Getting started

SpecSync for Azure DevOps synchronizes the scenarios of a SpecFlow project with the test cases of a Azure DevOps project. For the supported Azure DevOps versions, please check the Compatibility list.

The necessary steps are slightly different depending on whether you use SpecSync with SpecFlow (.NET) or with other Gherkin-based BDD tools, like Cucumber. Please follow the guide for your context.

- Getting started using SpecFlow
- Getting started using Cucumber or other Gherkin-based BDD tool

Getting started using SpecFlow

This chapter goes through the setup and the synchronization steps for SpecFlow projects. For non-SpecFlow projects, like Cucumber, please check page Getting started using Cucumber.

SpecSync is a synchronization tool that can be invoked from the command line. For SpecFlow projects, there is also a SpecFlow plugin that enables synchronizing automated test cases if that is necessary. This guide shows you step-by-step how the synchronization tool and the SpecFlow plugin can be configured.

Preparation

For setting up SpecSync for Azure DevOps, you need a SpecFlow project and a Azure DevOps project. For the supported Azure DevOps versions, please check the Compatibility list.

In our guide, we will use a calculator example (MyCalculator) that uses SpecFlow v2.3 with MsTest. The SpecFlow project is called `MyCalculator.Specs`. The sample project can be downloaded from GitHub.

For a synchronization target we use an Azure DevOps project: `https://specsyncdemo.visualstudio.com/MyCalculator`. (An Azure DevOps project for testing SpecSync can be created for free from the Azure DevOps website).

Installation

The SpecSync for Azure DevOps synchronization tool can be installed by adding the `SpecSync.AzureDevOps` package from NuGet.org:

```
PM> Install-Package SpecSync.AzureDevOps
```

The package contains the synchronization command line tool (`tools\SpecSync4AzureDevOps.exe`) and some documentation files (`docs` folder).

It also adds a `specsync4azuredevops.cmd` script file to the project for calling the SpecSync command line tool conveniently. SpecSync can also be used without the script file, but in this case you have to provide the full path of `SpecSync4AzureDevOps.exe` downloaded into the NuGet packages folder.

> NuGet does not allow adding content files for **SDK-style .NET projects (e.g. .NET Core)**. For these projects you should either add these files manually from the `content` folder of the package or use the alternative methods described in the Installation page.

Basic configuration

The NuGet package has added a configuration file (`specsync.json`) to your project that contains all SpecSync related settings. Before the first synchronization we have to review and change a few settings in this file.

1. Open the `specsync.json` file in Visual Studio from your project folder.
2. Set the value of the `remote/projectUrl` setting to the **project URL** of your Azure DevOps project. The project URL is usually in `https://server-name/project-name` or in `http://server-name:8080/tfs/project-name` form and it is not necessarily the URL of the dashboard you open normally. See What is my Azure DevOps project URL for more details.
3. Optionally you can set your personal access token (PAT) as user name (`remote/user` setting) or choose one of the other Azure DevOps authentication options. If you don't specify credentials here, SpecSync will show an interactive authentication prompt.

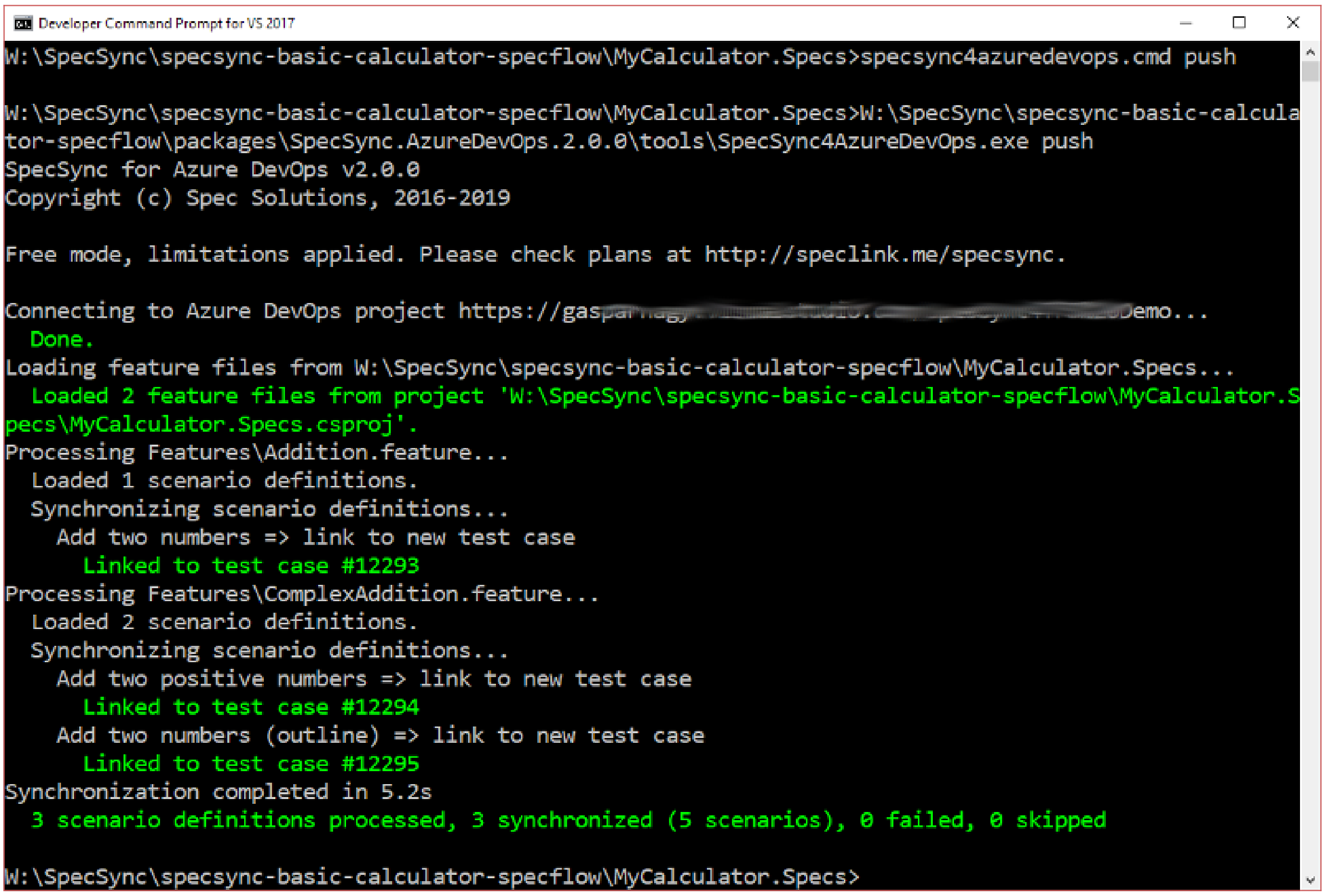
The `specsync.json` after basic configuration has been set

```
{
  "$schema": "http://schemas.specsolutions.eu/specsync4azuredevops-config-la

  // See configuration options and samples at http://speclink.me/specsynccor
  // You can also check the 'specsync-sample.json' file in the 'docs' folde

  "remote": {
    "projectUrl": "https://specsyncdemo.visualstudio.com/MyCalculator",
    "user": "52yny......................................ycsetda"
  }
}
```

First synchronization

1. Make sure your project compiles.
2. We recommend starting from a state where
   - all tests pass,
   - the modified files are checked in to source control.
3. Open a command line prompt and navigate to the SpecFlow project folder (`MyCalculator.Specs`)
4. Call `specsync4azuredevops.cmd push` to invoke the synchronization.
5. If you haven't specified any credentials in the configuration file, an authentication dialog will popup, where you have to specify your credentials for accessing the Azure DevOps project.

As a result, the scenarios from the project will be linked to newly created Azure DevOps test cases, and you will see a result like this.



```
W:\SpecSync\specsync-basic-calculator-specflow\MyCalculator.Specs>specsync4azuredevops.cmd push

W:\SpecSync\specsync-basic-calculator-specflow\MyCalculator.Specs>W:\SpecSync\specsync-basic-calcula
tor-specflow\packages\SpecSync.AzureDevOps.2.0.0\tools\SpecSync4AzureDevOps.exe push
SpecSync for Azure DevOps v2.0.0
Copyright (c) Spec Solutions, 2016-2019

Free mode, limitations applied. Please check plans at http://speclink.me/specsync.

Connecting to Azure DevOps project https://gaspar[redacted]Demo...
  Done.
Loading feature files from W:\SpecSync\specsync-basic-calculator-specflow\MyCalculator.Specs...
  Loaded 2 feature files from project 'W:\SpecSync\specsync-basic-calculator-specflow\MyCalculator.S
pecs\MyCalculator.Specs.csproj'.
Processing Features\Addition.feature...
  Loaded 1 scenario definitions.
  Synchronizing scenario definitions...
    Add two numbers => link to new test case
      Linked to test case #12293
Processing Features\ComplexAddition.feature...
  Loaded 2 scenario definitions.
  Synchronizing scenario definitions...
    Add two positive numbers => link to new test case
      Linked to test case #12294
    Add two numbers (outline) => link to new test case
      Linked to test case #12295
Synchronization completed in 5.2s
  3 scenario definitions processed, 3 synchronized (5 scenarios), 0 failed, 0 skipped

W:\SpecSync\specsync-basic-calculator-specflow\MyCalculator.Specs>
```

Result of the first synchronization

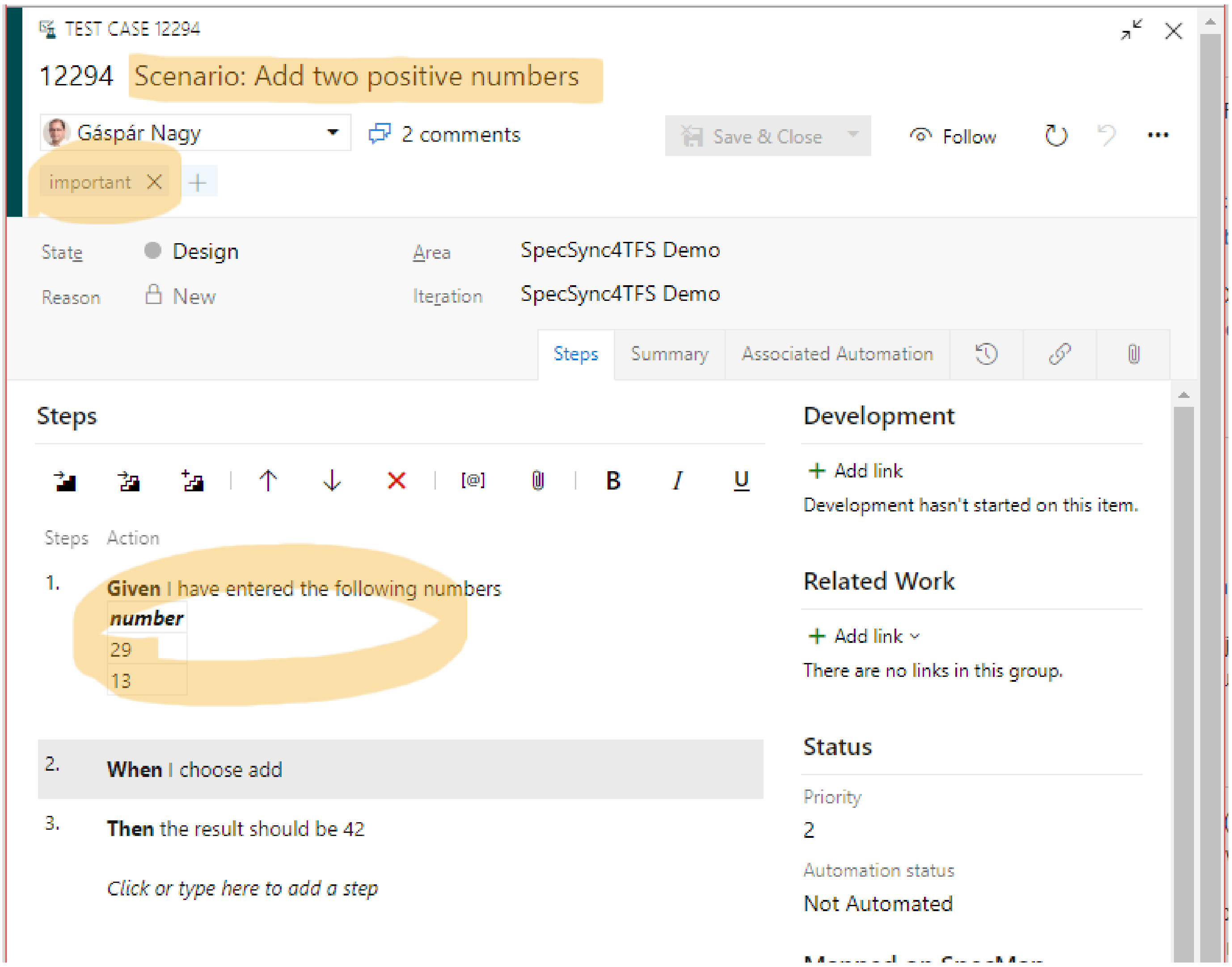
Note: Scenarios are synchronized to normal, Scenario Outlines to parametrized test cases.

