confy Documentation

Release 0.3.7

Kuba Janoszek

August 18, 2013

CONTENTS

Pragmatic & flexible **configuration loader** that makes your app settings clean and sexy. It reads configuration from many differnt sources including: python modules, environment variables and *.ini files.

The project aims to provide not only simple and easy solution to read settings in various formats, but also to give strong convention on how to keep code responsible for loading configuration.

CHAPTER

ONE

INSTALLATION

Simple and straight forward:

```
pip install confy
```

CHAPTER

TWO

SOURCE

All the code is hosted on github https://github.com/jqb/confy

2.1 Quick start

All the examples here assumes that settings directory looks exactly the same:

```
settings/
  |-- __init__.py
  |-- base.py
  |-- development.py
  |-- production.py
  `-- local.py
```

Content of settings/__init__.py is also the same:

```
1  import confy
2
3  confy = confy.loader(__file__)
4  config = confy.merge(
5      confy.from_modules('base', 'development'),
6      confy.from_modules('local', silent=True),
7  )
```

Let's just explain what short snippet in settings/__init__.py means.

In line no 3 we're defining a loader. The important thing here is that we are passing __file__ variable into loader constructor. This makes our loader aware of possition in the file system where are settings are and moreover gives usefull "confy.rootpath" method (we'll explain it later).

Lines 4-7 loades and merges settings from the sources. For sake of simplicity here we're loading settings only from python modules, but later we gonna explain other possible configuration sources, too.

Line no 5 loads two modules ('base' and 'development') one after another.

Line no 6 loads module 'local'. 'silent' param set to True means that confy won't complain if there's no such module like 'local'. In other words you can use it to conditionaly load modules (and other settings sources). This is convinent to configure environment specific settings.

2.1.1 Paths to resources

This is a common problem for web aplications, that you need to configure absolute path to the folder with your static files or other resources.

With “confy” you can simply define the paths relatively to your settings folder. Here’s how to do that.

Inside “settings/base.py” you can put the following snippet:

```
# SETTINGS_ROOT = confy.rootpath()                # /path/to/project/settings
# PROJECT_ROOT = confy.rootpath('..')             # /path/to/project
STATIC_FILES = confy.rootpath('..', 'static')     # /path/to/project/static
```

As you can see the path inside “confy.rootpath” is always calculated relatively to “settings” folder.

2.1.2 Variables interpolation

Interpolation is an nice feature in order to make your configuration clean and as simple as possible. Confy uses standard python “{variable_name}” interpolation, so you don’t need to learn anything new. Let’s see simple example.

So if you are using some kind of service inside your application you might want to easily manage url’s you need to work with to avoid simple and annoying problems with misstyped protocol (http vs https) and super-annoing duplication of copy-and-pasting the same root of url.

Assuming this is content of your “settings/base.py” file:

```
# -*- coding: utf-8 -*-


# Some immaginary API settings
api_domain = "http://api.com"
API_ADD = '{api_domain}/add/'
API_DELETE = '{api_domain}/delete/'
```

you can easily change values of “api_domain” in development.py / production.py and you don’t need to rewrite all the urls once again.

Contents of “settings/development.py”:

```
# Let's just change interesting parts
api_domain = "http://development.api.com"

>>> import os; os.environ['CONFIGURATION_MODE'] == 'development'
>>>
>>> from settings import config
>>> assert config.API_ADD == 'http://api-development.com/add/'
>>> assert config.API_DELETE == 'http://api-development.com/delete/'
```

Contents of “settings/production.py”:

```
# Let's just change interesting parts
api_domain = "http://production.api.com"

>>> import os; os.environ['CONFIGURATION_MODE'] == 'production'
>>>
>>> from settings import config
>>> assert config.API_ADD == 'http://production.api.com/add/'
>>> assert config.API_DELETE == 'http://production.api.com/delete/'
```

2.1.3 Neasted structures - collections

In general keeping settings as flat’n’simple variables is the best idea, however it makes sense sometimes to avoid typing the same prefix again and again.

Contents of your "base.py" might look like this

```
# settings/base.py
API = confy.collection(
    domain = "http://api.com",
    ADD    = '{domain}/add/',
    DELETE = '{domain}/delete/',
)
```

Then again, changing domain url is very simple, inside your "development.py"

```
# settings/development.py
API.update(
    domain = "http://api-development.com",
)
```

```
>>> import os; os.environ['CONFIGURATION_MODE'] == 'development'
>>>
>>> from settings import config
>>> assert config.API.ADD == 'http://api-development.com/add/'
>>> assert config.API.DELETE == 'http://api-development.com/delete/'
```

As you can see it's preatty simple, but two things might be interesting to you.

