Decovent

Version 1.1.1

Python 2.6.4 & Python 3

Python events rising and handling using @decorators(with arguments)

Adrian Cristea (adrian.cristea@gmail.com)

Example – import & create

```
from decovent import *          # the only import required

class Mouse(object):            # no inheritance is required
    def __init__(self):
        self.on_click()         # handler registration – no arguments

    @raise_event()
    def click(self, x, y):
        return (x, y)

    @set_handler(‘click’)
    def on_click(self, x, y):
        return (x, y)
```

Example – rising the event

```
mouse = Mouse()

# raises the event and executes registered handlers
mouse.click(10, 20)
```

Example – execution result

```
# the result of the event execution
>>(True, (10, 20), <class '__main__.Mouse'>, <function
    click at 0x00BCAFB0>)

# the result of the handler execution
>> ((True, >> (10, 20), <class '__main__.Mouse'>,
    <function on_click at 0x00BD10B0>),)
```

Example – log output

- Registering handler for <class '__main__.Mouse'>.click
- Handler was registered successfully
- Raising event <class '__main__.Mouse'>.click():12
- Event intercepted by <class '__main__.Mouse'>.on_click():16
- [MainThread] Processing event <class '__main__.Mouse'>.click()
- [MainThread] Processing of event <class '__main__.Mouse'>.click() is completed
- [Thread-1] Processing handler <class '__main__.Mouse'>.on_click()
- [Thread-1] Processing of handler <class '__main__.Mouse'>.on_click() is completed

Features (I)

- events and handlers are tied to the local-thread
- event name is case sensitive, Unicode safe and not required if it equals the decorated method name
- for an event can be registered as many handlers as necessary
- handlers are registered for (class, event) pair

Features (II)

- a handler can be registered many times, but will be executed only once for (class, event) pair
- handlers call order == registration order
- handlers are always executed in parallel threads, synchronous or asynchronous
- @classmethods can be raised as events or registered as handlers

Features (III)

- events and handlers can be memoized at local or global level
- events and handlers can be synchronized
- the time allocated for the execution of an event or handler is controllable
- the number of active executions is controllable

Restrictions

- events and handlers must be methods that belong to new-style classes
- @staticmethods can’t be raised as events or registered as handlers
- one handler can be registered for one event only

Handle own events

```
class Mouse(object):
    def __init__(self):
        self.on_click()                    # handler registration

    @raise_event()
    def click(self, x, y):
        return (x, y)

    @set_handler(‘click’)
    def on_click(self, x, y):
        return (x, y)

mouse = Mouse()
mouse.click(10, 20)
```

Handle events of another class

```
class Mouse(object):
    @raise_event()
    def click(self, x, y):
        return (x, y)

class Screen(object):
    @set_handler('click', Mouse)       # handles Mouse.click
    def on_click(self, x, y):
        return (x, y)

screen = Screen()
screen.on_click()                      # handler registration

mouse = Mouse()
mouse.click(10, 20)
```

@classmethod event or handler

```
class Mouse(object):
    @classmethod
    @raise_event()
    def click(self, x, y):
        return (x, y)

    @classmethod
    @set_handler('click')
    def on_click(self, x, y):
        return (x, y)

Mouse.on_click()
Mouse.click(10, 20)
```

Different event name

```
class Mouse(object):
    def __init__(self):
        self.on_move()

    @raise_event('move')             # event name != method name
    def click(self, x, y):
        return (x, y)

    @set_handler('move')             # handles event 'move'
    def on_move(self, x, y):
        return (x, y)

mouse = Mouse()
mouse.click(10, 20)
```

Execute handlers asynchronous

```
decovent.asynchronous = True
```

Unregister handler after 1st exec

```
class Mouse(object):
    def __init__(self):
        self.on_click()

    @raise_event()
    def click(self, x, y):
        return (x, y)

    @set_handler(‘click’, unregister=True)      # executed only once
    def on_click(self, x, y):
        return (x, y)

mouse = Mouse()
mouse.click(10, 20)                    # this event is handled
mouse.click(30, 40)                    # this event is NOT handled
```

Unregister handlers

- decovent.reset(Mouse, ‘click’)
  - removes all handlers for Mouse.click
- decovent.reset(Mouse)
  - removes all handlers for Mouse
- decovent.reset()
  - removes all handlers

Integrate with other decorators

- The example is a bit longish, please see it in the documentation
  - http://packages.python.org/Decovent/#how_to_12

Memoization

- decovent.memoize = True
  - activates memoization at global level
- @raise_event(‘click’, memoize_=True)
- @set_handler(‘click’, memoize_=True)

Synchronization

```
lock = threading.RLock()

class Mouse(object):
    # event & registered handlers are synchronized on this lock
    @raise_event(lock=lock)
    def click(self, x, y):
        return (x, y)

    @set_handler(‘click’)
    def on_click(self, x, y):
        return (x, y)
```

Timeout

```
class Mouse(object):
    @raise_event(timeout=1)
    def click(self, x, y):
        return (x, y)

    @set_handler('click', timeout=2)
    def on_click(self, x, y):
        return (x, y)
```

Active executions

To allow maximum *n* methods to be active at one time set decovent.active(n).

By default, 3 methods can be executed in parallel at one time.

Execution result (synch)

On success:

```
(True, (10, 20), <class '__main__.Mouse'>, <function
    click at 0x00BC5F30>)
((True, (10, 20), <class '__main__.Mouse'>, <function
    on_click at 0x00BC5FB0>),)
```

On error:

```
(False, error, <class '__main__.Mouse'>, <function click
    at 0x00BC5F30>)
((False, error, <class '__main__.Mouse'>, <function
    on_click at 0x00BC5FB0>),)
```

Execution result (asynch)

On success:

```
(True, (10, 20), <class '__main__.Mouse'>, <function click at 0x00BC5F30>)
((None, None, <class '__main__.Mouse'>, <function on_click at 0x00BC5FB0>),)
```

On error:

```
(False, error, <class '__main__.Mouse'>, <function click at 0x00BC5F30>)
((None, None, <class '__main__.Mouse'>, <function on_click at 0x00BC5FB0>),)
```

Error return

- decovent.exc_info = False & decovent.traceback = False
  - sys.exc_info()[1]

- decovent.exc_info = True & decovent.traceback = False
  - sys.exc_info()[:2]

- decovent.exc_info = True & decovent.traceback = True
  - sys.exc_info()

Download

http://pypi.python.org/pypi/Decovent

Documentation

http://packages.python.org/Decovent

Thank you

If you’ll use Decovent in your projects, please drop me a line, I’d like to know about it ☺

adrian.cristea@gmail.com

May, 2010