BehaveToolkit Documentation

Release 0.2.0

Mitchel Cabuloy

June 29, 2015

Contents

BehaveToolkit provides integration between Sublime Text 3 and Behave.

CHAPTER 1

Features:

- Run specific scenarios
- Generate step functions
- Go to step function
- Highlight unimplemented steps

CHAPTER 2

Contents:

2.1 Installation

2.1.1 Package Control

1. Install the Sublime Text Package Control plugin if you don't have it already.
2. Open the command palette and start typing `Package Control: Install Package`.
3. Enter `BehaveToolkit`.

2.1.2 Via Git

You can also clone the repo directly to your Packages folder if you so wish:

```
# on a Mac
$ cd "$HOME/Library/Application Support/Sublime Text 3/Packages"
# on Linux
$ cd $HOME/.config/sublime-text-3/Packages
# on Windows (PowerShell)
$ cd "$env:appdata\Sublime Text 3\Packages\"

$ git clone https://github.com/mixxorz/BehaveToolkit
```

2.2 Getting Started

2.2.1 Setup

Make sure you have `behave` and `BehaveToolkit` installed.

Let's create our project structure

```
myproject/
  features/
    steps/
    myfeature.feature
```

Open the project directory in sublime:

```
$ subl myproject/
```

Then, let's add a simple scenario.

```
# myproject/features/myfeature.feature
Feature: My feature

  Scenario: Addition between two numbers
    Given the first number is "1"
      And the second number is "1"
     When I add them together
     Then I should get "2"
```

Once you hit save, you should see undefined steps get highlighted.

2.2.2 Generating step functions

Let's try to generate some step functions. Place the cursor over a step, open up the command palette and select `Behave: Generate Step Function`. Choose "Create a new file". You should see a new file open with the following content.

```
from behave import given, when, then


@given(u'the first number is "1"')
def the_first_number_is_1(context):
    raise NotImplementedError(u'STEP: the first number is "1"')
```

Let's save this file under `myproject/features/steps/steps.py`. When you go back to `myfeature.feature`, you should see the step be cleared.

2.2.3 Generating missing step functions

Let's generate the rest of the steps. With the feature file open, let's open up the command palette and select `Behave: Generate Missing Step Functions`.

You're now given a choice of either creating a new file, or an existing step file. Let's choose `steps.py`.

You should now see the generated step functions pasted inside `steps.py`.

```
# myproject/features/steps/steps.py
from behave import given, when, then


@given(u'the first number is "1"')
def the_first_number_is_1(context):
    raise NotImplementedError(u'STEP: the first number is "1"')


@then(u'I should get "2"')
def i_should_get_2(context):
    raise NotImplementedError(u'STEP: I should get "2"')


@given(u'the second number is "1"')
def the_second_number_is_1(context):
    raise NotImplementedError(u'STEP: the second number is "1"')
```

```
@when(u'I add them together')
def i_add_them_together(context):
    raise NotImplementedError(u'STEP: I add them together')
```

Once you hit save and go back to the feature file, you should see that all steps are now cleared.

2.2.4 Running behave

In lieu with the spirit of TDD, let's watch the tests fail.

Place the cursor over a scenario, open the command palette and select `Behave: Run Behave`. You should see the test failing.

```
Feature: My feature # features/myfeature.feature:1

  Scenario: Addition between two numbers  # features/myfeature.feature:3
    Given the first number is "1"         # features/steps/steps.py:14
      Traceback (most recent call last):
        File "/Users/mixxorz/.pyenv/versions/2.7.10/lib/python2.7/site-packages/behave/model.py", lin
          match.run(runner.context)
        File "/Users/mixxorz/.pyenv/versions/2.7.10/lib/python2.7/site-packages/behave/model.py", lin
          self.func(context, *args, **kwargs)
        File "features/steps/steps.py", line 16, in the_first_number_is_1
          raise NotImplementedError(u'STEP: the first number is "1"')
      NotImplementedError: STEP: the first number is "1"

    And the second number is "1"          # None
    When I add them together              # None
    Then I should get "2"                 # None


Failing scenarios:
  features/myfeature.feature:3  Addition between two numbers

0 features passed, 1 failed, 0 skipped
0 scenarios passed, 1 failed, 0 skipped
0 steps passed, 1 failed, 3 skipped, 0 undefined
Took 0m0.000s
```

Let's implement the tests.

```
# myproject/features/steps/steps.py
from behave import given, when, then


@given(u'the first number is "{num:d}"')
def the_first_number_is_1(context, num):
    context._first_num = num


@then(u'I should get "{num:d}"')
def i_should_get_2(context, num):
    assert num == context._sum


@given(u'the second number is "{num:d}"')
```

```
def the_second_number_is_1(context, num):
    context._second_num = num


@when(u'I add them together')
def i_add_them_together(context):
    context._sum = context._first_num + context._second_num
```

