
Python Finite State Machine

Release 2.0.0

Aug 26, 2018

Contents

| | | |
|----------|---|-----------|
| 1 | Overview | 1 |
| 1.1 | Usage | 1 |
| 1.2 | Installation | 3 |
| 1.3 | Django integration | 3 |
| 1.4 | Documentation | 3 |
| 1.5 | Development | 3 |
| 2 | Installation | 5 |
| 3 | Usage | 7 |
| 4 | Reference | 9 |
| 4.1 | fsm package | 9 |
| 5 | Contributing | 13 |
| 5.1 | Bug reports | 13 |
| 5.2 | Documentation improvements | 13 |
| 5.3 | Feature requests and feedback | 13 |
| 5.4 | Development | 14 |
| 6 | Authors | 15 |
| 7 | Changelog | 17 |
| 7.1 | 2.0.0 (2018-08-26) | 17 |
| 7.2 | 1.0.0 (2018-07-03) | 17 |
| 7.3 | 0.1.3 (2017-15-09) | 17 |
| 7.4 | 0.1.0 (2016-04-18) | 17 |
| 8 | Indices and tables | 19 |
| | Python Module Index | 21 |

CHAPTER 1

Overview

| | |
|---------|--|
| docs | |
| tests | |
| package | |

Minimal state machine

- Free software: BSD license

Contents

- *Overview*
 - *Usage*
 - *Installation*
 - *Django integration*
 - *Documentation*
 - *Development*

1.1 Usage

```
import fsm

class MyTasks(fsm.FiniteStateMachineMixin):
```

(continues on next page)

(continued from previous page)

```

"""An example to test the state machine.

Contains transitions to everywhere, nowhere and specific states.
"""

state_machine = {
    'created': '__all__',
    'pending': ('running',),
    'running': ('success', 'failed'),
    'success': None,
    'failed': ('retry',),
    'retry': ('pending', 'retry'),
}

def __init__(self, state):
    """Initialize setting a state."""
    self.state = state

def on_before_pending(self):
    print("I'm going to a pending state")

```

```

In [4]: m = MyTasks(state='created')

In [5]: m.change_state('pending')
I'm going to a pending state
Out[5]: 'pending'

In [6]: m.change_state('failed')
-----
InvalidTransition                                Traceback (most recent call last)
<ipython-input-6-71d2461eee74> in <module>()
----> 1 m.change_state('failed')

~/pyfsm/src/fsm/fsm.py in change_state(self, next_state, **kwargs)
    90         msg = "The transition from {0} to {1} is not valid".
->format(previous_state,
    91                                             next_
->state)
----> 92         raise InvalidTransition(msg)
    93
    94         name = 'pre_{0}'.format(next_state)

InvalidTransition: The transition from pending to failed is not valid

```

There are hooks that can be included before a state transition happens and after.

fsm will look for these functions

```

pre_<state_name>
post_<state_name>

```

And will give them any extra argument given to change_state

E.g:

Running `m.change_state('pending', name='john')` will trigger `pre_pending(name='john')`

1.2 Installation

```
pip install fsm.py
```

1.3 Django integration

```
import fsm
from django.db import models

class MyModel(models.Model, fsm.FiniteStateMachineMixin):
    """An example to test the state machine.

    Contains transitions to everywhere, nowhere and specific states.
    """

    CHOICES = (
        ('created', 'CREATED'),
        ('pending', 'PENDING'),
        ('running', 'RUNNING'),
        ('success', 'SUCCESS'),
        ('failed', 'FAILED'),
        ('retry', 'RETRY'),
    )

    state_machine = {
        'created': '__all__',
        'pending': ('running',),
        'running': ('success', 'failed'),
        'success': None,
        'failed': ('retry',),
        'retry': ('pending', 'retry'),
    }

    state = models.CharField(max_length=30, choices=CHOICES, default='created')

    def on_change_state(self, previous_state, next_state, **kwargs):
        self.save()
```

1.4 Documentation

<https://pyfsm.readthedocs.org/>

1.5 Development

To run the tests run:

```
tox
```

Note, to combine the coverage data from all the tox environments run:

| | |
|---------|--|
| Windows | <pre>set PYTEST_ADDOPTS=--cov-append tox</pre> |
| Other | <pre>PYTEST_ADDOPTS=--cov-append tox</pre> |

CHAPTER 2

Installation

At the command line:

```
pip install fsm.py
```


To use Python Finite State Machine in a project:

```
import fsm
```

Then add the Mixin to any class where the state machine is required.

```
class Foo(fsm.FiniteStateMachineMixin):  
  
    state_machine = {  
        'my_first_state': '__all__',  
        'my_state': ('my_second_state',),  
        'my_second_state': ('my_state', 'my_second_state', 'last_state'),  
        'last_state': None  
    }  
  
    state = 'my_first_state'
```

Instantiate the class and use it. Remember that in order to work as intended, `change_state` must be used to transition from one state to the other.

```
>>> foo = Foo()  
  
>>> foo.current_state()  
'my_first_state'  
  
>>> foo.change_state('my_state')  
'my_state'  
  
>>> foo.current_state()  
'my_state'  
  
>>> foo.can_change('last_state')  
False
```

(continues on next page)

(continued from previous page)

```
>>> foo.get_valid_transitions()  
('my_second_state',)
```

You can also use `BaseFiniteStateMachineMixin` for more flexibility. Implementing `current_state` and `set_state` is required. Doing this allows using more complex behavior, but it is **not recommended**.

