

# Testing using Cucumber in Java

- [Overview](#)
- [Usage scenarios](#)
- [Example](#)
  - [Using Jira and Xray as master](#)
    - [Step-by-step](#)
  - [Using Git or other VCS as master](#)
    - [Step-by-step](#)
- [FAQ and Recommendations](#)
- [References](#)

## Overview

In this tutorial, we will create some tests in Cucumber using Java.

Cucumber is mainly a collaboration framework used in BDD context in order to improve shared understanding within the team, usually during "3 Amigos" sessions. That's its main fit.

However, some teams use it in other contexts (e.g. after software has being built) for implementing automated tests and take advantage of Gherkin syntax to have visibility/abstraction of the underlying automation code and have reusable automation code.

(Test) Scenarios derived from Cucumber are executable specifications; their statements will have a corresponding code implementation. These test scenarios are feature and more business oriented; they're not unit/integration tests.

Your specification is made using Gherkin (i.e. Given, When, That) statements in Scenario(s) or Scenario Outline(s), eventually complemented with a Background. Implementation of each Gherkin statement (i.e. "step") is done in code; the Cucumber framework finds the code based on regular or cucumber expressions.

### Source-code for this tutorial

Code is available in [GitHub](#); the repo contains some auxiliary scripts.

## Usage scenarios

Cucumber is used in diverse scenarios. Next you may find some usage patterns, even though Cucumber usage is mostly recommended only if you are adopting BDD.

1. Teams adopting BDD, start by defining a user story and clarify it using Cucumber Scenario(s); usually, Cucumber Scenario(s)/Scenario Outline(s) are specified directly in Jira, using Xray
2. Teams adopting BDD but that favour a more Git based approach (e.g. GitOps). In this case, stories would be defined in Jira but Cucumber .feature files would be specified using some IDE and would be stored in Git, for example
3. Teams not adopting BDD but still using Cucumber, more as an automation framework. Sometimes focused on regression testing; sometimes, for non-regression testing. In this case, cucumber would be used...
  - a. With a user story or some sort of "requirement" described in Jira
  - b. Without any story/"requirement" described in Jira

You may be adopting, or aiming to, one of the previous patterns.

Before moving into the actual implementation, we need to decide which workflow we'll use: do we want to use Xray/Jira as the master for writing the declarative specification (i.e. the Gherkin based Scenarios), or do we want to manage those outside using some editor and store them in Git, for example?



### Learn more

Please see [Testing in BDD with Gherkin based frameworks \(e.g. Cucumber\)](#) for an overview of the possible workflows.

The place that you'll use to edit the Cucumber Scenarios will affect your workflow. There are teams that prefer to edit Cucumber Scenarios in Jira using Xray, while there others that prefer to edit them by writing the .feature files by hand using some IDE.

## Example

For the purpose of this tutorial, we'll use a simple, dummy Calculator implemented in a Java class as our target for testing.

### Try it yourself!

The code on this tutorial is available in the [cucumber-java-calc](#) GitHub repository.

You can fork it and try it for yourself.

#### src/main/java/com/xray/tutorials/Calculator.java

```
package com.xray.tutorials;

public class Calculator
{
    // Square function
    public static int Square(int num)
    {
        return num*num;
    }
    // Add two integers and returns the sum
    public static int Add(int num1, int num2 )
    {
        return num1 + num2;
    }
    // Add two integers and returns the sum
    public static double Add(double num1, double num2 )
    {
        return num1 + num2;
    }
    // Multiply two integers and returns the result... this code is buggy on purpose
    public static int Multiply(int num1, int num2 )
    {
        if (num1==0) {
            return num2;
        } else if (num2==0) {
            return num1;
        } else {
            return num1 * num2;
        }
    }

    public static int Divide(int num1, int num2 )
    {
        return num1 / num2;
    }

    // Subtracts small number from big number
    public static int Subtract(int num1, int num2 )
    {
        if ( num1 > num2 )
        {
            return num1 - num2;
        }
        return num2 - num1;
    }
}
```

**This tutorial, has the following requirements:**

- Java
- Add the dependency of cucumber-jvm (i.e. [cucumber-java](#)) to your maven "pom.xml" file

In case you need to interact with Xray REST API at low-level using scripts (e.g. Bash/shell scripts), this tutorial uses an auxiliary file with the credentials (more info in [Global Settings: API Keys](#)).

#### Example of cloud\_auth.json used in this tutorial

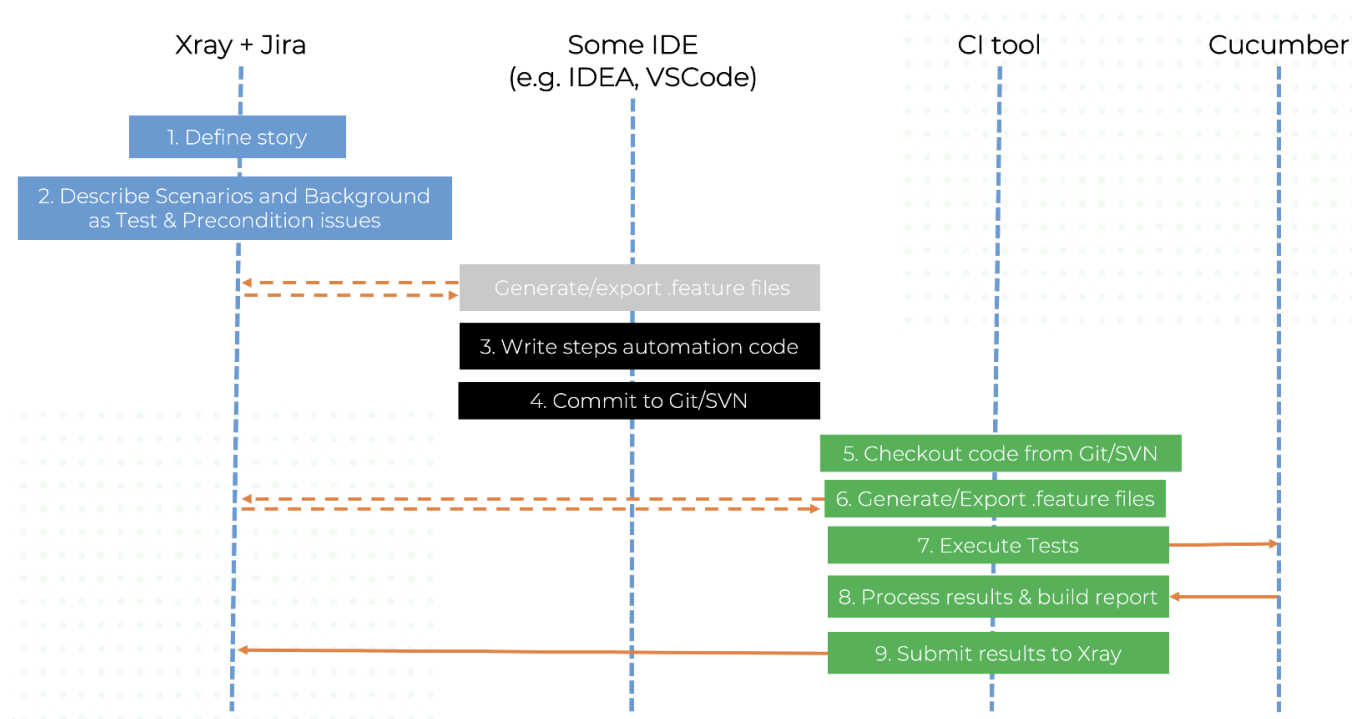
```
{ "client_id": "215FFD69FE4644728C72180000000000", "client_secret":  
  "1c00f8f22f56a8684d7c18cd6147ce2787d95e4da9f3bfb0af8f020000000000" }
```

