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 sofware 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.

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); usualy, 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 youself.

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 retuns 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:

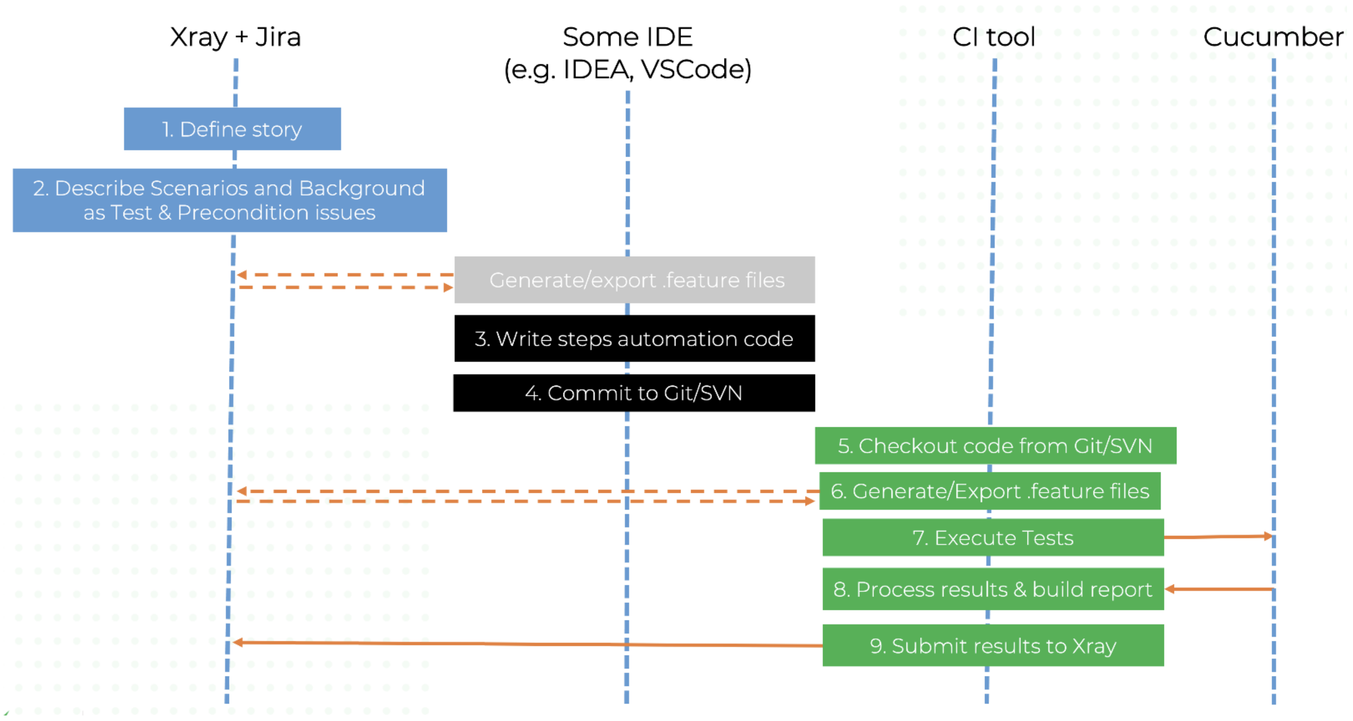
- Java
- Add the dependency of cucumber-jvm (i.e. cucumber-java) to your maven "pom.xml" file

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

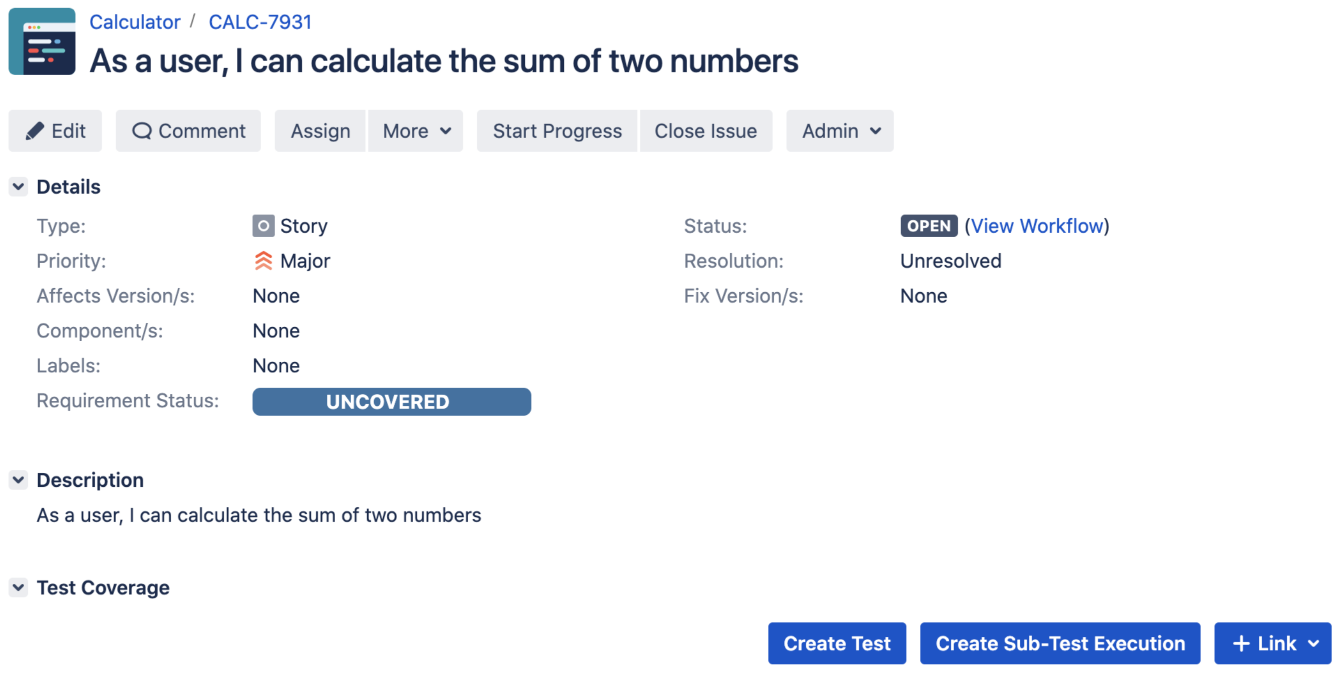

Xray + Jira
Some IDE (e.g. IDEA, VSCode)
CI tool
Cucumber
1. Define story
2. Describe Scenarios and Background as Test & Precondition issues
Generate/export .feature files
3. Write steps automation code
4. Commit to Git/SVN
5. Checkout code from Git/SVN
6. Generate/Export .feature files
7. Execute Tests
8. Process results & build report
9. Submit results to Xray

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-7931).


Calculator / CALC-7931

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

Details

Type:	Story	Status:	OPEN (View Workflow)
Priority:	Major	Resolution:	Unresolved
Affects Version/s:	None	Fix Version/s:	None
Component/s:	None		
Labels:	None		
Requirement Status:	UNCOVERED		

Description

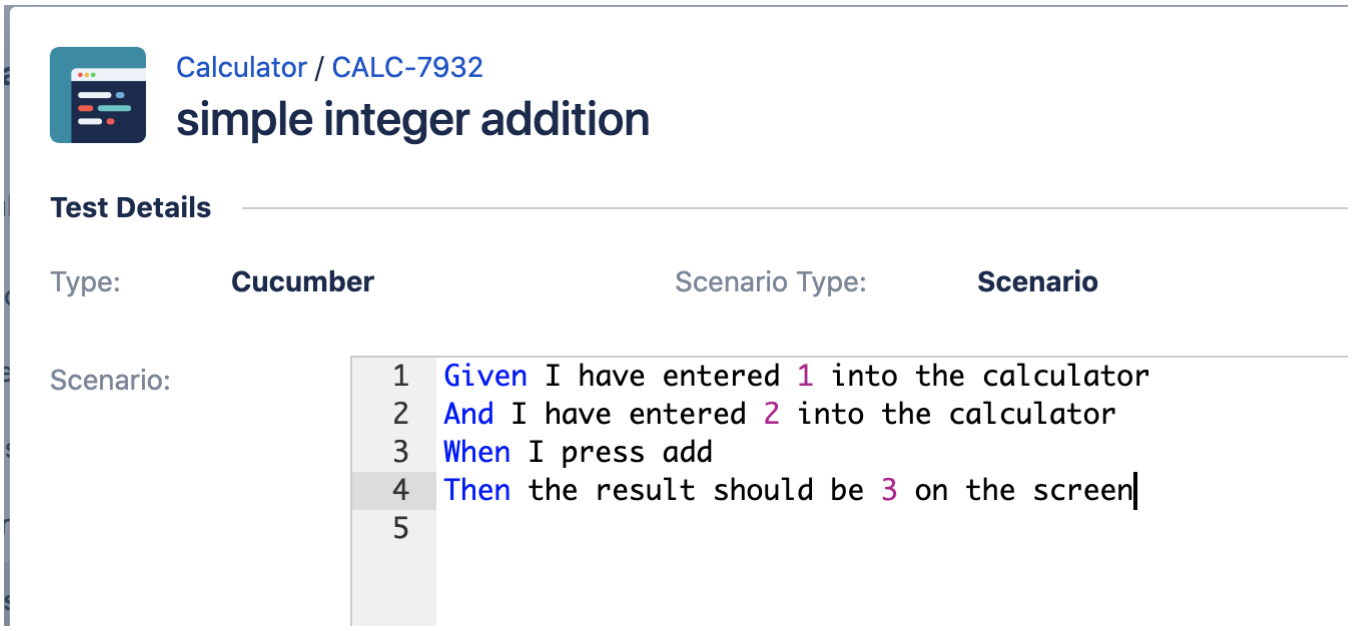
As a user, I can calculate the sum of two numbers

Test Coverage

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

In this case, we'll create a Cucumber Test, of Cucumber Type "Scenario".

We can fill out the Gherkin statements immediately on the Jira issue create dialog or we can create the Test issue first and fill out the details on the next screen, from within the Test issue. In the latter case, we can take advantage of the built-in Gherkin editor which provides auto-complete of Gherkin steps.


Calculator / CALC-7932

simple integer addition

Test Details

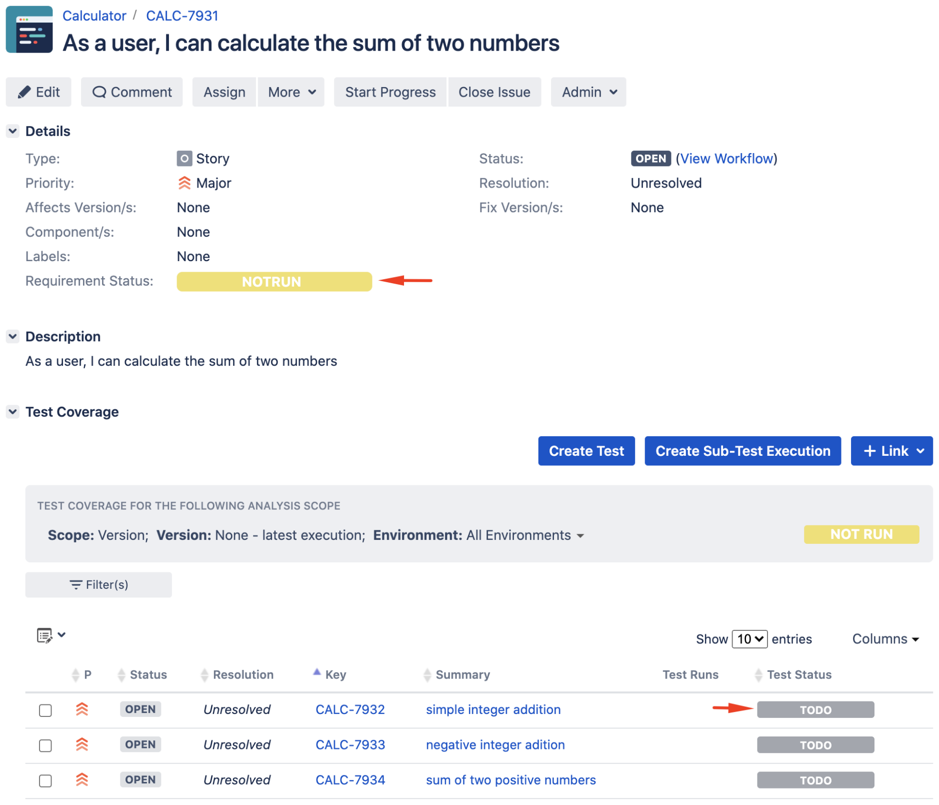
Type:	Cucumber	Scenario Type:	Scenario

Scenario:

```
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
```

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).


