Python State Machine Documentation

Release 0.7.1

Fernando Macedo

Jan 17, 2019

Contents

1 Python State Machine **3**
1.1 Getting started . . . 3

2 Installation **9**
2.1 Stable release . . . 9
2.2 From sources . . . 9

3 Usage **11**

4 Contributing **13**
4.1 Types of Contributions . . . 13
4.2 Get Started! . . . 14
4.3 Pull Request Guidelines . . . 15
4.4 Tips . . . 15

5 Credits **17**
5.1 Development Lead . . . 17
5.2 Contributors . . . 17
5.3 Credits . . . 17

6 History **19**
6.1 0.7.0 (2018-04-01) . . . 19
6.2 0.6.2 (2017-08-25) . . . 19
6.3 0.6.1 (2017-08-25) . . . 19
6.4 0.6.0 (2017-08-25) . . . 19
6.5 0.5.1 (2017-07-24) . . . 19
6.6 0.5.0 (2017-07-13) . . . 20
6.7 0.4.2 (2017-07-10) . . . 20
6.8 0.3.0 (2017-03-22) . . . 20
6.9 0.2.0 (2017-03-22) . . . 20
6.10 0.1.0 (2017-03-21) . . . 20

7 Indices and tables **21**

Contents:

CHAPTER 1

Python State Machine

Python finite-state machines made easy.

- Free software: MIT license
- Documentation: https://python-statemachine.readthedocs.io.

1.1 Getting started

To install Python State Machine, run this command in your terminal:

```
$ pip install python-statemachine
```

Define your state machine:

```
from statemachine import StateMachine, State

class TrafficLightMachine(StateMachine):
    green = State('Green', initial=True)
    yellow = State('Yellow')
    red = State('Red')

    slowdown = green.to(yellow)
    stop = yellow.to(red)
    go = red.to(green)
```

You can now create an instance:

```
>>> traffic_light = TrafficLightMachine()
```

And inspect about the current state:

```
>>> traffic_light.current_state
State('Green', identifier='green', value='green', initial=True)
>>> traffic_light.current_state == TrafficLightMachine.green == traffic_light.green
True
```

For each state, there's a dinamically created property in the form `is_<state.identifier>`, that returns `True` if the current status matches the query:

```
>>> traffic_light.is_green
True
>>> traffic_light.is_yellow
False
>>> traffic_light.is_red
False
```

Query about metadata:

```
>>> [s.identifier for s in m.states]
['green', 'red', 'yellow']
>>> [t.identifier for t in m.transitions]
['go', 'slowdown', 'stop']
```

Call a transition:

```
>>> traffic_light.slowdown()
```

And check for the current status:

```
>>> traffic_light.current_state
State('Yellow', identifier='yellow', value='yellow', initial=False)
>>> traffic_light.is_yellow
True
```

You can't run a transition from an invalid state:

```
>>> traffic_light.is_yellow
True
>>> traffic_light.slowdown()
Traceback (most recent call last):
...
LookupError: Can't slowdown when in Yellow.
```

You can also trigger events in an alternative way, calling the `run(<transition.identificer>)` method:

```
>>> traffic_light.is_yellow
True
>>> traffic_light.run('stop')
>>> traffic_light.is_red
True
```

A state machine can be instantiated with an initial value:

```
>>> machine = TrafficLightMachine(start_value='red')
>>> traffic_light.is_red
True
```

1.1.1 Models

If you need to persist the current state on another object, or you're using the state machine to control the flow of another object, you can pass this object to the `StateMachine` constructor:

```
>>> class MyModel(object):
...     def __init__(self, state):
...         self.state = state
...
>>> obj = MyModel(state='red')
>>> traffic_light = TrafficLightMachine(obj)
>>> traffic_light.is_red
True
>>> obj.state
'red'
>>> obj.state = 'green'
>>> traffic_light.is_green
True
>>> traffic_light.slowdown()
>>> obj.state
'yellow'
>>> traffic_light.is_yellow
True
```

1.1.2 Callbacks

Callbacks when running events:

```
from statemachine import StateMachine, State

class TrafficLightMachine(StateMachine):
    "A traffic light machine"
    green = State('Green', initial=True)
    yellow = State('Yellow')
    red = State('Red')

    slowdown = green.to(yellow)
    stop = yellow.to(red)
    go = red.to(green)

    def on_slowdown(self):
        print('Calma, lá!')

    def on_stop(self):
        print('Parou.')

    def on_go(self):
        print('Valendo!')
```

```
>>> stm = TrafficLightMachine()
>>> stm.slowdown()
Calma, lá!
>>> stm.stop()
Parou.
>>> stm.go()
Valendo!
```

