codi Documentation

Release 1.0.2

Mohammad Alghafli

Oct 10, 2018

Contents

1 The problem **3**

2 The solution **5**

3 Content **7**

3.1 Installation 7
3.2 Quick Guide 8
3.3 codi Reference 10
3.4 Indices and tables 13

Python Module Index **15**

This tutorial gives an introduction to how to use codi python library and its features.

This is not a python tutorial. You are expected to have general knowledge in python before you start this tutorial.

CHAPTER 1

The problem

Imagin your program needs to read some files at start up that change its behaviour . For example, it reads a config file that contains an array of values. Configuration files are usually stored in user files. However, when the program runs for the first time, the configuration files do not exist and you need to find fallback configuration files.

CHAPTER 2

The solution

This library solves this problem. You specify the user configuration directory and a fallback directory.

When you ask the library to read a file for you, the library tries to open the file from the user configuration directory. If the file (or the whole directory) is not found, the library looks for the file in the fallback directory. That only happens when you try to open a file for reading.

On the other hand, when you try to write to a file, the library only writes in the user configuration directory. It never writes to the fallback directory. You do not need to create any subdirectories since the library will create all subdirectories you need when you try to open a file for writing.

The library also gives you a *Config* class which is useful to store configuration values as well as set default values for your configuration.

CHAPTER 3

Content

3.1 Installation

3.1.1 Requirements

The major requirement of codi is python 3. codi is a python 3 library and was never tested in python 2. So first make sure your version of python is 3.

3.1.2 Installation

Make sure you have pip for python 3 installed.

On windows install using pip by running the command:

```
pip install codi
```

Or on linux:

```
pip3 install codi
```

Of course, pip command should be in your PATH environment variable. If you are using windows there is a good chance pip is not in your PATH. In this case you should specify the full pip path. Search about how to use pip on windows if you are having trouble.

Try to import codi to be sure it was installed successfully:

```
>>> import codi
>>>
```

If it is imported without errors, you are ready to use it. You may want to have a look at one or more of the following documents:

- *Quick Guide* For usage examples.

- *codi Reference* **This is the library reference. All classes and functions** documentation is here.

3.2 Quick Guide

3.2.1 Usage example

This is a typical usage example:

```
from codi import *

#assume we have 2 config directories:
#   * default-cfg/
#   * user-cfg/
#
#in other words, we have the following directory structure:
#
#
#default-cfg/
#|
#--path/
#  |
#  --to/
#    |
#    |-file.txt
#    |
#    --file.bin
#
#
#
#user-cfg/


#user config dir
user_dir = 'user-cfg/'
#default config dir
default_dir = 'default-cfg/'

#create a Codi object. you can give more than 2 dirs if you need.
config_dirs = Codi(user_dir, default_dir)

#read a file. args are same as builtin open
#will first try to open `user-cfg/path/to/file.txt`. because the file does
#not exist, will go to the next config dir and open
#`default-cfg/path/to/file.txt`.
f = config_dirs.open('path/to/file.txt')
print(f.read())
f.close()

#write a file.
#will always write in `user-cfg/path/to/file.txt`. any parent directories
#that do not exist will be created
f = config_dirs.open('path/to/file.txt', 'w')
print('hello world', file=f)
f.close()
```

(continues on next page)

(continued from previous page)

```
#convinience method to read data
#text. default encoding is utf8
#will open `user-cfg/path/to/file.txt` because it exists from our previous
#write operation.
print(config_dirs.read('path/to/file.txt', encoding='ascii'))
#binary
#will open `default-cfg/path/to/file.bin`
print(config_dirs.read('path/to/file.bin', text=False))

#convinience method to write data
#text. default encoding is utf8
#will always write in `user-cfg/path/to/file.txt`.
config_dirs.write('path/to/file.txt', 'hello world', encoding='ascii')
#binary
#again, will always write in `user-cfg/path/to/file.bin`.
config_dirs.write('path/to/file.bin', b'some binary data')
```

3.2.2 Config class usage example

This is a typical usage example for the *Config* class:

```
import codi

#Config objects are dictionary objects. you can pass the constructor
#anything you pass to a dictionary.
config = codi.Config()

#set default values
config.set_default('b', 2)
config.set_default('c', 3)

#set config values
config['a'] = -1
config['b'] = -2

#print config values
#will print -1 because we set its value previously
print(config['a'])

#will print -2 because we set its value. default is ignored.
print(config['b'])

#we did not set its value. will take the default value and print 3.
print(config['c'])

#no value and no default value. will raise KeyError
print(config['d'])
```

3.2.3 Further readings

In *codi Reference* you will find the library reference.

3.3 codi Reference

Date 2018-10-06

Version 1.0.2

Authors

- Mohammad Alghafli <thebsom@gmail.com>

Multiple configuration directories for your program. This library is useful if you have the following situation: you have a config directory where you store default configuration and another config directory where you store user custom configuration. When a user runs your program for the first time, no files exist in the user config directory and you want to read everything from the default config directory. When you write a config file, it must always be written in the user config directory. This library does this for you. You specify user config directory and any more directories you want to use for default config files. When you open a file for reading, the library opens the file from the user config directory if it exists. Otherwise, it searches for the file in the default config directories. When you open a file for writing, it is always opened in the user config directory. A typical usage example is below:

```
from codi import *

#assume we have 2 config directories:
#   * default-cfg/
#   * user-cfg/
#
#in other words, we have the following directory structure:
#
#
#default-cfg/
#|
#--path/
#  |
#  --to/
#    |
#    |-file.txt
#    |
#    --file.bin
#
#
#
#user-cfg/


#user config dir
user_dir = 'user-cfg/'
#default config dir
default_dir = 'default-cfg/'

#create a Codi object. you can give more than 2 dirs if you need.
config_dirs = Codi(user_dir, default_dir)

#read a file. args are same as builtin open
#will first try to open `user-cfg/path/to/file.txt`. because the file does
#not exist, will go to the next config dir and open
#`default-cfg/path/to/file.txt`.
f = config_dirs.open('path/to/file.txt')
print(f.read())
f.close()
```