## Using Jira and Xray as master

This section assumes using Xray as master, i.e. the place that you'll be using to edit the specifications (e.g. the scenarios that are part of .feature files).

The overall flow would be something like this, assuming Git as the source code versioning system:

1. define the story (skip if you already have it)
2. create Scenario/Scenario Outline as a Test in Jira; usually, it would be linked to an existing "requirement"/Story (i.e. created from the respective issue screen)
3. implement the code related to Gherkin statements/steps and store it in Git, for example. To start, and during development, you may need to generate/export the .feature file to your local environment
4. commit previous code to Git
5. checkout the code from Git
6. generate .feature files based on the specification made in Jira
7. run the tests in the CI
8. obtain the report in Cucumber JSON format
9. import the results back to Jira



Note that steps (5-9) performed by the CI tool are all automated, obviously.

To generate .feature file(s) based on Scenarios defined in Jira (i.e. Cucumber Tests and Preconditions), we can do it directly from Jira, by the REST API or using a CI tool; we'll see that ahead in more detail.

## Step-by-step

All starts with a user story or some sort of "requirement" that you wish to validate. This is materialized as a Jira issue and identified by the corresponding issue key (e.g. CALC-640).

## As a user, I can calculate the sum of 2 numbers

 Attach  Create subtask  Link issue  Test Coverage 

### Description

As a user, I can calculate the sum of 2 numbers

### Test Coverage

...

Create new Sub Test Execution

Create new Test

No Tests are associated with this issue.



UNCOVERED

We can promptly check that it is "UNCOVERED" (i.e. that it has no tests covering it, no matter their type/approach).

If you have this "requirement" as a Jira issue, then you can just use the "Create Test" on that issue to create the Scenario/Scenario Outline and have it automatically linked back to the Story/"requirement".

Otherwise, you can create the Test using the standard (issue) Create action from Jira's top menu.

We need to create the Test issue first and fill out the Gherkin statements later on in the Test issue screen.

Create issue

Import issuesConfigure fields

Project

Calculator (CALC)

Issue Type

Test

Some issue types are unavailable due to incompatible field configuration and/or workflow associations.

Summary

simple integer addition

Components

None

Attachment

Drop files to attach, or browse.

Description

Style B I U A A Link Image List Table Embed More

simple integer addition

Linked Issues

tests

Issue

Begin typing to search for issues to link. If you leave it blank, no link will be made.

Fix versions

Create another

Create

Cancel

## simple integer addition



### Description

simple integer addition

### Linked issues



tests

CALC-640 As a user, I can calculate the sum of 2 numbers **TO DO**

### Test Details



Manual

Generic

Cucumber

Exploratory



Test Repository

There are no steps defined.

Create Step

Open Dialog

Import

## simple integer addition



### Description

simple integer addition

### Linked issues



tests

CALC-640 As a user, I can calculate the sum of 2 numbers **TO DO**

### Test Details



Cucumber

Test Repository

Scenario

- 1 Given I have entered 1 into the calculator
- 2 And I have entered 2 into the calculator
- 3 When I press add
- 4 Then the result should be 3 on the screen

After the Test is created, and since we have done it from the user story screen, it will impact the coverage of related "requirement"/story.

The coverage and the test results can be tracked in the "requirement" side (e.g. user story). In this case, you may see that coverage changed from being UNCOVERED to NOTRUN (i.e. covered and with at least one test not run).

## As a user, I can calculate the sum of 2 numbers



### Description

As a user, I can calculate the sum of 2 numbers

### Linked issues

is tested by

CALC-642 simple integer addition ↑ TO DO

### Test Coverage

Calculate the Test Coverage for the following scopes.

Create new Sub Test Execution

Create new Test

Latest

Version

Test Plan

Test Environment

All Environments

NOTRUN

☒ Final statuses have precedence over non-final.

Status	Key	Summary	Test Status
↑ .. TO DO	CALC-642	simple integer addition	→ TO DO

Prev 1 Next

Additional tests could be created, eventually linked to the same Story or linked to another one (e.g. multiplication).

The related statement's code is managed outside of Jira and stored in Git, for example.

The tests related code is stored under `src/test` directory, which itself contains several other directories. In this case, they're organized as follows:

- `java/calculator`: step implementation files and test runner class.
  - The steps "glue-code" is defined in the `StepDefinitions` class.

**src/test/java/calculator/StepDefinitions.java**

```
package calculator;

import io.cucumber.java.en.Given;
import io.cucumber.java.en.Then;
import io.cucumber.java.en.When;
import com.xray.tutorials.Calculator;

import static org.junit.Assert.*;

public class StepDefinitions {
    private Integer int1;
    private Integer int2;
    private Integer result;

    @Given("I have entered {int} into the calculator")
    public void i_have_entered_into_the_calculator(Integer int1) {
        this.int2 = this.int1;
        this.int1 = int1;
    }

    @When("I press add")
    public void i_press_add() {
        this.result = Calculator.Add(this.int1, this.int2);
    }

    @When("I press multiply")
    public void i_press_multiply() {
        this.result = Calculator.Multiply(this.int1, this.int2);
    }

    @Then("the result should be {int} on the screen")
    public void the_result_should_be_on_the_screen(Integer value) {
        assertEquals(value, this.result);
    }
}
```

- the test runner is defined in the RunCucumberTest class. Cucumber options can be overridden from the command line, whenever executing Maven.

