ConfigLoader Documentation

Release 1.0.1

Arthur Blair

October 14, 2015

Contents

ConfigLoader is a Python dictionary subclass that provides convenience methods for common app configuration-loading scenarios, inspired by flask.Config.

CHAPTER 1

Features

Easily load config settings from:

- Python modules, classes or objects
- JSON files
- YAML files
- Environment variables

Supports Python 2.6+ and 3.3+.

CHAPTER 2

Installation

Install ConfigLoader from PyPI using pip:

```
pip install configloader[all]
```

The `[all]` indicates that all optional dependencies (AttrDict and PyYAML) should be installed.

CHAPTER 3

Example usage

```
>>> from configloader import ConfigLoader
>>> config = ConfigLoader
>>> config.update_from_object('my_app.settings')
>>> config.update_from_yaml_env('YAML_SETTINGS_PATH')
```

CHAPTER 4

Documentation

https://configloader.readthedocs.org/en/stable/

CHAPTER 5

API Reference

class configloader.**ConfigLoader**(**args*, ***kwargs*)

A dict that supports common app configuration-loading scenarios.

If AttrDict is installed, then elements can be accessed as both keys and attributes.

update_from_object(*obj*, *criterion=<function <lambda>>*)

Update dict from the attributes of a module, class or other object.

By default only attributes with all-uppercase names will be retrieved. Use the `criterion` argument to modify that behaviour.

Parameters

- **obj** – Either the actual module/object, or its absolute name, e.g. 'my_app.settings'.
- **criterion** (`function`) – Callable that must return True when passed the name of an attribute, if that attribute is to be used.

New in version 1.0.

update_from_yaml_env(*env_var*)

Update dict from the YAML file specified in an environment variable.

The PyYAML package must be installed before this method can be used.

Parameters **env_var** (`str`) – Environment variable name.

update_from_yaml_file(*file_path_or_obj*)

Update dict from a YAML file.

The PyYAML package must be installed before this method can be used.

Parameters **file_path_or_obj** – Filepath or file-like object.

update_from_json_env(*env_var*)

Update dict from the JSON file specified in an environment variable.

Parameters **env_var** (`str`) – Environment variable name.

update_from_json_file(*file_path_or_obj*)

Update dict from a JSON file.

Parameters **file_path_or_obj** – Filepath or file-like object.

update_from_env_namespace(*namespace*)

Update dict from any environment variables that have a given prefix.

The common prefix is removed when converting the variable names to dictionary keys. For example, if the following environment variables were set:

```
MY_APP_SETTING1=foo
MY_APP_SETTING2=bar
```

Then calling `.update_from_env_namespace('MY_APP')` would be equivalent to calling `.update({'SETTING1': 'foo', 'SETTING2': 'bar'})`.

Parameters **namespace** – Common environment variable prefix.

update_from(*obj=None*, *yaml_env=None*, *yaml_file=None*, *json_env=None*, *json_file=None*, *env_namespace=None*)

Update dict from several sources at once.

This is simply a convenience method that can be used as an alternative to making several calls to the various `update_from_*()` methods.

Updates will be applied in the order that the parameters are listed below, with each source taking precedence over those before it.

Parameters

- **obj** – Object or name of object, e.g. 'myapp.settings'.
- **yaml_env** – Name of an environment variable containing the path to a YAML config file.
- **yaml_file** – Path to a YAML config file, or a file-like object.
- **json_env** – Name of an environment variable containing the path to a JSON config file.
- **json_file** – Path to a JSON config file, or a file-like object.
- **env_namespace** – Common prefix of the environment variables containing the desired config.

namespace(*namespace*, *key_transform=<function <lambda>>*)

Return a copy with only the keys from a given namespace.

The common prefix will be removed in the returned dict. Example:

```
>>> from configloader import ConfigLoader
>>> config = ConfigLoader(
...     MY_APP_SETTING1='a',
...     EXTERNAL_LIB_SETTING1='b',
...     EXTERNAL_LIB_SETTING2='c',
... )
>>> config.namespace('EXTERNAL_LIB')
ConfigLoader({'SETTING1': 'b', 'SETTING2': 'c'})
```

Parameters

- **namespace** – Common prefix.
- **key_transform** – Function through which to pass each key when creating the new dictionary.

Returns New config dict.

Return type *ConfigLoader*

namespace_lower(*namespace*)

Return a copy with only the keys from a given namespace, lower-cased.

The keys in the returned dict will be transformed to lower case after filtering, so they can be easily passed as keyword arguments to other functions. This is just syntactic sugar for calling *namespace()* with `key_transform=lambda key: key.lower()`.

Example:

```
>>> from configloader import ConfigLoader
>>> config = ConfigLoader(
...     MY_APP_SETTING1='a',
...     EXTERNAL_LIB_SETTING1='b',
...     EXTERNAL_LIB_SETTING2='c',
... )
>>> config.namespace_lower('EXTERNAL_LIB')
ConfigLoader({'setting1': 'b', 'setting2': 'c'})
```

Parameters **namespace** – Common prefix.

Returns New config dict.

Return type *ConfigLoader*

CHAPTER 6

Development

- history
- authors
- contributing

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

C

C

N

U