Testing using Cucumber in Perl

Overview

In this tutorial, we will create some tests in Cucumber for Perl.

The test (specification) is initially created in Jira as a Cucumber Test and afterwards, it is exported using the UI or the REST API.

Requirements

- Install Test::BDD::Cucumber module
- Clone of the Github repository "test-bdd-cucumber-perl"

Description

We will use the code from the Github repository "test-bdd-cucumber-perl" with slight changes.

The first step is to create two Cucumber Tests, one "Scenario" and one "Scenario Outline", in Jira. The specification would be similar to the calculator example provided in the Github repository.

After creating the Tests in Jira and associating them with requirements, etc., you can export the specifications of the test to a Cucumber .feature file via the REST API or the **Export to Cucumber** UI action from within the Test Execution issue.

The created file will be similar to the original, but will contain the references to the Test issue key and the covered requirement issue key.

With the code below, you'll create a simple feature file. Note that we introduced a bug in the Scenario Outline specification on purpose (i.e., "6/3=3").

features/basic_functions.feature

```
@CALC-xx
Feature: Basic Calculator Functions
  In order to check I've written the Calculator class correctly
  As a developer I want to check some basic operations
  So that I can have confidence in my Calculator class.

  @CALC-885 @CALC-886
  Scenario: First Key Press on the Display
    Given a new Calculator object
    And having pressed 1
    Then the display should show 1

  @CALC-886 @CALC-887
  Scenario Outline: Basic arithmetic
    Given a new Calculator object
    And having keyed <first>
    And having keyed <operator>
    And having keyed <second>
    And having pressed =
    Then the display should show <result>
    Examples:
      | first | operator | second | result |
      | 5.0   | +        | 5.0    | 10     |
      | 6     | /        | 3      | 3      |
      | 10    | *        | 7.550  | 75.5   |
      | 3     | -        | 10     | -7     |
```

Please check if the Scenario Outline is specified using the "Scenario Outline" keywords.

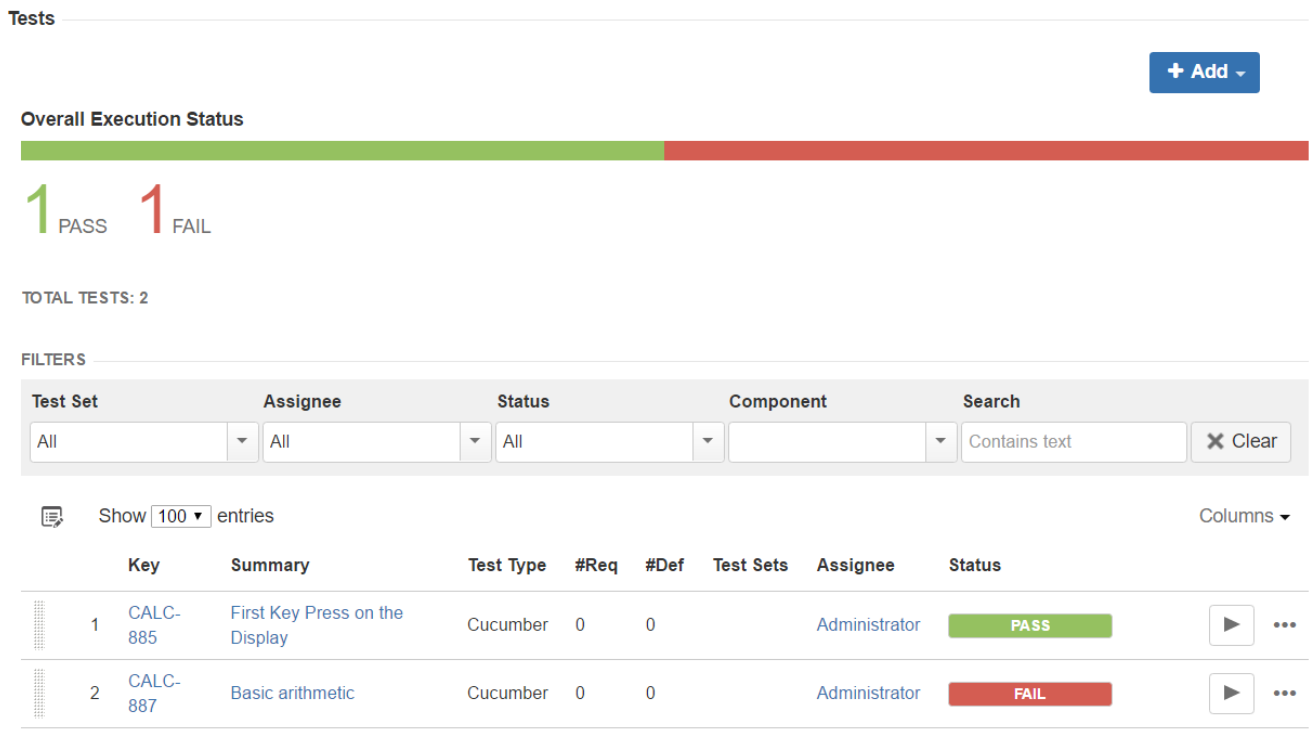
The steps are implemented in Perl code.