**src/test/java/calculator/RunCucumberTest.java**

```
package calculator;

import io.cucumber.junit.Cucumber;
import io.cucumber.junit.CucumberOptions;
import org.junit.runner.RunWith;

@RunWith(Cucumber.class)
@CucumberOptions(plugin = {"pretty"})
public class RunCucumberTest {

}
```

You can then export the specification of the test to a Cucumber .feature file via the REST API, or the **Export to Cucumber** UI action from within the Test /Test Execution issue or even based on an existing saved filter. As source, you can identify Test, Test Set, Test Execution, Test Plan or "requirement" issues. A plugin for your CI tool of choice can be used to ease this task.

So, you can either:

- use one of the available CI/CD plugins (e.g. see details of [Integration with Jenkins](#))

**Xray: Cucumber Features Export Task**

Jira Instance

xray cloud

Issues:

CALC-640;CALC-641

Filter:

File Path:

features

[Click here for more details](#)

- use the REST API directly (more info [here](#))

◦ **example of a shell script to export/generate .features from Xray**

```
#!/bin/bash

token=$(curl -H "Content-Type: application/json" -X POST --data @"cloud_auth.json" https://xray.cloud.getxray.app/api/v2/authenticate| tr -d '\n')
curl -H "Content-Type: application/json" -X GET -H "Authorization: Bearer $token" "https://xray.cloud.getxray.app/api/v2/export/cucumber?keys=CALC-640;CALC-641" -o features.zip

rm -rf features/*.feature
unzip -o features.zip -d features
```

- ... or even use the UI (e.g. from a Test issue)

The screenshot shows a Jira issue titled 'CALC-642' with the description 'simple integer addition'. On the right side, there is a menu with various actions. The option 'Xray - Export to Cucumber' is highlighted with a red box. Other visible options include 'Log work', 'Add flag', 'Xray - Document Generator', 'Convert to Subtask', 'Move', 'Clone', 'Print', 'Export XML', 'Export Word', and 'NEW JIRA ISSUE VIEW'.

We will export the features to a new directory named `features/` on the root folder of your Java project (we'll need to tell Maven to use this folder).

After being exported, the created .feature(s) will contain references to the Test issue key, eventually prefixed (e.g. "TEST\_") depending on an Xray global setting, and the covered "requirement" issue key, if that's the case. The naming of these files is detailed in [Generate Cucumber Features](#).



## features/2\_CALC-640.feature

```
@REQ_CALC-640
Feature: As a user, I can calculate the sum of 2 numbers
  #As a user, I can calculate the sum of 2 numbers

  #simple integer addition
  @TEST_CALC-642
  Scenario: simple integer addition
    Given I have entered 1 into the calculator
    And I have entered 2 into the calculator
    When I press add
    Then the result should be 3 on the screen

  #negative integer addition
  @TEST_CALC-643
  Scenario: negative integer addition
    Given I have entered -1 into the calculator
    And I have entered 2 into the calculator
    When I press add
    Then the result should be 1 on the screen

  #sum of two positive numbers
  @TEST_CALC-644
  Scenario Outline: sum of two positive numbers
    Given I have entered <input_1> into the calculator
    And I have entered <input_2> into the calculator
    When I press <button>
    Then the result should be <output> on the screen

    Examples:
      | input_1 | input_2 | button | output |
      | 20      | 30      | add    | 50      |
      | 2        | 5        | add    | 7        |
      | 0        | 40       | add    | 40       |
      | 4        | 50       | add    | 54       |
      | 5        | 50       | add    | 55       |
```

## features/1\_CALC-641.feature

```
@REQ_CALC-641
Feature: As a user, I can multiply two numbers
  #As a user, I can multiply two numbers

  #simple integer multiplication
  @TEST_CALC-645
  Scenario: simple integer multiplication
    Given I have entered 3 into the calculator
    And I have entered 0 into the calculator
    When I press multiply
    Then the result should be 0 on the screen
```

To run the tests and produce a Cucumber JSON report, we can run Maven and specify that we want a report in Cucumber JSON format and that it should process .features from the features/ directory.

```
mvn compile test -Dcucumber.plugin="json:report.json" -Dcucumber.features="features/"
```



#### Please note

As the report format in Cucumber JSON is being deprecated in favour of [Cucumber Messages](#), a protocol buffer based implementation, the previous command needs to be adapted slightly.

The report starts by being generated in Cucumber Messages, using "-f message" argument, and then converted to the legacy Cucumber JSON report using the tool [cucumber-json-formatter](#).

```
mvn compile test -Dcucumber.plugin="json:report.ndjson" -Dcucumber.features="features/"
cat report.ndjson | cucumber-json-formatter --format ndjson > report.json
```

This will produce one Cucumber JSON report with all results.

After running the tests, results can be imported to Xray via the REST API, or the **Import Execution Results** action within an existing Test Execution, or by using one of the available CI/CD plugins (e.g. see an example of [Integration with Jenkins](#)).

#### example of a Bash script to import results using the standard Cucumber endpoint

```
#!/bin/bash

BASE_URL=https://xray.cloud.getxray.app
token=$(curl -H "Content-Type: application/json" -X POST --data @"cloud_auth.json" "$BASE_URL/api/v2/authenticate" | tr -d '\n')
curl -H "Content-Type: application/json" -X POST -H "Authorization: Bearer $token" --data @"merged-test-results.json" "$BASE_URL/api/v2/import/execution/cucumber"
```

## Post-build Actions



### Xray: Results Import Task

Jira Instance

xray cloud

Format

Cucumber JSON

Parameters

Execution Report File (file path with file name)

report.json

Import in parallel



Import all results files in parallel, using

[Click here for more details](#)



### Which Cucumber endpoint to use?

To import results, you can use two different endpoints/"formats" (endpoints described in [Import Execution Results - REST](#)):

1. the "standard cucumber" endpoint
2. the "multipart cucumber" endpoint

The standard cucumber endpoint (i.e. `/import/execution/cucumber`) is simpler but more restrictive: you cannot specify values for custom fields on the Test Execution that will be created. This endpoint creates new Test Execution issues unless the Feature contains a tag having an issue key of an existing Test Execution.

