user_config Documentation

Release 1.0a10.post1

nihlaeth

Apr 09, 2017

Contents

manage user configuration for python projects

Contents:

CHAPTER 1

user_config

user_config package

Submodules

user_config.ini module

ini configuration file format.

user_config.ini.**ini_read**(_, *path*, *elements*)

Read ini configuration file and populate *data*.

Parameters

- **_** (user_config.Config) – IGNORED
- **path** (*pathlib.Path*) – path to configuration file
- **elements** (*Dict[ConfigElement]*) – configuration element tree

Raises None

Returns

Return type None

Examples

..doctest:

```
>>> TODO
```

user_config.ini.**ini_validate**(_, *elements*)

Make sure element tree is suitable for ini files.

Parameters

- **_** (user_config.Config) – IGNORED
- **elements** (*Dict[ConfigElement]*) – element tree

Raises InvalidConfigTree: – if the config tree is inappropriate for ini files

Returns

Return type None

Examples

..doctest:

```
>>> TODO
```

user_config.ini.**ini_write**(*_*, *elements*, *doc*)

Print default ini file.

This includes data already set in the existing configuration files.

Parameters

- **_** (user_config.Config) – IGNORED
- **elements** (*Dict[ConfigElement]*) – configuration element tree
- **doc** (*Option[str]*) – *Config* class docstring

Raises None

Returns

Return type None

Examples

..doctest:

```
>>> TODO
```

user_config.ini.**register_extension**()

Register ini file format functions with *user_config*.

Returns

Return type Dict

Examples

..doctest:

```
>>> register_extension()
{'read': <function ini_read at 0x...>, 'write': <function ini_write at 0x...>,
↪'validate': <function ini_validate at 0x...>, 'extension': 'cfg'}
```

Module contents

User config management.

class user_config.**BooleanListOption**(*doc=None*, *default=None*, *required=True*, *short_name=None*, *long_name=None*, *validate=None*, *additive=False*)

Bases: *user_config.StringListOption*

List configuration element with boolean content.

subtype
alias of bool

class user_config.**BooleanOption**(*doc=None*, *default=None*, *required=True*, *short_name=None*, *long_name=None*, *validate=None*)

Bases: *user_config.ConfigElement*

Configuration element with boolean value.

type_
alias of bool

class user_config.**Config**(*file_name='config'*, *global_path=None*, *user_path=None*, *cli=True*)

Bases: *user_config.MappingMixin*

Base class for application configuration.

Keyword Arguments

- **file_name** (*str, optional*) – name of the configuration file, defaults to config
- **global_path** (*pathlib.Path, optional*) – overwrite system global configuration path, defaults to None
- **user_path** (*pathlib.Path, optional*) – overwrite system user configuration path, defaults to None
- **cli** (*bool, optional*) – whether to parse commandline arguments, defaults to True

Raises

- AttributeError: – if *application* or *author* is not set
- InvalidConfigTree: – if configuration tree is inappropriate for *file_type*
- InvalidData: – if user supplied invalid data for a configuration element
- MissingData: – if an element marked as required has no value

file_type
str – file type to use for configuration files

application
str – application name

author
str – application author

version
str, optional – application version (set if your configuration is version dependent)

Examples

..doctest:

```
>>> TODO
```

application = None

author = None

version = None

class user_config.**ConfigElement**(*doc=None*, *default=None*, *required=True*, *short_name=None*, *long_name=None*, *validate=None*)

Bases: object

Base class for configuration elements.

Keyword Arguments

- **doc** (*str, optional*) – documentation for this option, defaults to None
- **default** (*Any, optional*) – fallback value, defaults to None
- **required** (*bool, optional*) – MUST a value be present? If no default is provided, this can result in a MissingData exception. Defaults to True
- **short_name** (*str, optional*) – short name for use with command line arguments, defaults to None
- **long_name** (*str, optional*) – overwrite default name for command line arguments, defaults to None
- **validate** (*Callable[Any, None], optional*) – additional validation function, defaults to None

Raises InvalidData: – if default value does not pass validation

creation_counter
int – global count of config elements, used to maintain field order

element_name
str – name of instance, provided by containing class

type_
type – python type of variable that this class represents

action
str – action for argparse

Examples

..doctest:

```
>>> TODO
```

action = 'store'

construct_parser(*parser*)
Add self to parser.

Parameters **parser** (*argparse.ArgumentParser*) – the argument parser to add an option to

Raises None

Returns

Return type None

Examples

..doctest:

```
>>> TODO
```

creation_counter = 0

element_name = None

extract_data_from_parser(*command_line_arguments*)
Get value from parser.

Parameters **command_line_arguments** (*argparse.Namespace*) – parsed arguments

Raises None

Returns

Return type None

Examples

..doctest:

```
>>> TODO
```

get_default()
Return default value.

get_value()
Return current option value.

has_default()
Return True if element has a default value.

set_value(*value*)
Validate and store value.

type_
alias of str

validate(*value*)
Validate individual value.