Calculator / CALC-7931

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

Edit | Comment | Assign | More | Start Progress | Close Issue | Admin

Details

Type:	Story	Status:	OPEN (View Workflow)
Priority:	Major	Resolution:	Unresolved
Affects Version/s:	None	Fix Version/s:	None
Component/s:	None		
Labels:	None		
Requirement Status:	NOTRUN		

Description

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

Test Coverage

Create Test | Create Sub-Test Execution | + Link

TEST COVERAGE FOR THE FOLLOWING ANALYSIS SCOPE

Scope: Version; **Version:** None - latest execution; **Environment:** All Environments — NOT RUN

Filter(s)

Show 10 entries — Columns

P	Status	Resolution	Key	Summary	Test Runs	Test Status
Major	OPEN	*Unresolved*	CALC-7932	simple integer addition		TODO
Major	OPEN	*Unresolved*	CALC-7933	negative integer adition		TODO
Major	OPEN	*Unresolved*	CALC-7934	sum of two positive numbers		TODO

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 overriden from the command line, whenever executing Maven.

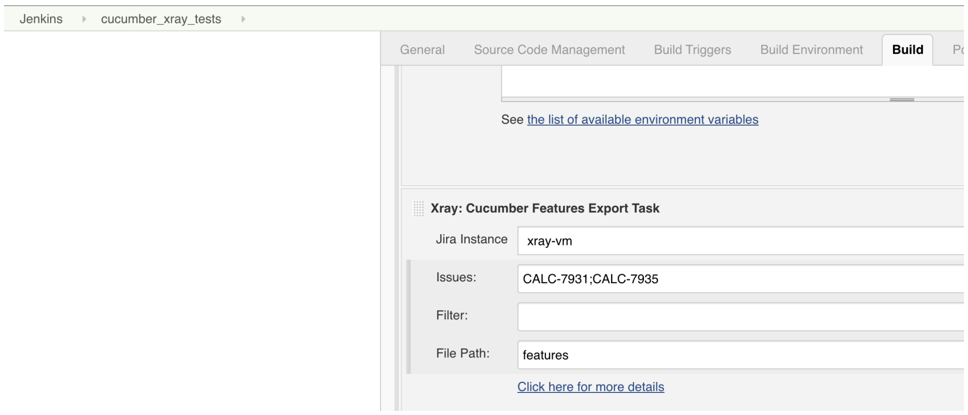
src/test/java/calculator/RunCucumberTest.java

```
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-vm

Issues: CALC-7931;CALC-7935

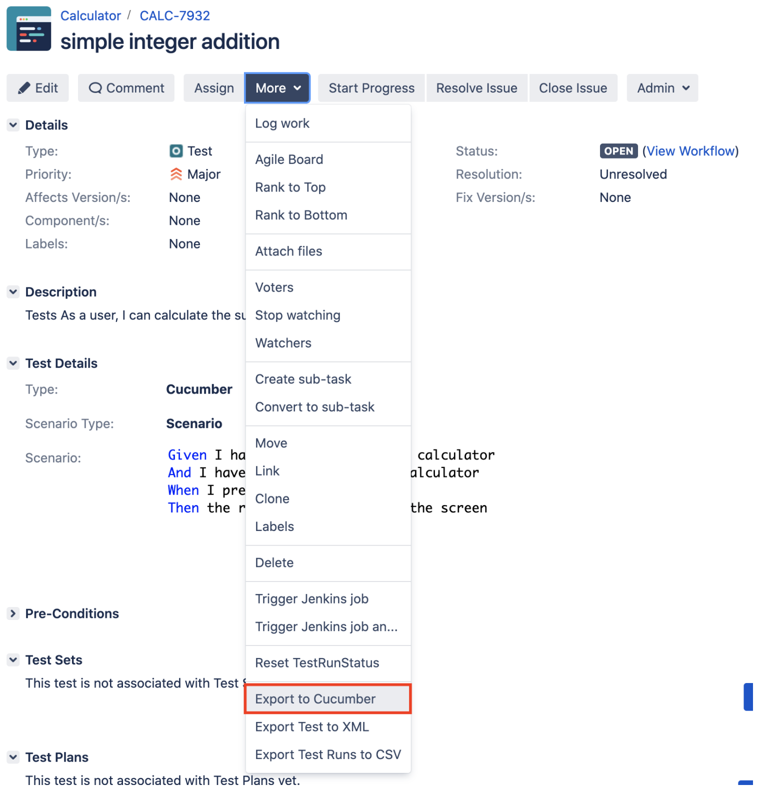
Filter:

File Path: features

- use the REST API directly (more info here)

```
#!/bin/bash

rm -f features/*.feature
curl -u admin:admin  "http://jiraserver.example.com/rest/raven/1.0/export/test?keys=CALC-7931;CALC-7935&fz=true" -o features.zip
unzip -o features.zip  -d features
```

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



Type: Test
Priority: Major
Affects Version/s: None
Component/s: None
Labels: None
Status: OPEN (View Workflow)
Resolution: Unresolved
Fix Version/s: None

Tests As a user, I can calculate the su

Type: Cucumber
Scenario Type: Scenario
Scenario:

```
Given I ha            calculator
And I have           alculator
When I pre
Then the r           the screen
```

This test is not associated with Test S

This test is not associated with Test Plans yet.

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 Export Cucumber Features.

features/1_CALC-7931.feature

```
@REQ_CALC-7931
Feature: As a user, I can calculate the sum of two numbers
	#As a user, I can calculate the sum of two numbers

	#Tests As a user, I can calculate the sum of two numbers
	@TEST_CALC-7934
	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     |


	@TEST_CALC-7933
	Scenario: negative integer adition
		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

	#Tests As a user, I can calculate the sum of two numbers
	@TEST_CALC-7932
	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
```

features/2_CALC-7935.feature

```
@REQ_CALC-7935
Feature: As a user, I can multiply two numbers
	#As a user, I can multiply two numbers

	#simple integer multiplication
	@TEST_CALC-7936
	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.

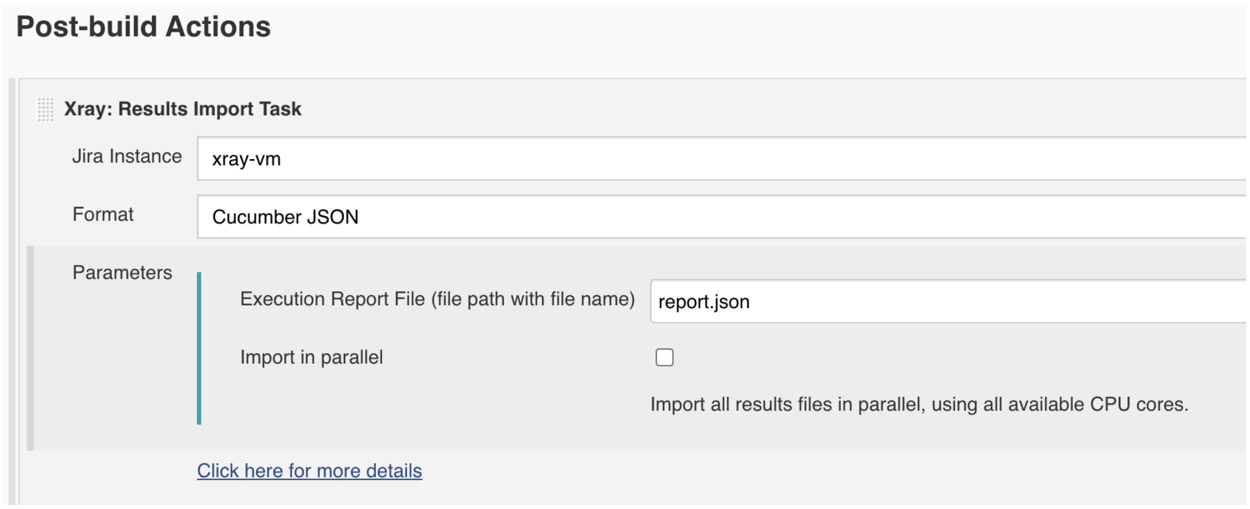
```
mvn compile test -Dcucumber.plugin="json:report.ndjson" -Dcucumber.features="features/"
cat report.ndjson | cucumber-json-formatter --format ndjson > report.json
```

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

```
curl -H "Content-Type: application/json" -X POST -u admin:admin --data @"report.json" http://jiraserver.example.
com/rest/raven/1.0/import/execution/cucumber
```


Post-build Actions

Xray: Results Import Task

Jira Instance: xray-vm

Format: Cucumber JSON

Parameters

Execution Report File (file path with file name): report.json

Import in parallel: ☐

Import all results files in parallel, using all available CPU cores.

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):

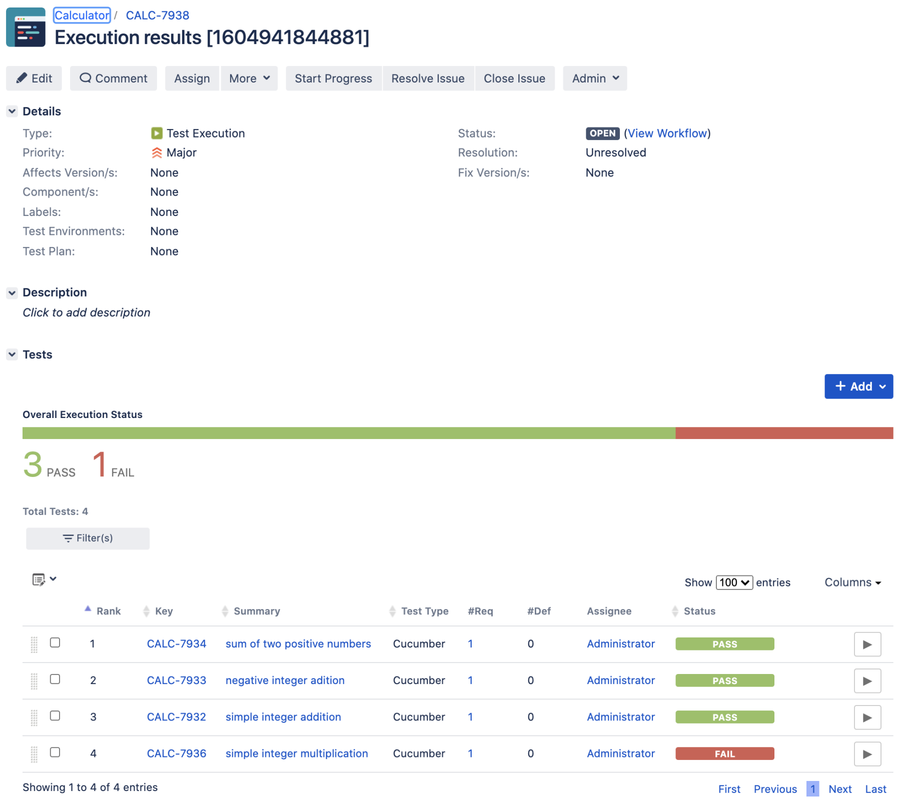
1. the "standard cucumber" endpoint
2. the "multipart cucumber" endpoint

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).


Calculator / CALC-7938

Execution results [1604941844881]

Edit | Comment | Assign | More | Start Progress | Resolve Issue | Close Issue | Admin

Details

Type:	Test Execution	Status:	OPEN (View Workflow)
Priority:	Major	Resolution:	Unresolved
Affects Version/s:	None	Fix Version/s:	None
Component/s:	None		
Labels:	None		
Test Environments:	None		
Test Plan:	None		

Description

Click to add description

Tests

Overall Execution Status

3 PASS 1 FAIL

Total Tests: 4

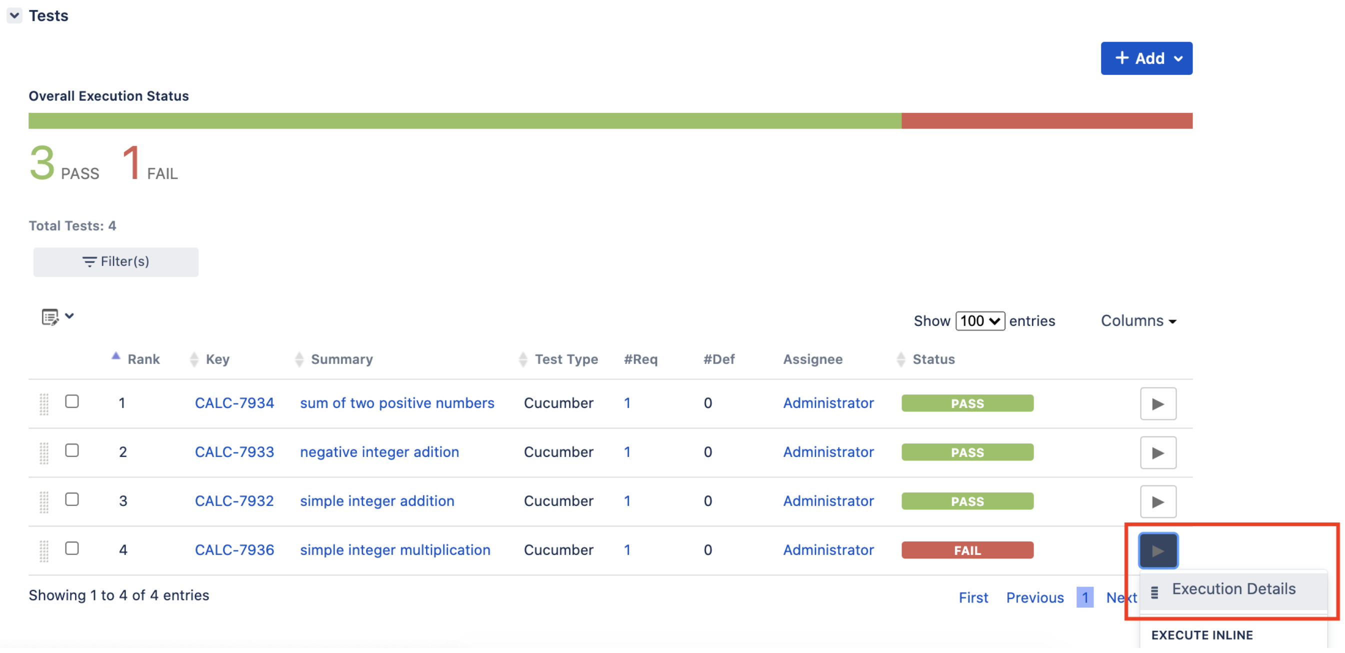
Show 100 entries

Rank	Key	Summary	Test Type	#Req	#Def	Assignee	Status
1	CALC-7934	sum of two positive numbers	Cucumber	1	0	Administrator	PASS
2	CALC-7933	negative integer adition	Cucumber	1	0	Administrator	PASS
3	CALC-7932	simple integer addition	Cucumber	1	0	Administrator	PASS
4	CALC-7936	simple integer multiplication	Cucumber	1	0	Administrator	FAIL

Showing 1 to 4 of 4 entries

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.