Useful hint for testing: Normally you cannot delete work items from Azure DevOps, so testing the initial linking is harder. If you have Visual Studio installed, there is a tool called `witadmin` available from the VS command prompt. With the `destroywi` command of this tool you can delete work items. See `witadmin help destroywi` for details, and use it carefully.

Check Test Case in Azure DevOps

1. Find one of the created test case in Azure DevOps. The easiest way to do this is to open the Azure DevOps URL in a browser and specify the test case ID (e.g. `#12294)` in the "Search" text box in the upper right corner of the web page.

You should see something like this.


TEST CASE 12294

12294 Scenario: Add two positive numbers

Gáspár Nagy | 2 comments | Save & Close | Follow

important

State: Design | Area: SpecSync4TFS Demo
Reason: New | Iteration: SpecSync4TFS Demo

Steps | Summary | Associated Automation

Steps

Steps	Action
1.	**Given** I have entered the following numbers ***number*** 29 13
2.	**When** I choose add
3.	**Then** the result should be 42
	Click or type here to add a step

Development

Add link

Development hasn't started on this item.

Related Work

Add link

There are no links in this group.

Status

Priority: 2

Automation status: Not Automated

A newly created test case in Azure DevOps

There are a couple of things you can note here.

- The name of the scenario has been synchronized as the title of the test case. (The "Scenario:" prefix can be omitted by changing the synchronization format configurations.)
- The tags of the scenario have been synchronized as test case tags.
- The steps of the scenario have been synchronized as test case steps. (The *Then* steps can also be synchronized into the *Expected result* column of the test case step list and you can change a couple of other formatting options as well.)

Verify feature file and commit changes

1. Open one of the feature files from the SpecFlow project in Visual Studio. SpecSync modified the file and added a few tags.

2. Each scenario and scenario outline has been tagged with a `@tc:...` tag making the link between the scenario and the created test case.

```
@tc:12294
@important
Scenario: Add two positive numbers
```

Note: The feature files are changed only when synchronizing new scenarios (linking). To avoid file changes (e.g. when running the synchronization from a build server) the `--buildServerMode` *command line switch can be used. See Synchronizing test cases from build for details.*

Verify if the project still compiles and the tests pass (they should, since we have only added tags), and commit (check-in) your changes.

Synchronize an update

Now let's make a change in one of the scenarios and synchronize the changes to the related test case.

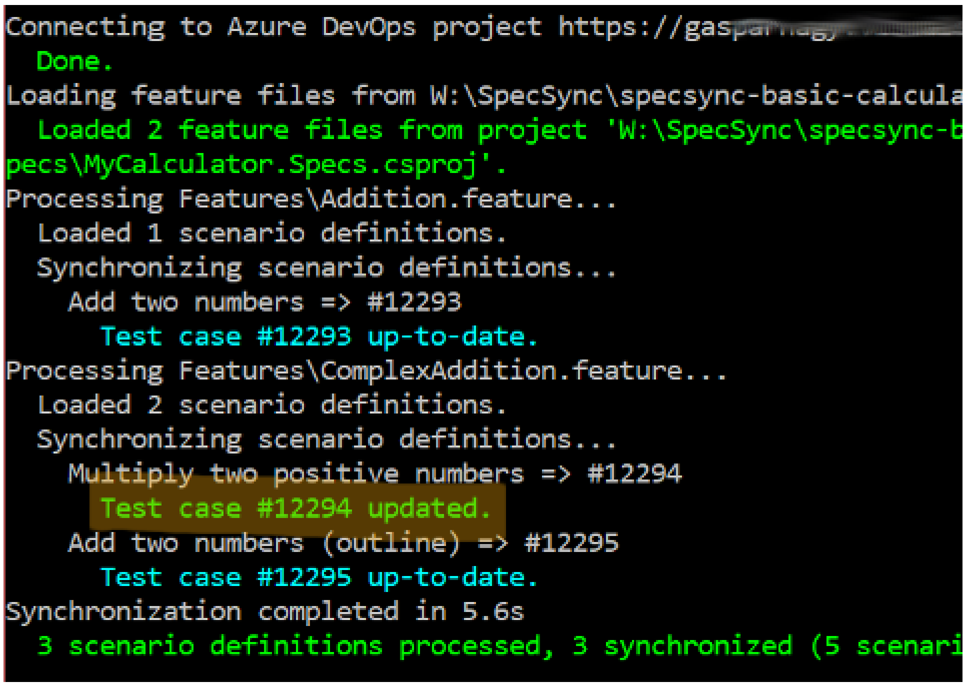
1. Update the title and the steps of the scenario, for example change the scenario `Add two positive numbers` to `Multiply two positive numbers`, change `add` to `multiply` in the *When* step and update the expected result to `377`:

```
@tc:12294
Scenario: Multiply two positive numbers
  Given I have entered the following numbers
      | number |
      | 29     |
      | 13     |
  When I choose multiply
  Then the result should be 377
```

2. Make sure it still compiles and the test passes.
3. Run the synchronization again:

```
specsync4azuredevops.cmd push
```

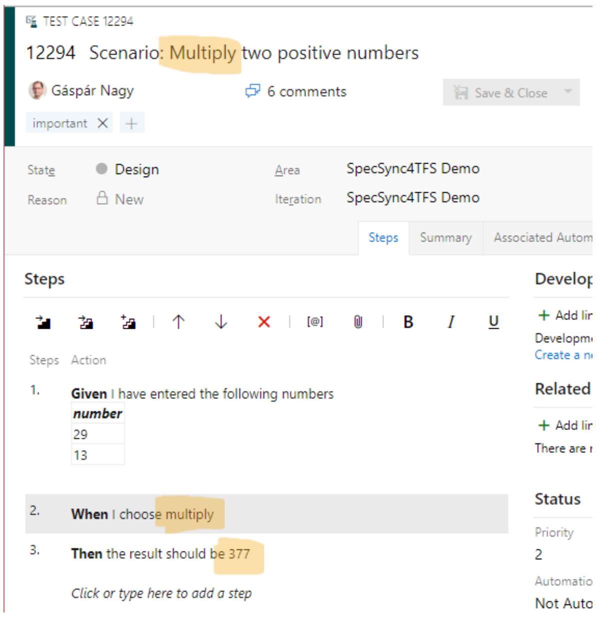
The result shows that the test case for the scenario has been updated, but the other test cases have remained unchanged (*up-to-date*).



```
Connecting to Azure DevOps project https://gaspar[redacted]
  Done.
Loading feature files from W:\SpecSync\specsync-basic-calcula
  Loaded 2 feature files from project 'W:\SpecSync\specsync-b
pecs\MyCalculator.Specs.csproj'.
Processing Features\Addition.feature...
  Loaded 1 scenario definitions.
  Synchronizing scenario definitions...
    Add two numbers => #12293
      Test case #12293 up-to-date.
Processing Features\ComplexAddition.feature...
  Loaded 2 scenario definitions.
  Synchronizing scenario definitions...
    Multiply two positive numbers => #12294
      Test case #12294 updated.
    Add two numbers (outline) => #12295
      Test case #12295 up-to-date.
Synchronization completed in 5.6s
  3 scenario definitions processed, 3 synchronized (5 scenari
```

Result of synchronizing an update

1. Refresh the test case in your browser to see the changed title and steps.



TEST CASE 12294

12294 Scenario: Multiply two positive numbers

Gáspár Nagy 6 comments Save & Close

important

State	Design	Area	SpecSync4TFS Demo
Reason	New	Iteration	SpecSync4TFS Demo

Steps Summary Associated Autom

Steps

Steps	Action
1.	**Given** I have entered the following numbers ***number*** 29 13
2.	**When** I choose multiply
3.	**Then** the result should be 377

Click or type here to add a step

Develop
+ Add lir
Developm
Create a n

Related
+ Add lir
There are r

Status
Priority
2
Automatio
Not Auto

Note: For executing complex test cases, further verification and planning steps might be required after the test case has been changed. SpecSync can reset the test case state to a configured *value (e.g.* `Design` *) in order to ensure that these steps are not forgotten. For more information on this, check the synchronization state configuration documentation.*

Group synchronized test cases to a test suite

We have seen already how to synchronize scenarios to test cases. To be able to easily find these test cases in Azure DevOps, they can be added to test suites. SpecSync can automatically add/remove the synchronized test cases to a test suite. For this you have to specify the name or the ID or the name of the test suite in the configuration.

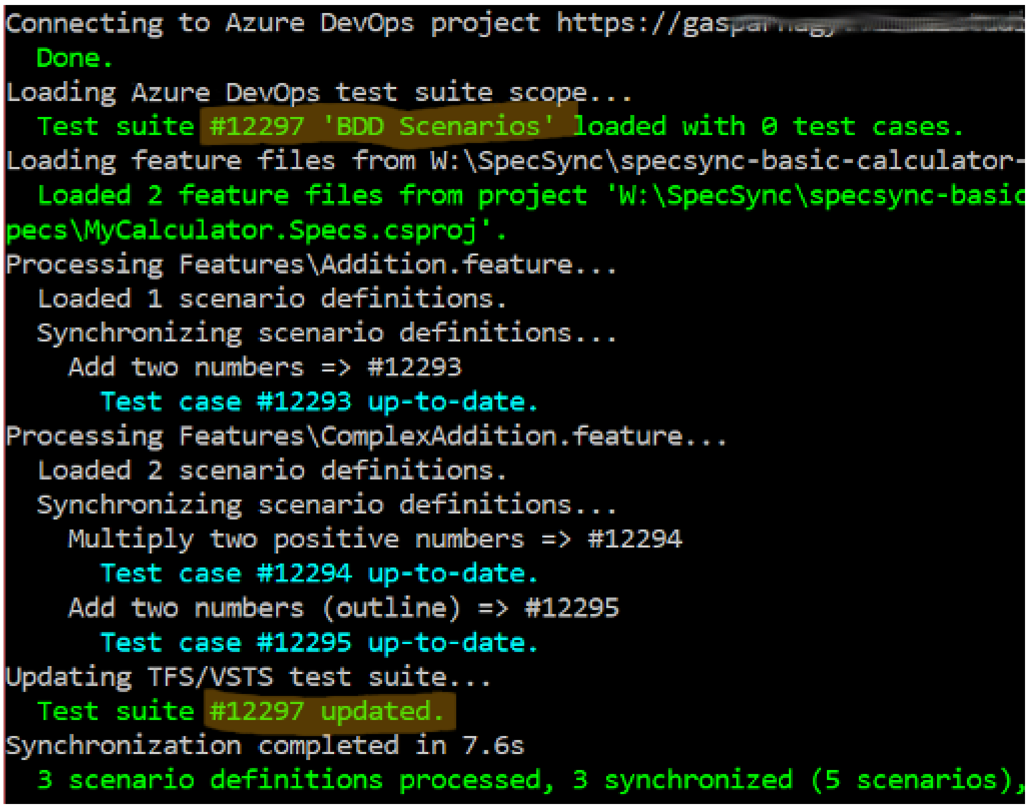
1. Create a "Static suite" (e.g. "BDD Scenarios") in Azure DevOps. (For that you have to navigate to "Test plans" and create and select a test plan first.)
2. Specify the name of the test suite in the `remote/testSuite/name` entry of the `specsync.json` file. (Alternatively you can specify the ID of the suite in `remote/testSuite/id` . The suite names are not unique in Azure DevOps!)

```
{
  "$schema": "http://schemas.specsolutions.eu/specsync4azuredevops-confi

  // See configuration options and samples at http://speclink.me/specsyr
  // You can also check the 'specsync-sample.json' file in the 'docs' fo

  "remote": {
    "projectUrl": "https://specsyncdemo.visualstudio.com/MyCalculator",
    "user": "52yny4a.....................................ycsetda",
    "testSuite": {
      "name": "BDD Scenarios"
    }
  }
}
```

