Contents

# Showing basic gherkin syntax

In order to see that gherkin is a very simple language

As a SpecFlow evangelist

I want to show that basic syntax

![Test Image](test.jpg)

## Simple GWT

**Given**  the initial state of the application is Running

**When**  I ask what the application state is

**Then**  I should see Running as the answer

## Using And and But

**Given**  the initial state of the application is Running

**And**  I have authorization to ask application state

**When**  I ask what the application state is

**Then**  I should see Running as the answer

**And**  I should see the time of the application

**But**  the state of the application should not be Stopped

# The test runner is not (very) important

In order to show that the test runner is just for the autogenerated stuff in SpecFlow

As a SpecFlow evanglist

I want to be able to call my steps in the same manner inspite of the testrunner configured

## A couple of simple steps

**Given**  I have step defintions in place

**When**  I call a step

**Then**  the step should have been called

# Addition

In order to explain the order in which hooks are run

As a SpecFlow evanglist

I wan to be able to hook into pre and post conditions in SpecFlow

## Hooking into pre conditions for Test Runs in SpecFlow

**Given**  the scenario is running

**Then**  the BeforeTestRun hook should have been executed

## Hooking into pre conditions for Features in SpecFlow

**Given**  the scenario is running

**Then**  the BeforeFeature hook should have been executed

## Hooking into pre conditions for Scenarios in SpecFlow

**Given**  the scenario is running

**Then**  the BeforeScenario hook should have been executed

## Hooking into pre conditions for ScenarioBlocks in SpecFlow

**Given**  the scenario is running

**Then**  the BeforeScenarioBlock hook should have been executed

## Hooking into pre conditions for Steps in SpecFlow

**Given**  the scenario is running

**Then**  the BeforeStep hook should have been executed

# Tag demonstrator

*(Tags: @allAboutTags, @important)*

In order to show the capabilities of tags in SpecFlow

As a SpecFlow evanglist

I want to write scenarios that has tags and show their usage in code

## Ignored scenario

*(Tags: @ignore)*

**Given**  that my scenario has the @ignore tag

**When**  I run the scenario

**Then**  the scenario is ignored

**And**  the missing step definitions are not reported

## A scenario without tags

**Given**  that my scenario has 0 tags

**When**  I run the scenario

**Then**  before scenario hook with '' is run

## A scenario with 1 tag

*(Tags: @testTag1)*

**Given**  that my scenario has 1 tags

**When**  I run the scenario

**Then**  before scenario hook with 'testTag1' is run

## A scenario with 3 tags

*(Tags: @testTag1, @testTag2, @testTag3)*

**Given**  that my scenario has 3 tags

**When**  I run the scenario

**Then**  before scenario hook with 'testTag1, testTag2, testTag3' is run

## A scenario with 2 tags

*(Tags: @testTag1, @testTag3)*

**Given**  that my scenario has 2 tags

**When**  I run the scenario

**Then**  before scenario hook with 'testTag1, testTag3' is run

# Scenario outline

In order to not have to type the same scenario over and over

As a SpecFlow evangelist

I want to show how to use ScenarioOutline

## Add two positive numbers with many examples

**Given**  I enter <number 1> into the calculator

**And**  I enter <number 2> into the calculator

**When**  I perform add

**Then**  the result should be <result>  
*# This is called Abstrakt Scenario in Swedish (!!!)*

### Examples:

*(Tags: @small, @short\_test)*

|  |  |  |
| --- | --- | --- |
| number 1 | number 2 | result |
| 10 | 20 | 30 |
| 20 | 20 | 40 |
| 20 | 30 | 50 |

### Examples:

*(Tags: @big)*

|  |  |  |
| --- | --- | --- |
| number 1 | number 2 | result |
| 100 | 20 | 120 |
| 1000 | 20 | 1020 |