Tests

Overall Execution Status

3 PASS 1 FAIL

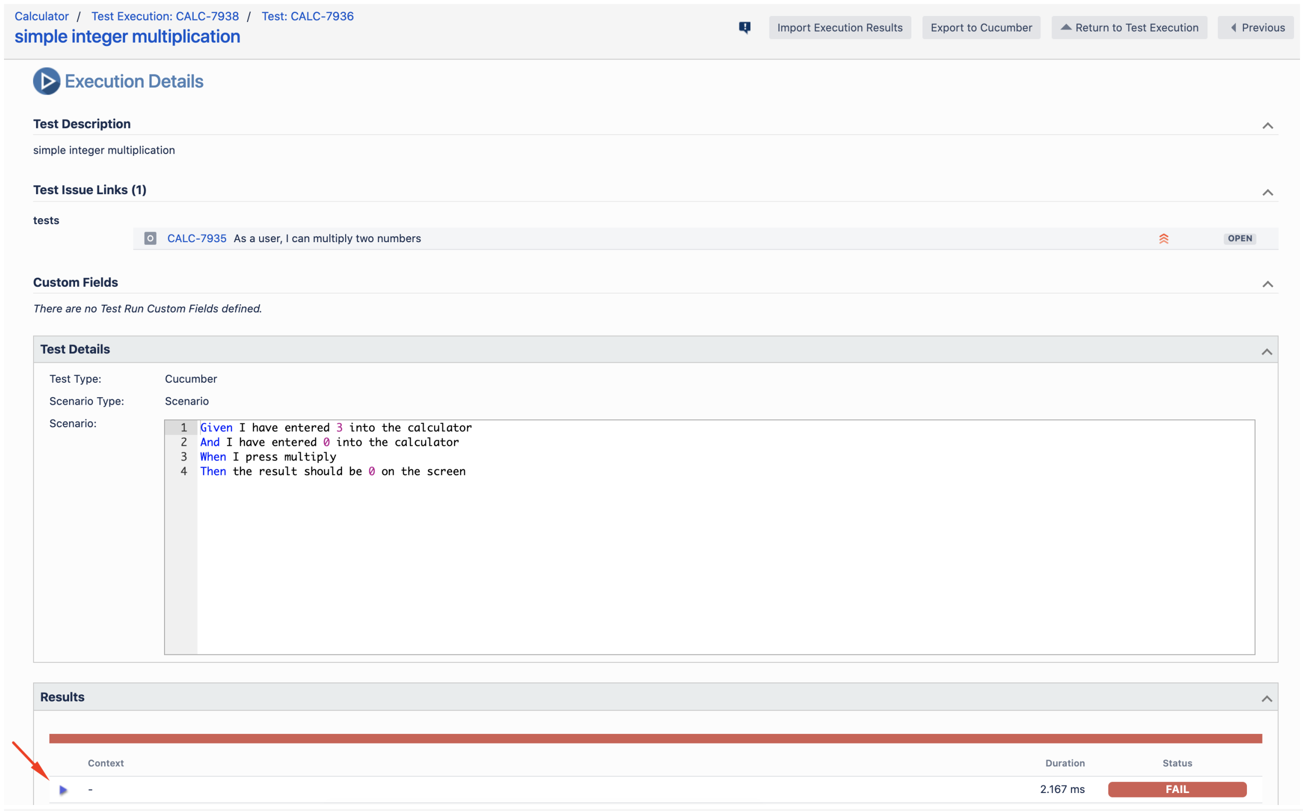
Total Tests: 4

Show 100 entries

Rank	Key	Summary	Test Type	#Req	#Def	Assignee	Status
1	CALC-7934	sum of two positive numbers	Cucumber	1	0	Administrator	PASS
2	CALC-7933	negative integer adition	Cucumber	1	0	Administrator	PASS
3	CALC-7932	simple integer addition	Cucumber	1	0	Administrator	PASS
4	CALC-7936	simple integer multiplication	Cucumber	1	0	Administrator	FAIL

Showing 1 to 4 of 4 entries

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



simple integer multiplication

Execution Details

Test Description

simple integer multiplication

Test Issue Links (1)

tests

CALC-7935 As a user, I can multiply two numbers OPEN

Custom Fields

There are no Test Run Custom Fields defined.

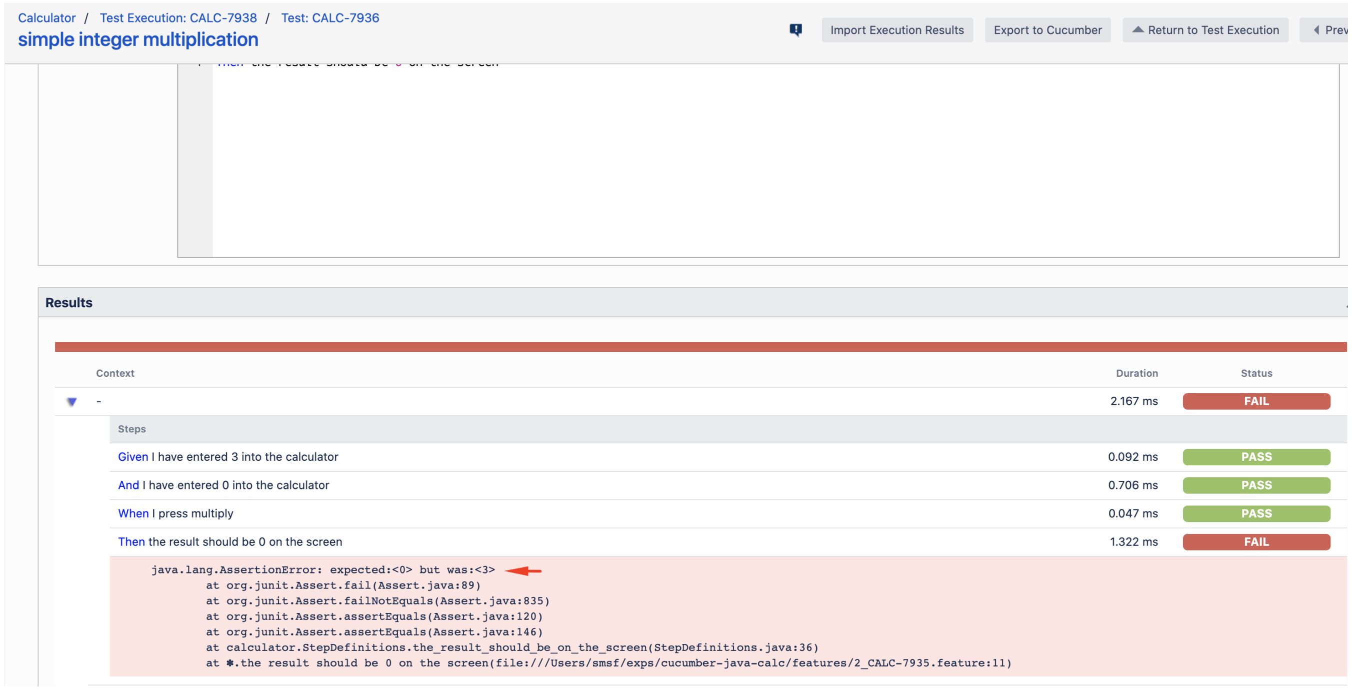
Test Details

Test Type:	Cucumber
Scenario Type:	Scenario
Scenario:	

```
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
```

Results

	Context	Duration	Status
▶	-	2.167 ms	FAIL



simple integer multiplication

Results

	Context	Duration	Status
▼	-	2.167 ms	FAIL

Steps		
Given I have entered 3 into the calculator	0.092 ms	PASS
And I have entered 0 into the calculator	0.706 ms	PASS
When I press multiply	0.047 ms	PASS
Then the result should be 0 on the screen	1.322 ms	FAIL

```
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 ✱.the result should be 0 on the screen(file:///Users/smsf/exps/cucumber-java-calc/features/2_CALC-7935.feature:11)
```

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

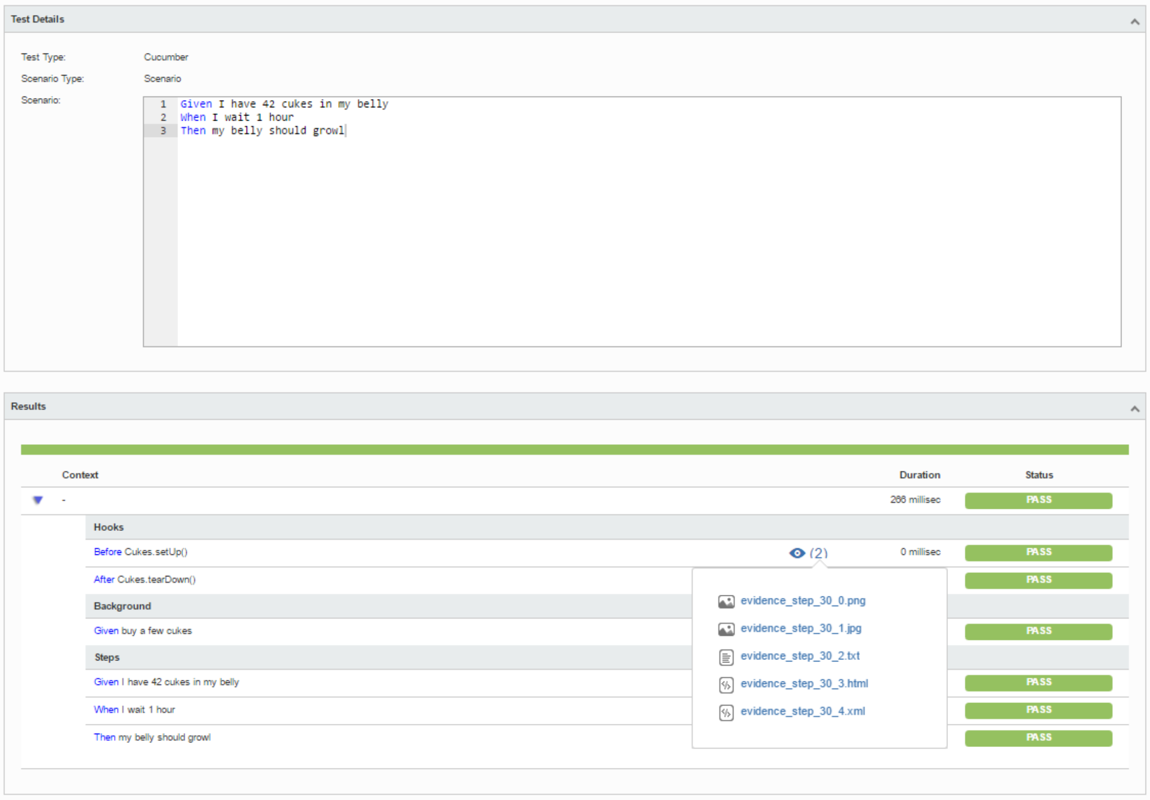
buggy Multiply() method in Calculator.java

```
// Multiply two integers and retuns 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;
    }
}
```

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 (2) represents the evidences ("embeddings") for each **Hook, Background** and **Steps**.



