
Commodity Documentation

Release 0.20130110

David Villa Alises

Nov 07, 2018

Contents

1	deco	3
2	args	5
3	log	7
4	mail	9
5	os	11
6	path	13
7	pattern	15
8	str	19
9	testing	21
10	thread	23
11	type	25
12	Indices and tables	27
	Python Module Index	29

`commodity` is a library of recurrent required utilities for Python programmers.

Contents:

CHAPTER 1

deco

`commodity.deco.add_attr(name, value)`

Set the given function attribute:

```
>>> @add_attr('timeout', 4)
... def func(args):
...     pass
...
>>> func.timeout
4
```

`commodity.deco.handle_exception(exception, handler)`

Add and exception handler with decorator

```
>>> import logging, math
>>> @handle_exception(ValueError, logging.error)
... def silent_sqrt(value):
...     return math.sqrt(value)
>>> silent_sqrt(-1)
ERROR:root:math domain error
```

`commodity.deco.suppress_errors(func=None, default=None, log_func=<function debug>, errors=[<type 'exceptions.Exception'>])`

Decorator that avoids to create try/excepts only to avoid valid errors. It can return a default value and ignore only some exceptions.

Example: It will return the file content or the empty string if it doesn't exists:

```
>>> @suppress_errors(errors=[IOError], default='')
... def read_file(filename):
...     with file(filename) as f:
...         return f.read()
```

```
>>> read_file('/some/not/exists')
''
```

Parameters

- **func** – Function to be called.
- **default** – Default value to be returned if any of the exceptions are launched.
- **log_func** – The log function. logging.debug by default. It is a good idea to use it to avoid ‘catch pokemon’ problems.
- **errors** – list of allowed Exceptions. It takes in mind inheritance.

`commodity.deco.tag(name)`

Add boolean attributes to functions:

```
>>> @tag('hidden')
... def func(args):
...     pass
...
>>> func.hidden
True
```

CHAPTER 2

args

Creating a powerful argument parser is very easy:

```
from commodity.args import parser, add_argument, args

add_argument('-f', '--foo', help='foo argument')
add_argument('-v', '--verbose', action='store_true', help='be verbose')

parser.parse_args('--foo 3'.split())

assert args.foo == 3
```

And it supports config files. That requires argument specs (config.specs):

```
[ui]
foo = integer(default=0)
verbose = boolean(default=False)
```

...the user config file (.app.config):

```
[ui]
foo = 4
verbose = True
```

...and the code:

```
from commodity.args import parser, add_argument, args

add_argument('-f', '--foo', help='foo argument')
add_argument('-v', '--verbose', action='store_true', help='be verbose')

parser.set_specs("config.specs")
parser.load_config("/etc/app.config")
parser.load_config("/home/user/.app.config")
parser.load_config("by-dir.config")
```

(continues on next page)

(continued from previous page)

```
parser.parse_args()  
assert args.foo == 4
```

CHAPTER 3

log

class commodity.log.**CallerData** (*depth=0*)
Get info about caller function, file, line and statement

```
>>> def a():
...     return b()
>>>
>>> def b():
...     print CallerData()
>>>
>>> a()
File "<stdin>", line 2, in a()
    return b()
```

class commodity.log.**CapitalLoggingFormatter** (*fmt=None, datefmt=None*)
Define variable “levelcapital” for message formating. You can do things like: [EE] foo bar

```
>>> formatter = CapitalLoggingFormatter('[%(levelcapital)s] %(message)s')
>>> console = logging.StreamHandler()
>>> console.setFormatter(formatter)
```

class commodity.log.**NullHandler** (*level=0*)

class commodity.log.**PrefixLogger** (*logger, prefix*)

class commodity.log.**UniqueFilter** (*name=""*)

Log messages are allowed through just once. The ‘message’ includes substitutions, but is not formatted by the handler. If it were, then practically all messages would be unique!

