easy_config Documentation

Release 1.0.0

Scott Colby

Feb 11, 2019

Contents

Parse configuration values from files, the environment, and elsewhere all in one place.

On this page:

CHAPTER 1

Installation

Installation should be as easy as executing this command in your chosen terminal:

```
$ pip install easy_config
```

The source code for this project is hosted on Github. Downloading and installing from source goes like this:

```
$ git clone https://github.com/scolby33/easy_config
$ cd easy_config
$ pip install .
```

If you intend to install in a virtual environment, activate it before running `pip install`.

easy_config officially supports Python 3.6 and later.

CHAPTER 2

Example Usage

Here is a full working example of using easy_config. First, write your configuration class:

```
# config.py
from easy_config import EasyConfig

class MyProgramConfig(EasyConfig):
   FILES = ['myprogram.ini']
   NAME = 'MyProgram'  # the name for the .ini file section and the namespace prefix␣
↪for environment variables

   # define the options like you would a dataclass
   number: int
   name: str
   check_bounds: bool = True  # options with defaults must all come after non-default␣
↪options
```

A sample configuration file:

```
# myprogram.ini
[MyProgram]
# section name matches `NAME` from the configuration class
number = 3
```

And a sample program to illustrate the usage:

```
# test_config.py
import sys

from config import MyProgramConfig

print(MyProgramConfig.load(name=sys.argv[1])
```

Running this program with various options:

```
$ python test_config.py Scott
MyProgramConfig(number=3, name='Scott', check_bounds=True)

$ env MYPROGRAM_CHECK_BOUNDS=False python test_config.py Scott
# environment variable names are the all-uppercase transformation of the NAME␣
↪concatenated with the option name and an underscore
MyProgramConfig(number=3, name='Scott', check_bounds=False)

$ env MYPROGRAM_NUMBER=10 MYPROGRAM_NAME=Charlie python test_config.py Scott
MyProgramConfig(number=10, name='Scott', check_bounds=True)
```

As you can see, values are taken in precedence, with arguments passed to `load` overriding values from the environment which, in turn, override values from configuration files.

Once you have the `MyProgramConfig` instance, you can use it just like any dataclass.

CHAPTER 3

API Reference

Information about each function, class, and method is included here.

3.1 API Reference

Parse configuration values from files, the environment, and elsewhere all in one place.

exception easy_config.**ConfigValueCoercionError**
Raised when a configuration value cannot be converted to the proper type.

Example: field type is `int` and the value is `None` or `'apple'`.

__weakref__
list of weak references to the object (if defined)

class easy_config.**EasyConfig**(**_kwargs*)
The parent class of all configuration classes.

__init__(**_kwargs*)
Do not instantiate the base class.

`TypeError` is raised instead of `NotImplementedError` to prevent IDEs (namely: PyCharm) from complaining that subclasses have not implemented all abstract methods. See https://github.com/scolby33/easy_config/issues/19

Raises **TypeError** – always; this class must be subclassed

Return type `None`

classmethod _read_file(*config_file*)
Read configuration values from a file.

This method parses ConfigParser-style INI files. To parse other formats, subclass EasyConfig and override this method.

Parameters **config_file** (Union[str, Path, Iterable[str]]) – the file from which configuration will be read. Note that this can be an Iterable[str], which includes open files and TextIO objects.

Return type Dict[str, Any]

Returns a mapping from string configuration value names to their values

Raises ***ConfigValueCoercionError*** – when an error occurs calling the type constructor on an input value

classmethod _read_environment()

Read configuration values from the environment.

Configuration values are looked up in the environment by the concatenation of the value name and the NAME class variable with an underscore separator.

For example, the configuration value "number" for an instance with the NAME "myprogram" will be read from the environment variable "MYPROGRAM_NUMBER".

Return type Dict[str, Any]

Returns a mapping from string configuration value names to their values

Raises ***ConfigValueCoercionError*** – when an error occurs calling the type constructor on an input value

classmethod _read_dict(*d*)

Read configuration values from a passed-in mapping.

Configuration values are extracted from the input mapping. Only keys in the mapping that are valid configuration values names are returned, others are ignored.