3. Make sure that the project compiles and the tests pass.
4. Run the synchronization again:

```
specsync4azuredevops.cmd push
```

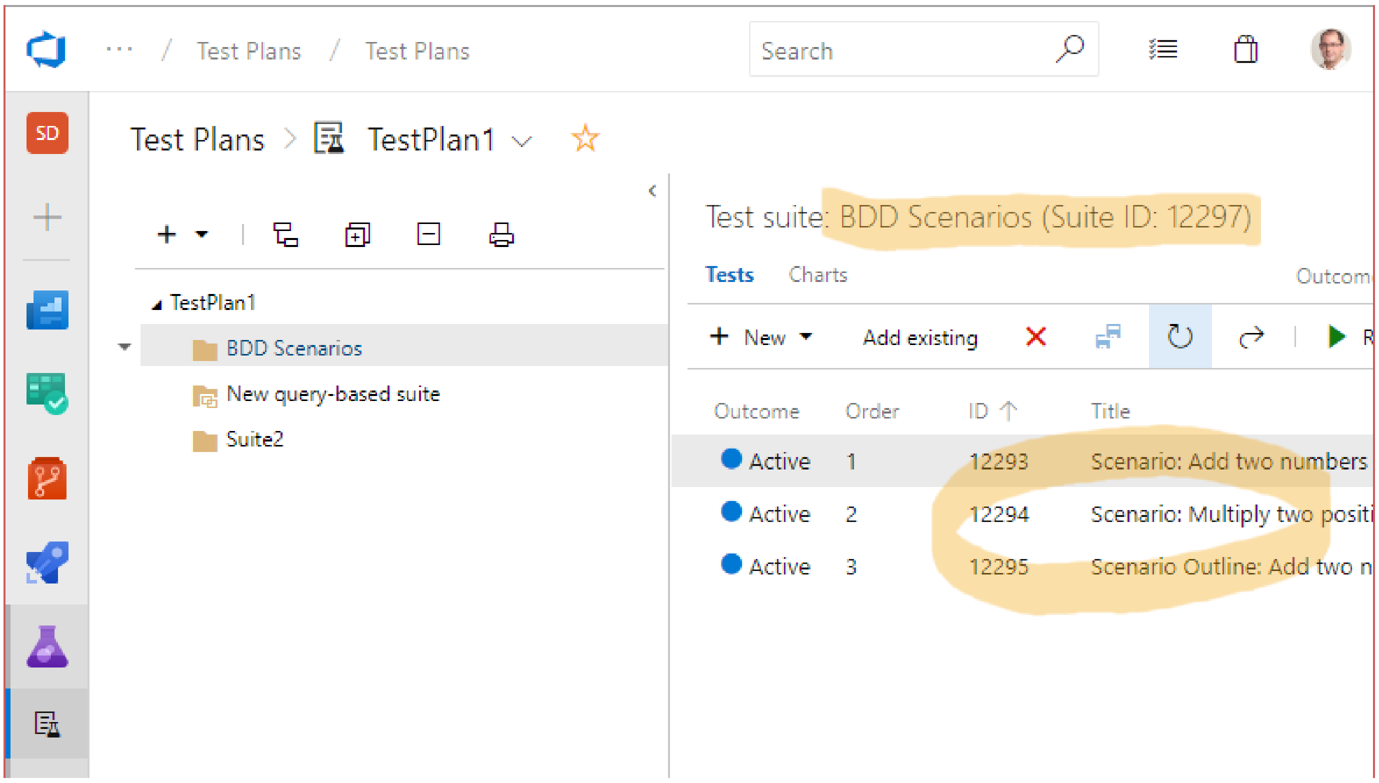
The synchronization will proceed with the result similar to this.



```
Connecting to Azure DevOps project https://gaspa[redacted]
  Done.
Loading Azure DevOps test suite scope...
  Test suite #12297 'BDD Scenarios' loaded with 0 test cases.
Loading feature files from W:\SpecSync\specsync-basic-calculator-
  Loaded 2 feature files from project 'W:\SpecSync\specsync-basic
pecs\MyCalculator.Specs.csproj'.
Processing Features\Addition.feature...
  Loaded 1 scenario definitions.
  Synchronizing scenario definitions...
    Add two numbers => #12293
      Test case #12293 up-to-date.
Processing Features\ComplexAddition.feature...
  Loaded 2 scenario definitions.
  Synchronizing scenario definitions...
    Multiply two positive numbers => #12294
      Test case #12294 up-to-date.
    Add two numbers (outline) => #12295
      Test case #12295 up-to-date.
Updating TFS/VSTS test suite...
  Test suite #12297 updated.
Synchronization completed in 7.6s
  3 scenario definitions processed, 3 synchronized (5 scenarios),
```

Synchronize scenarios to test suite

SpecSync has added the test cases to the test suite.



Test suite: BDD Scenarios (Suite ID: 12297)

Outcome	Order	ID	Title
Active	1	12293	Scenario: Add two numbers
Active	2	12294	Scenario: Multiply two positi
Active	3	12295	Scenario Outline: Add two n

Test cases were added to the test suite

Note: Since the test suite names are not unique in Azure DevOps, you can also specify the test suite ID in the `remote/testSuite/id` *setting.*

Synchronizing automated test cases (optional)

So far the test cases we synchronized from the scenarios were marked as "Not Automated". This means that although it is possible to execute the SpecFlow scenarios both locally and on the build server (from the assembly built from the SpecFlow project), the synchronized tests cases were not linked to the test method generated by SpecFlow.

If the team needs the Azure DevOps test cases for documentation and traceability and runs the scenarios from assembly, then we have already reached the desired outcome. But if the test cases have to be executed as automated test cases, we need to perform a few further steps and you have to choose a test execution strategy.

For more information on test execution strategies and a step-by-step instruction on how they can be configured, please check the Synchronizing automated test cases article.

Getting started using Cucumber or other Gherkin-based BDD tool

This chapter goes through the setup and the synchronization steps for non-SpecFlow projects. For SpecFlow projects, please check page Getting started using SpecFlow.

SpecSync can synchronize any scenarios that are written in Gherkin format. Gherkin format is used by many tools in many platforms, like Cucumber, Cucumber JVM, Cucumber.js, Behat, Behave and also SpecFlow.

If your scenarios are automated with a tool other than SpecFlow, SpecSync will synchronize them as non-automated Azure DevOps Test Cases, because currently Azure DevOps only supports specifying .NET automation for the test cases. The synchronized non-automated test cases can be managed, linked and structured in Azure DevOps. You can also run them manually.

The SpecSync synchronization tool can be executed as a command line tool from Windows, OSX and Linux-based systems. See Using SpecSync on OSX/Linux page for details.

In this guide we will use Cucumber.js as an example, but the steps can also be applied for other tools as well.

Preparation

For setting up SpecSync for Azure DevOps, you need a Cucumber project and a Azure DevOps project. For the supported Azure DevOps versions, please check the Compatibility list.

In our guide, we will use a calculator example (my_calculator) that uses Cucumber.js v5.1. The sample project can be downloaded from GitHub.

For a synchronization target we use an Azure DevOps project: `https://specsyncdemo.visualstudio.com/MyCalculator`. (An Azure DevOps project for testing SpecSync can be created for free from the Azure DevOps website).

Installation

Download SpecSync from the downloads page and unzip it to a folder on your system.

The package contains the synchronization command line tool (`tools/SpecSync4AzureDevOps.exe`) and some documentation files (`docs` folder).

Basic configuration

Create a configuration file (`specsync.json`) to your project root, based on the `docs/specsync-empty.json` file. The empty file can also be downloaded from http://schemas.specsolutions.eu/specsync-empty.json.

```
{
  "$schema": "http://schemas.specsolutions.eu/specsync4azuredevops-config-la

  // See configuration options and samples at http://speclink.me/specsynccon
  // You can also check the 'specsync-sample.json' file in the 'docs' folde

  "remote": {
    "projectUrl": "<specify your Azure DevOps project ULR>"
  }
}
```

Before the first synchronization we have to review and change a few settings in this file. For this example we will synchronize all feature files within the `test/features` folder. For synchronizing only a specific set of feature files, please check the `local` Configuration documentation.

1. Open the `specsync.json` file in your IDE (e.g. Visual Studio Code) from your project folder.
2. Set the value of the `remote/projectUrl` setting to the **project URL** of your Azure DevOps project. The project URL is usually in `https://server-name/project-name` or in `http://server-name:8080/tfs/project-name` form and it is not necessarily the URL of the dashboard you open normally. See What is my Azure DevOps project URL for more details.

3. Optionally you can set your personal access token (PAT) as user name (`remote/user` setting) or choose one of the other Azure DevOps authentication options. If you don't specify credentials here, SpecSync will show an interactive authentication prompt.
4. Set the value of the `local/featureFileSource/type` setting to `folder` and the `local/featureFileSource/folder` setting to `test/features`. This will instruct SpecSync to process the feature files from that specific folder.

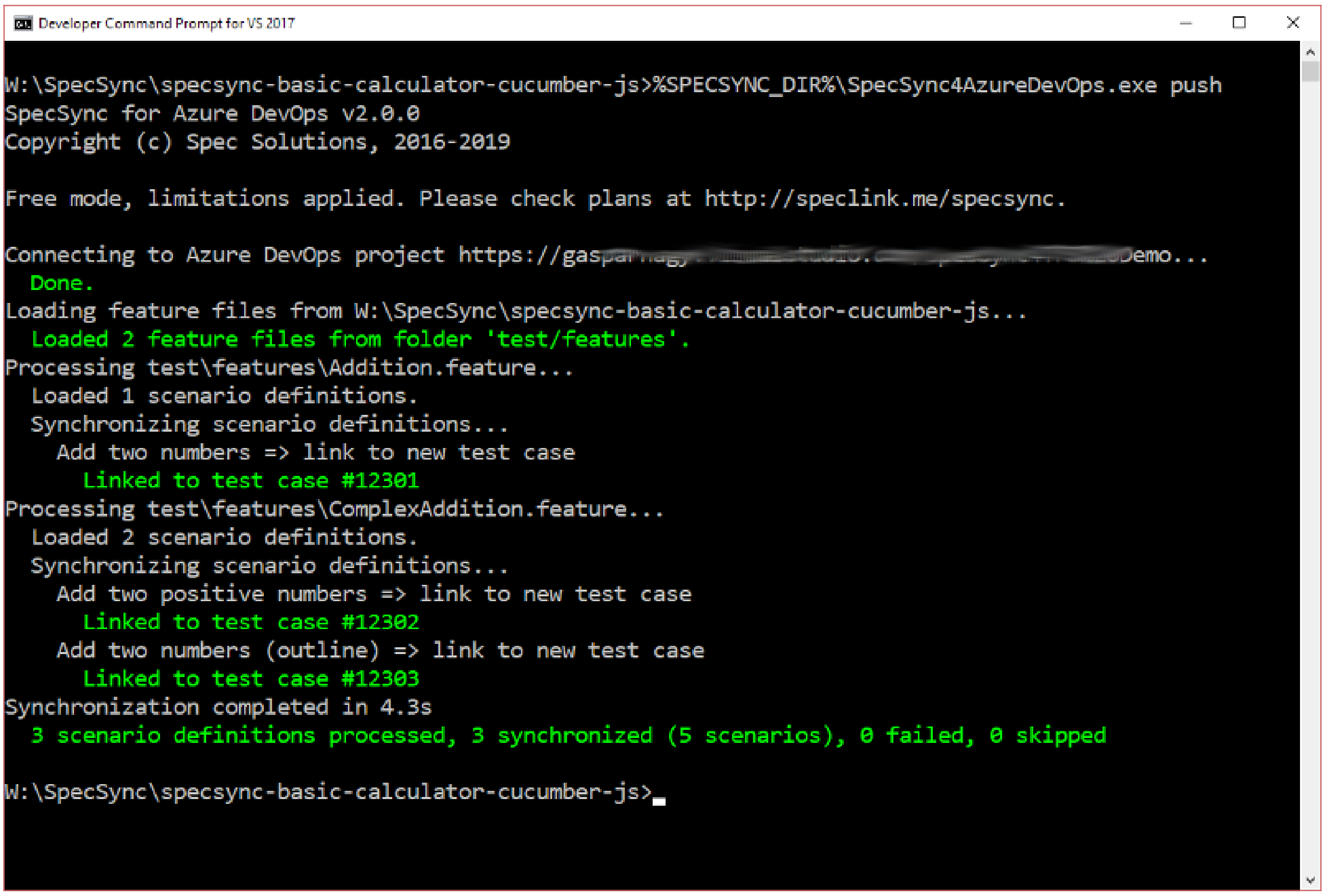
The `specsync.json` after basic configuration has been set

```
{
  "$schema": "http://schemas.specsolutions.eu/specsync4azuredevops-config-la

  // See configuration options and samples at http://speclink.me/specsynccon
  // You can also check the 'specsync-sample.json' file in the 'docs' folder

  "remote": {
    "projectUrl": "https://specsyncdemo.visualstudio.com/MyCalculator",
    "user": "52yny....................................ycsetda"
  }
}
```

First synchronization

1. Make sure your project runs.
2. We recommend starting from a state where
   - all tests pass,
   - the modified files are checked in to source control.
3. Open a command line prompt and navigate to the project root folder
4. Call `path-to-specsync-package/SpecSync4AzureDevOps.exe push` to invoke the synchronization. See Using SpecSync on OSX/Linux page for more details on how to invoke the synchronization tool on different platforms.
5. If you haven't specified any credentials in the configuration file, an authentication dialog will popup, where you have to specify your credentials for accessing the Azure DevOps project.