from: <http://code.activestate.com/recipes/412552-using-the-logging-module/>

CHAPTER 4

mail

CHAPTER 5

OS

`commodity.path.abs_dirname(path)`

Return absolute path for the directory containing *path*

```
>>> abs_dirname('test/some.py')
'/home/john/devel/test'
```

`commodity.path.child_relpath(path, start='.')`

Convert a path to relative to the current directory if possible

```
>>> os.getcwd()
/home/user
>>> child_relpath('/home/user/doc/last')
'doc/last'
>>> child_relpath('/usr/share/doc')
'/usr/share/doc'
```

`commodity.path.find_in_ancestors(fname, path)`

Search 'fname' file in the given 'path' and its ancestors

```
>>> find_in_ancestors('passwd', '/etc/network')
'/etc'
>>> find_in_ancestors('passwd', '/etc/network/interfaces')
'/etc'
```

`commodity.path.get_parent(path)`

```
>>> get_parent('/usr/share/doc')
'/usr/share'
```

`commodity.path.get_project_dir(fpath)`

`commodity.path.resolve_path(fname, paths, find_all=False)`

Search 'fname' in the given paths and return the first full path that has the file. If 'find_all' is True it returns all matching paths. It always returns a list.

```
>>> resolve_path('config', ['/home/user/brook', '/etc/brook'])
['/etc/brook/config']
```

`commodity.path.resolve_path_ancestors` (*fname, path, find_all=False*)

Search ‘fname’ in the given path and its ancestors. It returns the first full path that has the file. If ‘find_all’ is True it returns all matching paths. Always returns a list

```
>>> resolve_path_ancestors('passwd', '/etc/network')
'/etc/passwd'
>>> resolve_path_ancestors('passwd', '/etc/network/interfaces')
'/etc/passwd'
```

class commodity.pattern.**Bunch** (*args, **kargs)

It provides dict keys as attributes and viceversa

```
>>> data = dict(ccc=2)
>>> bunch = Bunch(data)
>>> bunch.ccc
2
>>> bunch.ddd = 3
>>> bunch['ddd']
3
>>> bunch['eee'] = 4
>>> bunch.eee
4
```

class commodity.pattern.**DummyLogger**

class commodity.pattern.**Flyweight** (name, bases, dct)

Flyweight design pattern (for identical objects) as metaclass

```
>>> class Sample(object):
...     __metaclass__ = Flyweight
```

class commodity.pattern.**MetaBunch** (dct=None)

A bunch of bunches. It allows to recursively access keys as attributes and viceversa. It may decorate any mapping type.

```
>>> b = MetaBunch()
>>> b['aaa'] = {'bbb': 1}
>>> b.aaa.bbb
1
>>> b.aaa.ccc = 2
>>> b['aaa']['ccc']
2
```

class commodity.pattern.NestedBunch(*args, **kwargs)
Bunch with recursive fallback in other bunch (its parent)

```
>>> a = NestedBunch()
>>> a.foo = 1
>>> a2 = a.new_layer()
>>> a2.bar = 2
>>> a2.foo
1
>>> a2.keys()
['foo', 'bar']
```

```
>>> b = NestedBunch()
>>> b.foo = 1
>>> b2 = b.new_layer()
>>> b2.foo
1
>>> b2['foo'] = 5000
>>> b2.foo
5000
>>> b['foo']
1
```

```
>>> c = NestedBunch()
>>> c.foo = 1
>>> c2 = c.new_layer().new_layer()
>>> c2.foo
1
```

class commodity.pattern.Observable

class commodity.pattern.TemplateBunch
A Bunch automatically templated with its own content

```
>>> t = TemplateBunch()
>>> t.name = "Bob"
>>> t.greeting = "Hi $name!"
>>> t.greeting
"Hi Bob!"
>>> t.person = "$name's sister"
>>> t.greeting = "Hi $person!"
>>> t.person
"Bob's sister"
>>> t.greeting
"Hi Bob's sister"
```

commodity.pattern.make_exception(name, message=)

commodity.pattern.memoized(method_or_arg_spec=None, ignore=None)

