Importing Cucumber Tests - REST

Importing Cucumber Tests

The following endpoint is provided to import a Cucumber .feature file or a zip file containing multiple .feature files. The files in the zip file may be in folders /subfolders.

Cucumber ".feature" file	**/rest/raven/1.0/import/feature**

Each .feature file will be processed and will try to find or create a Test/Pre-Condition inside the project given in the *projectKey* parameter. We use the following rules:

Tests:

1. try to find the Test by key, if found then update it; else...
2. try to find the Test having:
    a. a label with the original relative path of .feature (e.g. "core/sample_addition.feature)
    b. a label named "id:xxx", where xxx is some number (e.g. "id:1", "id:32"); this label comes from a scenario/scenario outline's tag
    c. no other label ending in ".feature"
3. try to find the Test having:
    a. a label with the original relative path of .feature (e.g. "core/sample_addition.feature)
    b. the same summary
    c. no other label ending in ".feature"
4. try to find the Test having:
    a. the same Summary
    b. no label ending in ".feature", except the one corresponding to the relative path of the Cucumber file being imported
5. create Test in that project and add a label with the relative path of the feature. The tags used in the scenario/scenario outline are also added as labels.

Pre-Conditions:

1. try to find the Pre-Condition by key, if found then update it; else...
2. try to find the Pre-Condition having both:
    a. a label with the original relative path of .feature (e.g. "core/sample_addition.feature)
    b. no other label ending in ".feature"
3. try to find the Pre-Condition having both:
    a. the same Summary
    b. no label ending in ".feature", except the one corresponding to the relative path of the Cucumber file being imported
4. create Pre-Condition in that project and add a label with the relative path of the feature

The mapping from the Scenario/Scenario Outline present in the .feature files to the Test issues in Jira would be as follows:

Scenario/Scenario Outline	Test in JIRA
name of the Scenario/Scenario Outline	"Summary" field
steps	"Scenario" field
tags of the Scenario/Scenario Outline	labels

The "Feature" section is not imported since the feature itself should exist previously as a Jira requirement issue (e.g., story).

The exception is the tags before the "Feature: " section; if a requirement issue is found for the specified key, then a "Tests" link is created between the Test and the requirement issue.

If the Cucumber feature has a background, a Pre-Condition issue will be created, if the issue key in the tag does not exist in JIRA, or updated, containing the information provided in that background.

If the background has no name, then the Summary of the Pre-Condition is going to be a string containing the keys of the Tests of that Cucumber feature (e.g. "Background for: CALC-1, CALC-2").

Below is an example of a .feature file containing a Scenario Outline and two Pre Conditions:

```
@REQ_CALC-889
Feature: As a user, I can calculate the sum of 2 numbers

        Background:
                #@PRECOND_TX-114
                Given that the calculator is turned on
                And the mode is to advanced
                #@PRECOND-TX-155
                Given that the calculator has been reset

        @UI @core
        Scenario Outline: Cucumber Test As a user, I can calculate the sum of 2 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     |
```

In this other hypotetical example, a feature file is shown containing one Background and two tests: one Scenario Outline and a Scenario. Each Scenario /Scenario Outline is identified by an internal "id:xxx", in order to uniquely identify the scenario within the feature file.

```
@REQ_CALC-1910
Feature: As a user, I can calculate the sum of two numbers

Background:
  Given I have a calculator
  And I have some fingers


@id:1 @fast
Scenario Outline: Cucumber Test As a user, I can calculate the 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         |


@id:2
Scenario: Cucumber Test As a user, I can calculate the sum of two negative numbers
  Given I have entered -1 into the calculator
  And I have entered -3 into the calculator
  When I press add
  Then the result should be -4 on the screen
```

Whenever this feature is imported for the first time, assuming it was imported from file named "features/addition.feature" in a zip file,

- a Pre-Condition with the summary "Background for: <issue_key_of_first_test>,<issue_key_of_second_test>" will be created;

- a Cucumber Test of type "Scenario Outline" will be created, having the summary "Cucumber Test As a user, I can calculate the sum of two positive numbers". The Test will have the label "fast";
- a Cucumber Test of type "Scenario" will be created, having the summary "Cucumber Test As a user, I can calculate the sum of two negative numbers";
- All previous Tests will be linked to the requirement CALC-1910

Whenever this same feature is imported for the second and following times, assuming it was imported from file named "features/addition.feature" in a zip file,

- a Pre-Condition with the summary "Background for: <issue_key_of_first_test>,<issue_key_of_second_test>" and the label "features/addition.feature" will be updated;
- a Cucumber Test having the labels "features/addition.feature" and "id:1" will be updated with the specification of the Scenario Outline; The Test will have the label "fast" (added if needed);
- a Cucumber Test having the labels "features/addition.feature" and "id:1" will be updated with the specification of the Scenario;
- All previous Tests will be linked to the requirement CALC-1910

This endpoint enables the control over newly-created or updated Test and Pre-Condition issues by allowing you to send, together with the features, two JSON files (one for Test and another for Pre-Condition issues) similar to the one Jira uses to create new issues. For more information about that JSON format, check the Jira documentation here.

Note that some field values in the JSON files will follow some rules:

- Issue Type: Overridden by the Test or Pre-Condition issue type;
- Summary: Overridden by the internally generated summary;
- Parent: This field is ignored;
- Scenario Type, Cucumber Scenario and Background: Not allowed to provide a value for these fields.

Request

QUERY PARAMETERS

parameter	type	description
projectKey	*String*	key of the project where the tests and pre-conditions are going to be created.

multipart/form-data:

***"file"** : a **MultipartFormParam** containing a **".feature" file** or a **ZIP file** to import.*

Example Request

curl -H "Content-Type: multipart/form-data" -u admin:admin -F "file=@1.feature" http://yourserver/rest/raven/1.0/import/feature?projectKey=DEV

curl -H "Content-Type: multipart/form-data" -u admin:admin -F "file=@features.zip" http://yourserver/rest/raven/1.0/import/feature?projectKey=DEV

curl -H "Content-Type: multipart/form-data" -u admin:admin -F "file=@features.zip" -F "testInfo=@testInfo.json" -F "preCondInfo=@precondInfo.json" http://yourserver/rest/raven/1.0/import/feature?projectKey=DEV

Responses

200 OK : **application/octet-stream** : Successful. The cucumber features where successfully imported to Jira.

Example Output

```
[
        {
                "id":"14400",
                "key":"DEV-915",
                "self":"http://localhost:8727/rest/api/2/issue/14400",
                "issueType": {
            "id": "10100",
            "name": "Test"
        }
        },
```

```
        {
                "id":"14401",
                "key":"DEV-916",
                "self":"http://localhost:8727/rest/api/2/issue/14401",
                "issueType": {
            "id": "10103",
            "name": "Pre-Condition"
        }
        }
]
```

400 BAD_REQUEST : **text/plain:** Returns the error.

401 UNAUTHORIZED : **text/plain** : The Xray license is not valid.

500 INTERNAL SERVER ERROR : **text/plain** : An internal error occurred when generating the *feature* file(s).