The multipart cucumber endpoint will allow you to customize fields (e.g. Fix Version, Test Plan), if you wish to do so, on the Test Execution that will be created. Note that this endpoint always creates new Test Executions (as of Xray v4.2).

In sum, if you want to customize the Fix Version, Test Plan and/or Test Environment of the Test Execution issue that will be created, you'll have to use the "multipart cucumber" endpoint.

A new Test Execution will be created (unless you originally exported the Scenarios/Scenario Outlines from a Test Execution).

Projects / Calculator / CALC-647

## Execution results [1605028733128]

[Attach](#) [Create subtask](#) [Link issue](#) [Tests](#) [...](#)

### Description

Add a description...

### Tests

Create Test

+ Add

### Overall Execution Status

TOTAL TESTS: 4

3 PASSED 1 FAILED





Rank	Key	Summary	Test Type	Status	Actions
<input type="checkbox"/> 1	<a href="#">CALC-642</a>	simple integer addition	Cucumber	<span>PASSED</span>	<a href="#">View</a> <a href="#">...</a>
<input type="checkbox"/> 2	<a href="#">CALC-643</a>	negative integer addition	Cucumber	<span>PASSED</span>	<a href="#">View</a> <a href="#">...</a>
<input type="checkbox"/> 3	<a href="#">CALC-644</a>	sum of two positive numbers	Cucumber	<span>PASSED</span>	<a href="#">View</a> <a href="#">...</a>
<input type="checkbox"/> 4	<a href="#">CALC-645</a>	simple integer multiplication	Cucumber	<span>FAILED</span>	<a href="#">View</a> <a href="#">...</a>

Prev 1 Next

Total 4 issues

One of the tests fails (on purpose).

The execution screen details of the Test Run will provide overall status information and Gherkin statement-level results, therefore we can use it to analyze the failing test.

	Rank ▾	Key ▾	Summary ▾	Test Type ▾	Status ▾		Actions	
<input type="checkbox"/>	1	<a href="#">CALC-642</a>	simple integer addition	Cucumber	<div><div></div></div> PASSED		...	
<input type="checkbox"/>	2	<a href="#">CALC-643</a>	negative integer addition	Cucumber	<div><div></div></div> PASSED		...	
<input type="checkbox"/>	3	<a href="#">CALC-644</a>	sum of two positive numbers	Cucumber	<div><div></div></div> PASSED		...	
<input type="checkbox"/>	4	<a href="#">CALC-645</a>	simple integer multiplication	Cucumber	<div><div></div></div> FAILED		...	
Prev	1	Next						Total 4 issues

Results, including for each example on Scenario Outline, can be expanded to see all Gherkin statements.

simple integer multiplication

Execution Status

FAILED

Started On: 10/Nov/2020 05:18 PMFinished On: 10/Nov/2020 05:18 PM

Assignee: Sérgio Freire

Versions: -Revision: -

Executed By: Sérgio Freire

Test Environments: -

Comment

Preview comment

Execution Defects (0)

+

Execution Evidence (0)

Add Evidence

Execution Details

Test Description

simple integer multiplication

Test Issue Links (1)

testsCALC-641As a user, I can multiply two numbers↑TO DO

Custom Fields

There are no Test Run Custom Fields defined.

Test Details

Test Type: Cucumber

Scenario Type: Scenario

Scenario:

1

Given

I have entered 3 into the calculator

2

And

I have entered 0 into the calculator

3

When

I press multiply

4

Then

the result should be 0 on the screen

Results

Context

Duration

Status

4ms

FAILED

Execution Details

Test Description

simple integer multiplication

Test Issue Links (1)

testsCALC-641As a user, I can multiply two numbers↑TO DO

Custom Fields

There are no Test Run Custom Fields defined.

Test Details

Test Type: Cucumber

Scenario Type: Scenario

Scenario:

1

Given

I have entered 3 into the calculator

2

And

I have entered 0 into the calculator

3

When

I press multiply

4

Then

the result should be 0 on the screen

Results

Context

Duration

Status

4ms

FAILED

Steps

Duration

Status

1 milliseccPASSED

0 milliseccPASSED

0 milliseccPASSED

2 milliseccFAILED

java.lang.AssertionError: expected:<0> but was:<3>  
at org.junit.Assert.fail(Assert.java:89)  
at org.junit.Assert.failNotEquals(Assert.java:835)  
at org.junit.Assert.assertEquals(Assert.java:120)  
at org.junit.Assert.assertEquals(Assert.java:146)  
at calculator.StepDefinitions.the\_result\_should\_be\_on\_the\_screen(StepDefinitions.java:36)  
at calculator.StepDefinitions.the\_result\_should\_be\_0\_on\_the\_screen(file:///Users/smsf/exps/cucumber-java-calc/features/1\_CALC-641.feature:11)

Note: in this case, the bug was added on purpose on the Calculator class.


#### buggy Multiply() method in Calculator.java

```
public static int Multiply(int num1, int num2 )
{
    if (num1==0) {
        return num2;
    } else if (num2==0) {
        return num1;
    } else {
        return num1 * num2;
    }
}
```



#### Screenshots and other attachments

If available, it is possible to see also attached screenshot(s). For this, you'll need to use Cucumber's API and do it in a After hook, for example (using `scenario.embed()`).

The icon , if shown, represents the evidences ("embeddings") for each **Hook**, **Background** and **Steps**.

Results are reflected on the covered items (e.g. Story issues) and can be seen in their issue screen.

Coverage now shows that the addition related user story (e.g. CALC-640) is OK based on the latest testing results; on the other hand, the multiplication related user story (CALC-641) is NOK since it has one test currently failing.

As a user, I can calculate the sum of 2 numbers

Attach Create subtask Link issue Test Coverage

Description

As a user, I can calculate the sum of 2 numbers

Linked issues

is tested by

CALC-642	simple integer addition	↑	TO DO
CALC-644	sum of two positive numbers	↑	TO DO
CALC-643	negative integer addition	↑	TO DO

Test Coverage

Calculate the Test Coverage for the following scopes.

Create new Sub Test Execution Create new Test

Latest Version Test Plan

Test Environment

All Environments

OK

Final statuses have precedence over non-final.