4.1 fsm package

4.1.1 Submodules

4.1.2 fsm module

class fsm.InvalidTransition

Bases: exceptions.Exception

Moving from an state to another is not possible.

class fsm.FiniteStateMachineMixin

Bases: fsm.fsm.BaseFiniteStateMachineMixin

A drop in implementation. Ready to be used.

Replace FIELD_NAME in order to automatically retrieve or set from a different field.

In order to use with django, just add a field `state` or as defined in `FIELD_NAME` and remember to use `change_state` instead of simply assigning it

FIELD_NAME = 'state'

current_state()

set_state(*state*)

class fsm.BaseFiniteStateMachineMixin

Base Mixin to add a `state_machine` behavior.

Represents the state machine for the object.

The states and transitions should be specified in the following way:

```
state_machine = {
    'some_state': '__all__'
    'another_state': ('some_state', 'one_more_state')
    'one_more_state': None
}
```

Requires the implementation of `current_state` and `set_state`

can_change (*next_state*)

Validates if the `next_state` can be executed or not.

It uses the `state_machine` attribute in the class.

change_state (*next_state*, ***kwargs*)

Performs a transition from current state to the given next state if possible.

Callbacks will be executed before and after changing the state. Specific state callbacks will also be called if they are implemented in the subclass.

Parameters `next_state` (*str or int*) – where the state must go

Returns new state.

Return type `str` or `int`

Raises `InvalidTransition` – If transitioning is not possible

current_state ()

Returns the current state in the FSM.

get_valid_transitions ()

Return possible states to whom a product can transition.

Returns valid transitions

Return type `tuple` or `list`

on_before_change_state (*previous_state*, *next_state*, ***kwargs*)

Called before a state changes.

Parameters

- **previous_state** (*str or int*) – type depends on the definition of the states.

- **next_state** (*str or int*) – type depends on the definition of the states.

on_change_state (*previous_state*, *next_state*, ***kwargs*)

Called after a state changes.

Parameters

- **previous_state** (*str or int*) – type depends on the definition of the states.

- **next_state** (*str or int*) – type depends on the definition of the states.

set_state (*state*)

Update the internal state field.

Parameters `state` (*str or int*) – type depends on the definition of the states.

state_machine = `None`

4.1.3 Module contents

Initializing package.

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

5.1 Bug reports

When **reporting a bug** please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.2 Documentation improvements

Python Finite State Machine could always use more documentation, whether as part of the official Python Finite State Machine docs, in docstrings, or even on the web in blog posts, articles, and such.

5.3 Feature requests and feedback

The best way to send feedback is to file an issue at <https://github.com/Woile/pyfsm/issues>.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that code contributions are welcome :)

5.4 Development

To set up *pyfsm* for local development:

1. Fork *pyfsm* (look for the “Fork” button).
2. Clone your fork locally:

```
git clone git@github.com:your_name_here/pyfsm.git
```

3. Create a branch for local development:

```
git checkout -b name-of-your-bugfix-or-feature
```

Now you can make your changes locally.

4. When you’re done making changes, run all the checks, doc builder and spell checker with *tox* one command:

```
tox
```

5. Commit your changes and push your branch to GitHub:

```
git add .  
git commit -m "Your detailed description of your changes."  
git push origin name-of-your-bugfix-or-feature
```

6. Submit a pull request through the GitHub website.

5.4.1 Pull Request Guidelines

If you need some code review or feedback while you’re developing the code just make the pull request.

For merging, you should:

1. Include passing tests (run *tox*)¹.
2. Update documentation when there’s new API, functionality etc.
3. Add a note to *CHANGELOG.rst* about the changes.
4. Add yourself to *AUTHORS.rst*.

5.4.2 Tips

To run a subset of tests:

```
tox -e envname -- py.test -k test_myfeature
```

To run all the test environments in *parallel* (you need to *pip install detox*):

```
detox
```

¹ If you don’t have all the necessary python versions available locally you can rely on Travis - it will run the tests for each change you add in the pull request.
It will be slower though ...

CHAPTER 6

Authors

- Santiago Fraire Willemoes - <https://github.com/woile>

7.1 2.0.0 (2018-08-26)

- BREAKING: Simpler implementation
- Docs and README updated
- Usage of pyproject.toml

7.2 1.0.0 (2018-07-03)

- Renamed hooks to `pre_<state_name>` and `post_<state_name>`

7.3 0.1.3 (2017-15-09)

- Updated docs
- Corrections to code
- ci updated

7.4 0.1.0 (2016-04-18)

- First release on PyPI.

CHAPTER 8

Indices and tables

- `genindex`
- `modindex`
- `search`

f

`fsm`, 11

B

BaseFiniteStateMachineMixin (class in fsm), 9

C

can_change() (fsm.BaseFiniteStateMachineMixin method), 10

change_state() (fsm.BaseFiniteStateMachineMixin method), 10

current_state() (fsm.BaseFiniteStateMachineMixin method), 10

current_state() (fsm.FiniteStateMachineMixin method), 9

F

FIELD_NAME (fsm.FiniteStateMachineMixin attribute), 9

FiniteStateMachineMixin (class in fsm), 9

fsm (module), 11

G

get_valid_transitions() (fsm.BaseFiniteStateMachineMixin method), 10

I

InvalidTransition (class in fsm), 9

O

on_before_change_state() (fsm.BaseFiniteStateMachineMixin method), 10

on_change_state() (fsm.BaseFiniteStateMachineMixin method), 10

S

set_state() (fsm.BaseFiniteStateMachineMixin method), 10

set_state() (fsm.FiniteStateMachineMixin method), 9

state_machine (fsm.BaseFiniteStateMachineMixin attribute), 10