Memoized decorator for functions and methods, including classmethods, staticmethods and properties

```
>>> import random
>>>
>>> @memoized
... def get_identifier(obj):
...     return random.randrange(1000)
...
>>> n1 = get_identifier("hi")
```

(continues on next page)

(continued from previous page)

```
>>> n2 = get_identifier("hi")
>>> assert n1 == n2
```

It allows to ignore some arguments in memoization, so different values for those arguments DO NOT implies new true invocations:

```
>>> @memoized(ignore=['b'])
... def gen_number(a, b):
...     return random.randrange(1000)
...
>>> n1 = gen_number(1, 2)
>>> n2 = gen_number(1, 3)
>>> print n1, n2
>>> assert n1 == n2
```

- <http://pko.ch/2008/08/22/memoization-in-python-easier-than-what-it-should-be/>
- <http://micheles.googlecode.com/hg/decorator/documentation.html>

class commodity.pattern.memoizedproperty(*method*)
Memoized properties

```
>>> import random
>>> class A:
...     @memoizedproperty
...     def a_property(self):
...         return random.randrange(1000)
...
>>> a1 = A()
>>> v1 = a1.a_property
>>> v2 = a1.a_property
>>> assert v1 == v2
```


class commodity.str_.**ASCII_TreeRender**

Draws an ASCII-art tree just with ASCII characters. norm_brother = '+-'

class commodity.str_.**Printable**

Class mixin that provides a `__str__` method from the `__unicode__` method.

class commodity.str_.**TemplateFile** (*template*)

Render a template (given as string or input file) and renders it with a dict over a string, given filename or tempfile.

```
>>> t = TemplateFile('$who likes $what')
>>> t.render(dict(who='kate', what='rice'))
'kate likes rice'
```

class commodity.str_.**TreeRender**

class commodity.str_.**UTF_TreeRender**

Draws an ASCII art tree with UTF characters.

commodity.str_.**add_prefix** (*prefix, items*)

Add a common prefix to all strings of a list:

```
>>> add_prefix('pre-', ['black', 'red', 'orange'])
['pre-black', 'pre-red', 'pre-orange']
```

commodity.str_.**convert_to_string** (*obj, encoding='utf-8'*)

Returns an encoded string from unicode or string.

CHAPTER 9

testing

CHAPTER 10

thread

class commodity.thread_.**ActualReaderThread** (*fdin, fdout*)
A thread that read from an actual file handler and write to an object that just has the write() method.

class commodity.thread_.**DummyReaderThread**

commodity.thread_.**ReaderThread** (*fin, fout*)

class commodity.thread_.**SimpleThreadPool** (*num_workers*)
A generic and simple thread pool. Function return value are received by means of callbacks.

```
>>> import math
>>>
>>> result = []
>>> pool = SimpleThreadPool(4)
>>> pool.add(math.pow, (1, 2), callback=result.append)
>>> pool.join()
```

Also implements a parallel map () for an argument sequence for a function executing each on a different thread:

```
>>> pool = SimpleThreadPool(4)
>>> pool.map(math.sqrt, ((2, 4, 5, 9)))
(1.4142135623730951, 2.0, 2.23606797749979, 3.0)
```

```
>>> pool.map(math.pow, [2, 3, 3], [2, 2, 4])
(4, 9, 81)
```

class commodity.thread_.**ThreadFunc** (*target, *args, **kargs*)
Execute given function in a new thread. It provides return value (or exception) as object attributes.

```
>>> import math
```

```
>>> tf = ThreadFunc(math.sin, 8)
>>> tf.join()
>>> tf.result
0.9893582466233818
```

```
>>> tf = ThreadFunc(math.sqrt, -1)
>>> tf.join()
>>> tf.exception
>>> ValueError('math domain error')
```

class commodity.thread_.**WorkerGroup**(*num_workers*)

A group of dedicated workers.

```
>>> import math
>>>
>>> results = []
>>> group = WorkerGroup(4)
>>> w1 = group.get_worker("some unique value")
>>> w1.add(math.square, (9,), callback=results.append)
>>> group.join()
>>>
>>> print results
[81]
```

commodity.thread_.**start_new_thread**(*target*, **args*, ***kargs*)

Execute given function in a new thread. It returns a *ThreadFunc* object.