As a result, the scenarios from the project will be linked to newly created Azure DevOps test cases, and you will see a result like this.



```
W:\SpecSync\specsync-basic-calculator-cucumber-js>%SPECSYNC_DIR%\SpecSync4AzureDevOps.exe push
SpecSync for Azure DevOps v2.0.0
Copyright (c) Spec Solutions, 2016-2019

Free mode, limitations applied. Please check plans at http://speclink.me/specsync.

Connecting to Azure DevOps project https://gaspa[redacted]Demo...
  Done.
Loading feature files from W:\SpecSync\specsync-basic-calculator-cucumber-js...
  Loaded 2 feature files from folder 'test/features'.
Processing test\features\Addition.feature...
  Loaded 1 scenario definitions.
  Synchronizing scenario definitions...
    Add two numbers => link to new test case
      Linked to test case #12301
Processing test\features\ComplexAddition.feature...
  Loaded 2 scenario definitions.
  Synchronizing scenario definitions...
    Add two positive numbers => link to new test case
      Linked to test case #12302
    Add two numbers (outline) => link to new test case
      Linked to test case #12303
Synchronization completed in 4.3s
  3 scenario definitions processed, 3 synchronized (5 scenarios), 0 failed, 0 skipped

W:\SpecSync\specsync-basic-calculator-cucumber-js>
```

Result of the first synchronization

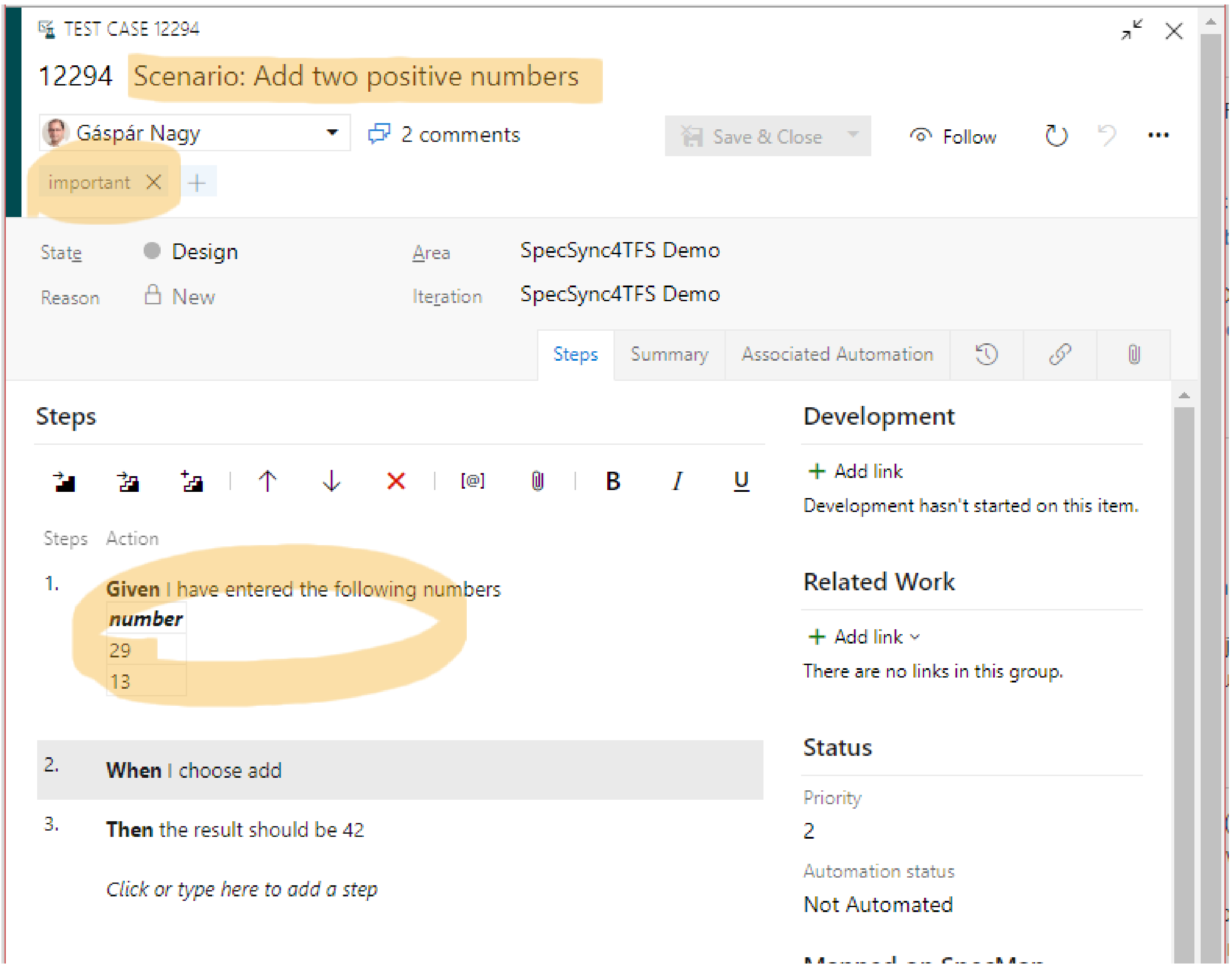
Note: Scenarios are synchronized to normal, Scenario Outlines to parametrized test cases.

Useful hint for testing: Normally you cannot delete work items from Azure DevOps, so testing the initial linking is harder. If you have Visual Studio installed, there is a tool called `witadmin` available from the VS command prompt. With the `destroywi` command of this tool you can delete work items. See `witadmin help destroywi` for details, and use it carefully.

Check Test Case in Azure DevOps

1. Find one of the created test case in Azure DevOps. The easiest way to do this is to open the Azure DevOps URL in a browser and specify the test case ID (e.g. `#12302)` in the "Search" text box in the upper right corner of the web page.

You should see something like this.


TEST CASE 12294

12294 Scenario: Add two positive numbers

Gáspár Nagy

2 comments

important

State: Design

Reason: New

Area: SpecSync4TFS Demo

Iteration: SpecSync4TFS Demo

Steps

Steps	Action
1.	**Given** I have entered the following numbers ***number*** 29 13
2.	**When** I choose add
3.	**Then** the result should be 42
	Click or type here to add a step

Development

Development hasn't started on this item.

Related Work

There are no links in this group.

Status

Priority: 2

Automation status: Not Automated

A newly created test case in Azure DevOps

There are a couple of things you can note here.

- The name of the scenario has been synchronized as the title of the test case. (The "Scenario:" prefix can be omitted by changing the synchronization format configurations.)
- The tags of the scenario have been synchronized as test case tags.
- The steps of the scenario have been synchronized as test case steps. (The *Then* steps can also be synchronized into the *Expected result* column of the test case step list and you can change a couple of other formatting options as well.)

Verify feature file and commit changes

1. Open one of the feature files from in the IDE. SpecSync modified the file and added a few tags.

2. Each scenario and scenario outline has been tagged with a `@tc:...` tag making the link between the scenario and the created test case.

```
@tc:12302
@important
Scenario: Add two positive numbers
```

Note: The feature files are changed only when synchronizing new scenarios (linking). To avoid file changes (e.g. when running the synchronization from a build server) the `--buildServerMode` *command line switch can be used. See Synchronizing test cases from build for details.*

Verify if the project still compiles and the tests pass (they should, since we have only added tags), and commit (check-in) your changes.

Synchronize an update

Now let's make a change in one of the scenarios and synchronize the changes to the related test case.

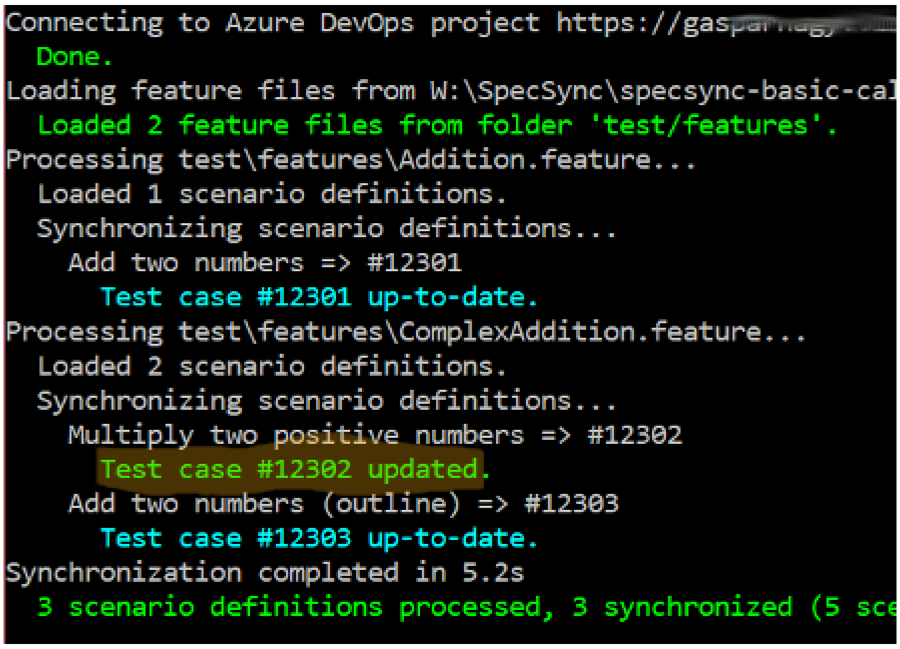
1. Update the title and the steps of the scenario, for example change the scenario `Add two positive numbers` to `Multiply two positive numbers`, change `add` to `multiply` in the *When* step and update the expected result to `377`:

```
@tc:12302
Scenario: Multiply two positive numbers
  Given I have entered the following numbers
      | number |
      | 29     |
      | 13     |
  When I choose multiply
  Then the result should be 377
```

2. Make sure it still compiles and the test passes.
3. Run the synchronization again:

```
path-to-specsync-package/SpecSync4AzureDevOps.exe push
```