## Add two negative numbers with many examples

**Given**  I enter <number 1> into the calculator

**And**  I enter <number 2> into the calculator

**When**  I perform add

**Then**  the result should be <result>

### Examples:

|  |  |  |
| --- | --- | --- |
| number 1 | number 2 | result |
| 10 | 20 | 30 |
| 20 | 20 | 40 |
| 20 | 30 | 50 |

### Examples:

|  |  |  |
| --- | --- | --- |
| number 1 | number 2 | result |
| 100 | 20 | 120 |
| 1000 | 20 | 1020 |

# Scenario Context features

In order to show how to use ScenarioContext

As a SpecFlow evangelist

I want to write some simple scenarios with data in ScenarioContext

## Store and retrive Person Marcus from ScenarioContext

**When**  I store a person called Marcus in the Current ScenarioContext

**Then**  a person called Marcus can easily be retrieved

## Showing information of the scenario

*(Tags: @showUpInScenarioInfo, @andThisToo)*

**When**  I execute any scenario

**Then**  the ScenarioInfo contains the following information

|  |  |
| --- | --- |
| Field | Value |
| Tags | showUpInScenarioInfo, andThisToo |
| Title | Showing information of the scenario |

## Show the type of step we're currently on

**Given**  I have a Given step

**And**  I have another Given step

**When**  I have a When step

**Then**  I have a Then step  
*#This is not so easy to write a scenario for but I've created an AfterScenario-hook*  
*#To see this in action remove the @ignore tag below*

## Display error information in AfterScenario

*(Tags: @ignore, @showingErrorHandling)*

**When**  an error occurs in a step

## Pending step

**When**  I set the ScenarioContext.Current to pending

**Then**  this step will not even be executed

# FeatureContext features

*(Tags: @showUpInScenarioInfo, @andThisToo)*

In order to show how to use FeatureContext

As a SpecFlow evangelist

I want to write some simple scenarios with data in FeatureContext

## Store and retrive Person Marcus from FeatureContext Current

**When**  I store a person called Marcus in the current FeatureContext

**Then**  a person called Marcus can easily be retrieved from the current FeatureContext

## Showing information of the feature

**When**  I execute any scenario in the feature

**Then**  the FeatureInfo contains the following information

|  |  |
| --- | --- |
| Field | Value |
| Tags | showUpInScenarioInfo, andThisToo |
| Title | FeatureContext features |
| TargetLanguage | CSharp |
| Language | en-US |
| Description | In order to |

# Show the use of background

In order to show how to use the Background keyword of Gherkin

As a SpecFlow evanglist

I want to show that background steps are called before any scenario step

|  |
| --- |
| **Background** **Given**  I have initialized the Sum-variable to 0  **When**  I add 1 to the Sum-variable |

## Add 1 to the sum

**When**  I add 1 to the Sum-variable

**Then**  the total sum should be 2

## Add 2 to the sum

**When**  I add 2 to the Sum-variable

**Then**  the total sum should be 3

# Showing table usage

In order to show how to use tables

As a SpecFlow evanglist

I want to write some simple scenarios that uses tables tables

## Using tables

**Given**  I have the following persons

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Style | Birth date | Cred |
| Marcus | Cool | 1972-10-09 | 50 |
| Anders | Butch | 1977-01-01 | 500 |
| Jocke | Soft | 1974-04-04 | 1000 |

**When**  I search for Jocke

**Then**  the following person should be returned

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Style | Birth date | Cred |
| Jocke | Soft | 1974-04-04 | 1000 |

## Using tables with SpecFlow Assist

**Given**  I have the following persons using assist

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Style | Birth date | Cred |
| Marcus | Very cool | 1972-10-09 | 50 |
| Anders | Butch | 1977-01-01 | 500 |
| Jocke | Soft | 1974-04-04 | 1000 |

**When**  I search for Jocke

**Then**  the following person should be returned using assist

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Style | Birth date | Cred |
| Jocke | Soft | 1974-04-04 | 1000 |

## Creating a entity from field value

**When**  I fill out the form like this

|  |  |
| --- | --- |
| Field | Value |
| Name | Marcus |
| Style | very cool |
| Birth date | 1972-10-09 |
| Cred | 100 |

