
ycsettings Documentation

Release 0.1.4

Yanchuan Sim

Oct 02, 2017

Contents

1	ycsettings	1
1.1	Example	1
2	Indices and tables	5

ycsettings is a utility module for handling app settings. It simplifies the searching of multiple sources (i.e., environment, files, etc) for settings and configuration variables.

Example

```
parser = ArgumentParser(description='Hello World!')
parser.add_argument('settings_uri', type=str, metavar='<config_file>', help=
    ↪ 'Positional option')
A = parser.parse_args()

settings_dict = {'A': 5}

settings = Settings(A, settings_dict, 's3://example/settings.yaml', search_first=['env',
    ↪ 'env_settings_uri'], warn_missing=False)

print(settings.getint('A', default=5, raise_exception=True))
```

Table of Contents:

Settings object API

This page documents the Settings object and its API.

```
class ycsettings.settings.Settings (*sources, search_first=['env',
    'env_settings_uri'], case_sensitive=False,
    raise_exception=False, warn_missing=False,
    env_settings_uri_keys=['SETTINGS_URI'],
    dict_settings_uri_keys=['settings', 'settings_uri'], ob-
    ject_settings_uri_keys=['settings', 'settings_uri'])
```

This class manages the lookup and prioritizing of setting variables in multiple different sources. Supported sources include:

- Environment (*env*): The OS environment, i.e., `os.environ`
- URI/files: Handles different file types including: JSON, YAML, and INI
- Python modules: Python modules similar to Django settings module; it can be a `.py` file or module path
- Dictionary-like objects: Objects with the `items` attribute
- Arbitrary objects: All `__dict__` entries not starting with `__` are used as settings

If `settings_uri` is found in the environment (`SETTINGS_URI`), dictionary, or arbitrary object, it will also load the corresponding settings URI in addition to the object itself. This way, it can autoloading from `SETTINGS_URI` in the environment and `argparse.Namespace`. For example,

```
parser = ArgumentParser(description='Hello World!')
parser.add_argument('settings_uri', type=str, metavar='<config_file>', help=
    ↪'Positional option')
A = parser.parse_args()

settings = Settings(A, warn_missing=False)
```

The file specified in `A.settings_uri` will be loaded.

```
__init__(*sources, search_first=['env', 'env_settings_uri'], case_sensitive=False,
    raise_exception=False, warn_missing=False, env_settings_uri_keys=['SETTINGS_URI'],
    dict_settings_uri_keys=['settings', 'settings_uri'], object_settings_uri_keys=['settings',
    'settings_uri'])
```

Initializes the `Settings` object.

Parameters

- **sources** (*list*) – list of sources to search for settings
- **search_first** (*list*) – list of sources which will be searched first before any other sources specified in `sources`.
- **case_sensitive** (*bool*) – whether to make case sensitive comparisons for settings key
- **raise_exception** (*bool*) – whether to raise a `MissingSettingException` exception when the setting is not found
- **warn_missing** (*bool*) – whether to display a warning when the setting is not found
- **env_settings_uri_keys** (*str*) – keys to find settings in the environment; if multiple keys are found, they'll all be used
- **dict_settings_uri_keys** (*str*) – keys to find settings in a `dict()`-like object; if multiple keys are found, they'll all be used
- **object_settings_uri_keys** (*str*) – keys to find settings in an arbitrary object; if multiple keys are found, they'll all be used

```
get(key, *, default=None, cast_func=None, case_sensitive=None, raise_exception=None,
    warn_missing=None, use_cache=True, additional_sources=[])
```

Gets the setting specified by `key`. For efficiency, we cache the retrieval of settings to avoid multiple searches through the sources list.

Parameters

- **key** (*str*) – settings key to retrieve
- **default** (*str*) – use this as default value when the setting key is not found
- **cast_func** (*func*) – cast the value of the settings using this function

- **case_sensitive** (*bool*) – whether to make case sensitive comparisons for settings key
- **raise_exception** (*bool*) – whether to raise a `MissingSettingException` exception when the setting is not found
- **warn_missing** (*bool*) – whether to display a warning when the setting is not found
- **additional_sources** (*list*) – additional sources to search for the key; note that the values obtained here could be cached in a future call

Returns the setting value

Return type `str`

getbool (*key*, ***kwargs*)

Gets the setting value as a `bool()` by cleverly recognizing true values.

Return type `bool`

getdict (*key*, ***kwargs*)

Gets the setting value as a `dict`.

Return type `dict`

getfloat (*key*, ***kwargs*)

Gets the setting value as a `float`.

Return type `float`

getint (*key*, ***kwargs*)

Gets the setting value as a `int`.

Return type `int`

getlist (*key*, *delimiter=' , '*, ***kwargs*)

Gets the setting value as a `list`; it splits the string using `delimiter`.

Parameters **delimiter** (*str*) – split the value using this delimiter

Return type `list`

getnjobs (*key*, ***kwargs*)

Gets the setting value as an integer relative to the number of CPU. See `ycsettings.settings.parse_n_jobs()` for parsing rules.

Return type `int`

getserialized (*key*, *decoder_func=None*, ***kwargs*)

Gets the setting value as a `dict` or `list` trying `json.loads()`, followed by `yaml.load()`.

Return type `dict`, `list`

geturi (*key*, ***kwargs*)

Gets the setting value as a `urllib.parse.ParseResult`.

Return type `urllib.parse.ParseResult`

`ycsettings.settings.parse_n_jobs` (*s*)

This function parses a “math”-like string as a function of CPU count. It is useful for specifying the number of jobs.

For example, on an 8-core machine:

```
assert parse_n_jobs('0.5 * n') == 4
assert parse_n_jobs('2n') == 16
assert parse_n_jobs('n') == 8
assert parse_n_jobs('4') == 4
```

Parameters *s* (*str*) – string to parse for number of CPUs

Installation

Stable release

To install ycsettings, run this command in your terminal:

```
$ pip install ycsettings
```

This is the preferred method to install ycsettings, as it will always install the most recent stable release.

If you don't have `pip` installed, this [Python installation guide](#) can guide you through the process.

From sources

The sources for ycsettings can be downloaded from the [Github repo](#).

You can either clone the public repository:

```
$ git clone git://github.com/skylander86/ycsettings
```

Or download the [tarball](#):

```
$ curl -OL https://github.com/skylander86/ycsettings/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```


CHAPTER 2

Indices and tables

- `genindex`
- `search`

Symbols

`__init__()` (`ycsettings.settings.Settings` method), 2

G

`get()` (`ycsettings.settings.Settings` method), 2
`getbool()` (`ycsettings.settings.Settings` method), 3
`getdict()` (`ycsettings.settings.Settings` method), 3
`getfloat()` (`ycsettings.settings.Settings` method), 3
`getint()` (`ycsettings.settings.Settings` method), 3
`getlist()` (`ycsettings.settings.Settings` method), 3
`getnjobs()` (`ycsettings.settings.Settings` method), 3
`getserialized()` (`ycsettings.settings.Settings` method), 3
`geturi()` (`ycsettings.settings.Settings` method), 3

P

`parse_n_jobs()` (in module `ycsettings.settings`), 3

S

`Settings` (class in `ycsettings.settings`), 1