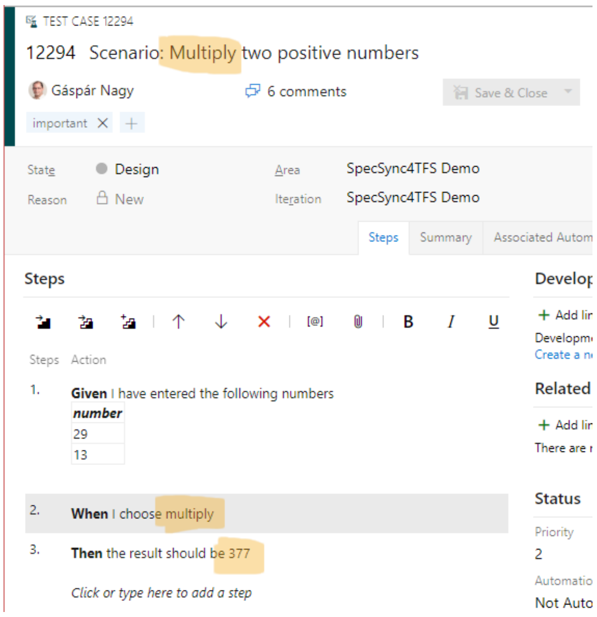
The result shows that the test case for the scenario has been updated, but the other test cases have remained unchanged (*up-to-date*).



```
Connecting to Azure DevOps project https://gaspar[redacted]
  Done.
Loading feature files from W:\SpecSync\specsync-basic-cal
  Loaded 2 feature files from folder 'test/features'.
Processing test\features\Addition.feature...
  Loaded 1 scenario definitions.
  Synchronizing scenario definitions...
    Add two numbers => #12301
      Test case #12301 up-to-date.
Processing test\features\ComplexAddition.feature...
  Loaded 2 scenario definitions.
  Synchronizing scenario definitions...
    Multiply two positive numbers => #12302
      Test case #12302 updated.
    Add two numbers (outline) => #12303
      Test case #12303 up-to-date.
Synchronization completed in 5.2s
  3 scenario definitions processed, 3 synchronized (5 sce
```

Result of synchronizing an update

1. Refresh the test case in your browser to see the changed title and steps.



TEST CASE 12294

12294 Scenario: Multiply two positive numbers

Gáspár Nagy   6 comments   Save & Close

important

State: Design   Area: SpecSync4TFS Demo
Reason: New   Iteration: SpecSync4TFS Demo

Steps | Summary | Associated Autom

Steps

Steps	Action
1.	**Given** I have entered the following numbers ***number*** 29 13
2.	**When** I choose multiply
3.	**Then** the result should be 377
	Click or type here to add a step

Develop
+ Add lir
Developm
Create a n

Related
+ Add lir
There are r

Status
Priority: 2
Automatio: Not Auto

Note: For executing complex test cases, further verification and planning steps might be required after the test case has been changed. SpecSync can reset the test case state to a configured value (e.g. `Design` *) in order to ensure that these steps are not forgotten. For more information on this, check the synchronization state configuration documentation.*

Group synchronized test cases to a test suite

We have seen already how to synchronize scenarios to test cases. To be able to easily find these test cases in Azure DevOps, they can be added to test suites. SpecSync can automatically add/remove the synchronized test cases to a test suite. For this you have to specify the name or the ID or the name of the test suite in the configuration.

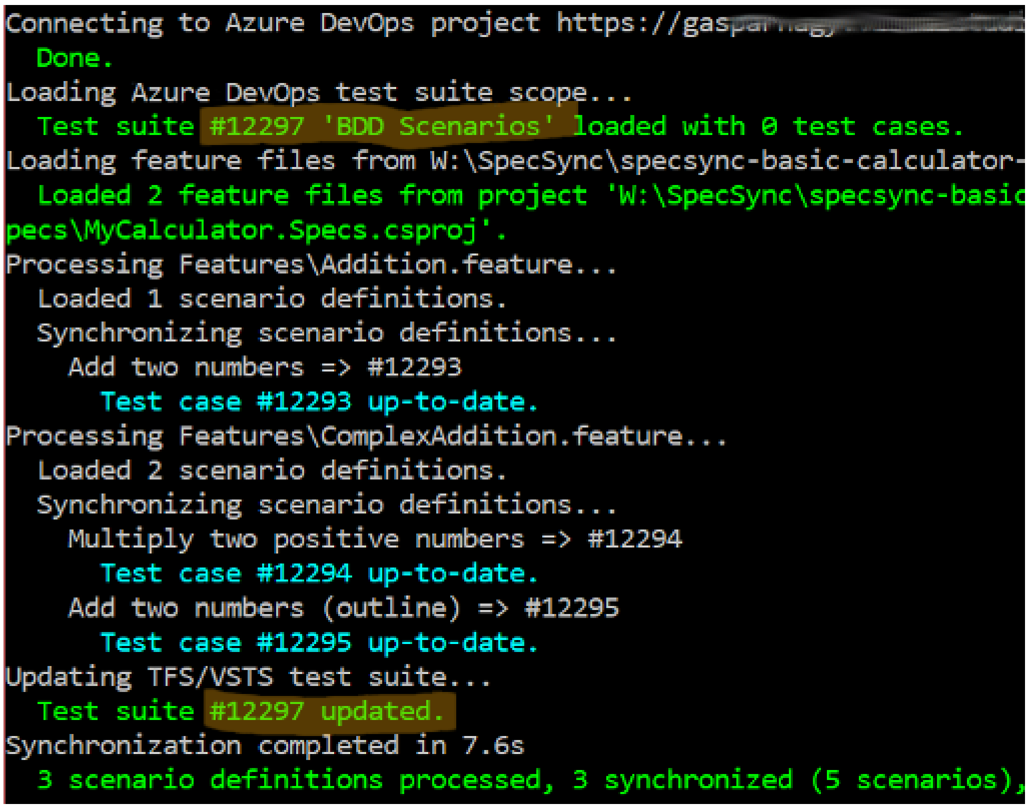
1. Create a "Static suite" (e.g. "BDD Scenarios") in Azure DevOps. (For that you have to navigate to "Test plans" and create and select a test plan first.)
2. Specify the name of the test suite in the `remote/testSuite/name` entry of the `specsync.json` file. (Alternatively you can specify the ID of the suite in `remote/testSuite/id`. The suite names are not unique in Azure DevOps!)

```
{
  "$schema": "http://schemas.specsolutions.eu/specsync4azuredevops-conf

  // See configuration options and samples at http://speclink.me/specsyn
  // You can also check the 'specsync-sample.json' file in the 'docs' fo

  "remote": {
    "projectUrl": "https://specsyncdemo.visualstudio.com/MyCalculator",
    "user": "52yny4a....................................ycsetda",
    "testSuite": {
      "name": "BDD Scenarios"
    }
  }
}
```

3. Make sure that the project compiles and the tests pass.
4. Run the synchronization again:

```
path-to-specsync-package/SpecSync4AzureDevOps.exe push
```

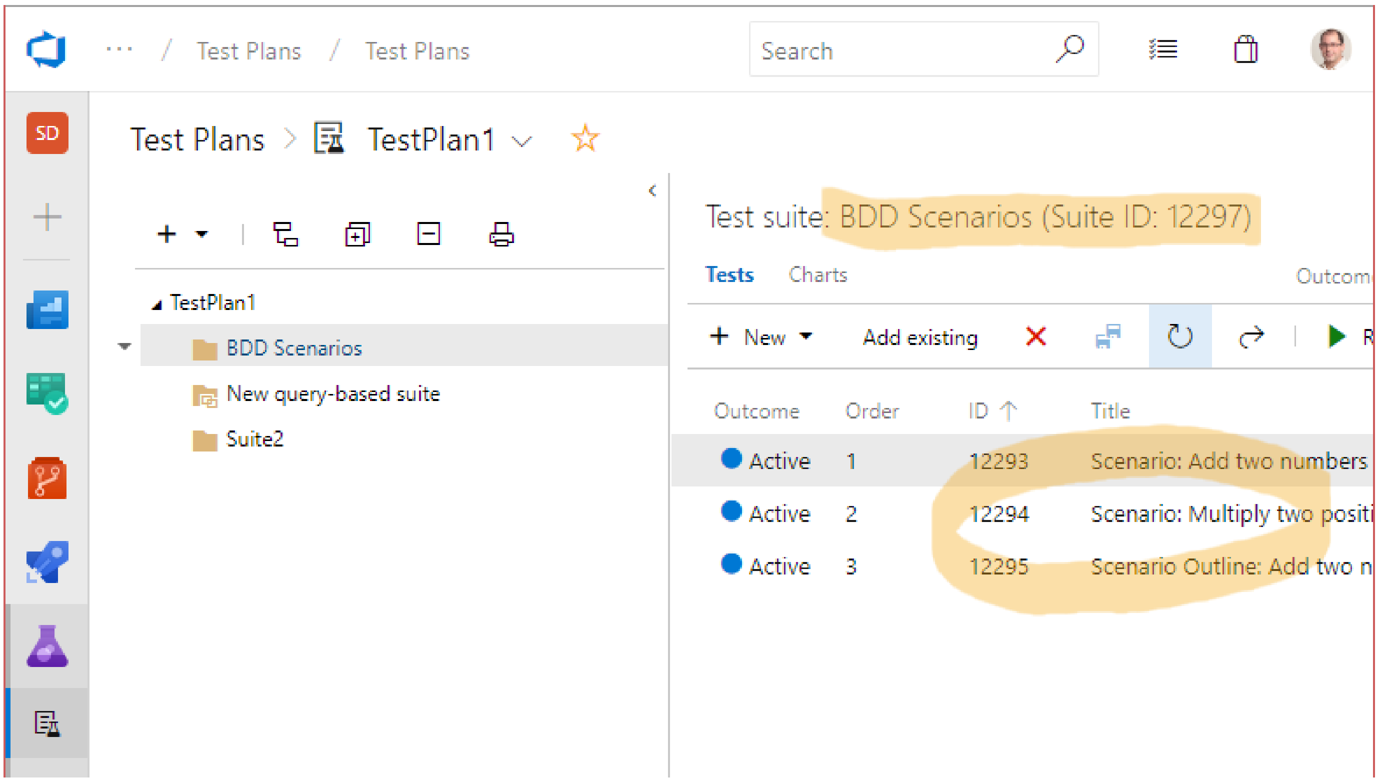
The synchronization will proceed with the result similar to this.



```
Connecting to Azure DevOps project https://gaspar[redacted]
  Done.
Loading Azure DevOps test suite scope...
  Test suite #12297 'BDD Scenarios' loaded with 0 test cases.
Loading feature files from W:\SpecSync\specsync-basic-calculator-
  Loaded 2 feature files from project 'W:\SpecSync\specsync-basic
pecs\MyCalculator.Specs.csproj'.
Processing Features\Addition.feature...
  Loaded 1 scenario definitions.
  Synchronizing scenario definitions...
    Add two numbers => #12293
      Test case #12293 up-to-date.
Processing Features\ComplexAddition.feature...
  Loaded 2 scenario definitions.
  Synchronizing scenario definitions...
    Multiply two positive numbers => #12294
      Test case #12294 up-to-date.
    Add two numbers (outline) => #12295
      Test case #12295 up-to-date.
Updating TFS/VSTS test suite...
  Test suite #12297 updated.
Synchronization completed in 7.6s
  3 scenario definitions processed, 3 synchronized (5 scenarios),
```

Synchronize scenarios to test suite

SpecSync has added the test cases to the test suite.



Test suite: BDD Scenarios (Suite ID: 12297)

Outcome	Order	ID ↑	Title
● Active	1	12293	Scenario: Add two numbers
● Active	2	12294	Scenario: Multiply two positi
● Active	3	12295	Scenario Outline: Add two n

Test cases were added to the test suite

Note: Since the test suite names are not unique in Azure DevOps, you can also specify the test suite ID in the `remote/testSuite/id` *setting.*

Installation

For a detailed installation instructions, please check the Getting started guide.

For .NET (SpecFlow) projects

For .NET (SpecFlow) projects, SpecSync can be installed via NuGet. The synchronization tool can be installed by adding the `SpecSync.AzureDevOps` package from NuGet.org.

```
PM> Install-Package SpecSync.AzureDevOps
```

For synchronizing automated test cases using the "Test Suite based execution" strategy with MsTest and NUnit, an SpecFlow plugin has to be installed additionally. The name of the package depends on the SpecFlow version you use, e.g. for SpecFlow v2.3 the `SpecSync.AzureDevOps.SpecFlow.2-3` package has to be used. See more details in Synchronizing automated test cases.

Notes for installing SpecSync to SDK-Style .NET Projects (e.g. .NET Core)

The `SpecSync.AzureDevOps` NuGet package contains the synchronization tool, but it also contains some supporting files to get started with SpecSync easier. Unfortunately when using NuGet for SDK-Style .NET Projects (e.g. .NET Core), the NuGet infrastructure does not allow including editable content files in the target project, so these helper files are not added to the project by default. You can add these files manually from the project from the `content` folder of the NuGet package.