**Then**  the following person should be returned using assist

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Style | Birth date | Cred |
| Marcus | Very cool | 1972-10-09 | 1000 |

## Example of a wide table

**Given**  this wide table

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Name | Style | Birth date | Cred | Name | Style | Birth date | Cred | Name | Style | Birth date | Cred | Name | Style | Birth date | Cred | Name | Style | Birth date | Cred | Name | Style | Birth date | Cred | Name | Style | Birth date | Cred | Name | Style | Birth date | Cred |
| Marcus | Very cool | 10/9/1972 | 1000 | Marcus | Very cool | 10/9/1972 | 1000 | Marcus | Very cool | 10/9/1972 | 1000 | Marcus | Very cool | 10/9/1972 | 1000 | Marcus | Very cool | 10/9/1972 | 1000 | Marcus | Very cool | 10/9/1972 | 1000 | Marcus | Very cool | 10/9/1972 | 1000 | Marcus | Very cool | 10/9/1972 | 1000 |

# Show the compare to feature

In order to show the compare to features of SpecFlow Assist

As a SpecFlow evanglist

I want to show how the different versions of compareTo works

## CompareToInstance

**Given**  I have the following person

|  |  |
| --- | --- |
| Field | Value |
| Name | Marcus |
| Style | Butch |
| Birth date | 1972-10-09 |

**Then**  CompareToInstance should match this guy

|  |  |
| --- | --- |
| Field | Value |
| Name | Marcus |
| Style | Butch |
| BirthDate | 10/9/1972 12:00:00 AM |

**And**  CompareToInstance should match this guy

|  |  |
| --- | --- |
| Field | Value |
| Name | Marcus |
| BirthDate | 10/9/1972 12:00:00 AM |

**But**  CompareToInstance should not match this guy  
*# CompareToSet will test only the properties that you define in the table.*  
*# CompareToSet does not test the order of the objects, only that one was found that matches*

|  |  |
| --- | --- |
| Field | Value |
| Name | Anders |
| Style | very cool |
| BirthDate | 10/9/1974 12:00:00 AM |

## CompareToSet

**Given**  I have the following persons using assist

|  |  |  |
| --- | --- | --- |
| Name | Style | Birth date |
| Marcus | Cool | 1972-10-09 |
| Anders | Butch | 1977-01-01 |
| Jocke | Soft | 1974-04-04 |

**Then**  CompareToSet should match this

|  |  |  |
| --- | --- | --- |
| Name | Style | BirthDate |
| Marcus | Cool | 10/9/1972 12:00:00 AM |
| Anders | Butch | 1/1/1977 12:00:00 AM |
| Jocke | Soft | 4/4/1974 12:00:00 AM |

**But**  CompareToSet should not match this

|  |  |  |
| --- | --- | --- |
| Name | Style | BirthDate |
| Marcus | Cool | 10/9/1972 12:00:00 AM |
| Anders | Butch | 1/1/1977 12:00:00 AM |

# Svenska - Summering

För att slippa att göra dumma fel

Som räknare

Vill jag kunna lägga summera

## Summera 5 och 7 ska vara 12

**Givet**  att jag har knappat in 5

**Och**  att jag har knappat in 7

**När**  jag summerar

**Så**  ska resultatet vara 12

# Test de Cultuur in het Vlaams

## Het Scenario

**Stel**  dat ik 50 ingeef

**En**  dat ik 70 ingeef

**Als**  ik plus druk

**Dan**  moet het resultaat 120 zijn

# Attribute overloading

In order to show that steps can be used with multiple attributes

As a SpecFlow Evangelist

I want to show that similar attributes can be applied to the same step definition