lib/Calculator.pm

```
package    # hide from PAUSE indexer
  Calculator;
use strict;
use warnings;
use Moose;
has 'left'     => ( is => 'rw', isa => 'Num', default => 0 );
has 'right'    => ( is => 'rw', isa => 'Str', default => '' );
has 'operator' => ( is => 'rw', isa => 'Str', default => '+' );
has 'display' => ( is => 'rw', isa => 'Str', default => '0' );
has 'equals'  => ( is => 'rw', isa => 'Str', default => '' );
sub key_in {
    my ( $self, $seq ) = @_;
    my @possible = grep { /\S/ } split( //, $seq );
    $self->press($_) for @possible;
}
sub press {
    my ( $self, $key ) = @_;
    # Numbers
    $self->digit($1) if $key =~ m/^([\d\.])$/;
    # Operators
    $self->key_operator($1) if $key =~ m/^([\+\-\/\*])$/;
    # Equals
    $self->equalsign if $key eq '=';
    # Clear
    $self->clear if $key eq 'C';
}
sub clear {
    my $self = shift;
    $self->left(0);
    $self->right('');
    $self->operator('+');
    $self->display('0');
    $self->equals('');
}
sub equalsign {
    my $self = shift;
    $self->key_operator('+');
    my $result = $self->left;
    $self->clear();
    $self->equals($result);
    $self->display($result);
}
sub digit {
    my ( $self, $digit ) = @_;
    # Deal with decimal weirdness
    if ( $digit eq '.' ) {
        return if $self->right =~ m/\./;
        $digit = '0.' unless length( $self->right );
    }
    $self->right( $self->right . $digit );
    $self->display( $self->right );
}
sub key_operator {
    my ( $self, $operator ) = @_;
    my $cmd =
        $self->left
      . $self->operator
      . (
        length( $self->right )
        ? $self->right
        : ( length( $self->equals ) ? $self->equals : '0' )
      );
    $self->right('');
    $self->equals('');
    $self->left( ( eval $cmd ) + 0 );
    $self->display( $self->left );
    $self->operator($operator);
}
1;
```

After running the tests and generating the Cucumber JSON report (e.g., data.json), it can be imported to Xray via the REST API or the **Import Execution Results** action within the Test Execution.

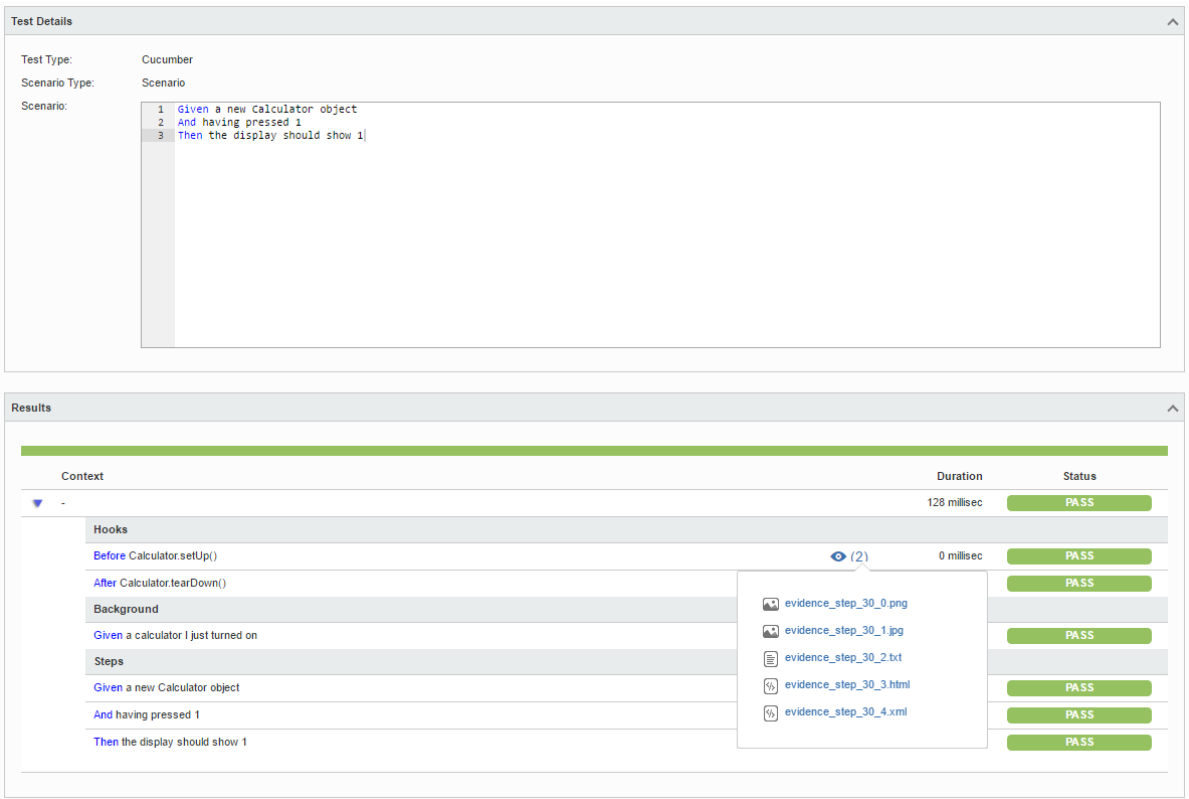
```
pherkin -I lib -o JSON features/basic_functions.feature > data.json
```



