Walter Documentation

Release 0.1

Adam Brenecki

Aug 18, 2017

Contents

Warning: Walter is pre-release software. Expect the API to change without notice, and expect this documentation to have lots of sharp edges.

Walter is a configuration library, inspired by python-decouple, and intended to replace direct access to `os.environ` in Django `settings.py` files (although it is by no means Django-specific). It currently supports Python 3.5+.

It differs from other, similar libraries for two reasons:

- It will let you specify your configuration parameters in one place and have auto-generated Sphinx documentation, just like with Python code. (Work on this hasn't been started yet.)
- When your users try to start up your app with invalid configuration, the error message they get shows a list of **all of the errors** with every configuration parameter, not just the first one.

CHAPTER 1

Installation

```
pip install walter
```

CHAPTER 2

Usage

```
from walter.config import Config

# Your configuration needs to be wrapped in a context manager,
# so Walter can collect all the errors and display them at the end.
with Config("SGC", "Dialer") as config:

    # Read a configuration value with config.get()
    SECRET_KEY = config.get('SECRET_KEY')

    # Convert the returned value to something other than a string with cast.
    DEBUG = config.get('DEBUG', cast=bool)

    # You can pass any function that takes a string to `cast`.
    # Here, we're using a third party function to parse a database URL
    # string into a Django-compatible dictionary.
    DATABASES = {
        'default': config.get('DATABASE_URL', cast=dj_database_url.parse),
    }

    # You can also make a parameter optional by giving it a default.
    RAVEN_DSN = config.get('RAVEN_DSN', default=None)

    # Last but not least, help_text is displayed in your Sphinx docs.
    SITE_NAME = config.get('SITE_NAME',
                           help_text="Displayed to users in the admin")
```

CHAPTER 3

Documentation Contents

API

Config

class `walter.config.`**`Config`**(*author*, *name*, *sources=None*, *search_path=None*)

Creates a config object.

Parameters

- **`author`** (*`str`*) – Name of the person or company that created this program. Used on Windows to set the default search path.
- **`name`** (*`str`*) – Name of this program. Used on Windows to set the default search path.
- **`sources`** (*`iterable`*) – An iterable of *`Source`* objects to pull configuration from. Defaults to the following:
  - *`EnvironmentSource`*
  - *`IniFileSource`*
- **`search_path`** (*`iterable`*) – An iterable of directories to search for configuration files. Defaults to the current directory, followed by an appropriate user and site config directory depending on the operating system.

`get`(*key*, *cast=None*, *help_text=None*)

Get a configuration parameter.

Parameters

- **`key`** (*`str`*) – The name of the configuration parameter to get.
- **`cast`** (*`function`*) – A function to call on the returned parameter to convert it to the appropriate value.
- **`help_text`** (*`str`*) – Help text to display to the user, explaining the usage of this parameter.

Sources

Built-In

class `walter.sources.`**`EnvironmentSource`**(*prefix=''*)

Source that extracts values from environment variables.

Parameters **`prefix`** (*`str`*) – Prefix to expect at the beginning of environment variable names.

class `walter.sources.`**`IniFileSource`**(*filename=None*, ***kwargs*)

Source that extracts values from `.ini` files.

Files should be in the format expected by `configparser.ConfigParser`.

Parameters **`section`** – Section header to look for settings under. Defaults

to `settings`. :type section: str

Creating Your Own

class `walter.sources.`**`Source`**

Base class for configuration sources.

To implement a simple (non-file-based) configuration source, subclass this class and override `__getitem__`.

`__getitem__` should return a string, or raise `KeyError` if a key isn't found in the configuration source.

If you are implementing an ambient configuration source (e.g. one that reads from environment variables, command-line args, a single file in a well-known location, or something else that doesn't depend on Walter's search path), you can expose your `Source` subclass to users directly. If instead you are implementing a file-based source, see also *`FileSource`*.

class `walter.sources.`**`FileSource`**(*filename=None*, ***kwargs*)

Base class for file-based configuration sources.

Because Walter implements searching for configuration files internally, and allows for a mix of different types of configuration files, a file-based configuration source consists of two classes.

One is the actual source itself. This is a subclass of *`Source`* — not this class — and behaves like a normal source, except it takes a file-like object as its first positional argument, and it is an implementation detail that is not exposed to your users.

The other is the "meta-source", which is a subclass of `FileSource`. It is responsible for two things: determining which filenames match the source, and creating new source objects from files. Users will create an instance of the meta-source and pass that to Walter, which will use it to create source instances.

While it is possible to override `match_filename()` and `create()` entirely, most meta-sources should be able to get by with simply setting two properties and adding a docstring:

- `source_class`, your actual source class.
- `pattern`, a default file pattern to match on, which can be either a shell glob or a compiled regular expression.

Unless you override `__init__`, your meta-source will accept a `filename` arg that allows users to override `pattern`; any other keyword arguments given to the meta-source will be passed through to the source itself.

`create`(*file_obj*)

Return a new source with the given file object.

Returns A new source object.

`match_filename` (*filename*)
Test a filename to see if it matches this source.

Returns Whether the filename matches this source.

Return type bool

Contribution Guide

Walter's code is currently hosted on GitLab at abre/walter. If you're not familiar with GitLab, it's very similar to GitHub; you can sign in with your GitHub account, and then fork, modify and file merge requests.

Setting Up

- To install Walter for development, run `pip install -e .[dev,docs]`.
- Tests are written using pytest; just run the command `pytest` to run them.
- Documentation is built with Sphinx. You can just run `cd docs; make html` and browse the generated HTML files, but if you install devd and modd, then run the command `modd`, you'll get a nice live-reloading view served on localhost port 8000 (or run e.g. `env PORT=1337 modd` to serve on a different port).

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Index