Test Details

Test Type:	Cucumber
Scenario Type:	Scenario
Scenario:	

```
Given I have 42 cukes in my belly
When I wait 1 hour
Then my belly should growl
```

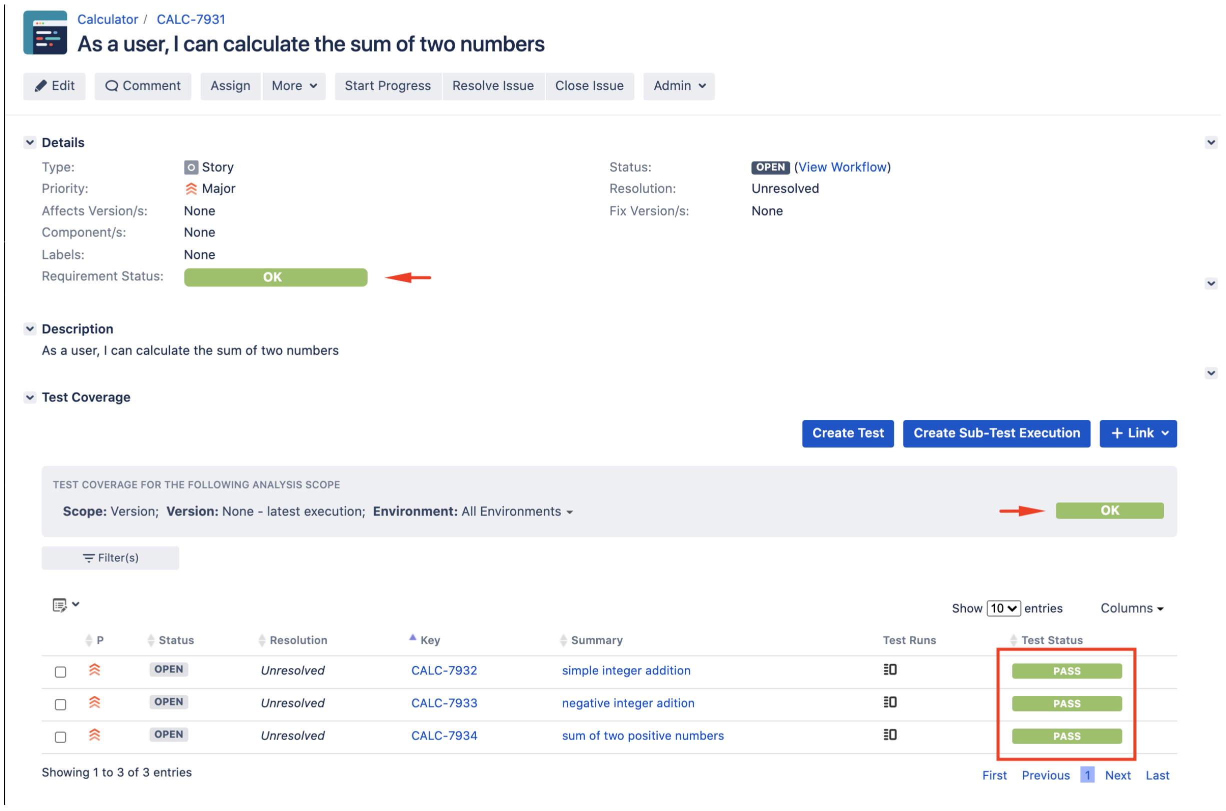
Results

Context		Duration	Status
-		266 millisec	PASS
Hooks			
Before Cukes.setUp()	(2)	0 millisec	PASS
After Cukes.tearDown()			PASS
Background			
Given buy a few cukes			PASS
Steps			
Given I have 42 cukes in my belly			PASS
When I wait 1 hour			PASS
Then my belly should growl			PASS

evidence_step_30_0.png
evidence_step_30_1.jpg
evidence_step_30_2.txt
evidence_step_30_3.html
evidence_step_30_4.xml

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

Coverage now shows that the addition related user story (e.g. CALC-7931) is OK based on the latest testing results; on the other hand, the multiplication related user story (CALC-7935) is NOK since it has one test currently failing.


Calculator / CALC-7931

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

Details

Type:	Story	Status:	OPEN (View Workflow)
Priority:	Major	Resolution:	Unresolved
Affects Version/s:	None	Fix Version/s:	None
Component/s:	None		
Labels:	None		
Requirement Status:	OK		

Description

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

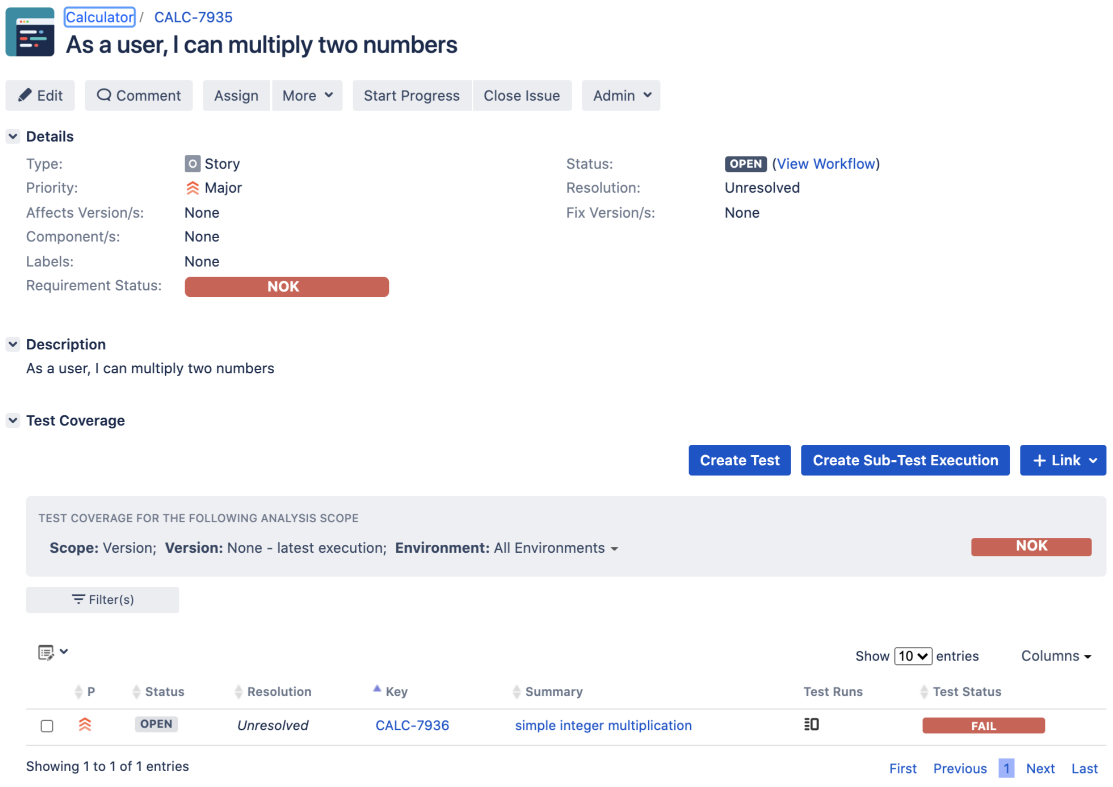
Test Coverage

TEST COVERAGE FOR THE FOLLOWING ANALYSIS SCOPE

Scope: Version; **Version:** None - latest execution; **Environment:** All Environments — OK

P	Status	Resolution	Key	Summary	Test Runs	Test Status
	OPEN	Unresolved	CALC-7932	simple integer addition		PASS
	OPEN	Unresolved	CALC-7933	negative integer adition		PASS
	OPEN	Unresolved	CALC-7934	sum of two positive numbers		PASS

Showing 1 to 3 of 3 entries


Calculator / CALC-7935

As a user, I can multiply two numbers

Details

Type:	Story	Status:	OPEN (View Workflow)
Priority:	Major	Resolution:	Unresolved
Affects Version/s:	None	Fix Version/s:	None
Component/s:	None		
Labels:	None		
Requirement Status:	NOK		

Description

As a user, I can multiply two numbers

Test Coverage

TEST COVERAGE FOR THE FOLLOWING ANALYSIS SCOPE

Scope: Version; **Version:** None - latest execution; **Environment:** All Environments — NOK

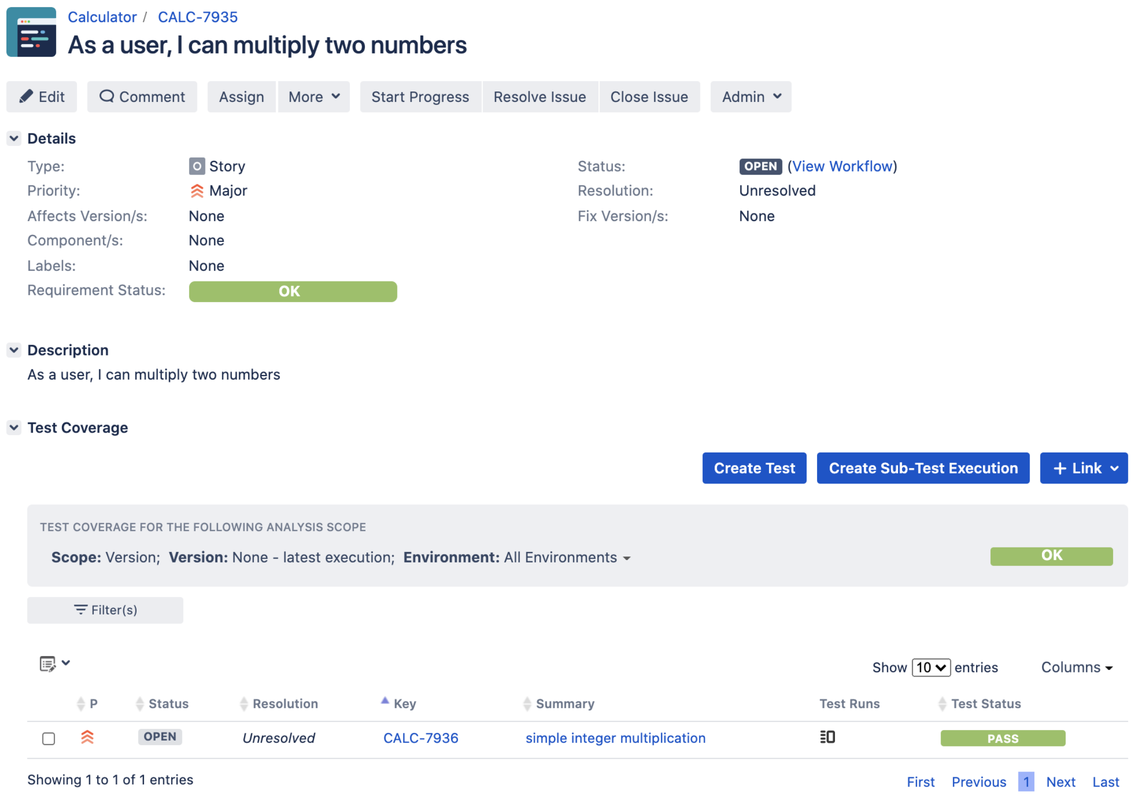
P	Status	Resolution	Key	Summary	Test Runs	Test Status
	OPEN	Unresolved	CALC-7936	simple integer multiplication		FAIL