(continues on next page)

(continued from previous page)

```
#write a file.
#will always write in `user-cfg/path/to/file.txt`. any parent directories
#that do not exist will be created
f = config_dirs.open('path/to/file.txt', 'w')
print('hello world', file=f)
f.close()

#convinience method to read data
#text. default encoding is utf8
#will open `user-cfg/path/to/file.txt` because it exists from our previous
#write operation.
print(config_dirs.read('path/to/file.txt', encoding='ascii'))
#binary
#will open `default-cfg/path/to/file.bin`
print(config_dirs.read('path/to/file.bin', text=False))

#convinience method to write data
#text. default encoding is utf8
#will always write in `user-cfg/path/to/file.txt`.
config_dirs.write('path/to/file.txt', 'hello world', encoding='ascii')
#binary
#again, will always write in `user-cfg/path/to/file.bin`.
config_dirs.write('path/to/file.bin', b'some binary data')
```

The library also provides *Config* class which acts as a dict for config values. It adds the ability to set default values.

class codi.**Codi**(**dirs*)

Class to set multiple directories for configuration files. The first config directory is the user config directory. The other config directories added are used as fallbacks when a file opened for reading is not found in the user config directory. If a path is opened for writing, it is always opened in the user config directory.

append(*dir*)

Append a config dir.

args:

- **dir (any type accepted by *pathlib.Path* constructor):** Directory to add.

extend(*dirs*)

Append new config dirs from iterable.

args:

- dirs (path-like object): iterable of directories to add.

glob(*pattern*)

Same as *pathlib.Path.glob* but searches all config dirs added to *self* for glob results. If a file is found in multiple config dirs, only the path of the file found in the first config dir is returned. The returned paths are absolute.

args:

- pattern (str): Glob pattern.

returns: The glob result as a *list* of *pathlib.Path* objects.

insert(*index*, *dir*)

Insert a config dir.

args:

- index (int): The index in which the new directory is added.
- dir (path-like object): Directory to add.

open (*file*, *mode='r'*, **args*, ***kwargs*)
Opens a file. This function is used in the same way as the builtin *open*. If *mode* contains w, x, a or +, the file is created in the first config dir and all directories in the path are created if they do not exist.

args: Same as builtin *open*.

returns: File object.

path (*path=''*, *writable=False*)
Return an absolute path as a pathlib.Path object for *path* in one of the config directories added to *self*.

args:

- path (path-like object): The relative path to return.
- **writable (bool): Whether the path is requested for writing or** reading. If *False*, the function searches the config directories in the order they were added for an existing path. If *writable* is *True*, the returned path is in the first config dir added to *self* whether it exists or not.

pop (*index=-1*)
Remove the config directory at index.

args:

- index (int): The index of the dir to be removed.

read (*path*, *text=True*, *encoding='utf-8'*)
Returns the content of the file *path*.

args:

- **path (path-like object): The path to read relative to the config** dirs added to *self*.
- **text (bool): If *True*, the file is read in text mode and str** object is returned . If *False*, the file is read in binary mode and bytes object is returned.
- **encoding: Text encoding to open the file with. Ignored if *text*** is *False*.

returns: The content of the file as str or bytes object.

remove (*value*)
Remove a config directory.

args:

- value (path-like object): The directory to remove.

write (*path*, *data*, *encoding='utf-8'*)
Writes data to the file *path*.

args:

- **path (path-like object): The path to write into** relative to the config dirs added to *self*.
- **data (str or bytes): Data to write. If an str object, file is** opened in text mode. If bytes object, file is opened in binary mode.
- **encoding: Text encoding to open the file with. Ignored if *data*** is a bytes object.

returns: None

class codi.**Config**(**args*, ***kwargs*)

Used to store config values. This class subclasses dict and adds the functionality of adding default values. Instead of raising a *KeyError* if the key is not in the dict, it looks for the key in an internal default values dict. If there is a default value for key, it adds it to itself and returns it. Otherwise, *KeyError* is raised.

get_default(*key*)

Returns the default value of *key*.

pop_default(*key*)

Removes and returns default value of *key*.

set_default(*key*, *value*)

Sets default value of *key* to *value*.

3.4 Indices and tables

- genindex
- modindex
- search

Python Module Index

C

codi, 10

Index

A

append() (codi.Codi method), 11

C

Codi (class in codi), 11
codi (module), 10
Config (class in codi), 12

E

extend() (codi.Codi method), 11

G

get_default() (codi.Config method), 13
glob() (codi.Codi method), 11

I

insert() (codi.Codi method), 11

O

open() (codi.Codi method), 12

P

path() (codi.Codi method), 12
pop() (codi.Codi method), 12
pop_default() (codi.Config method), 13

R

read() (codi.Codi method), 12
remove() (codi.Codi method), 12

S

set_default() (codi.Config method), 13

W

write() (codi.Codi method), 12