Parameters **value** (*Any*) – to be validated

Raises InvalidData: – if validation fails

Returns

Return type None

Examples

..doctest:

```
>>> TODO
```

validate_data()
Validate data.

Raises

- InvalidData: – if validation fails
- MissingData: – if *self.required* is *True* and our value is *None*

Returns

Return type None

Examples

..doctest:

```
>>> TODO
```

class user_config.**ConfigMeta**
Bases: type

ORM-like magic for configuration class.

Gather all *ConfigElement* attributes into *_elements* and get correct *_validate*, *_read* and *_writer* functions.

Parameters

- **cls_name** (*str*) – class name
- **cls_parents** (*Tuple[class]*) – class parents
- **cls_attributes** (*Dict*) – class attributes

Raises

- AttributeError: – if class tries to overwrite a reserved attribute
- ImportError: – if no appropriate *entry_point* could be found for *file_type*

Examples

..doctest:

```
>>> TODO
```

class user_config.**FloatListOption**(*doc=None*, *default=None*, *required=True*, *short_name=None*, *long_name=None*, *validate=None*, *additive=False*)
Bases: *user_config.StringListOption*

List configuration element with float content.

subtype
alias of float

class user_config.**FloatOption**(*doc=None*, *default=None*, *required=True*, *short_name=None*, *long_name=None*, *validate=None*)
Bases: *user_config.ConfigElement*

Configuration element with float value.

type_
alias of float

class user_config.**IntegerListOption**(*doc=None*, *default=None*, *required=True*, *short_name=None*, *long_name=None*, *validate=None*, *additive=False*)
Bases: *user_config.StringListOption*

List configuration element with integer content.

subtype
alias of int

class user_config.**IntegerOption**(*doc=None*, *default=None*, *required=True*, *short_name=None*, *long_name=None*, *validate=None*)
Bases: *user_config.ConfigElement*

Configuration element with integer value.

type_
alias of int

exception user_config.**InvalidConfigTree**
Bases: Exception

Inappropriate configuration tree for file type.

exception user_config.**InvalidData**
Bases: Exception

User supplied invalid data for a configuration element.

class user_config.**MappingMixin**
Bases: object

Methods for emulating a mapping type.

get(*key*, *default*)
Get items without risking a KeyError.

items()
Return a view of dictionary key, value pairs.

keys()
Return a view of dictionary keys.

update(**args*, ***kwargs*)
Update more than one key at a time.

values()
Return a view of dictionary values.

exception user_config.**MissingData**
Bases: Exception

An element marked as required is missing a value.

class user_config.**Section**(*required=True*, *validate=None*)
Bases: *user_config.ConfigElement*, *user_config.MappingMixin*

Named container that contains ConfigElements.

Keyword Arguments

- **required** (*bool, optional*) – MUST section be present? If no default is provided for any required content elements, this can result in a MissingData exception. to find out if an optional section is complete, see *self.incomplete_count*. Defaults to True
- **validate** (*Callable[Any, None], optional*) – additional validation function, defaults to None

Raises AttributeError: – if content element is not a *ConfigElement*

creation_counter
int – global count of config elements, used to maintain field order

element_name
str – name of instance, provided by containing class

type_
type – python type of variable that this class represents

action
str – action for argparse

incomplete_count
int – Number of content elements which are required, but do not have a value. Useful for sections which are not marked as required, but do have required elements.

Examples

..doctest:

```
>>> TODO
```

construct_parser(*parser*)

extract_data_from_parser(*command_line_arguments*)

get_elements()
Return raw element tree, use with caution.

get_value()
Return current content value.

has_default()
Return True because Section always has a default value.

incomplete_count = 0

set_value(*value*)
Validate and store value.

type_
alias of dict

validate(*value*)

validate_data()

class user_config.**StringListOption**(*doc=None*, *default=None*, *required=True*, *short_name=None*, *long_name=None*, *validate=None*, *additive=False*)
Bases: *user_config.ConfigElement*

Configuration element with list value.

Keyword Arguments

- **doc** (*str, optional*) – documentation for this option, defaults to None
- **default** (*Any, optional*) – fallback value, defaults to None
- **required** (*bool, optional*) – MUST a value be present? If no default is provided, this can result in a MissingData exception. Defaults to True
- **short_name** (*str, optional*) – short name for use with command line arguments, defaults to None
- **long_name** (*str, optional*) – overwrite default name for command line arguments, defaults to None
- **validate** (*Callable[Any, None], optional*) – additional validation function, defaults to None
- **additive** (*bool, optional*) – whether to add all found lists together instead of overwrite them, defaults to False

Raises InvalidData: – if default value does not pass validation

creation_counter
int – global count of config elements, used to maintain field order

element_name
str – name of instance, provided by containing class

type_
type – python type of variable that this class represents

subtype
type – python type of list items

action
str – action for argparse

Examples

..doctest:

```
>>> TODO
```

action = 'append'

append(*value*)
Append value to option.

count(*value*)
Count occurrence of value.

extend(*extension*)
Extend value of option with extension.

extract_data_from_parser(*command_line_arguments*)

index(*value*)
Return index of first occurrence of value.

insert (*index*, *value*)
Insert value at index.

pop (*index=-1*)
Remove and return value at index.

remove (*value*)
Remove value.

reverse ()
Reverse list in place.

set_value (*value*)

sort (*key=None*, *reverse=False*)
Sort list in place.

subtype
alias of `str`

type_
alias of `list`

validate (*value*)

class user_config.**StringOption** (*doc=None*, *default=None*, *required=True*, *short_name=None*, *long_name=None*, *validate=None*)
Bases: *user_config.ConfigElement*

Configuration element with string value.

type_
alias of `str`

user_config.**with_metaclass** (*meta*, **bases*)
Create a base class with a metaclass.