1. global "confy" object?

   yes - it is global helper **buy ONLY inside your settings folder** and it is global only for the time when module is beeing loaded. Thats why It's been decided to use "with confy.loader" statement instead of simple assigment.

2. "confy.collection"

   creates confy collection object. Basicaly all you need to know is that it behaves exactly as a dictionary, and has additional features like to recognize "{interpolation_variables}" and ability to use __getitem__ notation for keys if you want to (keys might be non-identifiers as well - but ofcourse you won't be able to get them with "." notation).

   ```
   >>> from settings import config
   >>> config.API.ADD == config.API['ADD']  # => True
   ```

2.1.4 Lazy property

Having interpolation property is nice feature but it very rarely happens that you need more flexibility. "lazy" property is allowing you to create property-like function that'll be invoked to calculate value.

```
API = confy.collection(
    domain = "http://api.com",
    ADD    = '{domain}/add/',
    DELETE = '{domain}/delete/',
    ALL    = confy.lazy(lambda self: "%s, %s" % (self.ADD, self.DELETE)),
)

all_urls = API.ALL  # the function that was passed to "confy.lazy" is invoked here
assert all_urls == "http://api.com/add/, http://api.com/delete/"
```

2.1.5 Lazy import property

It's often practice to store complete path to "somekind.of.BackendClass" in settings file. Hovever you always need to write code that will later use it to acctualy import the think. You can stop thinking about it:

```
# settings/base.py
SUPER_DUPER_BACKEND = confy.lazyimport('somekind.of.BackendClass')
```

```
>>> from settings import config  # no import here...
>>> config.SUPER_DUPER_BACKEND   # ...but here the import is done and BackendClass is ready for you
```

2.1.6 Raw propery

Ok - but you really want to use "{}" chars inside your setting string - exactly as they are. - No problem:

```
# settings/base.py
RAW_STRING = confy.raw('use as many {} specia; {{{ }}}}} () characters as you want')
```

2.2 Source

confy supports number of formats from which you can read configuration. You can see how it works with with modules in quick start tutorial. Here I'm gonna explain other features.

1. Let's extend our example from quick start tutorial. Settings directory now looks like this:

```
settings/
  |-- __init__.py
  |-- base.py
  |-- development.py
  |-- production.py
  |-- local.py
  |-- sample.ini              # <= ini file
  `-- envvars/                # <= directory with variables like for "envdir"
        |-- DATABASE/         #    http://cr.yp.to/daemontools/envdir.html
        |     |-- USER
        |     |-- PASSWORD
        |     |-- PORT
        |     |-- NAME
        |     `-- HOST
        `-- HELLO
```

   If you don't have idea how example envvars could look like, please visit https://github.com/jqb/confy/tree/master/tests/tconf/envvars inside confy tests directory.

2. Content for settings/__init__.py for all examples below goes as follows:

```
1  import confy
2
3  confy = confy.loader(__file__)
4  config = confy.merge(
5      confy.from_modules('base', 'development'),
6      confy.from_modules('local', silent=True),
7      confy.from_ini('sample.ini'),
8      confy.from_dirs('envvars'),
9  )
```

2.2.1 INI files

confy can easily read standard "ini" files. If - let's say - content of sample.ini is

```
[DEFAULT]
root = /home/user

[static]
project_dir = %{root}/static

[media]
project.dir = %{root}/media    # note there's "." not "_" in variable name
```

when this is what you gonna get when you import `config` from `settings/__init__.py`

```
from settings import config

assert config.static.project_dir == "/home/user/static"   # OK
assert config.media['project.dir'] == "/home/user/media"  # OK
```

As you can see `confy` supports "." notation as far as variable names allows it to do so.

2.2.2 Envdir source

If you know `deamontools'` envdir that will be easy (http://cr.yp.to/daemontools/envdir.html). `confy` reads data inside env directory easily. It's a little bit more powerfull since you're not restricted to flat names only. Every directory inside pointed directory is treaded as key in dictionary, so you can have neasted structures as well.

In my extended example you can see `envvars` directory which is read by `confy`. This is what you gonna get (all the values are from confy tests folder: https://github.com/jqb/confy/tree/master/tests/tconf/envvars)

```
from settings import config

assert config.DATABASE.USER == "testdb"       # OK
assert config.DATABASE.PASSWORD == "testdb"   # OK
assert config.DATABASE.POST == "9000"         # OK
# etc...

assert config.HELLO == "world!"               # OK
```

CHAPTER

THREE

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*