Status	Key	Summary	Test Status
↑ .. TO DO	CALC-642	simple integer addition	PASSED
↑ .. TO DO	CALC-643	negative integer addition	PASSED
↑ .. TO DO	CALC-644	sum of two positive numbers	PASSED

Prev 1 Next

As a user, I can multiply two numbers

Attach Create subtask Link issue Test Coverage

Description

As a user, I can multiply two numbers

Linked issues

is tested by

CALC-645	simple integer multiplication	↑	TO DO
----------	-------------------------------	---	-------

Test Coverage

Calculate the Test Coverage for the following scopes.

Create new Sub Test Execution Create new Test

Latest Version Test Plan

Test Environment

All Environments

NOK

Final statuses have precedence over non-final.

Status	Key	Summary	Test Status
↑ .. TO DO	CALC-645	simple integer multiplication	→ FAILED

Prev 1 Next

If we fix the code on the Calculator class, run the tests and import results, coverage for the multiplication related user story will be shown as OK.

### fix of Multiply() method in Calculator.java

```
public static int Multiply(int num1, int num2 )
{
    return num1 * num2;
}
```

Projects /  Calculator /  CALC-641

## As a user, I can multiply two numbers

 Attach  Create subtask  Link issue  Test Coverage 

### Description

As a user, I can multiply two numbers

### Linked issues

is tested by

 CALC-645 simple integer multiplication  **TO DO**

### Test Coverage

Calculate the Test Coverage for the following scopes.

Create new Sub Test Execution

Create new Test



Latest Version Test Plan

Test Environment

All Environments

OK

☒ Final statuses have precedence over non-final.

Status	Key	Summary	Test Status
 <b>TO DO</b>	<b>CALC-645</b>	simple integer multiplication	 <b>PASSED</b>

Prev 1 Next

## Using Git or other VCS as master

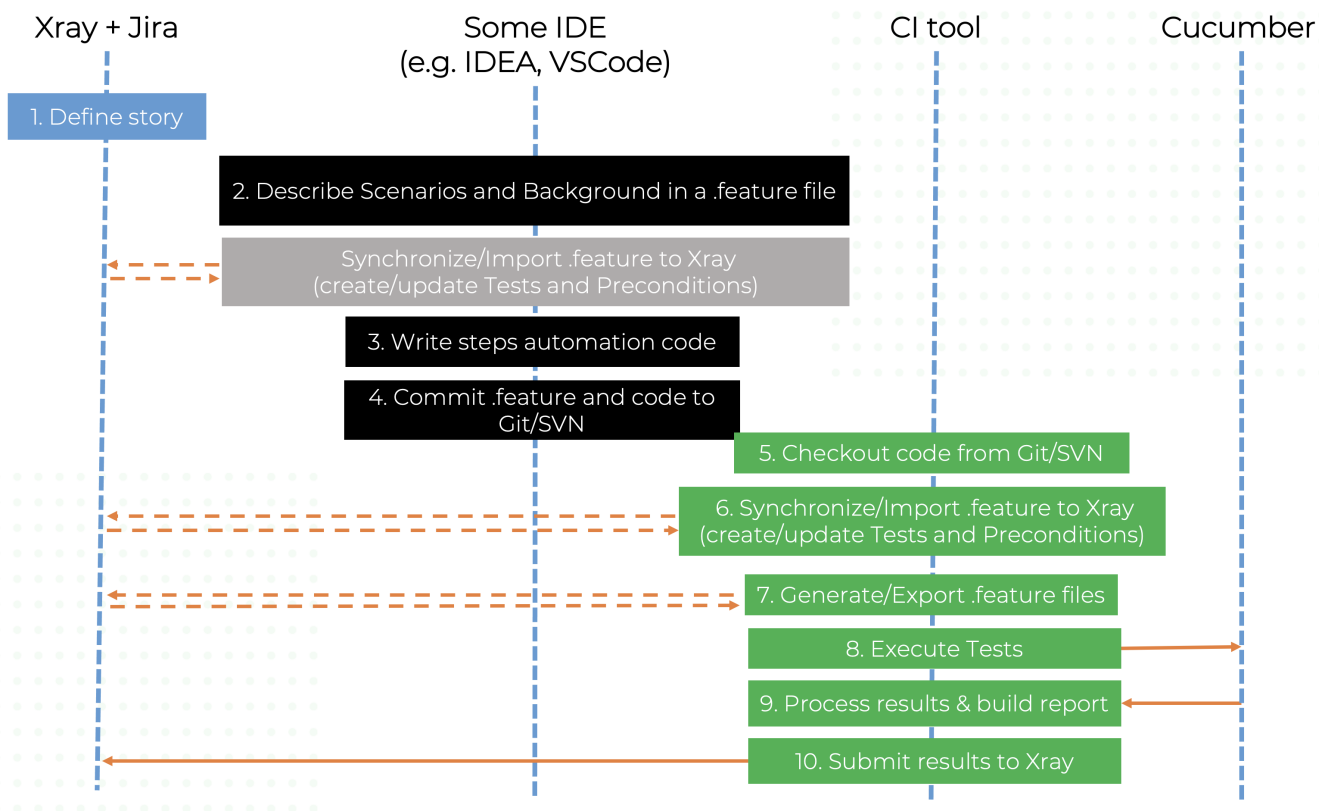
You can edit your .feature files using your IDE outside of Jira (eventually storing them in your VCS using Git, for example) alongside with remaining test code.

In any case, you'll need to synchronize your .feature files to Jira so that you can have visibility of them and report results against them.

The overall flow would be something like this:

1. look at the existing "requirement"/Story issue keys to guide your testing; keep their issue keys
2. specify Cucumber/Gherkin .feature files in your IDE supporting Cucumber/Gherkin and store it in Git, for example. Meanwhile, you may decide to import/synchronize them Xray to provision or update corresponding Test and/or Precondition entities
3. implement the code related to Gherkin statements/steps and store it in Git, for example.
4. commit code and .feature file(s) to Git
5. checkout the code from Git
6. import/synchronize the .feature files to Xray to provision or update corresponding Test and/or Precondition entities
7. export/generate .feature files from Jira, so that they contain references to Tests and requirements in Jira
8. run the tests in the CI
9. obtain the report in Cucumber JSON format
10. import the results back to Jira





Note that steps (5-10) performed by the CI tool are all automated, obviously.

To import .features to Jira we can either use the REST API or a CI tool. To export tagged .features from Jira, we can do it directly from Jira, by the REST API or using a CI tool; we'll see that ahead in more detail.

## Step-by-step

All starts with a user story or some sort of "requirement" that you wish to validate. This is materialized as a Jira issue and identified by the corresponding issue key (e.g. CALC-640).