- `specsync.json` -- this is a default configuration file. Alternatively you can also create a new JSON file based on the the samples shown in the Configuration page of the documentation.
- specsync4azuredevops.cmd -- this file makes it easier to execute the synchronization tool. Alternatively you can also invoke the `SpecSync4AzureDevOps.exe` directly from the NuGet packages folder. (See more details in the Usage page.)

For any platforms (e.g. for Cucumber)

SpecSync can be downloaded as a ZIP file, the file contains the synchronization tool inside the `tools` folder. The download links can be found on the Downloads page.

Prerequisites for running the synchronization tool.

- On Windows systems: .NET framework 4.5 or later
- On OSX/Linux: Mono

If you need to use SpecSync but cannot ensure the prerequisites, please contact support (specsync@specsolutions.eu).

Install SpecSync as .NET Core tool

> This install method is available for machines with .NET Core SDK 2.1 installed regardless of the .NET version of the project that is going to be synchronized. If this cannot be ensured, you can use the the SpecSync.AzureDevOps.Console NuGet package or download one of the pre-compiled binaries.

> Installing SpecSync as a .NET Core tool is available from SpecSync v2.2 or later (including v2.2 pre-releases).

The most convenient way to use SpecSync is to install it as a local .NET Core tool. This way you can use different SpecSync versions for different projects and the required SpecSync version is registered in the project repository. You can read more about .NET Core local tools on Microsoft Docs.

.NET Core local tools are only supported with .NET Core SDK 3.0 or later. In case you only have .NET Core 2.1 SDK installed, you can install SpecSync as a global tool or use the SpecSync.AzureDevOps.Console NuGet package.

Step 1 - Initialize .NET Core local tool configuration (if needed)

If you haven't used any .NET Core local tool in your project, you need to create the necessary configuration file. Otherwise this step can be skipped.

For initializing the configuration files, you need to run the `dotnet new tool-manifest` command from the solution or repository root directory.

```
dotnet new tool-manifest
```

This command creates a manifest file named `dotnet-tools.json` under the `.config` directory.

Step 2 - Install SpecSync as a .NET Core local tool

Once the .NET local tool configuration is initialized SpecSync can be easily installed using the `dotnet tool install` command.

```
dotnet tool install SpecSync.AzureDevOps --version 2.2.0-pre20200414
```

> The `--version` setting is only required until the .NET Core tool support is only available as pre-release. It is recommended to specify the latest pre-release version listed on NuGet.org.

Step 3 - Verify installation

SpecSync is ready to run using the `dotnet specsync` command. You can test the installation by checking the installed SpecSync version.

```
dotnet specsync version
```

Step 4 - Restore SpecSync for other developers working with your project

To be able to use the .NET Core local tools you have installed by other developers of the same project, they need to "restore" the installed tools. This can be performed using the `dotnet tool restore` command, that restores all .NET Core local tools of the project. See more information about this command on Microsoft Docs.

```
dotnet tool restore
```

Usage

The SpecSync install package contains a command line tool (`SpecSync4AzureDevOps.exe`) inside the `tools` folder. All synchronization operations can be performed by invoking this tool from the local environment or from the CI build process. (For .NET projects, the package adds a `specsync4azuredevops.cmd` script file to the project for calling the SpecSync command line tool conveniently.)

The synchronization tool provides different commands. For synchronizing the scenarios to Azure DevOps, the `push` command can be used. The configuration options have to be provided in a json configuration file, called `specsync.json` by default. It is recommended to invoke the command line tool from the project folder, otherwise the path of the configuration file has to be specified explicitly.

```
path-to-specsync-package\tools\SpecSync4AzureDevOps.exe push
```

For a detailed setup instructions, please check the Getting started guide. For a complete list of configuration options check the Configuration documentation.

Note: SpecSync collects anonymous error diagnostics and statistics. Neither user nor machine names, nor Azure DevOps URLs, nor test case & test suite names nor IDs are collected! This can be disabled with the `--disableStats` *parameter.*

Available SpecSync commands

- `push` -- Pushes changes of the scenarios on the local repository to the Azure DevOps server. This by default includes linking of new scenarios to new test cases (link) and updating test cases of linked scenarios (update).
- `pull` -- Pulls changes from Azure DevOps server to the local repository. This by default includes creation of new scenarios from unlinked test cases (create) and changing scenarios of linked test cases (change). See Two-way synchronization for details.

- `publish-test-results` -- Publish local test results to Azure DevOps server. See more details about the command in the "Assembly based execution strategy" section of the Synchronizing automated test cases article.
- `help` -- Displays more information on a specific command.
- `version` -- Displays version information.

Common command line options

The following command line options are available for all synchronization commands.

- `[config-file-path]` -- The path of the SpecSync configuration file, absolute path or relative to the current folder, e.g. `MyProject.Specs\specsync.json` . (Default: use `specsync.json` from the current folder)
- `--user [user-name]` -- The Azure DevOps user name or personal access token (PAT). Overrides `remote/user` setting of the configuration file. See Azure DevOps authentication options for details. (Default: [use from config file or interactive prompt])
- `--password` -- The password for the Azure DevOps user. Overrides `remote/password` setting of the configuration file. See Azure DevOps authentication options for details. (Default: [use from config file or interactive prompt])
- `--buildServerMode` -- If specified, only those changes will be performed that do not need any change in the local feature file. Linking new scenarios or pulling changes from Azure DevOps are skipped. Overrides `synchronization/enableLocalChanges` setting of the configuration file. See Synchronizing test cases from build for details. (Default: false)
- `--tagFilter` -- A tag expression of scenarios that should be included in the current synchronization (e.g. `@current_sprint and @done`). See Filters and scopes for details. (Default: [not filtered by tags])
- `--force` -- If specified, SpecSync update test cases even if there is no local change and the test case was not modified remotely. (Default: false)
- `--license` -- The path to the license file; can be relative to the project folder. Overrides `toolSettings/licensePath` setting of the configuration file. See Licensing for details. (Default: [use from config file or `specsync.lic`])
- `--baseFolder` -- The base folder where SpecSync searches for project, feature and license files by default. (Default: [folder of the configuration file])
- `--disableStats` -- If specified, SpecSync will not collect anonymous error diagnostics and statistics. Overrides `toolSettings/disableStats` setting of the configuration file. (Default: false)

- `-v`, `--verbose` -- If specified, error messages and trace information will be displayed more in detail. Overrides `toolSettings/outputLevel` setting of the configuration file. (Default: false)

Synchronization command line options (for push and pull)

All common command line options can be used and in addition to that the following options can be specified for `push` and `pull` commands.

- `--buildServerMode` -- If specified, only those changes will be performed that do not need any change in the local feature file. Linking new scenarios or pulling changes from Azure DevOps are skipped. Overrides `synchronization/enableLocalChanges` setting of the configuration file. See Synchronizing test cases from build for details. (Default: false)
- `--tagFilter` -- A tag expression of scenarios that should be included in the current synchronization (e.g. `@current_sprint and @done`). See Filters and scopes for details. (Default: [not filtered by tags])
- `--force` -- If specified, SpecSync update test cases even if there is no local change and the test case was not modified remotely. (Default: false)

Publish test results command line options (for publish-test-results)

All common command line options can be used and in addition to that the following options can be specified for `publish-test-results` command.

See more details about the command in the "Assembly based execution strategy" section of the Synchronizing automated test cases article.

- `--testConfiguration` -- The Azure DevOps test configuration name or ID to publish the results for. For specifying an ID, use `#1234` format. (Default: [use from config file])
- `--testResultFile` -- The file path of the TRX test result file to publish. (Default: [use from config file])

Examples

Synchronize local changes to Azure DevOps (`specsync.json` config file is in the current folder):

```
path-to-specsync-package\tools\SpecSync4AzureDevOps.exe push
```

Get help about command line options for `push` command:

```
path-to-specsync-package\tools\SpecSync4AzureDevOps.exe help push
```

Synchronize local changes to Azure DevOps with custom config file:

```
path-to-specsync-package\tools\SpecSync4AzureDevOps.exe push MyProject.Specs\
```

Synchronize local changes to Azure DevOps on build server:

```
path-to-specsync-package\tools\SpecSync4AzureDevOps.exe push --buildServerMod
```

Configuration

This section contains a detailed reference of the SpecSync configuration options.

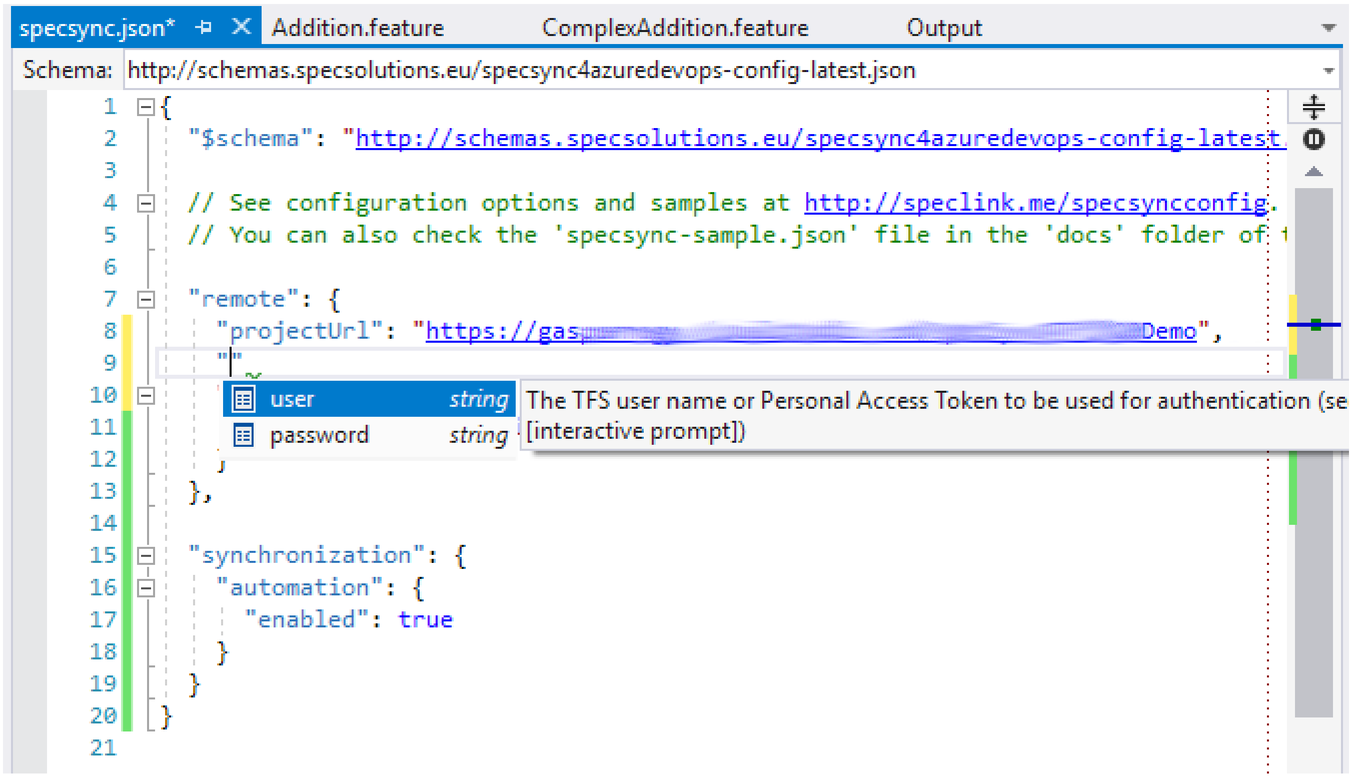
For a detailed setup instructions, please check the Getting started guide. For a complete list of command line options of the synchronization tool check the Usage documentation.

The `specsync.json` configuration file.

SpecSync can be configured using a json configuration file, by default called `specsync.json`. This file contains all information required to perform the different synchronization tasks. Some settings of the configuration file can be also overridden from the command line options of the synchronization tool, these are listed in the Usage guide.

The `specsync.json` configuration file is a standard JSON file, but it also allows `//` style comments. There is a JSON schema available for the configuration file that contains the available configuration options and a short description.

Open the SpecSync config files in Visual Studio (or other editor that supports JSON schema, like Visual Studio Code) to get auto-completion for editing and documentation hints if you hover your mouse over a setting.



