee button ☑ (0s 0ms)

Then I should be served a coffee ❗ (0s 0ms)

and steps following a failed step should be skipped ⊘ (0s 0ms)
This step is still visible in the report, but was actually not executed. It is marked as skipped in the report.

A scenario with a failing test case for demonstration purposes

❗ (0s 10ms)

Tags: *FailingOnPurpose*

Case 1

```
withCoffees = true
```

Given a coffee machine
An empty coffee machine that is already turned on.<br>The coffee price is set to 2 EUR.

and there are *2* coffees left in the machine
The number of coffees in the machine is set to the given value.

When I insert *2* one euro coins

and I press the coffee button

Then I should be served a coffee

Case 2

```
withCoffees = false
```

Given a coffee machine ☑ (0s 0ms)
An empty coffee machine that is already turned on.<br>The coffee price is set to 2 EUR.

When I insert *2* one euro coins ☑ (0s 0ms)

and I press the coffee button ☑ (0s 0ms)

Then I should be served a coffee ❗ (0s 4ms)

Long error messages should wrapped

❗ (0s 1ms)

Tags: *FailingOnPurpose*

Given an exception with a very long message ❗ (0s 0ms)

Should fail with unexpected runtime exception

❗ (1s 4ms)

Tags: *FailingOnPurpose*

Then *should throw a runtime exception* ❗ (0s 999ms)

TagsWithCustomStyle

Buy a coffee

☑ (0s 22ms)

Tags: *TagsWithCustomStyle*

Given a coffee machine
An empty coffee machine that is already turned on.<br>The coffee price is set to 2 EUR.

and there are <**coffees**> coffees left in the machine
The number of coffees in the machine is set to the given value.

and the machine is <**onOrOff**>

and the coffee costs *2* euros

When I insert <**dollars**> one euro coins

and I press the coffee button

Then I <**shouldOrShouldNot**> be served a coffee

Table 9. Cases

#	coffees	onOrOff	dollars	shouldOrShouldNot	Status
1	1	on	1	should not	☑
2	1	on	2	should	☑
3	0	on	2	should not	☑
4	1	off	2	should not	☑