Drops the middle class upon creation. Source: http://lucumr.pocoo.org/2013/5/21/porting-to-python-3-redux/

Parameters

- **meta** (*type*) – meta class or function
- ***bases** (*object*) – parents of class to be created

Raises None

Returns

Return type class of type *name*

Examples

..doctest:

```
>>> TODO
```

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

CHAPTER 3

user_config

Manage user configuration for python projects.

For easy and well-documented user-defined configuration.

CHAPTER 4

Links

- home: https://github.com/nihlaeth/user_config
- pypi: https://pypi.python.org/pypi/user-config
- documentation: http://user-config.readthedocs.io/en/latest/

CHAPTER 5

Badges

CHAPTER 6

Fallback order

1. command line arguments
2. user config files in `~/.config/<app>/config.<extension>`
3. global config files in `/etc/xdg/<app>/config.<extension>`
4. default values

For directories on operating systems than linux, see: https://github.com/ActiveState/appdirs

CHAPTER 7

Config format

Supported out of the box: ini

Other config formats can be supported via plug-ins.

CHAPTER 8

Requirements

- Linux, or Os X, or Windows (but not Windows Vista)
- python 2.7 or newer (python 3.6 supported)
- relatively new versions of setuptools and pip (version requirement to follow)

CHAPTER 9

Examples

Simple configuration example

```
"""Usage example for user_config."""
from user_config import Config, Section, StringOption, IntegerOption

class MyConfig(Config):

    """This will be displayed in the configuration documentation."""

    application = "my_application"
    author = "me"

    class GeneralSection(Section):
        """General information."""
        name = StringOption(
            doc="your name",
            default="unknown person")
        age = IntegerOption(
            doc="your age",
            required=True)
    general = GeneralSection()
    class AddressSection(Section):
        """shipping address"""
        street = StringOption(
            doc="street including house number",
            required=True)
        city = StringOption(required=True)
    address = AddressSection(required=False)

if __name__ == "__main__":
    CONFIG = MyConfig()
    print("hello there, {}!".format(CONFIG.general.name))
```

Command line help text:

```
$ python examples/simple_example.py -h
usage: my_application [-h] [--generate-config] [--city CITY] [--street STREET]
                      [--age AGE] [--name NAME]

This will be displayed in the configuration documentation. Command line
arguments overwrite configuration found in:
/root/.config/my_application/config.cfg /etc/xdg/my_application/config.cfg

optional arguments:
  -h, --help         show this help message and exit
  --generate-config  print a complete configuration file with current settings
  --city CITY
  --street STREET    street including house number
  --age AGE          your age
  --name NAME        your name
```

Command line use with default value:

```
$ python examples/simple_example.py --age 211
hello there, unknown person!
```

Command line use without required value:

```
$ python examples/simple_example.py
Traceback (most recent call last):
  File "examples/simple_example.py", line 29, in <module>
    CONFIG = MyConfig()
  File "/git/user_config/user_config/user_config/__init__.py", line 622, in __init__
    self._elements[element].validate_data(self._data)
  File "/git/user_config/user_config/user_config/__init__.py", line 464, in validate_
↪data
    self._elements[element].validate_data(self._data)
  File "/git/user_config/user_config/user_config/__init__.py", line 380, in validate_
↪data
    self.element_name))
user_config.MissingData: no value was provided for required option age
```

Command line use:

```
$ python examples/simple_example.py --age 211 --name mystery_user
hello there, mystery_user!
```

Generate configuration file:

```
$ python examples/simple_example.py --generate-config
## This will be displayed in the configuration documentation.

[general]
## General information.

## your name
# name = unknown person
name = tamara

## your age
## REQUIRED
# age =
age =
```

```
[address]
## shipping address
## OPTIONAL_SECTION

## street including house number
## REQUIRED
# street =
street =

## REQUIRED
# city =
city =
```

CHAPTER 10

Documentation

```
$ pip install -e ".[doc]"
$ python setup.py build_sphinx
```

CHAPTER 11

Testing

- pytest
- pytest-cov
- coverage
- codacy-coverage

```
$ python -m pytest --cov=user_config --cov-report xml
```

CHAPTER 12

Planned features

- multi matching sections / wildcard sections
- yaml config format
- json config format
- hook for overwriting config from database or other storage function

Python Module Index

u

Index

R

S

T

U

V

W