flattree

Release 2.0.2

Mar 16, 2020

Contents:

FlatTree is a lightweight tool that implements basic operations on nested Python dictionaries, "trees". It allows to

- merge trees into single tree
- access leaf nodes or branches using path-like "flat" keys
- use aliases for keys
- assign to or delete leaves or branches

The package has no dependencies other than The Python Standard Library.

CHAPTER 1

Usage example

FlatTree is quite useful when working with application configurations. Consider an application module that stores temporary objects in a file system cache. While in development, it's convenient to store objects in JSON format because of its human-readable nature. In production, objects are saved as pickles for performance.

Use FlatTree to merge configurations as needed:

```
>>> cfg_dev = {'processor': {'cache': {'format': 'json'}}}
>>> cfg_prod = {'processor': {'cache': {'format': 'pickle'}}}
>>> cfg_common = {'processor': {'cache': {'folder_options': ['.cache', 'cache']}},
>>>               'logging': None}
>>> cfg = FlatTree(cfg_dev, cfg_common)
>>> cfg['processor.cache.format']
'json'
>>> cfg['processor.cache.folder.0']  # List item can be addressed individually
'.cache'
>>> cfg.update_aliases({'FMT': 'processor.cache.format'})
>>> cfg['FMT']  # Access with an alias
'json'
```

It's possible to update leaves and branches. For example, consider adding logging configuration:

```
cfg['logging'] = {
    'version': 1,
    'disable_existing_loggers': False,
    'loggers': {
        '': {
            'level': 'INFO',
        },
        'my.module': {
            'level': 'DEBUG',
        },
    }
}
```

Values are accessible both as "scalar" leaves and as subtrees:

```
>>> cfg.update_aliases({'loglevel': 'logging.loggers..level'})
>>> cfg['loglevel']
'INFO'
>>> cfg.update_aliases({'loggers': 'logging.loggers'})
>>> cfg['loggers']
{'': {'level': 'INFO'}, 'my.module': {'level': 'DEBUG'}}
```

CHAPTER 2

Installation

```
pip install flattree
```

2.1 API (sphinx-autodoc)

2.1.1 flattree package

FlatTree is a tool to work with nested Python dictionaries.

Submodules

flattree.api module

class flattree.api.**FlatTree**(**trees*, *root=None*, *sep='.'*, *esc='\'*, *aliases=None*, *default=None*, *raise_key_error=False*)

Main tool to work with nested dictionaries using "flat" keys.

Flat keys are path-like strings with key components joined by "sep": e.g. 'level01.level02.level03.leaf' where dot is a sep.

***trees**
flat or regular trees, merged initialization

root
flat key prefix (puts tree in branch rather than root)

Type str

sep
symbol to use when joining key components

Type str

esc
symbol to escape sep in key components

Type str

aliases
dictionary in a form of {alias: flat_key}. Aliases are flat key shortcuts.

default
value to return if key is not found during dictionary access when raise_key_error is not set

raise_key_error
if True, raise exception rather than return `default`

classmethod flatten(**trees*, *root=None*, *sep='.'*, *esc='\\'*)
Merges nested dictionaries into a flat key dictionary.

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

tree
Regular tree dynamically recovered from the flat tree.

update_aliases(*aliases*)
Updates alias dictionary, removes aliases if value is None

Parameters **aliases** – new aliases

flattree.logic module

flattree.logic.**desparse**(*tree*, *na=None*, *reindex=True*)
Converts branch(es) with integer keys into lists within a dictionary.

Dictionary with (all) integer keys acts as a sparse list with only non-void values actually stored. This function would convert sparse list into the regular one.

Examples

{1: 'one', 3: 'three'} -> ['one', 'three'] # if reindex {1: 'one', 3: 'three'} -> [na, 'one', na, 'three'] # if not reindex

Parameters

- **tree** (*dict*) – dictionary
- **na** – value to fill in gaps
- **reindex** (*bool*) – if True, keep compact but change non-consecutive indices

Returns dict or list

flattree.logic.**flatkey_to_keylist**(*flatkey*, *sep='.'*, *esc='\\'*)
Converts flatkey to a list of key components, extracts list indices

Components that look like integers, e.g. '1000' get converted to integers, int('1000') in this example.

Parameters

- **flatkey** (*str*) – flatkey string
- **sep** (*str*) – symbol to use when joining flat key components
- **esc** (*str*) – symbol to escape sep in key components

Returns key components, int if

Return type list

flattree.logic.**genleaves**(**trees*, *pre=None*, *sep='.'*, *esc='\\'*, *idxbase=0*, *list_merger=<function list_merger_list0>*)

Generator used internally to merge trees and decompose them into leaves

Parameters

- **trees** – nested dictionaries to merge
- **pre** – list of key components to prepend to resulting flatkey strings
- **sep** (*str*) – symbol to use when joining flat key components
- **esc** (*str*) – symbol to escape sep in key components
- **idxbase** (*int*) – number at which list indices would start
- **list_merger** – function called on trees when leading tree is a list

Yields tuples (flatkey, scalar leaf value) Example: ('my.branch.x', 0)

flattree.logic.**keylist_to_flatkey**(*keylist*, *sep='.'*, *esc='\\'*)

Converts list of key components to a flatkey string

Integer key components are considered list indices and get converted.

Parameters

- **keylist** (*list*) – list of key components
- **sep** (*str*) – symbol to use when joining flat key components
- **esc** (*str*) – symbol to escape sep in key components

Returns flatkey string

Return type str

flattree.logic.**list_merger_list0**(**lists*)

Picks leading list, discards everything else

flattree.logic.**unflatten**(*flatdata*, *root=None*, *sep='.'*, *esc='\\'*, *default=None*, *raise_key_error=False*)

Restores nested dictionaries from a flat tree starting with a branch.

Parameters

- **flatdata** (*dict*) – dictionary of values indexed by flatkeys
- **root** – branch to restore (None for the whole tree)
- **sep** (*str*) – symbol to use when joining flat key components
- **esc** (*str*) – symbol to escape sep in key components
- **default** – default value Returned in case no branch is found and raise_key_error is False.
- **raise_key_error** (*bool*) – if True, raise exception rather than return the default value in case no branch is found

Returns Tree or leaf value or default.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

3.1 Closing remarks

Author is aware that this kind of functionality has already been implemented a number of times elsewhere. However, reinventing the wheel seemed a useful practice.

Python Module Index

f

Index