Showing 1 to 1 of 1 entries

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;
}
```


Calculator / CALC-7935

As a user, I can multiply two numbers

Details

Type:	Story	Status:	OPEN (View Workflow)
Priority:	Major	Resolution:	Unresolved
Affects Version/s:	None	Fix Version/s:	None
Component/s:	None		
Labels:	None		
Requirement Status:	OK		

Description

As a user, I can multiply two numbers

Test Coverage

TEST COVERAGE FOR THE FOLLOWING ANALYSIS SCOPE

Scope: Version; **Version:** None - latest execution; **Environment:** All Environments OK

P	Status	Resolution	Key	Summary	Test Runs	Test Status
	OPEN	*Unresolved*	CALC-7936	simple integer multiplication		PASS

Showing 1 to 1 of 1 entries

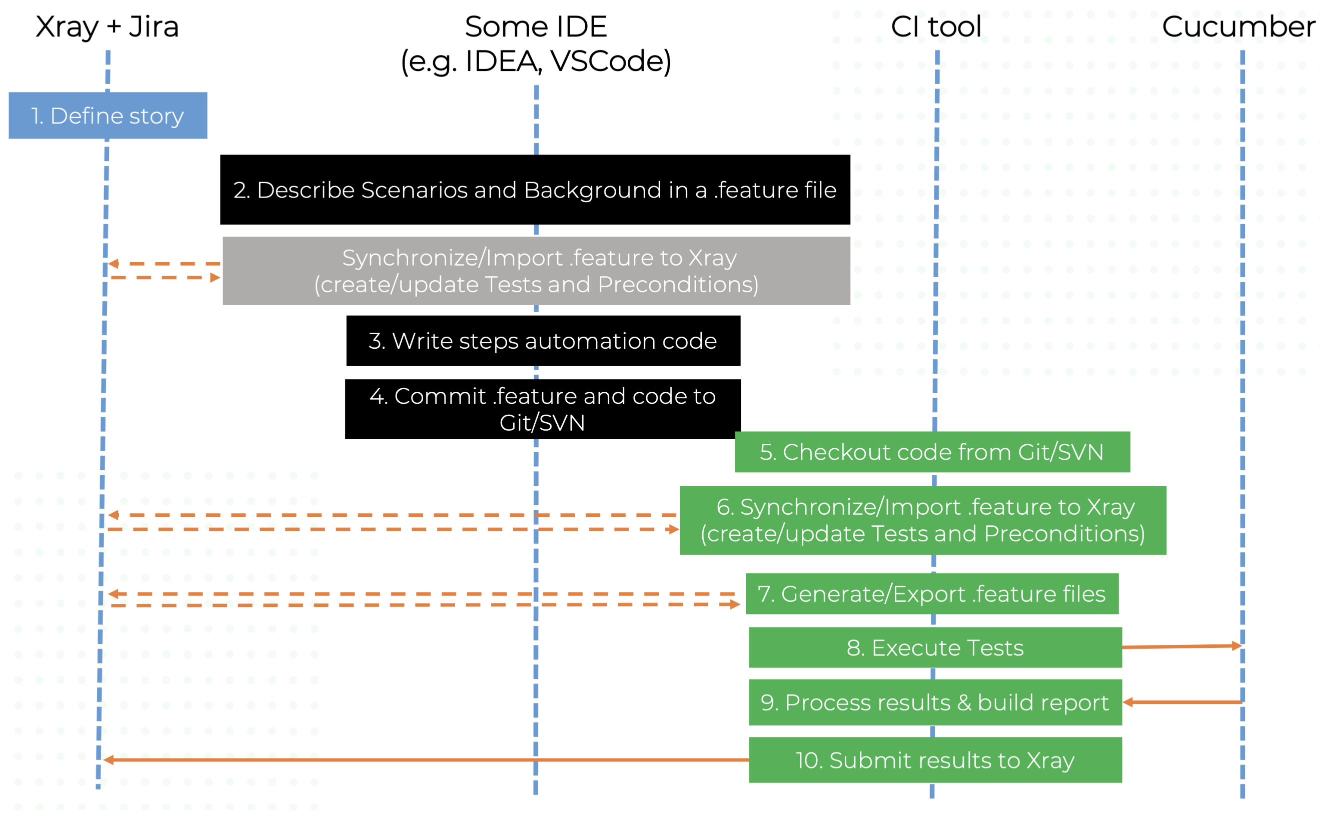
Using Git or other VCS as master

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

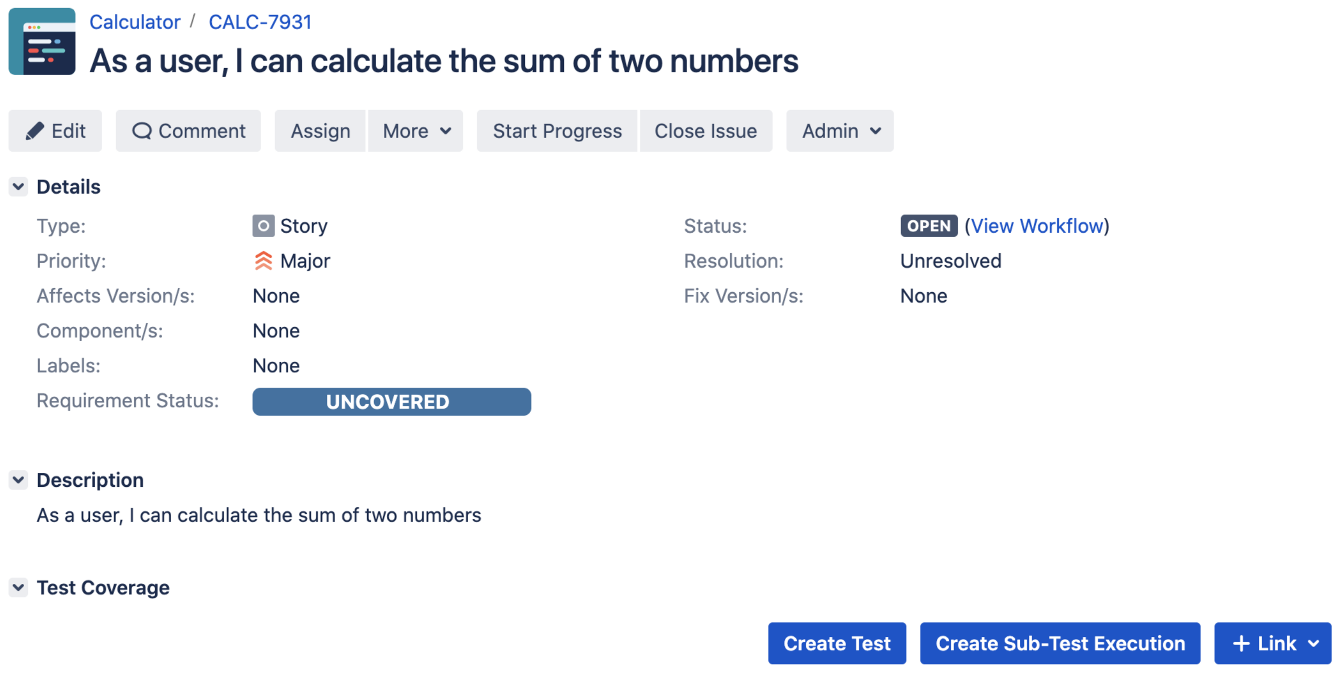

Xray + Jira
Some IDE (e.g. IDEA, VSCode)
CI tool
Cucumber
1. Define story
2. Describe Scenarios and Background in a .feature file
Synchronize/Import .feature to Xray (create/update Tests and Preconditions)
3. Write steps automation code
4. Commit .feature and code to Git/SVN
5. Checkout code from Git/SVN
6. Synchronize/Import .feature to Xray (create/update Tests and Preconditions)
7. Generate/Export .feature files
8. Execute Tests
9. Process results & build report
10. Submit results to Xray

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-7931).



Calculator / CALC-7931

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

Details

Type:	Story	Status:	OPEN (View Workflow)
Priority:	Major	Resolution:	Unresolved
Affects Version/s:	None	Fix Version/s:	None
Component/s:	None		
Labels:	None		
Requirement Status:	UNCOVERED		

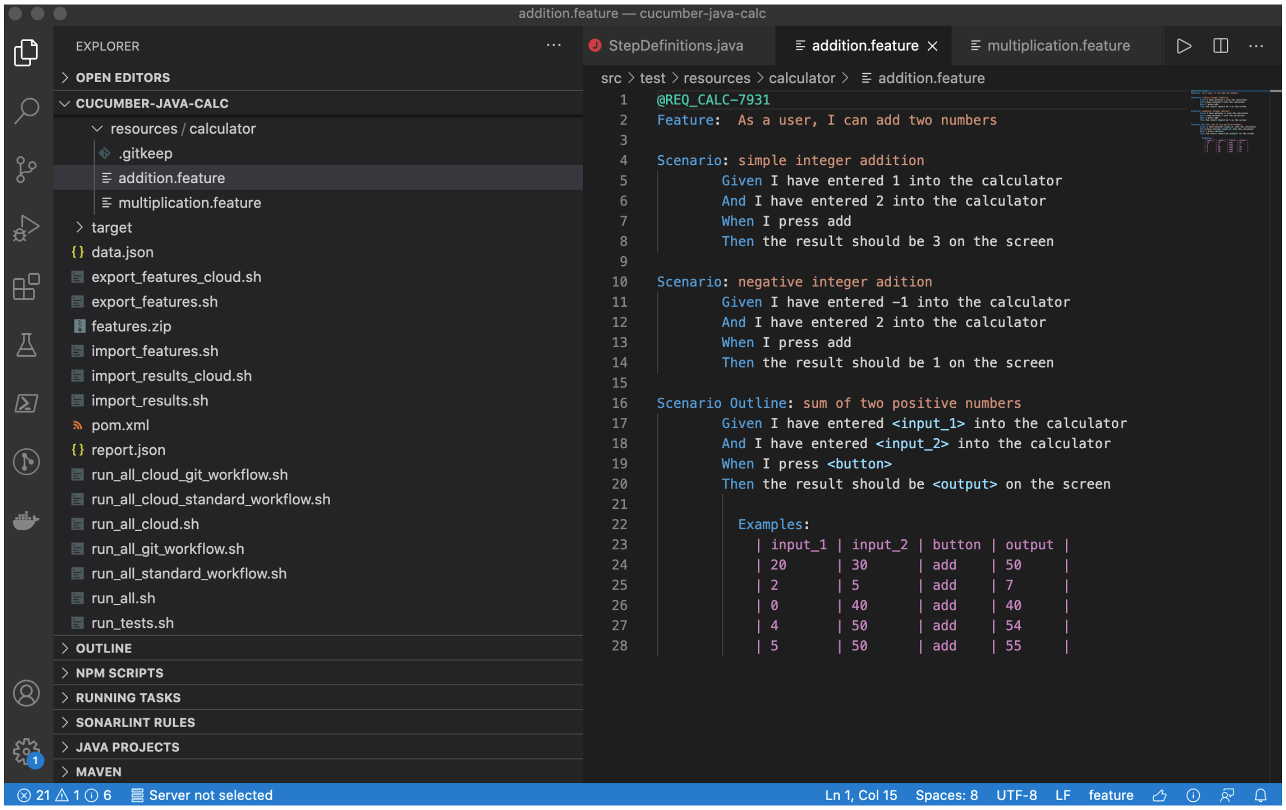
Description

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

Test Coverage

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.



```
@REQ_CALC-7931
Feature:  As a user, I can add two numbers

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

Scenario: negative integer adition
        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

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     |
```

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 overriden from the command line, whenever executing Maven.

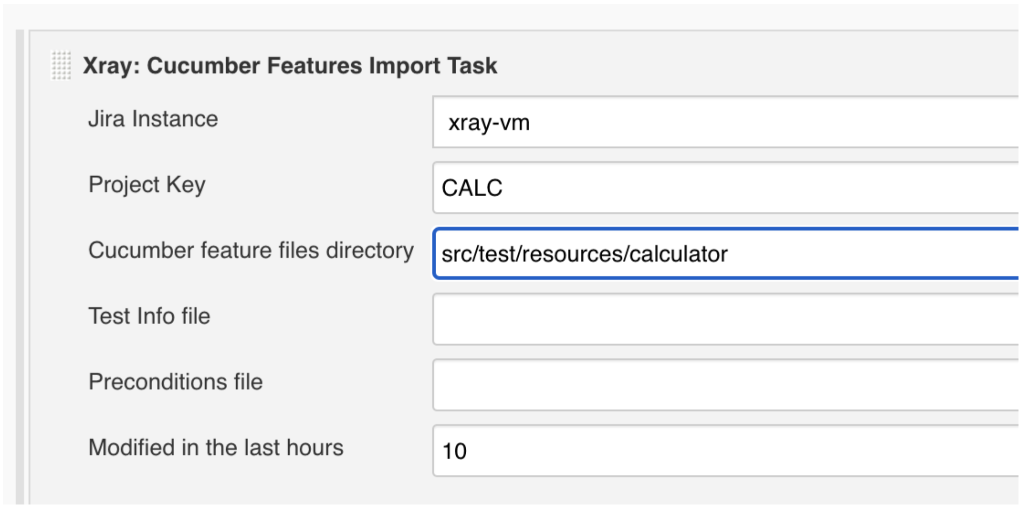
src/test/java/calculator/RunCucumberTest.java

```
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

```
rm -f features.zip
zip -r features.zip src/test/resources/calculator/ -i \*.feature
curl -H "Content-Type: multipart/form-data" -u admin:admin -F "file=@features.zip" "http://jiraserver.example.
com/rest/raven/1.0/import/feature?projectKey=CALC"
```

