Confyg Documentation

Release 0.1.0

Wilberto Morales

December 11, 2015

Contents

Welcome to Confyg's documentation. Confyg is a class based configuration module. By defining configuration classes, Confyg will load the values from their sources.

A source can be anything like a file or environment variables. The advantage to using Confyg is that you will have a unifying syntax to load configurations, independent of the data source.

CHAPTER 1

Tutorial

Here is a simple example that uses environment variables for a web server.

server_config.py

```
from confyg import OsConfyg


class Config(OsConfyg):
    SECRET_KEY = 'key'
    LOGGER_NAME = 'logger_name'
    DEBUG = 'debug'

Config.load()
assert Config.SECRET_KEY == '5b6ba13f79129a74a3e819b78e36b922'
assert Config.LOGGER_NAME == 'web_server'
assert Config.DEBUG == 'True'
```

```
export SECRET_KEY='5b6ba13f79129a74a3e819b78e36b922'
export LOGGER_NAME='web_server'
export DEBUG='True'
python server_config.py
```

We changed our mind and want to use a configuration file. The file needs to be in some format. For simplicity we choose JSON. The code doesn't change much:

```
from confyg import JsonConfyg


class Config(JsonConfyg):

    __source__ = 'config.json'

    SECRET_KEY = 'key'
    LOGGER_NAME = 'logger-name'
    DEBUG = 'debug'

Config.load()
assert Config.SECRET_KEY == '5b6ba13f79129a74a3e819b78e36b922'
assert Config.LOGGER_NAME == 'web_server'
assert Config.DEBUG == True
```

config.json

```
{
    "key": "5b6ba13f79129a74a3e819b78e36b922",
    "logger-name": "web_server",
    "debug": true
}
```

There is one thing to notice here. With JSON, *DEBUG* has a different type. This is because the operating system environment only supports strings. Different sources support different types. We think this can lead to bugs, and hope to fix it.

CHAPTER 2

Transformations

Transformations change the keys used to get values from the *__source__*. They are specified in the *__transformation__* class attribute. They must be wrapped around a *transformation* function call.

```
from confyg import JsonConfyg
from confyg.transformations import upper_case, composite, transformation,         hyphens_to_undersco


class Config(JsonConfyg):

    __source__ = 'config.json'

    SECRET_KEY = 'key'
    LOGGER_NAME = 'logger-name'
    DEBUG = 'debug'

Config.load()
```

This would work as before but imagine we changed our mind from JSON configuration file to a OS environment one. We could easily replace the inheriting *JsonConfyg* class to a *OSConfyg* one but things like 'logger-name' are not valid valid environment variable identifiers. All we have to do is tweak the keys that we use.

```
from confyg import OSConfyg
from confyg.transformations import upper_case, composite, transformation,         hyphens_to_undersco


class Config(OSConfyg):

    __source__ = 'config.json'

    __transformation__ = transformation(
        composite(
            upper_case,
            hyphens_to_underscore
        )
    )

    SECRET_KEY = 'key'
    LOGGER_NAME = 'logger-name'
    DEBUG = 'debug'

Config.load()
```

In here we are taking two transformations and mixing them together. The first *upper_case* will take a key and turn it

into a all upper cased string.

```
assert upper_case('key') == 'KEY'
assert upper_case('logger-name') == 'LOGGER-NAME'
assert upper_case('debug') == 'DEBUG'
```

The second replaces all hyphens to underscores:

```
assert hyphens_to_underscore('key') == 'key'
assert hyphens_to_underscore('logger-name') == 'logger_name'
assert hyphens_to_underscore('debug') == 'debug'
```

When mashed together we get:

```
assert composite(upper_case, hyphens_to_underscore)('key') == 'KEY'
assert composite(upper_case, hyphens_to_underscore)('logger-name') == 'LOGGER_NAME'
assert composite(upper_case, hyphens_to_underscore)('debug') == 'DEBUG'
```

Just like that we keep our nice looking keys in our source code while we keep them in whatever style the *__source__* prefers.

CHAPTER 3

Interface

Every Confyg has a *__source__*. The source is used to find the configuration values.

The *__config_store__* holds the serialized key values after being loaded from the *__source__*.

New Confyg subclasses should only have to override the *load_store* method. This method should return a dictionary like object. If so the default *get* and *set* methods should work fine.

We encourage you to read the source code for the project. It is tested, small, simple and documented.

You might wonder why we use names like *__source__*, *__config_store__*, and *__transformation__* instead of regular names. This is so that they do not clash with the values that we fill in the classes.

class `confyg.`**`Confyg`**
: The base configuration class implementing the common functionality. Subclassing this class should make it easy to add new configuration sources.

class `confyg.`**`JsonConfyg`**
: JsonConfyg let's you load configuration from a JSON file. The path to the file should be specified in *__source__*.

class `confyg.`**`DictConfyg`**
: The DictConfyg loads configuration from the dictionary set in *__source__*.

class `confyg.`**`OsConfyg`**
: The OSConfyg class loads configuration from environment variables.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

C

C

D

J

O