Schema: http://schemas.specsolutions.eu/specsync4azuredevops-config-latest.json

```
{
  "$schema": "http://schemas.specsolutions.eu/specsync4azuredevops-config-latest.

  // See configuration options and samples at http://speclink.me/specsyncconfig.
  // You can also check the 'specsync-sample.json' file in the 'docs' folder of t

  "remote": {
    "projectUrl": "https://gas[redacted]Demo",
    "|"

    }
  },

  "synchronization": {
    "automation": {
      "enabled": true
    }
  }
}
```

user string
password string

The TFS user name or Personal Access Token to be used for authentication (se
[interactive prompt])

Code completion of the configuration file in Visual Studio

Examples

The following example shows a minimal configuration file.

```
{
  "$schema": "http://schemas.specsolutions.eu/specsync4azuredevops-config-la

  "remote": {
    "projectUrl": "https://specsyncdemo.visualstudio.com/MyCalculator",
  }
}
```

A detailed sample configuration file that contains nearly all settings can be found at http://schemas.specsolutions.eu/specsync-sample.json.

Configuration sections

The settings in the configuration file are grouped into different configuration sections. Each configuration section is a JSON object following the syntax:

```
{  // start of the file
  ...
  "section1": {
    // settings for section 'section1'
  },
  ...
}  // end of the file
```

Note: The leading comma (`,` *) after the curly brace close (* `}` *) of* `section1` *is not needed if this is the last section in the file.*

Available configuration sections

The following configuration sections can be used. Click to the name of the section for detailed documentation.

- `toolSettings` -- settings for the synchronization tool
- `local` -- settings for the local repository (file system) containing the feature files
- `remote` -- Settings for accessing the test cases on the remote Azure DevOps server
- `synchronization` -- synchronization settings
- `specFlow` -- settings related to synchronizing SpecFlow projects
- `customizations` -- configure customizations

toolSettings

This configuration section contains settings for the synchronization tool.

The following example shows the available options within this section.

```
{
  ...
  "toolSettings": {
    "licensePath": "specsync.lic",
    "disableStats": false,
    "outputLevel": "normal"
  },
  ...
}
```

Settings

- `licensePath` -- Path for the license file. Can contain an absolute or a relative path to the config file folder. It may contain environment variables in `...%MYENV%...` form. Can be overridden by the `--license` command line option. See Licensing for details. (Default: `specsync.lic`)
- `disableStats` -- If set to true, SpecSync will not collect anonymous error diagnostics and statistics. Can be overridden by the `--disableStats` command line option. (Default: `false`)
- `outputLevel` -- Set the detail level of error messages and trace information displayed by the tool. Available options: `normal`, `verbose` and `debug`. Can be overridden by the `--verbose` command line option. (Default: `normal`)

[Back to the Configuration guide]

local

This configuration section contains settings for the local repository (file system) containing the feature files.

The following example shows the available options within this section.

```
{
  ...
  "local": {
    "featureFileSource": {
      "type": "projectFile",
      "filePath": "MyProject.Specs\\MyProject.Specs.csproj"
    },

    "tags": "@done and not (@ignored or @planned)",

    "defaultFeatureLanguage": "en-US"
  },
  ...
}
```

Settings

- `featureFileSource` -- The feature file source configuration. (Default: *[Detect project file in the folder of the configuration file]*)
  - `featureFileSource/type` -- The type of the feature file source configuration. Available options: `projectFile` , `folder` , `listFile` and `stdIn` .
    - `projectFile` -- Loads feature file list from a .NET C# project file (`.csproj`). SpecSync detects the project file by extension in the folder of the configuration file by default, but the project file path can also be specified in the `local/featureFileSource/filePath` setting.
    - `folder` -- Loads the feature files from a particular folder and its sub-folders. The folder can be specified in the `featureFileSource/folder` setting.

        - `listFile` -- Loads the feature file list from a text file. Each line of the text file should contain the path of a feature file. Empty lines and lines start with `#` are ignored. The feature file path can be absolute or a relative path to the config file folder. See example below and Getting started using Cucumber for more details.
        - `stdIn` -- Loads the feature file list from the standard input stream. The content of the input stream are handled in the same was as the `listFile` option. See example below and Getting started using Cucumber for more details.
    - `featureFileSource/filePath` -- The path of the feature file source file. Can contain an absolute or a relative path to the config file folder. It may contain environment variables in `...%MYENV%...` form.
    - `featureFileSource/folder` -- The folder to search the feature files in when `type` was set to `folder`. Can contain an absolute or a relative path to the config file folder. It may contain environment variables in `...%MYENV%...` form. (Default: *[load feature files from the folder of the config file]*)
- `tags` -- A tag expression of scenarios that should be included in synchronization (e.g. `not @ignore` or `@done and not (@ignored or @planned)`). See Filters and scopes for details. (Default: *[all scenarios included]*)
- `defaultFeatureLanguage` -- The default feature file language, e.g. `de-AT`. (Default: *[get from SpecFlow config or use `en-US`]*)

Example: Synchronize feature files of a SpecFlow project

The SpecFlow project can be detected in the folder of the configuration file usually, in this case **no additional configuration is required**. (SpecSync tries to find a `.csproj` file in the folder.) In case there are multiple .NET project in the folder or the configuration file is not stored in the project root, you should configure SpecSync as below:

```
{
  ...
  "local": {
    "featureFileSource": {
      "type": "projectFile",
      "filePath": "MyProject\\MyProjectFile.csproj"
    }
  },
  ...
}
```

Example: Synchronize feature files from the `features` folder

For Cucumber-based projects, it is common to store the feature files in a folder called `features`. In order to synchronize the feature files with this setup, the feature file source has to be configured to `folder` and the required folder path has to be specified in the `folder` setting:

```
{
  ...
  "local": {
    "featureFileSource": {
      "type": "folder",
      "folder": "features"
    }
  },
  ...
}
```

You can invoke the syncronization as usual:

```
path-to-specsync-package/tools/SpecSync4AzureDevOps.exe push
```

Example: Synchronize feature files from a sub-folder

Let's imagine a folder structure as the following:

```
features/
features/feature_a.feature
features/group_a/feature_b.feature
features/group_a/feature_c.feature
features/group_a/area_1/feature_d.feature
features/group_b/feature_e.feature
```

In this example SpecSync is configured to synchronize all feature files within `features/group_a` (so currently `feature_b.feature`, `feature_c.feature`,

`feature_d.feature`), but without listing them explicitly in a file.

For this, the feature file source has to be configured to `folder` and the required folder path has to be specified in the `folder` setting:

```
{
  ...
  "local": {
    "featureFileSource": {
      "type": "folder",
      "folder": "features/group_a"
    }
  },
  ...
}
```

You can invoke the syncronization as usual:

```
path-to-specsync-package/tools/SpecSync4AzureDevOps.exe push
```

Example: Synchronize specific feature files

The following example synchronizes a specific set of feature files.

We need a text file with the list of feature files to be synchronized, let's call it `specsync-features.txt` . Save it in the project root folder, where the `specsync.json` configuration file is located.

```
features/addition.feature
features/special/complex_addition.feature
# this line is ignored
features/multiplication.feature
```

All paths in this file can be relative to the folder of the config file. On Windows platform the `\` character has to be used instead of the `/` .

The list file has to be specified in the config file:

```
{
  ...
  "local": {
    "featureFileSource": {
      "type": "listFile",
      "filePath": "specsync-features.txt"
    }
  },
  ...
}
```

You can invoke the syncronization as usual:

```
path-to-specsync-package/tools/SpecSync4AzureDevOps.exe push
```

Example: Synchronize feature files from a dynamic folder list

In this example we will synchronize the same feature files as in the previous example (all feature files within `features/group_a`), but now the list of feature file is provided by an external tool and SpecSync will receive it through through the standard input stream with piping.

First, let's set the config file to:

```
{
  ...
  "local": {
    "featureFileSource": {
      "type": "stdIn"
    }
  },
  ...
}
```

After that we can generate the file list and invoke the synchronization. Let's suppose the current folder is the project root.

On Windows (CMD):

```
dir /s /b features\group_a\*.feature | path-to-specsync-package\tools\SpecSyn
```

On Windows (PowerShell):

```
gci -Path .\features\group_a -r *.Feature | % FullName | path-to-specsync-pac
```

On OSX and Linux:

```
find features/group_a/ -name *.feature | path-to-specsync-package/tools/SpecS
```

[Back to the Configuration guide]

remote

This configuration section contains settings for accessing the test cases on the remote Azure DevOps server.

The following example shows the available options within this section.

```
{
  ...
  "remote": {
    "projectUrl": "https://dev.azure.com/myorganization/MyProject",
    "user": "myuser",
    "password": "%MYPWD%",
    "testSuite": {
      "name": "BDD Scenarios"
    }
  },
  ...
}
```

Settings

- `projectUrl` -- The full URL of the Azure DevOps or VSTS project (including project collection name if necessary). Must not include project team name: for multi-team projects the root project URL has to be specified. See What is my Azure DevOps project URL for details. (Mandatory.)
- `user` -- The Azure DevOps user name or personal access token (PAT) to be used for authentication. It may contain environment variables in `...%MYENV%...` form. See Azure DevOps authentication options for details. (Default: *[interactive prompt]*)
- `password` -- The password to be used for authentication. It may contain environment variables in `...%MYENV%...` form. See Azure DevOps authentication options for details. (Default: *[interactive prompt]*)
- `testSuite` -- Specifies a test suite within the Azure DevOps project as a target container of the synchronized test cases. If the test suite is specified, SpecSync will add the

synchronized test cases to it for `push` command and consider the test cases within the suite for `pull` command. See Group synchronized test cases to a test suite for details. (Default: *[test cases are not included to a test suite]*)

  - `testSuite/name` -- The name of the test suite. For suites with non-unique names, please use the `testSuite/id` setting.
  - `testSuite/id` – "The ID of the test suite as a number (e.g. `id: 12345`)."

[Back to the Configuration guide]

synchronization

This configuration section contains synchronization settings.

The following example shows the available options within this section.

```
{
  ...
  "synchronization": {
    "enableLocalChanges": true,
    "forceUpdate": true,
    "testCaseTagPrefix": "tc",

    "pull": {
      "enabled": true,
      "enableCreatingScenariosForNewTestCases": false
    },

    "automation": {
      "enabled": true,
      "skipForTags": "@manual"
    },

    "state": {
      "setValueOnChangeTo": "Design"
    },

    "areaPath": {
      "mode": "setOnLink",
      "value": "\\MyArea"
    },
    "iterationPath": {
      "mode": "setOnLink",
      "value": "\\Iteration1"
    },
    "links": [
      {
        "targetWorkItemType": "ProductBacklogItem",
        "tagPrefix": "pbi",
        "relationship": "Child",
        "mode": "createIfMissing"
      },
      {
        "tagPrefix": "bug"
      }
    ],
    "format": {
      "useExpectedResult": false,
      "syncDataTableAsText": false,
      "prefixBackgroundSteps": true,
      "prefixTitle": true
    }
  },
  ...
}
```

Settings

- `enableLocalChanges` -- Enables changing feature files in the local repository. If set to false (called *build server mode*), only those changes will be performed that do not need any change in the local feature files. Linking new scenarios or pulling changes from test cases will be skipped. Can be overridden by the `--buildServerMode` command line option. See Synchronizing test cases from build for details. (Default: `true`)
- `forceUpdate` -- If set to true, SpecSync update test cases even if there is no local change and the test case was not modified remotely. Can be overridden by the `--force` command line option. (Default: `false`)
- `testCaseTagPrefix` -- The tag prefix for specifying the test cases. E.g. specify `testcase` for referring to test cases using a tag, like `@testcase:1234`. (Default: `tc`)

Sub-sections

The following configuration sub-sections can be used. Click to the name of the section for detailed documentation.

- `pull` -- Settings to configure the pull behavior. See Two-way synchronization for details.
- `automation` -- Settings to configure synchronizing automated test cases. See Synchronizing automated test cases for details.
- `state` -- Settings to configure how the test case *state* field should be updated.
- `areaPath` -- Settings to configure how the test case *area path* field should be updated. See Add new test cases to an Area or an Iteration for details.

- `iterationPath` -- Settings to configure how the test case *iteration path* field should be updated. See Add new test cases to an Area or an Iteration for details.
- `links` -- Settings to configure how test case links should be updated based on the tags of the scenario. See Linking work items using tags for details.
- `format` -- Settings to configure the format of the synchronized test case. See Configuring the format of the synchronized test cases for details.

