edgy.workflow Documentation

Release 0.1.1

Romain Dorgueil

July 23, 2016

Contents

Workflow is a simple library that allows you to manage lightweight workflows/state machines, and easily add this logic to business objects you already use.

It supports both python 2 and 3, and does not assume you use any framework. In the future, we may include a light adapter for the most popular frameworks, but the code should be very trivial.

By design choice, the library does not enforce the validity of objects. If you want to set an invalid state on an object, you will be allowed to do so. That's a tradeoff that makes the library more flexible, at the price of less data interity (in the plans: add a «strict» mode).

CHAPTER 1

Kick-start

```
from edgy.workflow import Workflow, Transition, StatefulObject

# Define transitions
@Transition(source='new', target='accepted')
def accept(self, subject):
    print('accepting {} using {}...'.format(subject, self))

@Transition(source='new', target='refused')
def refuse(self, subject):
    print('refusing {} using {}...'.format(subject, self))

# Create a workflow object
workflow = Workflow()
workflow.add_transition(accept)
workflow.add_transition(refuse)

# Create a stateful object
class Issue(StatefulObject):
    initial_state = 'new'
    workflow = workflow

# Play with your newly workflow-enabled object.
iss42 = Issue()
iss42.accept()

iss43 = Issue()
iss43.refuse()

iss44 = Issue(state='invalid')
```

CHAPTER 2

Dive-in

2.1 Transitions

The smallest atom of `edgy.workflow` is a `Transition`, which basically is a regular python callable with additional metadata to make the system aware of when it can be applied.

class `edgy.workflow.`**`Transition`**(*handler=None*, *name=None*, *source=None*, *target=None*)

Defines when and how to go from one state to another, eventually applying a user-defined side-effect while being applied.

Example:

```
>>> t = Transition(name='sleep', source='awake', target='asleep')

>>> class Person(object):
...     state = 'awake'

>>> me = Person()
>>> t(me)
>>> me.state
'asleep'
```

This class can also be used as a decorator:

```
>>> @Transition(source='asleep', target='awake')

>>> def wakeup(self, subject):
...     print('HEY!')

>>> wakeup(me)
>>> me.state
'awake'
```

A special wildcard source can make transitions work from any state. Just specify "*" as a transition source and you'll be able to transition from any state.

2.2 Workflows

class `edgy.workflow.`**`Workflow`**

A `Workflow` is a coherent set of Transitions meant to define a state machine system.

add_transition (*transition*, *name=None*)
Add a transition to this workflow instance, to be used on stateful subjects later.

Parameters **transition** (edgy.workflow.Transition) – Transition to add.

states
Set of valid known states for this workflow. Beware, if you're using wildcard as source, there can be states you expect as valid that this instance does not know about, and will treat as invalid.

Returns set[str]

transitions
Set of transitions living in this state machine system.

Returns set[edgy.workflow.Transition]

2.3 Stateful objects

For now, a "stateful object" is, to this library, anything that as a "state" attribute that can be read. If it quacks, then it's a duck.

However, as an helper class to demonstrate how a workflow can be bound to an object, we provide `StatefulObject` as an example implementation that you can use.

class edgy.workflow.**StatefulObject**
Example stateful object.

To use it, subclass me and set the workflow attribute to a `edgy.workflow.Workflow` instance.

workflow
A workflow instance, setting the system in which the instances of this object live.

initial_state
The default initial state of this object.

current_state
The current state of this object.

state
Helper for getting the actual state of an object. You should use this instead of `initial_state` and `current_state` if your only aim is to read or write a new state to this object.

Beware though, the setter of this property will override the state, without going through the transitions. If you wanna run the transitions (and in 95% of the cases, you should, otherwise this library is a pretty bad choice for you), then a proxy attribute exist on the object for each transition name, and you should just call it (for example, if a transition is named `wakeup`, you can just call `instance.wakeup()`).

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

e

A

C

E

I

S

T

W