## Checking number for evenness

**Given**  I have this simple step

**And**  this simple step

**And**  also this step

**When**  I do something

**Then**  I could validate that the number 2 is even

**And**  that the number 4 is even

**But**  the number 3 is odd

# Calling Steps from StepDefinitions

In order to create steps of a higher abstraction

As a SpecFlow evangelist

I want reuse other steps in my step definitions

## Log in

**Given**  I am on the index page

**When**  I enter my unsername nad password

**And**  I click the login button

**Then**  the welcome page should be displayed

## Do something meaningful

**Given**  I am logged in

**When**  I dosomething meaningful

**Then**  I should get rewarded

# Step Argument Transformations

In order to reduce the amount of code and repetitive tasks in my steps

As a SpecFlow evanglist

I want to define reusable transformations for my step arguments

## Steps with non-string arguments

**Given**  Dan has been registered at date 2003/03/13

**And**  Aslak has been registered at terminal 2

**Then**  I should be able to see Aslak at terminal 2

# Injecting context into step specifications

In order to don't have to rely on the global shared state

and to be able to define the contexts required for each scenario.

As a SpecFlow Evanglist

I would like to have the system automatically inject an instance of any class as

defined in the constructor of a step file

## Feature with no context

**Given**  a feature which requires no context

**Then**  everything is dandy

## Feature with a single context

**Given**  a feature which requires a single context

**Then**  the context is set

## Feature with multiple contexts

**Given**  a feature which requires multiple contexts

**Then**  the contexts are set

## Feature with recursive contexts

**Given**  a feature which requires a recursive context

**Then**  the context is set

**And**  its sub-context is set

## Feature with a dependent context

**Given**  a feature which requires a single context

**Then**  the context is set

**And**  the context was created by the feature with a single context scenario

# Nested Folder Example

In order to test nested folder output

As a silly contributer

I want to create an example of something several folders deep

## Nested - Add two numbers

*(Tags: @nestedFolders)*

**Given**  I have entered 50 into the calculator

**And**  I have entered 70 into the calculator

**When**  I press add

**Then**  the result should be 120 on the screen

# Multiline Feature Example

In order capture this particular Gherkin feature

As a Pickles contributer

I want to demonstrate an example of using multiline text in a Scenario

## Mutliline Output

*(Tags: @mytag)*

**Given**  I have read in some text from the user

This is line 1.

This is line 2!

This is line 3!!

**When**  I process this input

**Then**  the result will be saved to the multiline text data store

# Sample Markdown Feature

# Header 1

## Header 2

### Header 3

#### Header 4

##### Header 5

###### Header 6

This is a \*significant\* word

1. Ordered #1

2. Ordered #2

3. Ordered #3

- Unordered #1

- Unordered #2

- Unordered #3

Horizontal Rule:

- - -

Table example:

| First Header | Second Header |

| ------------- | ------------- |

| Content Cell | Content Cell |

| Content Cell | Content Cell |

- - -

Including a picture: ![](./image.png)

|  |
| --- |
| **Background** This is the \*coolest\* background  **Given**  I have initialized the Sum-variable to 0  **When**  I add 1 to the Sum-variable |

## Sample Markdown Scenario Example

This is \*\*important\*\* text
Code Block:
```
var x = 2;
```
Apple
: Pomaceous fruit of plants of the genus Malus in
the family Rosaceae.
: An American computer company.
Orange
: The fruit of an evergreen tree of the genus Citrus.

**Given**  this

**Then**  that

## Sample Markdown Scenario Outline Example

*(Tags: @AddingATag)*

This is [an example link to pickles](https://github.com/picklesdoc/pickles/wiki "Pickles") inline link.
[This link to pickles](https://github.com/picklesdoc/pickles/wiki) has no title attribute.

**Given**  this: <test>

**Then**  that: <test2>

### Examples:

This \_\_message\_\_ is important too and is for an \*Example\* table.

|  |  |
| --- | --- |
| test | test2 |
| value | value2 |

# Interactive DHTML View

In order to increase stakeholder engagement with pickled specs

As a SpecFlow evangelist

I want to adjust the level of detail in the DHTML view to suite my audience

So that I do not overwhelm them.