[Back to the Configuration guide]

pull

This configuration section contains settings to configure the pull behavior.

Read more about the pull behavior in the Two-way synchronization concept description.

Note: The two-way synchronization is an Enterprise feature.

The following example shows the available options within this sub-section.

```
{
  ...
  "synchronization": {
    ...
    "pull": {
      "enabled": true,
      "enableCreatingScenariosForNewTestCases": false
    },
    ...
  },
  ...
}
```

Settings

- `enabled` -- Enables changing the scenarios in the local repository based on the remote test cases. (Default: `false`)
- `enableCreatingScenariosForNewTestCases` -- Enables creating new scenarios from test cases that are not linked to any scenarios yet. (Default: `false`)

[Back to the `synchronization` Configuration]

[Back to the Configuration guide]

automation

This configuration section contains settings to configure synchronizing automated test cases.

Read more about synchronizing automated test cases in the Synchronizing automated test cases concept description.

The following example shows the available options within this sub-section.

```
{
  ...
  "synchronization": {
    ...
    "automation": {
      "enabled": true,
      "skipForTags": "@manual",
      "testExecutionStrategy": "assemblyBasedExecution"
    },
    ...
  },
  ...
}
```

Settings

- `enabled` -- Specifies whether SpecSync should attempt to create automated test cases. (Default: `false`)
- `skipForTags` -- A tag expression of scenarios that should be excluded from automation (e.g. `@manual or @planned`). (Default: *[all test cases synced as automated]*)
- `testExecutionStrategy` -- Specifies the test execution strategy for the automated test cases. Check Synchronizing automated test cases for details about the execution strategies. Available options: `assemblyBasedExecution` , `testSuiteBasedExecution` , `testSuiteBasedExecutionWithScenarioOutlineWrappers` and `none` (Default: not set)

[Back to the `synchronization` Configuration]

[Back to the Configuration guide]

state

This configuration section contains settings to configure how the test case *state* field should be updated.

The following example shows the available options within this sub-section.

```
{
  ...
  "synchronization": {
    ...
    "state": {
      "setValueOnChangeTo": "Design"
    },
    ...
  },
  ...
}
```

Settings

- `setValueOnChangeTo` -- A state value (e.g. `Design`) to set test case state to when updating or creating a test case during synchronization. Useful for setting back `Ready` test cases to `Design` on change. (Default: *[don't change test case state]*)

[Back to the `synchronization` Configuration]

[Back to the Configuration guide]

areaPath

This configuration section contains settings to configure how the test case *area path* field should be updated.

Read more about the setting the *area path* and *iteration path* fields in the Add new test cases to an Area or an Iteration concept description.

The following example shows the available options within this sub-section.

```
{
  ...
  "synchronization": {
    ...
    "areaPath": {
      "mode": "setOnLink",
      "value": "\\MyArea"
    },
    ...
  },
  ...
}
```

Settings

- `mode` -- Specifies how the area path of the test case should be updated. Available options: `notSet` and `setOnLink`. (Default: `notSet`)
  - `notSet`: the path is not set
  - `setOnLink`: set the path when the test case created and linked to the scenario, but not to update later on.
- `value` -- The area path to set for test cases (e.g. `\\MyArea`). The project name prefix can be omitted.

[Back to the `synchronization` *Configuration]*

[Back to the Configuration guide]

iterationPath

This configuration section contains settings to configure how the test case *iteration path* field should be updated.

Read more about the setting the *area path* and *iteration path* fields in the Add new test cases to an Area or an Iteration concept description.

The following example shows the available options within this sub-section.

```
{
  ...
  "synchronization": {
    ...
    "iterationPath": {
      "mode": "setOnLink",
      "value": "\\Iteration1"
    },
    ...
  },
  ...
}
```

Settings

- `mode` -- Specifies how the iteration path of the test case should be updated. Available options: `notSet` and `setOnLink`. (Default: `notSet`)
  - `notSet`: the path is not set
  - `setOnLink`: set the path when the test case created and linked to the scenario, but not to update later on.
- `value` -- The iteration path to set for test cases (e.g. `\\Iteration1`). The project name prefix can be omitted.

[Back to the `synchronization` *Configuration]*

[Back to the Configuration guide]

links

This configuration section contains settings to configure how test case links should be updated based on the tags of the scenario.

Read more about synchronizing test case links in the Linking work items using tags concept description.

The following example shows how this sub-section can be used to specify a single link type.

```
{
  ...
  "synchronization": {
    ...
    "links": [
      {
        "tagPrefix": "bug"
      }
    ],
    ...
  },
  ...
}
```

The following example shows the available options within this sub-section. This example configures two link types.

```
{
  ...
  "synchronization": {
    ...
    "links": [
      {
        "tagPrefix": "pbi",
        "targetWorkItemType": "ProductBacklogItem",
        "relationship": "Parent",
        "mode": "createIfMissing"
      },
      {
        "tagPrefix": "bug"
      }
    ],
    ...
  },
  ...
}
```

Settings

The `links` sub-section contains an array of link type configurations. Each of them is used to configure a link type. (The second example above configures two link types.)

For each link type configuration the following settings can be used.

- `tagPrefix` -- A tag prefix for specifying the relation for the scenario. E.g. specify `pbi` for linking product backlog items using a tag, like `@pbi:1234` . (Mandatory.)
- `targetWorkItemType` -- The type of the Azure DevOps work item the link refers to. (Default: *[Can link to any work item type]*)
- `relationship` -- Specify the relationship for the created link. E.g. specifying `Parent` means that the linked work item will be the parent of the test case work item. (Default: `Tests` relationship is established)
- `mode` -- Specifies how the links are updated. Available options: `createIfMissing` . (Default: `createIfMissing`)
  - `createIfMissing` -- SpecSync only creates links but never removes them, even if the tag has been removed from the scenario.

[Back to the `synchronization` *Configuration]*

[Back to the Configuration guide]

format

This configuration section contains settings to configure the format of the synchronized test case.

Read more about the test case format options in the Configuring the format of the synchronized test cases concept description.

The following example shows the available options within this sub-section.

```
{
  ...
  "synchronization": {
    ...
    "format": {
      "useExpectedResult": false,
      "syncDataTableAsText": false,
      "prefixBackgroundSteps": true,
      "prefixTitle": true
    }
  },
  ...
}
```

Settings

- `useExpectedResult` – If set to true, *Then* steps will be synchronized to the *Expected result* field of the test case steps. (Default: `false`)
- `syncDataTableAsText` – If set to true, DataTables will be synchronized as plain text instead of HTML tables. (Default: `false`)
- `prefixBackgroundSteps` – If set to true, *Background* steps will be synchronized with the `Background:` prefix. (Default: `true`)
- `prefixTitle` – If set to true, test case title will be synchronized with the `Scenario:` or `Scenario Outline:` prefix. (Default: `true`)

[Back to the `synchronization` Configuration]

[Back to the Configuration guide]

publishTestResults

This configuration section contains settings related to publishing TRX test results.

To read more about publishing test results see the "Assembly based execution strategy" section of the Synchronizing automated test cases article.

The following example shows the available options within this section.

```
{
  ...
  "publishTestResults": {
    "testConfiguration": {
      "name": "Windows 10"
    },
    "testResult": {
      "filePath": "test-result.trx"
    }
  },
  ...
}
```

Settings:

- `testConfiguration` -- Specifies a test configuration within the Azure DevOps project as a target configuration for publishing test results.
  - `testConfiguration/name` -- The name of the test configuration.
  - `testConfiguration/id` -- The ID of the test configuration.
- `testResult` -- The TRX test result configuration.
  - `testResult/filePath` -- The path of the TRX file. Can also be specified as a command line parameter.

specFlow

This configuration section contains settings related to synchronizing SpecFlow projects. These settings are only required for synchronizing automated test cases. See Synchronizing automated test cases for more details.

The following example shows the available options within this section.

```
{
  ...
  "specFlow": {
    "specFlowGeneratorFolder": "..\\packages\\SpecFlow.2.3.0\\tools",
    "scenarioOutlineAutomationWrappers": "iterateThroughExamples",
    "wrapperMethodPrefix": "_SpecSyncWrapper_",
    "wrapperMethodCategory": "SpecSyncWrapper"
  },
  ...
}
```

Settings

- `specFlowGeneratorFolder` -- The path of the SpecFlow generator folder used by the project, that is usually the `tools` folder of the SpecFlow NuGet package, e.g. `packages\\SpecFlow.2.3.0\\tools`. (Default: *[detect generator of the project]*)
- `scenarioOutlineAutomationWrappers` -- Specifies how automation wrapper methods should be generated for synchronizing scenario outlines to automated test cases. Available options: `useTestCaseData` and `iterateThroughExamples`. (Default: `iterateThroughExamples`)
    - `useTestCaseData` -- the generated wrapper method loads the test data for the iterations from the test case. Running the test cases through this wrapper in Azure DevOps generates a detailed report about each iteration, but it cannot be executed locally and also does not work in from Azure DevOps pipeline build. See related section of the Synchronizing automated test cases article for details.

    - `iterateThroughExamples` -- the generated wrapper method iterates through the examples and runs the test for each. A failure of an iteration does not block the remaining iterations. Running the test cases through this wrapper in Azure DevOps generates a single entry in the report, but the details of the entry contain all executed data set.
- `wrapperMethodPrefix` -- The method prefix to be used for the generated automation wrapper methods. (Default: `_SpecSyncWrapper_`)
- `wrapperMethodCategory` -- The test category (trait) be added for the generated automation wrapper methods. (Default: `SpecSyncWrapper`)

[Back to the Configuration guide]

customizations

This configuration section contains settings for configuring customizations.

Note: The customizations described here are Enterprise features.

The following example shows the available options within this section.

```
{
  ...
 "customizations": {
    "branchTag": {
      "enabled": true,
      "prefix": "tc.mybranch"
    },
    "fieldDefaults": {
      "enabled": true,
      "defaultValues": {
        "MyCompany.MyCustomField": "Default 1",
        "MyCompany.OtherCustomField": "Default 2"
      }
    },
    "ignoreTestCaseSteps": {
      "enabled": true,
      "ignoredPrefixes": [ "COMMENT" ]
    },
    "customFieldUpdates": {
      "enabled": true,
      "updates": {
        "System.Description": "Syncronized from feature {feature-name}{br}{
      }
    }
  }
  ...
}
```

Settings

- `branchTag` -- Supports synchronization of scenarios on a feature branch.
  - `branchTag/enabled` -- Enables the customization. (Default: `false`)
  - `branchTag/prefix` -- The tag prefix to be used for linking scenarios that are updated on a branch. E.g. the prefix `tc.mybranch` will generate tags, like `@tc.mybranch:1234` .
- `fieldDefaults` -- Enables setting default values to test case fields. Useful for custom Azure DevOps process templates.
  - `fieldDefaults/enabled` -- Enables the customization. (Default: `false`)
  - `fieldDefaults/defaultValues` -- A list of key-value pair, where the key is the canonical name of the field to be updated (e.g. `System.Description`) and the value is the default value to be used when the test case is created.
- `ignoreTestCaseSteps` -- Can ignore (leave unchanged) test case steps with a specific prefix.
  - `ignoreTestCaseSteps/enabled` -- Enables the customization. (Default: `false`)
  - `ignoreTestCaseSteps/ignoredPrefixes` -- An array of prefixes. The test case steps that start with any of the listed prefixes (case-insensitive) will be ignored by the synchronization.
- `customFieldUpdates` -- Enables updating test case fields that are normally not changed by SpecSync.
  - `customFieldUpdates/enabled` -- Enables the customization. (Default: `false`)
  - `customFieldUpdates/updates` -- A list of key-value pair, where the key is the canonical name of the field to be updated (e.g. `System.Description`) and the value is the template to be used to update the field. The template can contain the following placeholders
    - `{feature-name}` -- the name of the feature (specified in the feature file header)
    - `{feature-description}` -- the description of the feature (the free-text block specified after the feature file header)
    - `{scenario-name}` -- the name of the scenario or scenario outline
    - `{scenario-description}` -- the description of the scenario or scenario outline
    - `{feature-file-name}` -- the file name of the feature file (without folder)
    - `{feature-file-folder}` -- the folder of the feature file, relative to the project root
    - `{feature-file-path}` -- the path (folder and file name) of the feature file, relative to the project root

[Back to the Configuration guide]