When you run behave, the tests should now pass:

```
Feature: My feature # features/myfeature.feature:1

  Scenario: Addition between two numbers  # features/myfeature.feature:3
    Given the first number is "1"         # features/steps/steps.py:14
    And the second number is "1"          # features/steps/steps.py:19
    When I add them together              # features/steps/steps.py:4
    Then I should get "2"                 # features/steps/steps.py:9

1 feature passed, 0 failed, 0 skipped
1 scenario passed, 0 failed, 0 skipped
4 steps passed, 0 failed, 0 skipped, 0 undefined
Took 0m0.001s
```

Specific scenarios

If you want, you can run only specific scenarios. Let's add a new scenario, with different numbers this time.

```
Scenario: Addition between different numbers
  Given the first number is "2"
    And the second number is "3"
   When I add them together
   Then I should get "5"
```

Place the cursor over the second scenario. When you run behave, it will only run the scenario under your cursor.

Run all scenarios in the current feature

If you want to run all scenarios in the current feature, just place your cursor on the first line of the feature file, and run behave.

Run all features

If you want to run all scenarios in all features, just run behave without a feautre file open.

2.3 Commands

2.3.1 Generate Step Function

Generates step functions for steps under the cursor(s).

- Command: `bt_generate_step_function`

Gives you the ability to quickly generate functions for steps. Place the cursor under a step (e.g. `Given the first number is "1"`), open the command palette and select `Behave: Generate Step Function`. You will be prompted whether you want to create a new file for this step function, or to paste it to an existing steps file.

You can also generate multiple step functions at once by placing cursors on multiple steps.

2.3.2 Generate Missing Step Functions

Generates step functions for missing steps.

- Command: `bt_generate_missing_step_functions`

This command is just a convenience command to generate step functions for all unimplemented steps in the open feature file.

2.3.3 Go To Step Function

Navigate to the step function of the step under the cursor.

- Command: `bt_go_to_step_function`

Does what it says on the tin.

2.3.4 Run behave

Runs behave within the current context.

- Command: `bt_run_behave`

This command is activated by selecting `Behave: Run behave` on the command palette.

- If the cursor is within a Scenario definition, the command will only run that scenario.
- If the cursor is above the first Scenario, the command will run all Scenarios in that Feature.
- If a feature file isn't visible, the command will run all Scenarios in all Features
- If there are multiple cursors, the command will run all Scenarios that are under the cursors.

2.4 Keybindings

These are the default key bindings:

- Go To Step Function
  - Linux `Super+G`
  - OSX `Alt+G`
  - Windows `Ctrl+Alt+Shift+G`
- Generate Step Function
  - Linux `Super+S`
  - OSX `Alt+S`
  - Windows `Ctrl+Alt+Shift+S`
- Generate Missing Step Functions

  - Linux `Super+F`
  - OSX `Alt+F`
  - Windows `Ctrl+Alt+Shift+F`
- Run behave
  - Linux `Super+T`
  - OSX `Alt+T`
  - Windows `Ctrl+Alt+Shift+T`

2.5 Snippets

step

Snippet for step functions

```
@given(u'step name')
def step_impl(context):
    raise NotImplementedError(u'STEP: step name')
```

2.6 Configuration

2.6.1 behave_command

The command used to run behave.

By default, `BehaveToolkit` tries to find `behave` in your environment. Change this setting if you want to specifically set how `behave` gets executed.

Open `Preferences > Package Settings > BehaveToolkit > Settings - User` and paste the following (for example):

```
{
  "behave_command": ["/Users/mixxorz/.virtualenvs/myproject/bin/behave"]
}
```

You can override the setting in your the sublime-project file of your project too. This will take priority over the global settings.

myproject.sublime-project

```
{
  "folders":
  [
    {
      "path": "/Users/mixxorz/Projects/myproject"
    }
  ],
  "settings":
  {
    "behave_command": ["/Users/mixxorz/.virtualenvs/myproject/bin/behave"]
  }
}
```

If you're using behave-django, another project by me which integrates `behave` and `Django`, you can configure `behave_command` like this.

```
{
  "behave_command": [
    "/Users/mixxorz/.virtualenvs/myproject/bin/python",
    "/Users/mixxorz/Projects/myproject/manage.py",
    "behave"
  ]
}
```

2.7 Troubleshooting

One of the popular reasons for `BehaveToolkit` not working is `behave` not working. A good rule of thumb is that, if `behave` works, then `BehaveToolkit` should work.

Project root directory

Sublime should be opened with the directory where you would run behave. If you followed the Getting Started guide, you would notice that we opened Sublime on the `myproject/` folder. This is also the directory where we would run `behave`. Sure enough, if we run `behave` in this folder, it will work.

Behave can't be found

There may be instances where the `behave` binary cannot be found on your system. In which case, you should look at Configuration.

Other issues

If you're still having problems, don't hesitate to ask questions by opening an issue on our GitHub issues page.

CHAPTER 3

Indices and tables

- genindex
- search