## Scenario with large data table

**Given**  a feature with a large table of data:

|  |  |
| --- | --- |
| heading | page # |
| Chapter 1 | 1 |
| Chapter 2 | 5 |
| Chapter 3 | 10 |
| Chapter 4 | 15 |
| Chapter 5 | 20 |
| Chapter 6 | 25 |
| Chapter 7 | 30 |
| Chapter 8 | 35 |
| Chapter 9 | 40 |
| Chapter 10 | 45 |
| Chapter 11 | 50 |
| Chapter 12 | 55 |
| Chapter 13 | 60 |
| Chapter 14 | 65 |
| Chapter 15 | 70 |
| Chapter 16 | 75 |
| Chapter 17 | 80 |
| Chapter 18 | 85 |
| Chapter 19 | 90 |
| Chapter 20 | 95 |
| Chapter 21 | 100 |
| Chapter 22 | 105 |

**When**  I click on the table heading

**Then**  the table body should collapse

# Kinds of verification

In order to increase stakeholder engagement with pickled specs

As a SpecFlow evangelist

I want to make clear how many scenarios are automated, manually tested and not tested

So that the stakeholders have a better overview of progress

## Verified automatically

*(Tags: @automated)*

**Given**  a feature that is verified automatically

**When**  I run pickles

**Then**  it appears in the @automated summary

## Verified manually

*(Tags: @manual)*

**Given**  a feature that is verified manually

**When**  I run pickles

**Then**  it appears in the @manual summary

## Verified automatically

*(Tags: @notTested\_waitingForAutomation)*

**Given**  a feature that is not verified

**When**  I run pickles

**Then**  it appears in the @notTested summary

# Arithmetic

In order to avoid silly mistakes

As a math idiot

I want to be able to perform arithmetic on the calculator

When $a \ne 0$, there are two solutions to $\(ax^2 + bx + c = 0\)$ and they are

$$x = {-b \pm \sqrt{b^2-4ac} \over 2a}.$$

## Add two numbers

*(Tags: @arithmetic, @fast)*

$50 + 70 = 120$

*# In the DHTML version, the description will be rendered with mathematical formulas if the experimental features are enabled.*  
**Given**  I have entered 50 into the calculator

**And**  I have entered 70 into the calculator

**When**  I press add

**Then**  the result should be 120 on the screen

## Subtract two numbers

*(Tags: @arithmetic, @fast)*

**Given**  I have entered 50 into the calculator

**And**  I have entered 70 into the calculator

**When**  I press subtract

**Then**  the result should be -20 on the screen

## Multiply two numbers

*(Tags: @arithmetic, @fast)*

**Given**  I have entered 50 into the calculator

**And**  I have entered 70 into the calculator

**When**  I press multiply

**Then**  the result should be 3500 on the screen

## Divide two numbers

*(Tags: @arithmetic, @fast)*

**Given**  I have entered 50 into the calculator

**And**  I have entered 2 into the calculator

**When**  I press divide

**Then**  the result should be 25 on the screen

# Trigonometry

In order to avoid perform more advanced calculations

As a math idiot

I want to be able to use trigonometric functions

## Sine

*(Tags: @trigonometric, @fast)*

**Given**  I have entered 90 into the calculator

**When**  I press sin

**Then**  the result should be 1 on the screen

## Cosine

*(Tags: @trigonometric, @fast)*

**Given**  I have entered 0 into the calculator

**When**  I press cos

**Then**  the result should be 1 on the screen

## Tangent

*(Tags: @trigonometric, @fast)*

**Given**  I have entered 45 into the calculator

**When**  I press tan

**Then**  the result should be 1 on the screen

# Clearing Screen

In order to restart a new set of calculations

As a math idiot

I want to be able to clear the screen

## Clear the screen

*(Tags: @workflow, @slow)*

**Given**  I have entered 50 into the calculator

**And**  I have entered 70 into the calculator

**When**  I press C

**Then**  the screen should be empty