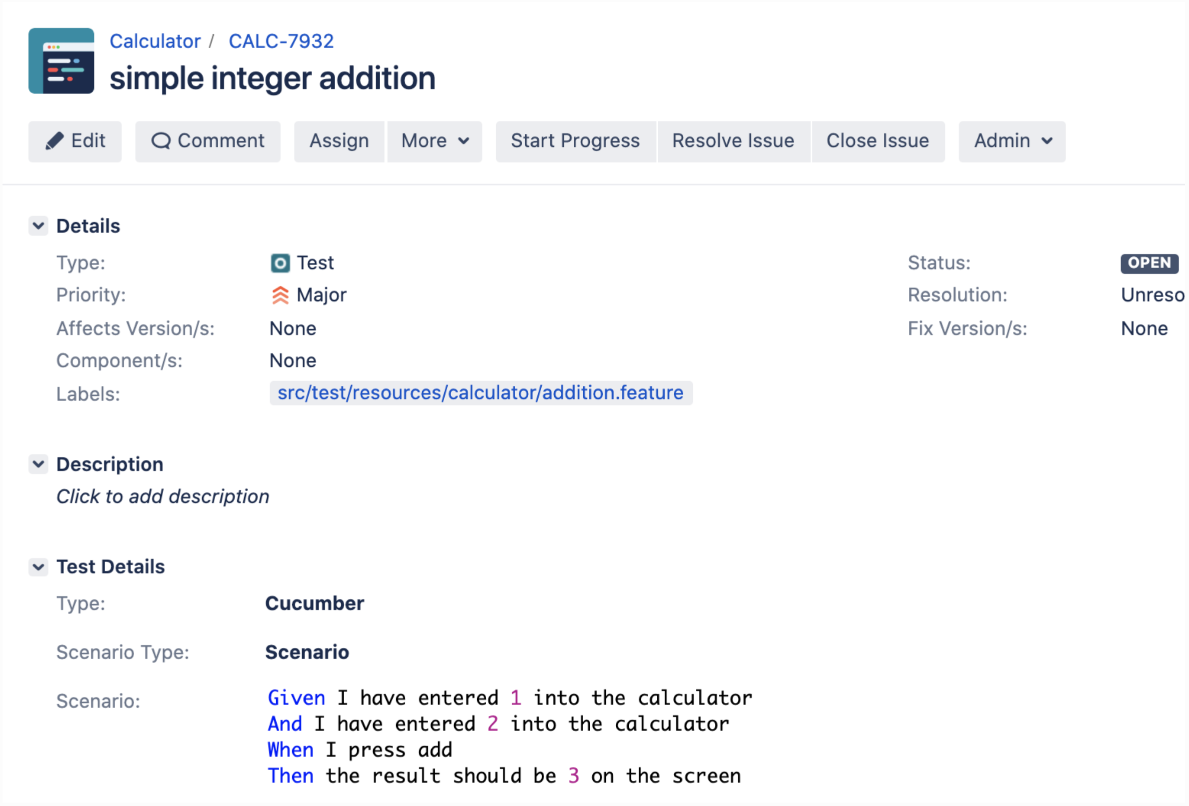


Xray: Cucumber Features Import Task

Jira Instance	xray-vm
Project Key	CALC
Cucumber feature files directory	src/test/resources/calculator
Test Info file	
Preconditions file	
Modified in the last hours	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.



simple integer addition

Details

Type:	Test	Status:	OPEN
Priority:	Major	Resolution:	Unreso
Affects Version/s:	None	Fix Version/s:	None
Component/s:	None		
Labels:	src/test/resources/calculator/addition.feature		

Description

Click to add description

Test Details

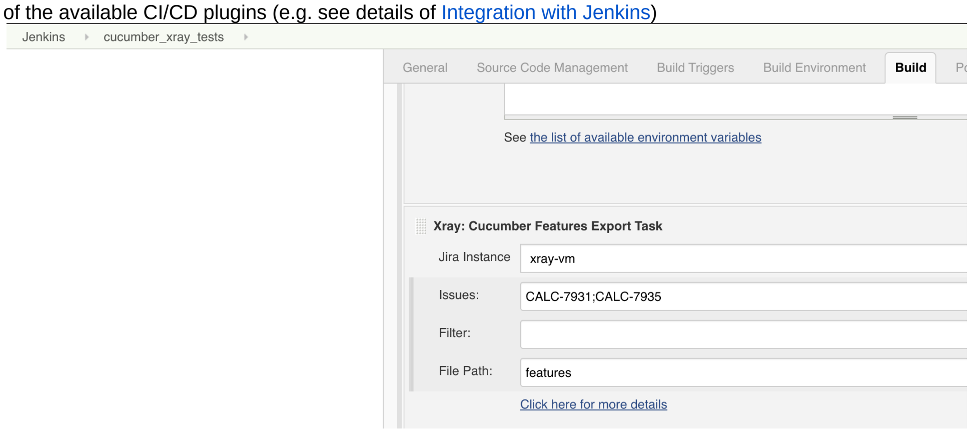
Type:	**Cucumber**
Scenario Type:	**Scenario**

Scenario:

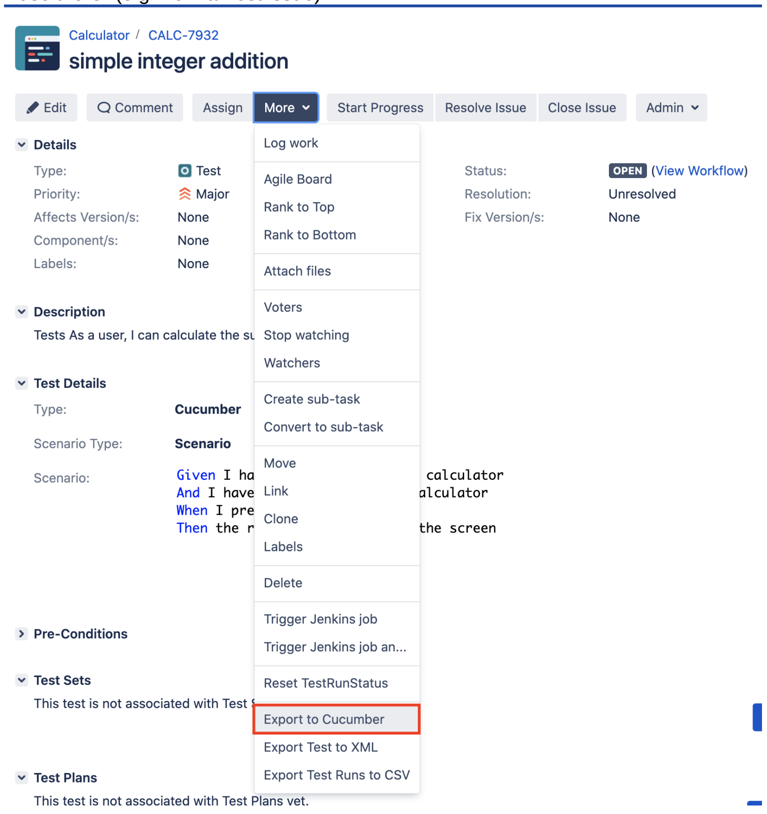
```
Given I have entered 1 into the calculator
And I have entered 2 into the calculator
When I press add
Then the result should be 3 on the screen
```

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)
    - 
      Jenkins
      cucumber_xray_tests
      General
      Source Code Management
      Build Triggers
      Build Environment
      Build
      See the list of available environment variables
      Xray: Cucumber Features Export Task
      Click here for more details

      Jira Instance: xray-vm
      Issues: CALC-7931;CALC-7935
      Filter:
      File Path: features
- use the REST API directly (more info here)

    -
      ```
      #!/bin/bash

      rm -f features/*.feature
      curl -u admin:admin  "http://jiraserver.example.com/rest/raven/1.0/export/test?keys=CALC-7931;CALC-7935&fz=true" -o features.zip
      unzip -o features.zip  -d features
      ```
- ... or even use the UI (e.g. from a Test issue)
    - 

      Type: Test
      Priority: Major
      Affects Version/s: None
      Component/s: None
      Labels: None
      Status: OPEN (View Workflow)
      Resolution: Unresolved
      Fix Version/s: None
      Tests As a user, I can calculate the su
      Type: Cucumber
      Scenario Type: Scenario
      Scenario:
      ```
      Given I ha            calculator
      And I have           alculator
      When I pre
      Then the r           the screen
      ```
      This test is not associated with Test S
      This test is not associated with Test Plans yet.

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 Export Cucumber Features.

features/1_CALC-7931.feature

```
@REQ_CALC-7931
Feature: As a user, I can calculate the sum of two numbers
        #As a user, I can calculate the sum of two numbers

        #Tests As a user, I can calculate the sum of two numbers
        @TEST_CALC-7934
        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     |


        @TEST_CALC-7933
        Scenario: negative integer adition
                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

        #Tests As a user, I can calculate the sum of two numbers
        @TEST_CALC-7932
        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
```

features/2_CALC-7935.feature

```
@REQ_CALC-7935
Feature: As a user, I can multiply two numbers
        #As a user, I can multiply two numbers

        #simple integer multiplication
        @TEST_CALC-7936
        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.

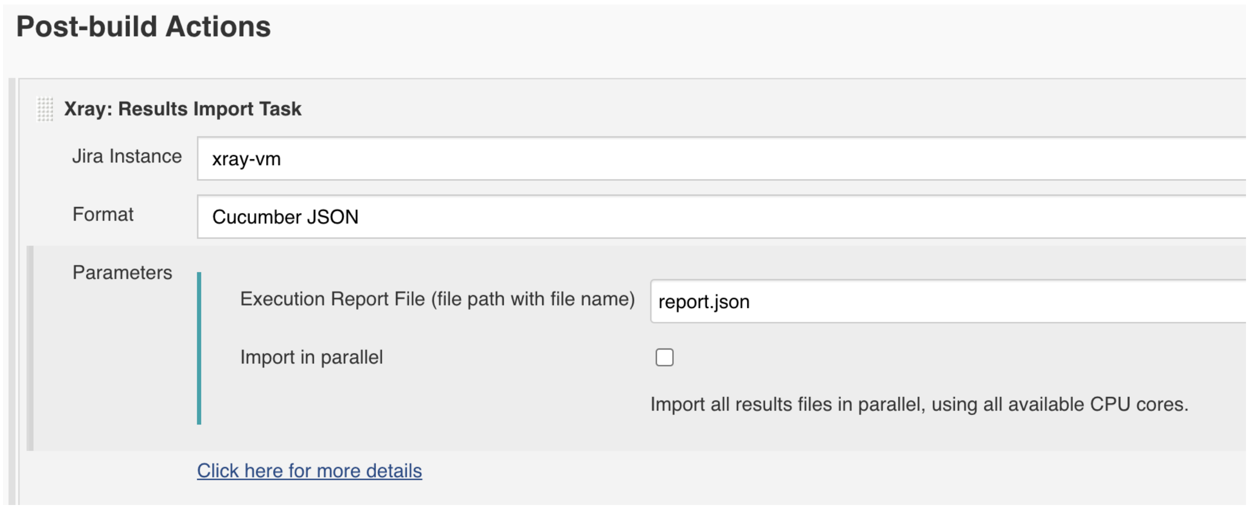
```
mvn compile test -Dcucumber.plugin="json:report.ndjson" -Dcucumber.features="features/"
cat report.ndjson | cucumber-json-formatter --format ndjson > report.json
```

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

