CascaDict Documentation

Release 0.8.1

Josef Nevrly

November 16, 2015

Contents

1 That's it 5

2 Class documentation 7

3 Indices and tables 9

Python Module Index 11

This little class aims to solve just another almost nonexistent problem - handling of cascading properties. To describe it simply, CascaDict implements something like class inheritance, but on dictionary-key level. See the examples:

```
from cascadict import CascaDict

fruit_properties = CascaDict({'type':'fruit', 'taste':'sweet', 'color':"I don't have any color, I'm a
blueberry = fruit_properties.cascade({'name': 'blueberry', 'color':'blue'})

print(blueberry['color'])
print(blueberry['taste'])
```

```
blue
sweet
```

Internally, it's not just copy and append, one can also access all the cascaded values:

```
print(blueberry.get_cascaded('color'))
```

```
['blue', "I don't have any color, I'm an abstract concept!"]
```

Cascaded ancestors are referenced, not copied:

```
fruit_properties['taste'] = 'bittersweet'
blueberry['taste']
```

```
'bittersweet'
```

CascaDicts can be nested, and any dict element put into CascaDict is also nested as CascaDict:

```
fruit_properties['classification'] = {'kingdom': 'Plantae',}
blueberry['classification'] = {'Order': 'Ericales', 'Family': 'Ericaceae', 'Genus': 'Vaccinium', 'Sec

blueberry['classification']['kingdom']
```

```
'Plantae'
```

CascaDicts are of course iterable...

```
for key, value in blueberry.items():
    print(key,value)
```

```
('color', 'blue')
('name', 'blueberry')
('classification', <{'Section': 'Cyanococcus', 'Genus': 'Vaccinium', 'Order': 'Ericales', 'Family':
('taste', 'bittersweet')
('type', 'fruit')
```

... and picklable

```
import pickle
blueberry_jam = pickle.loads(pickle.dumps(blueberry))
for key, value in blueberry_jam.items():
    print(key,value)
```

```
('color', 'blue')
('name', 'blueberry')
('classification', <{'Section': 'Cyanococcus', 'Genus': 'Vaccinium', 'Order': 'Ericales', 'Family':
('taste', 'bittersweet')
('type', 'fruit')
```

If needed, CascaDict can be "flattened" into normal (nested) dict:

```
blueberry.copy_flat()
```

```
{'classification': {'Family': 'Ericaceae',
  'Genus': 'Vaccinium',
  'Order': 'Ericales',
  'Section': 'Cyanococcus',
  'kingdom': 'Plantae'},
 'color': 'blue',
 'name': 'blueberry',
 'taste': 'bittersweet',
 'type': 'fruit'}
```

Or only the top (final) level of CascaDict, without any ancestor properties, can be copied:

```
blueberry.copy_flat(level='skim')
```

```
{'classification': {'Family': 'Ericaceae',
  'Genus': 'Vaccinium',
  'Order': 'Ericales',
  'Section': 'Cyanococcus'},
 'color': 'blue',
 'name': 'blueberry'}
```

Combined with (e.g.) yaml, it makes any configuration processing a breeze:

```
import yaml

config = '''
defaults:
    port: 5556
    login_required: True
    logging:
        level: DEBUG
        handler: stream

process_1:
    max_runtime: 100
    login_required: False
    logging:
        handler: file

process_2:
    port: 6005
    halt_on_error: True
    logging:
        level: ERROR

'''

raw_config = yaml.load(config)
defaults = CascaDict(raw_config.pop('defaults'))
properties = {} #no dict comprehension, remember Python 2.7 folk
for k,v in raw_config.items():
    properties[k] = CascaDict(v, ancestor=defaults)

for k,v in properties.items():
    print("{0}: {1}".format(k, v.copy_flat()))
```

```
process_2: {'login_required': True, 'logging': {'handler': 'stream', 'level': 'ERROR'}, 'port': 6005,
process_1: {'logging': {'handler': 'file', 'level': 'DEBUG'}, 'login_required': False, 'max_runtime':
```

CHAPTER 1

That's it

This whole thing is just one small file, works in both Python 2.7 and 3.x and is released under MIT License. Now, cascade!

CHAPTER 2

Class documentation

class `cascadict.`**`CascaDict`**(**args*, ***kwargs*)

`cascade`(**args*, ***kwargs*)
Create new empty `CascaDict` cascading from this one.

`copy_flat`(*level='top'*, *recursive=True*)
Return flat copy (`dict`) of the `CascaDict`. Wrapper function for `__flatten__()`

`get_ancestor`()
Return `CascaDict` from which is current `CascaDict` cascaded.

`get_cascaded`(*key, default=[None]*)

Get item. If key is contained also in ancestors, a list of items from all ancestor for given key is retuned, sorted from top to bottom.

Parameters

- **`key`** –
- **`default`** – Default value to be returned when no key is found.

`get_root`()
Returns root ancestor for given CascaDict.

`is_root`()
Returns `True` if CascaDict has no ancestors (is root of the ancestor tree).

classmethod `new_cascadict`(*dict*)
Helper constructor for automatically cascading new CascaDict from object, regardless if it's another `CascaDict` or simple `dict`.

Parameters **`dict`** – `CascaDict` or `dict` object which will be cascaded.

`CascaDict.`**`__flatten__`**(*level='top'*, *recursive=True*)

Create flat `dict` containing all keys (even from ancestors). In case of overlapping values, value according to the 'level' argument will be selected.

Parameters

- **`level`** – ['top', 'bottom', 'skim'] Default: 'top'
  - 'top' level flattens with top level values for overlapping keys.
  - 'bottom' level flattens with bottom level (=closer to root) for overlapping keys.

  - **'skim' means that only values which were added to the final** ***CascaDict*** will be returned. Ancestor values are ignored, even those which are not overlapped.

- **recursive** – [`True`, `False`] Default `True`. If `True`, same flattening protocol is used for nested CascaDicts. Otherwise nested CascaDicts are simply referenced.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

C

cascadict, 7

Symbols

__flatten__() (cascadict.CascaDict method), 7

C

cascade() (cascadict.CascaDict method), 7
CascaDict (class in cascadict), 7
cascadict (module), 7
copy_flat() (cascadict.CascaDict method), 7

G

get_ancestor() (cascadict.CascaDict method), 7
get_cascaded() (cascadict.CascaDict method), 7
get_root() (cascadict.CascaDict method), 7

I

is_root() (cascadict.CascaDict method), 7

N

new_cascadict() (cascadict.CascaDict class method), 7