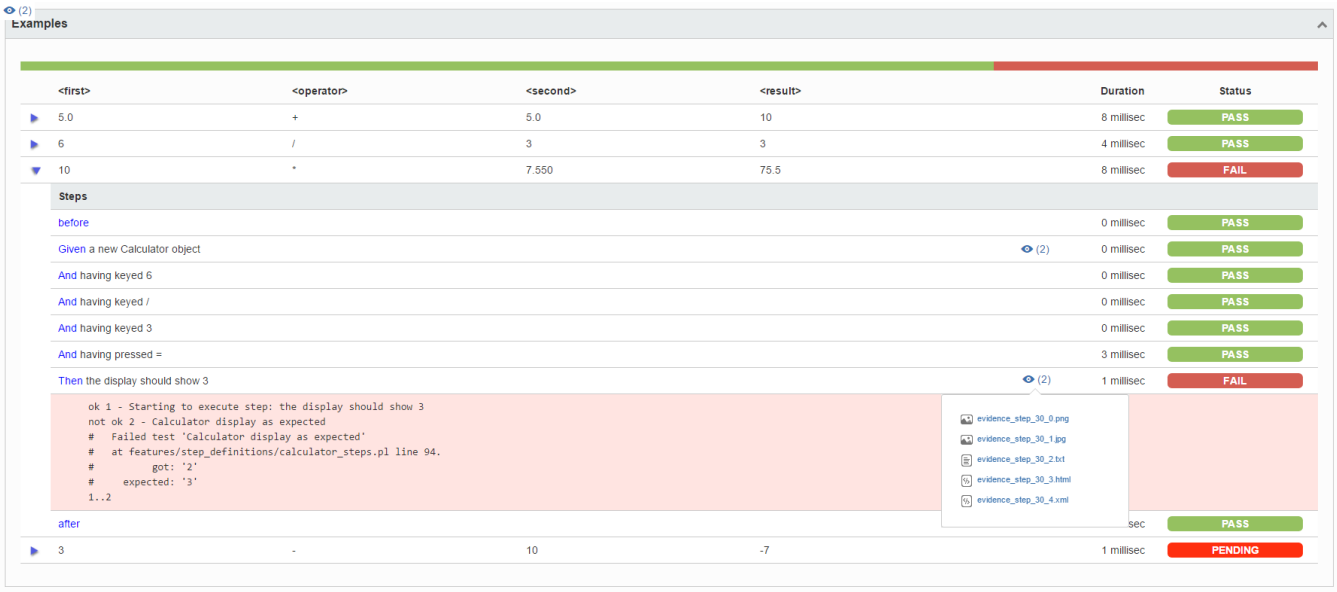
The execution screen details will provide information on the test run result.

> The Cucumber Scenarios Example/Result details (i.e., **Hooks, Backgrounds** and **Steps**) are only available for executions done in Xray v2.2.0 and above.

For a Cucumber Scenario Test:



For a Cucumber Scenario Outline Test:



The icon (2) represents the evidences ("embeddings") for each **Hook, Background and Step**, but is only available for executions done in Xray v2.3.0 and above.

Learn more

Please see Testing in BDD with Gherkin based frameworks (e.g. Cucumber) for an overview on how to use Cucumber Tests with Xray.

References

- https://github.com/pjlsergeant/test-bdd-cucumber-perl/tree/master/examples/calculator
- http://search.cpan.org/~sargie/Test-BDD-Cucumber/
- Automated Tests (Import/Export)
- Exporting Cucumber Tests - REST