```
curl -H "Content-Type: application/json" -X POST -u admin:admin --data @"report.json" http://jiraserver.example.
com/rest/raven/1.0/import/execution/cucumber
```


Post-build Actions

Xray: Results Import Task

Jira Instance: xray-vm

Format: Cucumber JSON

Parameters

Execution Report File (file path with file name): report.json

Import in parallel: ☐

Import all results files in parallel, using all available CPU cores.

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):

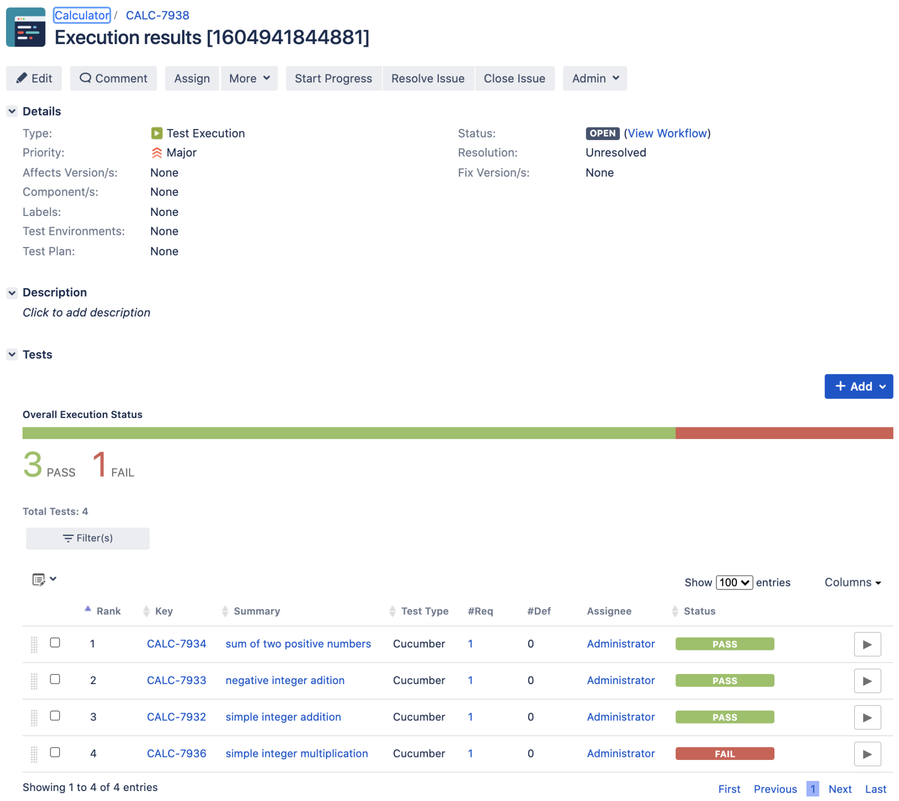
1. the "standard cucumber" endpoint
2. the "multipart cucumber" endpoint

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).



Calculator / CALC-7938

Execution results [1604941844881]

Edit | Comment | Assign | More | Start Progress | Resolve Issue | Close Issue | Admin

Details

Type:	Test Execution	Status:	OPEN (View Workflow)
Priority:	Major	Resolution:	Unresolved
Affects Version/s:	None	Fix Version/s:	None
Component/s:	None		
Labels:	None		
Test Environments:	None		
Test Plan:	None		

Description

Click to add description

Tests

Overall Execution Status

3 PASS 1 FAIL

Total Tests: 4

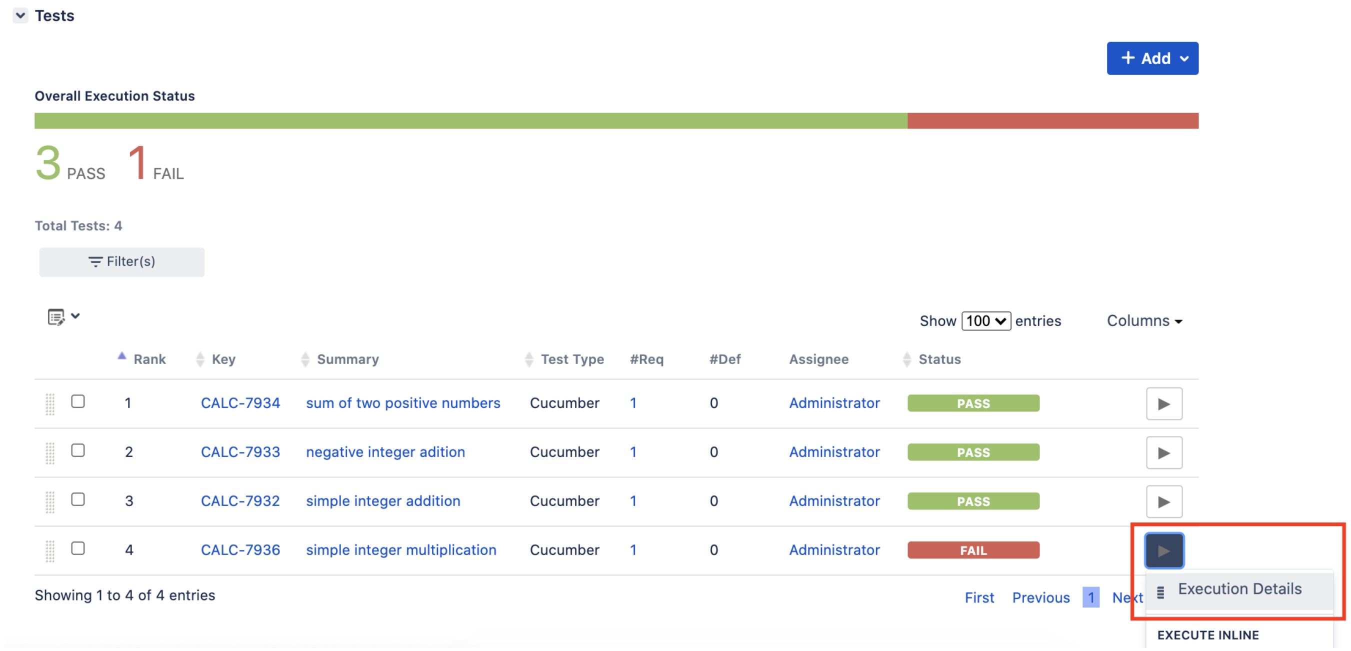
Show 100 entries

Rank	Key	Summary	Test Type	#Req	#Def	Assignee	Status
1	CALC-7934	sum of two positive numbers	Cucumber	1	0	Administrator	PASS
2	CALC-7933	negative integer adition	Cucumber	1	0	Administrator	PASS
3	CALC-7932	simple integer addition	Cucumber	1	0	Administrator	PASS
4	CALC-7936	simple integer multiplication	Cucumber	1	0	Administrator	FAIL

Showing 1 to 4 of 4 entries

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.



Tests

Overall Execution Status

3 PASS 1 FAIL

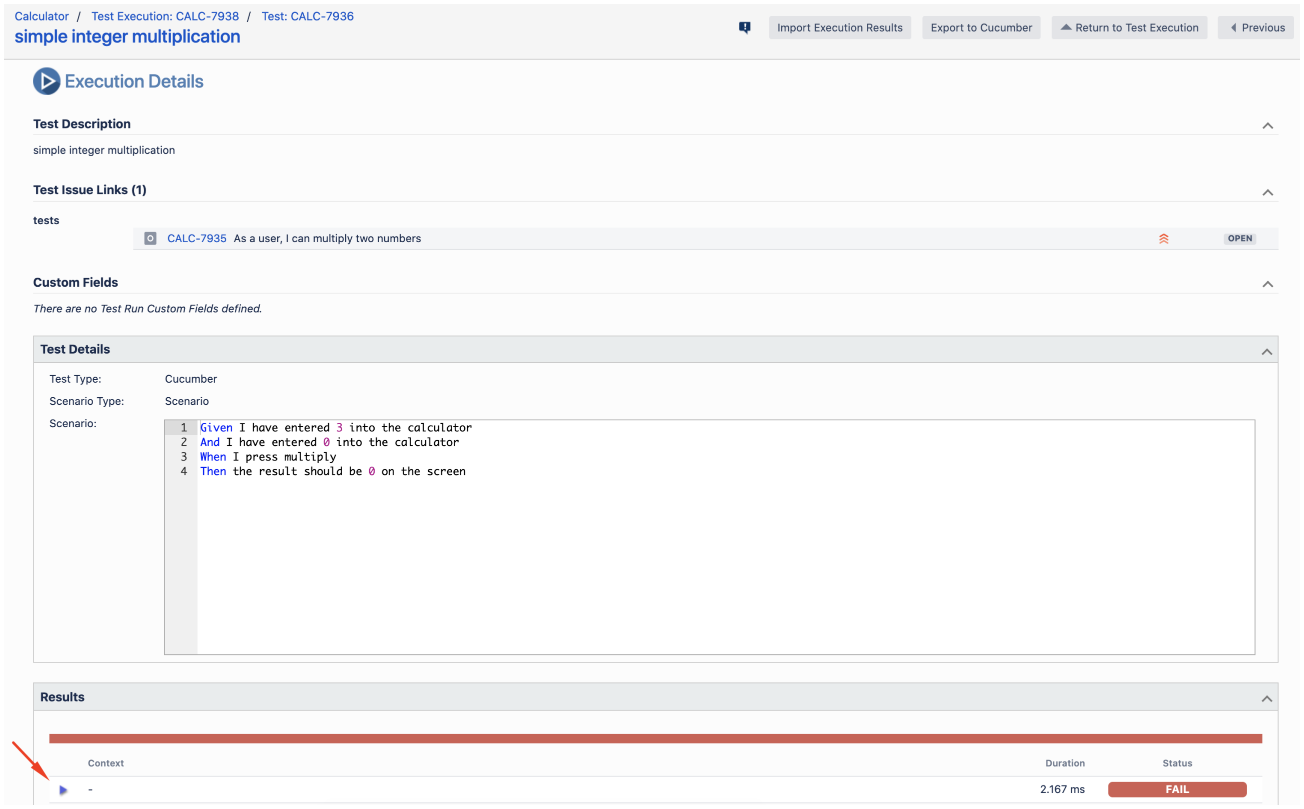
Total Tests: 4

Show 100 entries

Rank	Key	Summary	Test Type	#Req	#Def	Assignee	Status
1	CALC-7934	sum of two positive numbers	Cucumber	1	0	Administrator	PASS
2	CALC-7933	negative integer adition	Cucumber	1	0	Administrator	PASS
3	CALC-7932	simple integer addition	Cucumber	1	0	Administrator	PASS
4	CALC-7936	simple integer multiplication	Cucumber	1	0	Administrator	FAIL

Showing 1 to 4 of 4 entries

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


Calculator / Test Execution: CALC-7938 / Test: CALC-7936

simple integer multiplication

Execution Details

Test Description

simple integer multiplication

Test Issue Links (1)

tests

CALC-7935 As a user, I can multiply two numbers OPEN

Custom Fields

There are no Test Run Custom Fields defined.

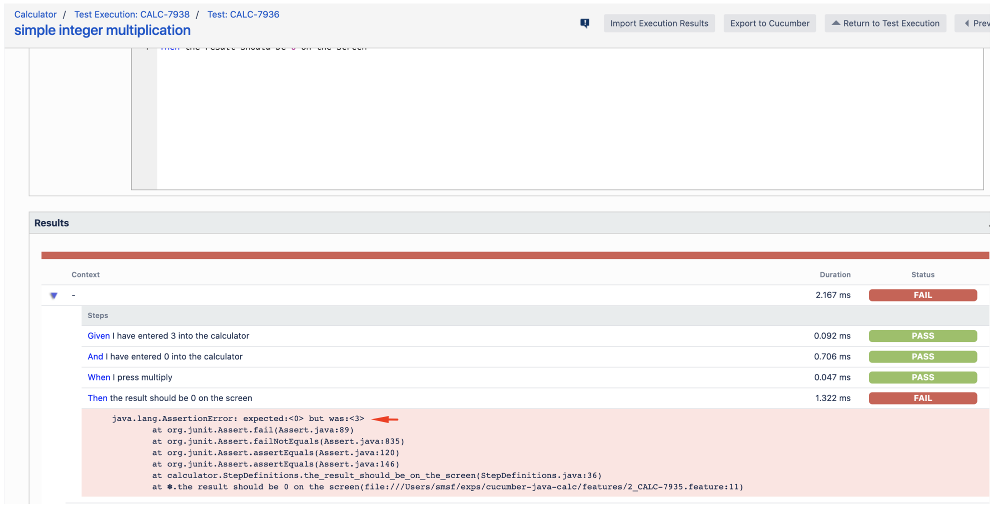
Test Details

Test Type:	Cucumber
Scenario Type:	Scenario
Scenario:	

```
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
```

Results

Context	Duration	Status
-	2.167 ms	FAIL


Calculator / Test Execution: CALC-7938 / Test: CALC-7936

simple integer multiplication

Results

Context	Duration	Status
-	2.167 ms	FAIL

Steps		
Given I have entered 3 into the calculator	0.092 ms	PASS
And I have entered 0 into the calculator	0.706 ms	PASS
When I press multiply	0.047 ms	PASS
Then the result should be 0 on the screen	1.322 ms	FAIL

```
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 ✱.the result should be 0 on the screen(file:///Users/smsf/exps/cucumber-java-calc/features/2_CALC-7935.feature:11)
```

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

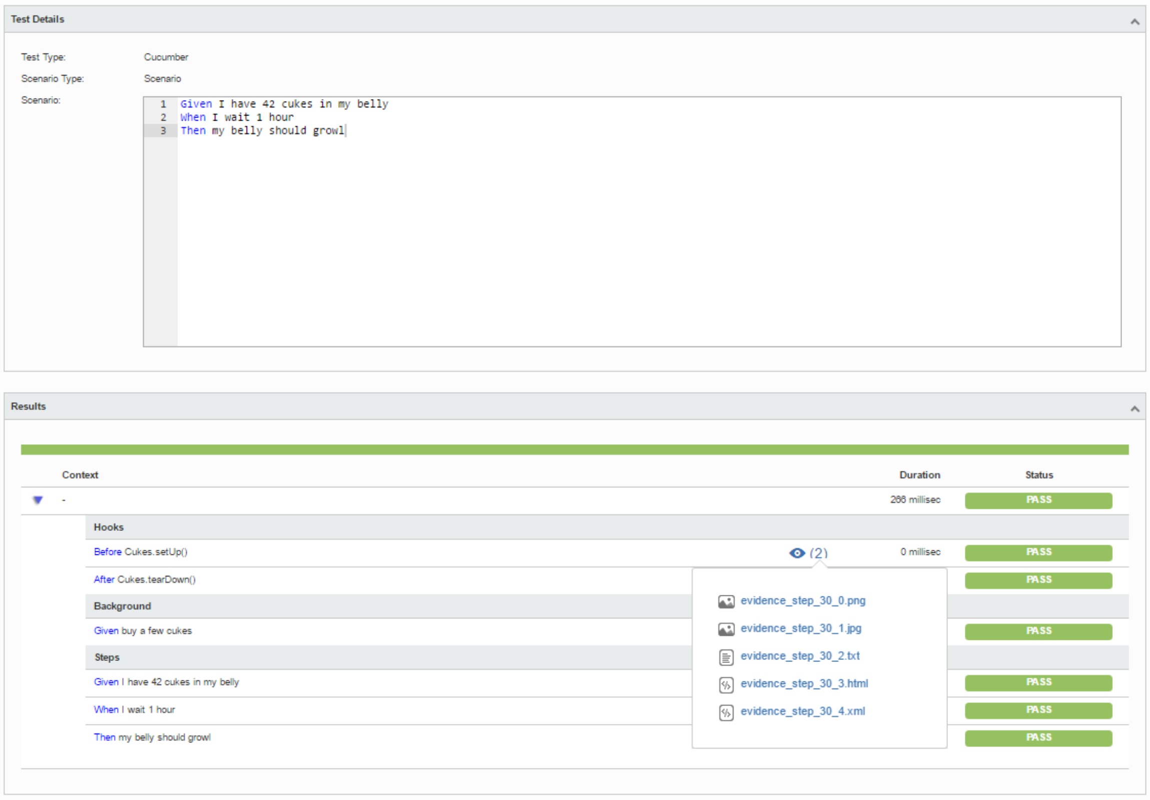
buggy Multiply() method in Calculator.java