Projects / Calculator / CALC-640

### As a user, I can calculate the sum of 2 numbers

Attach Create subtask Link issue Test Coverage ...

#### Description

As a user, I can calculate the sum of 2 numbers

#### Test Coverage

Create new Sub Test Execution

Create new Test

No Tests are associated with this issue.



UNCOVERED

We can promptly check that it is "UNCOVERED" (i.e. that it has no tests covering it, no matter their type/approach).

Having those to guide testing, we could then describe and implement the Cucumber test scenarios using our favourite IDE.

The screenshot shows an IDE with a file explorer on the left and a code editor on the right. The file explorer shows a project structure for 'cucumber-java-calc' with directories like 'src', 'test', and 'resources'. The code editor displays the content of 'addition.feature' in the 'src > test > resources > calculator >' directory. The file contains three Cucumber scenarios: 'simple integer addition', 'negative integer addition', and 'sum of two positive numbers'. The third scenario includes an 'Examples:' table with test data.

```
2 @REQ_CALC-640 You, 16 hours ago • fix test step assert and c
3 Feature: As a user, I can add two numbers
4
5 Scenario: simple integer addition
6   Given I have entered 1 into the calculator
7   And I have entered 2 into the calculator
8   When I press add
9   Then the result should be 3 on the screen
10
11 Scenario: negative integer addition
12   Given I have entered -1 into the calculator
13   And I have entered 2 into the calculator
14   When I press add
15   Then the result should be 1 on the screen
16
17 Scenario Outline: sum of two positive numbers
18   Given I have entered <input_1> into the calculator
19   And I have entered <input_2> into the calculator
20   When I press <button>
21   Then the result should be <output> on the screen
22
23   Examples:
24   | input_1 | input_2 | button | output |
25   | 20      | 30      | add    | 50     |
26   | 2        | 5        | add    | 7       |
27   | 0        | 40       | add    | 40      |
28   | 4        | 50       | add    | 54      |
29   | 5        | 50       | add    | 55      |
```

The related statement's code is managed outside of Jira and stored in Git, for example.

The tests related code is stored under `src/test` directory, which itself contains several other directories. In this case, they're organized as follows:

- `java/calculator`: step implementation files and test runner class.
  - The steps "glue-code" is defined in the `StepDefinitions` class.

**src/test/java/calculator/StepDefinitions.java**

```
package calculator;

import io.cucumber.java.en.Given;
import io.cucumber.java.en.Then;
import io.cucumber.java.en.When;
import com.xray.tutorials.Calculator;

import static org.junit.Assert.*;

public class StepDefinitions {
    private Integer int1;
    private Integer int2;
    private Integer result;

    @Given("I have entered {int} into the calculator")
    public void i_have_entered_into_the_calculator(Integer int1) {
        this.int2 = this.int1;
        this.int1 = int1;
    }

    @When("I press add")
    public void i_press_add() {
        this.result = Calculator.Add(this.int1, this.int2);
    }

    @When("I press multiply")
    public void i_press_multiply() {
        this.result = Calculator.Multiply(this.int1, this.int2);
    }

    @Then("the result should be {int} on the screen")
    public void the_result_should_be_on_the_screen(Integer value) {
        assertEquals(value, this.result);
    }
}
```

- the test runner is defined in the RunCucumberTest class. Cucumber options can be overridden from the command line, whenever executing Maven.

**src/test/java/calculator/RunCucumberTest.java**

```
package calculator;

import io.cucumber.junit.Cucumber;
import io.cucumber.junit.CucumberOptions;
import org.junit.runner.RunWith;

@RunWith(Cucumber.class)
@CucumberOptions(plugin = {"pretty"})
public class RunCucumberTest {
}
}
```

Before running the tests in the CI environment, you need to import your .feature files to Xray/Jira; you can invoke the REST API directly or use one of the available plugins/tutorials for CI tools.

### example of a shell script to import/synchronize .features to Jira and Xray

```
#!/bin/bash
BASE_URL=https://xray.cloud.getxray.app
PROJECT=CALC

rm -f features.zip
zip -r features.zip src/test/resources/calculator/ -i \*.feature

token=$(curl -H "Content-Type: application/json" -X POST --data @"cloud_auth.json" "$BASE_URL/api/v2/authenticate" | tr -d '\n')
curl -H "Content-Type: multipart/form-data" -H "Authorization: Bearer $token" -F "file=@features.zip" "$BASE_URL/api/v2/import/feature?projectKey=$PROJECT"
```

## Build



### Xray: Cucumber Features Import Task

Jira Instance	<input type="text" value="xray cloud"/>
Project Key	<input type="text" value="CALC"/>
Cucumber feature files directory	<input type="text" value="src/test/resources/calculator"/>
Test Info file	<input type="text"/>
Preconditions file	<input type="text"/>
Modified in the last hours	<input type="text" value="10"/>



### Please note

Each Scenario of each .feature will be created as a Test issue that contains unique identifiers, so that if you import once again then Xray can update the existent Test and don't create any duplicated tests. See [Importing Cucumber Tests - REST](#) for details on how it works.

Projects / Calculator / CALC-645

## simple integer multiplication



### Description

simple integer multiplication

### Linked issues



tests

CALC-641 As a user, I can multiply two numbers

**TO DO**

### Test Details



Cucumber



Test Repository

#### Scenario

- 1 **Given** I have entered 3 into the calculator
- 2 **And** I have entered 0 into the calculator
- 3 **When** I press multiply
- 4 **Then** the result should be 0 on the screen

You can then export the specification of the test to a Cucumber .feature file via the REST API, or the **Export to Cucumber** UI action from within the Test /Test Execution issue or even based on an existing saved filter. As source, you can identify Test, Test Set, Test Execution, Test Plan or "requirement" issues. A plugin for your CI tool of choice can be used to ease this task.