CHAPTER 11

type

`commodity.type_.checked_type(cls, val)`

If 'val' is a instance of 'cls' returns 'val'. Otherwise raise TypeError.

```
>>> checked_type(int, 3)
3
>>> checked_type((str, int), '4')
'4'
>>> checked_type(str, 5)
TypeError: str is required, not int: '5'.
```

`commodity.type_.module_to_dict(module)`

Return module global vars as a dict

CHAPTER 12

Indices and tables

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

c

- `commodity.deco`, 3
- `commodity.log`, 7
- `commodity.path`, 13
- `commodity.pattern`, 15
- `commodity.str_`, 19
- `commodity.thread_`, 23
- `commodity.type_`, 25

p

- `pattern`, 15

A

`abs_dirname()` (in module `commodity.path`), 13
`ActualReaderThread` (class in `commodity.thread_`), 23
`add_attr()` (in module `commodity.deco`), 3
`add_prefix()` (in module `commodity.str_`), 19
`ASCII_TreeRender` (class in `commodity.str_`), 19

B

`Bunch` (class in `commodity.pattern`), 15

C

`CallerData` (class in `commodity.log`), 7
`CapitalLoggingFormatter` (class in `commodity.log`), 7
`checked_type()` (in module `commodity.type_`), 25
`child_relpath()` (in module `commodity.path`), 13
`commodity.deco` (module), 3
`commodity.log` (module), 7
`commodity.path` (module), 13
`commodity.pattern` (module), 15
`commodity.str_` (module), 19
`commodity.thread_` (module), 23
`commodity.type_` (module), 25
`convert_to_string()` (in module `commodity.str_`), 19

D

`DummyLogger` (class in `commodity.pattern`), 15
`DummyReaderThread` (class in `commodity.thread_`), 23

F

`find_in_ancestors()` (in module `commodity.path`), 13
`Flyweight` (class in `commodity.pattern`), 15

G

`get_parent()` (in module `commodity.path`), 13
`get_project_dir()` (in module `commodity.path`), 13

H

`handle_exception()` (in module `commodity.deco`), 3

M

`make_exception()` (in module `commodity.pattern`), 16
`memoized()` (in module `commodity.pattern`), 16
`memoizedproperty` (class in `commodity.pattern`), 17
`MetaBunch` (class in `commodity.pattern`), 15
`module_to_dict()` (in module `commodity.type_`), 25

N

`NestedBunch` (class in `commodity.pattern`), 15
`NullHandler` (class in `commodity.log`), 7

O

`Observable` (class in `commodity.pattern`), 16

P

`pattern` (module), 15
`PrefixLogger` (class in `commodity.log`), 7
`Printable` (class in `commodity.str_`), 19

R

`ReaderThread()` (in module `commodity.thread_`), 23
`resolve_path()` (in module `commodity.path`), 13
`resolve_path_ancestors()` (in module `commodity.path`), 14

S

`SimpleThreadPool` (class in `commodity.thread_`), 23
`start_new_thread()` (in module `commodity.thread_`), 24
`suppress_errors()` (in module `commodity.deco`), 3

T

`tag()` (in module `commodity.deco`), 4
`TemplateBunch` (class in `commodity.pattern`), 16
`TemplateFile` (class in `commodity.str_`), 19
`ThreadFunc` (class in `commodity.thread_`), 23
`TreeRender` (class in `commodity.str_`), 19

U

`UniqueFilter` (class in `commodity.log`), 7

UTF_TreeRender (class in commodity.str_), [19](#)

W

WorkerGroup (class in commodity.thread_), [24](#)