```java
// Multiply two integers and retuns 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;
    }
}
```

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 👁 (2) represents the evidences ("embeddings") for each **Hook, Background** and **Steps**.



Test Details

Test Type:	Cucumber
Scenario Type:	Scenario
Scenario:	

```
Given I have 42 cukes in my belly
When I wait 1 hour
Then my belly should growl
```

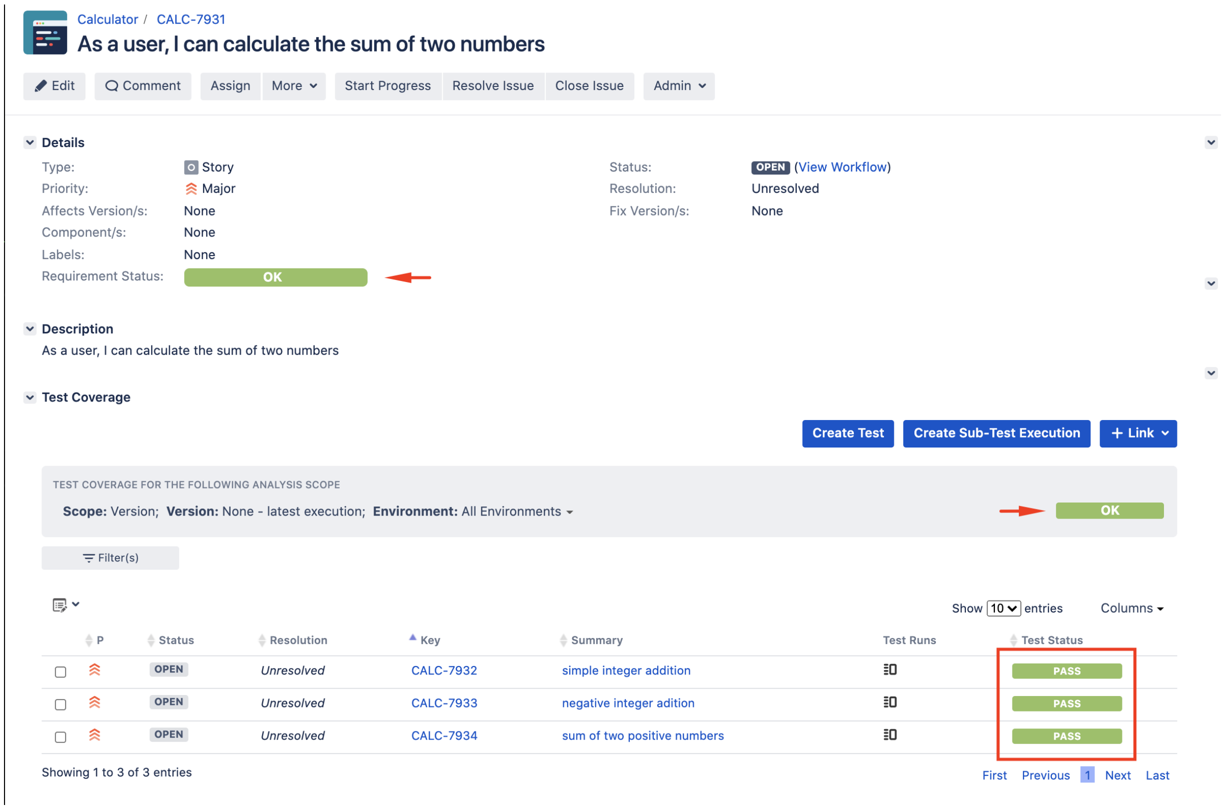
Results

Context		Duration	Status
-		266 millisec	PASS
Hooks			
Before Cukes.setUp()	👁 (2)	0 millisec	PASS
After Cukes.tearDown()			PASS
Background			
Given buy a few cukes			PASS
Steps			
Given I have 42 cukes in my belly			PASS
When I wait 1 hour			PASS
Then my belly should growl			PASS

evidence_step_30_0.png
evidence_step_30_1.jpg
evidence_step_30_2.txt
evidence_step_30_3.html
evidence_step_30_4.xml

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

Coverage now shows that the addition related user story (e.g. CALC-7931) is OK based on the latest testing results; on the other hand, the multiplication related user story (CALC-7935) is NOK since it has one test currently failing.



Calculator / CALC-7931

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

Details

Type:	Story	Status:	OPEN (View Workflow)
Priority:	Major	Resolution:	Unresolved
Affects Version/s:	None	Fix Version/s:	None
Component/s:	None		
Labels:	None		
Requirement Status:	OK		

Description

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

Test Coverage

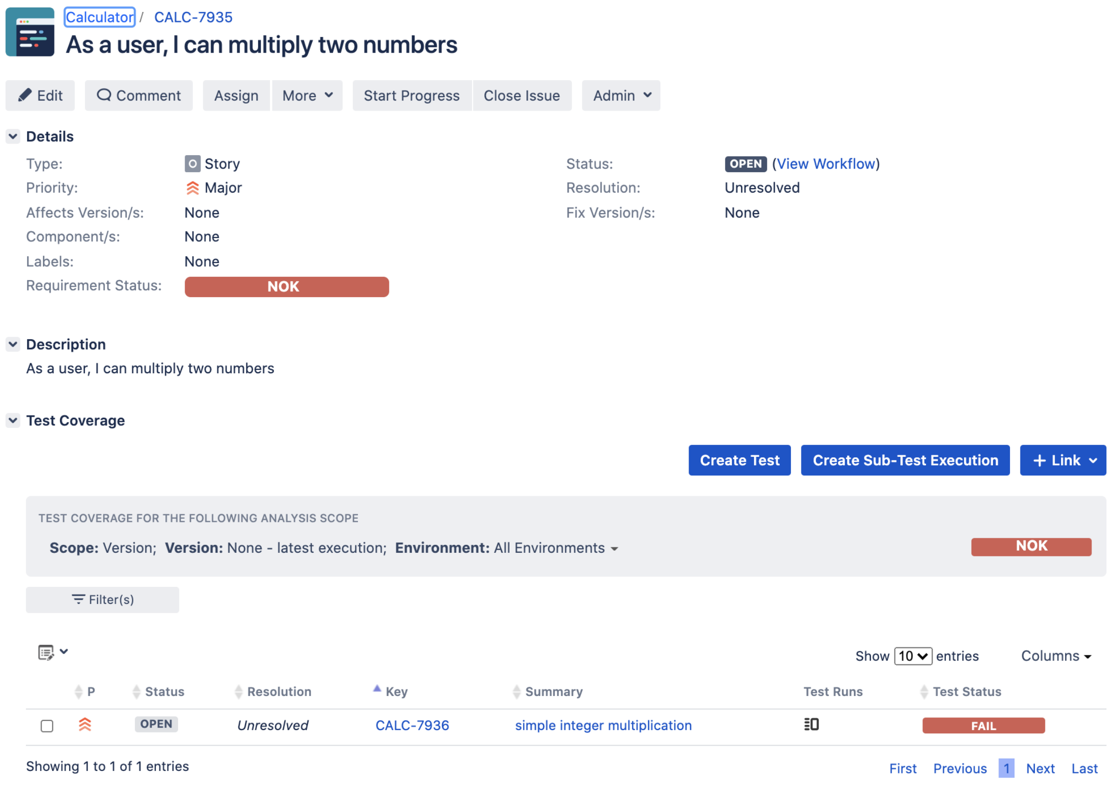
TEST COVERAGE FOR THE FOLLOWING ANALYSIS SCOPE

Scope: Version; **Version:** None - latest execution; **Environment:** All Environments OK

Show 10 entries Columns

	P	Status	Resolution	Key	Summary	Test Runs	Test Status
☐		OPEN	*Unresolved*	CALC-7932	simple integer addition		PASS
☐		OPEN	*Unresolved*	CALC-7933	negative integer adition		PASS
☐		OPEN	*Unresolved*	CALC-7934	sum of two positive numbers		PASS

Showing 1 to 3 of 3 entries First Previous 1 Next Last



Calculator / CALC-7935

As a user, I can multiply two numbers

Details

Type:	Story	Status:	OPEN (View Workflow)
Priority:	Major	Resolution:	Unresolved
Affects Version/s:	None	Fix Version/s:	None
Component/s:	None		
Labels:	None		
Requirement Status:	NOK		

Description

As a user, I can multiply two numbers

Test Coverage

TEST COVERAGE FOR THE FOLLOWING ANALYSIS SCOPE

Scope: Version; **Version:** None - latest execution; **Environment:** All Environments NOK

Show 10 entries Columns

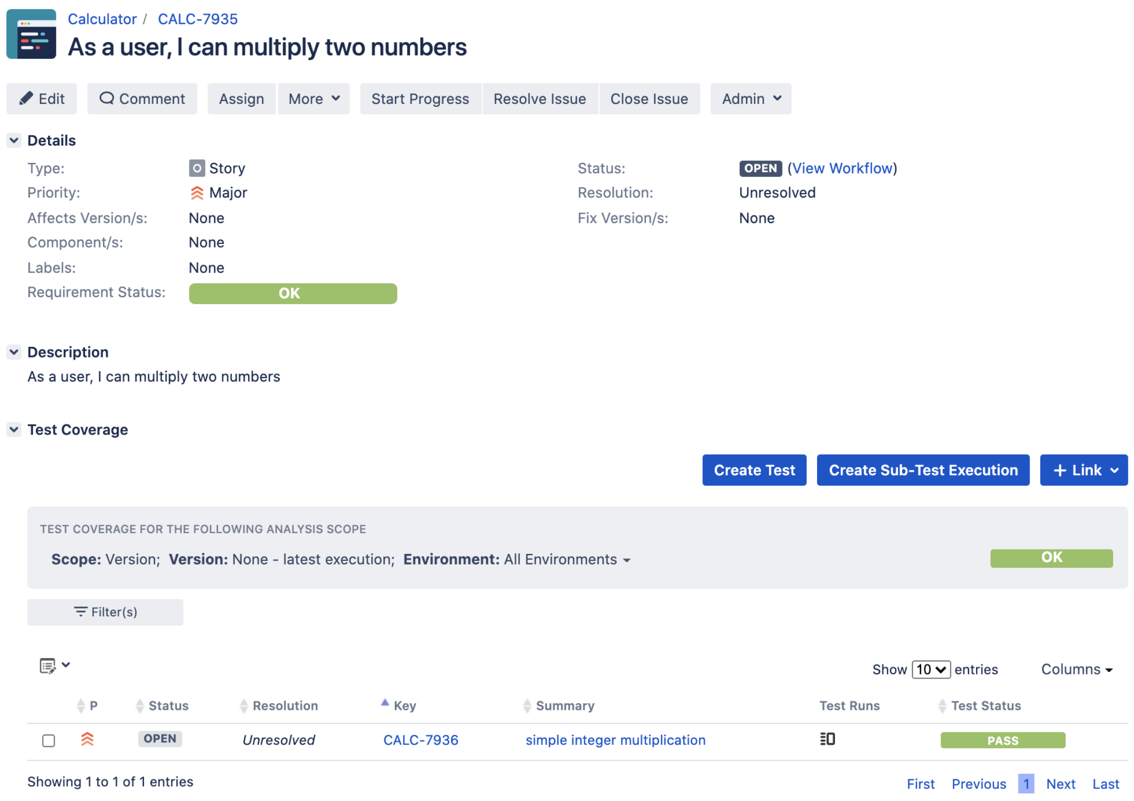
	P	Status	Resolution	Key	Summary	Test Runs	Test Status
☐		OPEN	*Unresolved*	CALC-7936	simple integer multiplication		FAIL

Showing 1 to 1 of 1 entries First Previous 1 Next Last

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;
}
```



Calculator / CALC-7935

As a user, I can multiply two numbers

Details

Type:	Story	Status:	OPEN (View Workflow)
Priority:	Major	Resolution:	Unresolved
Affects Version/s:	None	Fix Version/s:	None
Component/s:	None		
Labels:	None		
Requirement Status:	OK		

Description

As a user, I can multiply two numbers

Test Coverage

TEST COVERAGE FOR THE FOLLOWING ANALYSIS SCOPE

Scope: Version; **Version:** None - latest execution; **Environment:** All Environments — OK

	P	Status	Resolution	Key	Summary	Test Runs	Test Status
☐		OPEN	*Unresolved*	CALC-7936	simple integer multiplication		PASS

Showing 1 to 1 of 1 entries

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