Or when entering/exiting states:

```
from statemachine import StateMachine, State

class TrafficLightMachine(StateMachine):
    "A traffic light machine"
    green = State('Green', initial=True)
    yellow = State('Yellow')
    red = State('Red')

    cycle = green.to(yellow) | yellow.to(red) | red.to(green)

    def on_enter_green(self):
        print('Valendo!')

    def on_enter_yellow(self):
        print('Calma, lá!')

    def on_enter_red(self):
        print('Parou.')
```

```
>>> stm = TrafficLightMachine()
>>> stm.cycle()
Calma, lá!
>>> stm.cycle()
Parou.
>>> stm.cycle()
Valendo!
```

1.1.3 Mixins

Your model can inherited from a custom mixin to auto-instantiate a state machine.

```
class CampaignMachineWithKeys(StateMachine):
    "A workflow machine"
    draft = State('Draft', initial=True, value=1)
    producing = State('Being produced', value=2)
    closed = State('Closed', value=3)

    add_job = draft.to.itself() | producing.to.itself()
    produce = draft.to(producing)
    deliver = producing.to(closed)


class MyModel(MachineMixin):
    state_machine_name = 'CampaignMachine'

    def __init__(self, **kwargs):
        for k, v in kwargs.items():
            setattr(self, k, v)
        super(MyModel, self).__init__()

    def __repr__(self):
        return "{}({!r})".format(type(self).__name__, self.__dict__)
```

(continues on next page)

(continued from previous page)

```
model = MyModel(state='draft')
assert isinstance(model.statemachine, campaign_machine)
assert model.state == 'draft'
assert model.statemachine.current_state == model.statemachine.draft
```

CHAPTER 2

Installation

2.1 Stable release

To install Python State Machine, run this command in your terminal:

```
$ pip install python-statemachine
```

This is the preferred method to install Python State Machine, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for Python State Machine can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/fgmacedo/python-statemachine
```

Or download the tarball:

```
$ curl  -OL https://github.com/fgmacedo/python-statemachine/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

To use Python State Machine in a project:

```
import statemachine
```

CHAPTER 4

Contributing

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

You can contribute in many ways:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/fgmacedo/python-statemachine/issues.

If you are 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.

4.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

4.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

4.1.4 Write Documentation

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

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/fgmacedo/python-statemachine/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 contributions are welcome :)

4.2 Get Started!

Ready to contribute? Here's how to set up *python-statemachine* for local development.

1. Fork the *python-statemachine* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/python-statemachine.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv python-statemachine
$ cd python-statemachine/
$ python setup.py develop
```

4. Create a branch for local development:

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

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 statemachine tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. 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
```

7. Submit a pull request through the GitHub website.

4.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.7, 3.3, 3.4 and 3.5. Check https://travis-ci.org/fgmacedo/python-statemachine/pull_requests and make sure that the tests pass for all supported Python versions.

4.4 Tips

To run a subset of tests:

```
$ py.test tests.test_statemachine
```

CHAPTER 5

Credits

5.1 Development Lead

- Fernando Macedo <fgmacedo@gmail.com>

5.2 Contributors

- Guilherme Nepomuceno <piercio@loggi.com>

5.3 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 6

History

6.1 0.7.0 (2018-04-01)

- New event callbacks: *on_enter_<state>* and *on_exit_<state>*.

6.2 0.6.2 (2017-08-25)

- Fix README.

6.3 0.6.1 (2017-08-25)

- Fix deploy issues.

6.4 0.6.0 (2017-08-25)

- Auto-discovering *statemachine/statemachines* under a Django project when they are requested using the mixin/registry feature.

6.5 0.5.1 (2017-07-24)

- Fix bug on `CombinedTransition._can_run` not allowing transitions to run if there are more than two transitions combined.

6.6 0.5.0 (2017-07-13)

- Custom exceptions.
- Duplicated definition of `on_execute` callback is not allowed.
- Fix bug on `StateMachine.on_<transition.identifier>` being called with extra `self` param.

6.7 0.4.2 (2017-07-10)

- Python 3.6 support.
- Drop official support for Python 3.3.
- *Transition* can be used as decorator for *on_execute* callback definition.
- *Transition* can point to multiple destination states.

6.8 0.3.0 (2017-03-22)

- README getting started section.
- Tests to state machine without model.

6.9 0.2.0 (2017-03-22)

- `State` can hold a value that will be assigned to the model as the state value.
- Travis-CI integration.
- RTD integration.

6.10 0.1.0 (2017-03-21)

- First release on PyPI.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search