So, you can either:

- use one of the available CI/CD plugins (e.g. see details of [Integration with Jenkins](#))

**Xray: Cucumber Features Export Task**

Jira Instance

xray cloud

Issues:

CALC-640;CALC-641

Filter:

File Path:

features

[Click here for more details](#)

- use the REST API directly (more info [here](#))

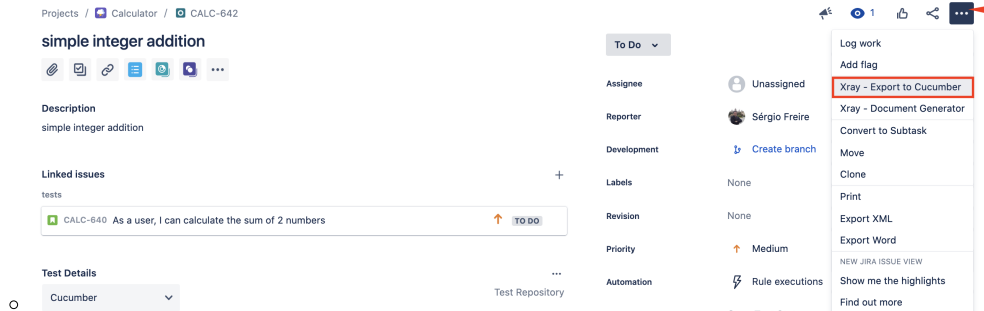
○ **example of a shell script to export/generate .features from Xray**

```
#!/bin/bash

token=$(curl -H "Content-Type: application/json" -X POST --data @"cloud_auth.json" https://xray.cloud.getxray.app/api/v2/authenticate | tr -d ' ')
curl -H "Content-Type: application/json" -X GET -H "Authorization: Bearer $token" "https://xray.cloud.getxray.app/api/v2/export/cucumber?keys=CALC-640;CALC-641" -o features.zip

rm -rf features/*.*feature
unzip -o features.zip -d features
```

- ... or even use the UI (e.g. from a Test issue)



We will export the features to a new directory named `features/` on the root folder of your Java project (we'll need to tell Maven to use this folder).

After being exported, the created `.feature(s)` will contain references to the Test issue key, eventually prefixed (e.g. "TEST\_") depending on an Xray global setting, and the covered "requirement" issue key, if that's the case. The naming of these files is detailed in [Generate Cucumber Features](#).

## features/2\_CALC-640.feature

```
@REQ_CALC-640
Feature: As a user, I can calculate the sum of 2 numbers
  #As a user, I can calculate the sum of 2 numbers

  #simple integer addition
  @TEST_CALC-642
  Scenario: simple integer addition
    Given I have entered 1 into the calculator
    And I have entered 2 into the calculator
    When I press add
    Then the result should be 3 on the screen

  #negative integer addition
  @TEST_CALC-643
  Scenario: negative integer addition
    Given I have entered -1 into the calculator
    And I have entered 2 into the calculator
    When I press add
    Then the result should be 1 on the screen

  #sum of two positive numbers
  @TEST_CALC-644
  Scenario Outline: sum of two positive numbers
    Given I have entered <input_1> into the calculator
    And I have entered <input_2> into the calculator
    When I press <button>
    Then the result should be <output> on the screen

    Examples:
      | input_1 | input_2 | button | output |
      | 20      | 30      | add    | 50      |
      | 2        | 5        | add    | 7        |
      | 0        | 40       | add    | 40       |
      | 4        | 50       | add    | 54       |
      | 5        | 50       | add    | 55       |
```

## features/1\_CALC-641.feature

```
@REQ_CALC-641
Feature: As a user, I can multiply two numbers
  #As a user, I can multiply two numbers

  #simple integer multiplication
  @TEST_CALC-645
  Scenario: simple integer multiplication
    Given I have entered 3 into the calculator
    And I have entered 0 into the calculator
    When I press multiply
    Then the result should be 0 on the screen
```

To run the tests and produce a Cucumber JSON report, we can run Maven and specify that we want a report in Cucumber JSON format and that it should process .features from the features/ directory.

```
mvn compile test -Dcucumber.plugin="json:report.json" -Dcucumber.features="features/"
```



#### Please note

As the report format in Cucumber JSON is being deprecated in favour of [Cucumber Messages](#), a protocol buffer based implementation, the previous command needs to be adapted slightly.

The report starts by being generated in Cucumber Messages, using "-f message" argument, and then converted to the legacy Cucumber JSON report using the tool [cucumber-json-formatter](#).

```
mvn compile test -Dcucumber.plugin="json:report.ndjson" -Dcucumber.features="features/"
cat report.ndjson | cucumber-json-formatter --format ndjson > report.json
```

This will produce one Cucumber JSON report with all results.

After running the tests, results can be imported to Xray via the REST API, or the **Import Execution Results** action within an existing Test Execution, or by using one of the available CI/CD plugins (e.g. see an example of [Integration with Jenkins](#)).

#### example of a Bash script to import results using the standard Cucumber endpoint

```
#!/bin/bash

BASE_URL=https://xray.cloud.getxray.app
token=$(curl -H "Content-Type: application/json" -X POST --data @"cloud_auth.json" "$BASE_URL/api/v2/authenticate" | tr -d '\n')
curl -H "Content-Type: application/json" -X POST -H "Authorization: Bearer $token" --data @"merged-test-results.json" "$BASE_URL/api/v2/import/execution/cucumber"
```

## Post-build Actions



### Xray: Results Import Task

Jira Instance

xray cloud

Format

Cucumber JSON

Parameters

Execution Report File (file path with file name)

report.json|

Import in parallel



Import all results files in parallel, using

[Click here for more details](#)





### Which Cucumber endpoint to use?

To import results, you can use two different endpoints/"formats" (endpoints described in [Import Execution Results - REST](#)):

1. the "standard cucumber" endpoint
2. the "multipart cucumber" endpoint

The standard cucumber endpoint (i.e. `/import/execution/cucumber`) is simpler but more restrictive: you cannot specify values for custom fields on the Test Execution that will be created. This endpoint creates new Test Execution issues unless the Feature contains a tag having an issue key of an existing Test Execution.

The multipart cucumber endpoint will allow you to customize fields (e.g. Fix Version, Test Plan), if you wish to do so, on the Test Execution that will be created. Note that this endpoint always creates new Test Executions (as of Xray v4.2).

In sum, if you want to customize the Fix Version, Test Plan and/or Test Environment of the Test Execution issue that will be created, you'll have to use the "multipart cucumber" endpoint.

A new Test Execution will be created (unless you originally exported the Scenarios/Scenario Outlines from a Test Execution).

Projects / Calculator / CALC-647

## Execution results [1605028733128]

Attach Create subtask Link issue Tests ...

### Description

Add a description...