Parameters **d** (Mapping[str, Any]) – the input mapping of string configuration value names to their values

Return type Dict[str, Any]

Returns a mapping from string configuration value names to their values

Raises ***ConfigValueCoercionError*** – when an error occurs calling the type constructor on an input value

classmethod load(*_additional_files=None*, ***, *_parse_files=True*, *_parse_environment=True*, *_lookup_config_envvar=None*, ***kwargs*)

Load configuration values from multiple locations and create a new instance of the configuration class with those values.

Values are read in the following order. The last value read takes priority.

1. values from the files listed in the FILES class variable, in order
2. values from files passed in the additional_files parameter, in order
3. values from the file specified by the config file specified by the environment variable _lookup_config_envvar
4. values from the environment
5. values passed as keyword arguments to this method (useful for values specified on the command line)

Parameters

- **_additional_files** (Optional[Iterable[Union[str, Path, Textio]]]) – files to be parsed in addition to those named in the FILES class variable; always parsed, no matter the value of the parse_files flag

- **_parse_files** (bool) – whether to parse files from the FILES class variable
- **_parse_environment** (bool) – whether to parse the environment for configuration values
- **_lookup_config_envvar** (Optional[str]) – the environment variable that contains the config file location. Like the loading from the environment, this value will be uppercased and appended to the program name. For example, the _lookup_config_envvar "config" for an instance with the NAME "myprogram" will result in a search for the environment variable "MYPROGRAM_CONFIG" for the path to the configuration file.
- **kwargs** (Any) – additional keyword arguments are passed through unchanged to the final configuration object

Return type ~EasyConfigOrSubclass

Returns an instance of the configuration class loaded with the parsed values

classmethod _load_helper(*_additional_files=None*, ***, *_parse_files=True*, *_parse_environment=True*, *_lookup_config_envvar=None*, ***kwargs*)

Help load the dictionaries in .load().

Return type Generator[Dict[str, Any], None, None]

dump(*fp*)

Serialize all current configuration values to fp as a ConfigParser-style INI.

Values will be placed in the section corresponding to the class value NAME.

Parameters **fp** (Textio) – a write()-supporting file-like object

Return type None

__weakref__

list of weak references to the object (if defined)

CHAPTER 4

Click Integration

easy_config ships with a contrib module integrating with the Click command line interface package.

4.1 Click

CHAPTER 5

License

5.1 License

This software is licensed under the MIT License. The full text of this license is below.

MIT License

Copyright (c) 2019 Scott Colby

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

`easy_config` is licensed under the MIT License, a permissive open-source license.

The full text of the license is available *here* and in the root of the source code repository.

CHAPTER 6

Changelog

6.1 Changelog

`easy_config` adheres to the Semantic Versioning ("Semver") 2.0.0 versioning standard. Details about this versioning scheme can be found on the Semver website. Versions postfixed with '-dev' are currently under development and those without a postfix are stable releases.

Changes as of 10 February 2019

6.1.1 1.0.0 <11 February 2019>

- Stabilization and 1.0.0 release!
- Add docs
- Use *ChainMap* instead of repeated *dict.update*'s in the loading code (@cthoyt)
- Add Click integration under *easy_config.contrib.click* (@cthoyt)
- Improve error messages when configuration value strings cannot be used to create configuration values of the appropriate type
- Change names of *_load_** private methods to *_read_** to better indicate their purpose
- Raise *TypeError* instead of *NotImplementedError* in the base *EasyConfig.__init__* to improve behavior in PyCharm and possibly other IDEs

6.1.2 0.2.0 <26 September 2018>

- Add *contrib* package for containing functionality that interacts with other packages, especially those outside the stdlib (@cthoyt)
- Add *click* extension to the contrib package for creating a *click* decorator based on an *EasyConfig* instance (@cthoyt)

6.1.3 0.1.0 <25 September 2018>

- Initial beta release to PyPI
- Implementation of most planned functionality
- 100% test coverage
- Clean bill of health from the various linters and MyPy
- Loading of file specified by an environment variable (@cthoyt)

easy_config adheres to the Semantic Versioning ("Semver") 2.0.0 versioning standard. Details about this versioning scheme can be found on the Semver website. Versions postfixed with '-dev' are currently under development and those without a postfix are stable releases.

You are reading the documents for version 1.0.0 of *easy_config*.

Full changelogs can be found on the *Changelog* page.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

e

Index

Symbols

C

D

E

L