### Tests

Create Test + Add

### Overall Execution Status

TOTAL TESTS: 4





3 PASSED 1 FAILED

Rank	Key	Summary	Test Type	Status	Actions
<input type="checkbox"/> 1	<a href="#">CALC-642</a>	simple integer addition	Cucumber	PASSED	...
<input type="checkbox"/> 2	<a href="#">CALC-643</a>	negative integer addition	Cucumber	PASSED	...
<input type="checkbox"/> 3	<a href="#">CALC-644</a>	sum of two positive numbers	Cucumber	PASSED	...
<input type="checkbox"/> 4	<a href="#">CALC-645</a>	simple integer multiplication	Cucumber	FAILED	...

Prev 1 Next Total 4 issues

One of the tests fails (on purpose).

The execution screen details of the Test Run will provide overall status information and Gherkin statement-level results, therefore we can use it to analyze the failing test.

	Rank ▾	Key ▾	Summary ▾	Test Type ▾	Status ▾		Actions	
<input type="checkbox"/>	1	<a href="#">CALC-642</a>	simple integer addition	Cucumber	<div><div></div></div> PASSED		...	
<input type="checkbox"/>	2	<a href="#">CALC-643</a>	negative integer addition	Cucumber	<div><div></div></div> PASSED		...	
<input type="checkbox"/>	3	<a href="#">CALC-644</a>	sum of two positive numbers	Cucumber	<div><div></div></div> PASSED		...	
<input type="checkbox"/>	4	<a href="#">CALC-645</a>	simple integer multiplication	Cucumber	<div><div></div></div> FAILED		...	
Prev	1	Next						Total 4 issues

Results, including for each example on Scenario Outline, can be expanded to see all Gherkin statements.

As a user, I can calculate the sum of 2 numbers

Attach

Create subtask

Link issue

Test Coverage

Description

As a user, I can calculate the sum of 2 numbers

Linked issues

is tested by

CALC-642	simple integer addition	↑	TO DO
CALC-644	sum of two positive numbers	↑	TO DO
CALC-643	negative integer addition	↑	TO DO

Test Coverage

Calculate the Test Coverage for the following scopes.

Create new Sub Test Execution

Create new Test

Latest

Version

Test Plan

Test Environment

All Environments

OK

☒ Final statuses have precedence over non-final.

Status	Key	Summary	Test Status
↑ .. TO DO	CALC-642	simple integer addition	PASSED
↑ .. TO DO	CALC-643	negative integer addition	PASSED
↑ .. TO DO	CALC-644	sum of two positive numbers	PASSED

Prev 1 Next

As a user, I can multiply two numbers

Attach

Create subtask

Link issue

Test Coverage

Description

As a user, I can multiply two numbers

Linked issues

is tested by

CALC-645	simple integer multiplication	↑	TO DO
----------	-------------------------------	---	-------

Test Coverage

Calculate the Test Coverage for the following scopes.

Create new Sub Test Execution

Create new Test

Latest

Version

Test Plan

Test Environment

All Environments

NOK

☒ Final statuses have precedence over non-final.

Status	Key	Summary	Test Status
↑ .. TO DO	CALC-645	simple integer multiplication	→ FAILED

Prev 1 Next

Note: in this case, the bug was added on purpose on the Calculator class.


#### buggy Multiply() method in Calculator.java

```
public static int Multiply(int num1, int num2 )
{
    if ((num1==1) || (num2==1)) {
        return 0;
    } else {
        return num1 * num2;
    }
}
```



#### Screenshots and other attachments

If available, it is possible to see also attached screenshot(s). For this, you'll need to use Cucumber's API and do it in a After hook, for example (using `scenario.embed()`).

The icon , if shown, represents the evidences ("embeddings") for each **Hook**, **Background** and **Steps**.

Results are reflected on the covered items (e.g. Story issues) and can be seen in their issue screen.

Coverage now shows that the addition related user story (e.g. CALC-640) is OK based on the latest testing results; on the other hand, the multiplication related user story (CALC-641) is NOK since it has one test currently failing.

As a user, I can calculate the sum of 2 numbers

Attach Create subtask Link issue Test Coverage

Description

As a user, I can calculate the sum of 2 numbers

Linked issues

is tested by

CALC-642	simple integer addition	↑ TO DO
CALC-644	sum of two positive numbers	↑ TO DO
CALC-643	negative integer addition	↑ TO DO

Test Coverage

Calculate the Test Coverage for the following scopes.

Latest Version Test Plan

Test Environment

All Environments

OK

Final statuses have precedence over non-final.

Status	Key	Summary	Test Status
↑ TO DO	CALC-642	simple integer addition	PASSED
↑ TO DO	CALC-643	negative integer addition	PASSED
↑ TO DO	CALC-644	sum of two positive numbers	PASSED

Prev 1 Next

As a user, I can multiply two numbers

Attach Create subtask Link issue Test Coverage

Description

As a user, I can multiply two numbers

Linked issues

is tested by

CALC-645	simple integer multiplication	↑ TO DO
----------	-------------------------------	---------

Test Coverage

Calculate the Test Coverage for the following scopes.

Latest Version Test Plan

Test Environment

All Environments

NOK

Final statuses have precedence over non-final.

Status	Key	Summary	Test Status
↑ TO DO	CALC-645	simple integer multiplication	→ FAILED

Prev 1 Next


If we fix the code on the Calculator class, run the tests and import results, coverage for the multiplication related user story will be shown as OK.

## fix of Multiply() method in Calculator.java

```
public static int Multiply(int num1, int num2 )
{
    return num1 * num2;
}
```

Projects /  Calculator /  CALC-641

## As a user, I can multiply two numbers


 Attach  Create subtask  Link issue  Test Coverage 

### Description

As a user, I can multiply two numbers

### Linked issues

is tested by

 CALC-645 simple integer multiplication  **TO DO**

### Test Coverage

Calculate the Test Coverage for the following scopes.

Create new Sub Test Execution

Create new Test


Latest Version Test Plan

Test Environment

All Environments

OK

☒ Final statuses have precedence over non-final.

Status	Key	Summary	Test Status
 <b>TO DO</b>	<b>CALC-645</b>	simple integer multiplication	 <b>PASSED</b>

Prev 1 Next

## FAQ and Recommendations

Please see [this page](#).

## References

- [Code used in this tutorial, along with some auxiliary scripts](#)
- [Sample project cucumber-java-skeleton](#)
- [Official Cucumber documentation](#)
- [Cucumber installation instructions for Java](#)
- [Cucumber API](#)
- [Cucumber expressions](#)
- [Testing in BDD with Gherkin based frameworks \(e.g. Cucumber\)](#)
- [Automated Tests \(Import/Export\)](#)
- [Exporting Cucumber Tests - REST](#)