pipcs

Göktuğ Karakaşlı

May 10, 2021

CONTENTS:

CHAPTER

ONE

REFERENCE

Config	Base class to create root configuration.
Choices	Specify valid choices for a variable.
Condition	Mark a variable as valid, only if the condition is hold.
required	Mark a variable as required.
InvalidChoiceError	Raised when the user tries to assign a non-valid variable to *pipcs.Choices* variable.
RequiredError	Raised if a user doesn't set *pipcs.required* variable in the inherited config.

class pipcs.**Config**(*dictionary={}*)

Base class to create root configuration.

Parameters **dictionary** (Union[dict, Config], optional) – If it is a *pipcs.Config*, it will inherit the base configuration.

check_config()

Check configuration if all of the variables are valid.

```
from pipcs import Config, Required, required

config = Config()

@config('example')
class Example():
    variable: Required[int] = required

config.check_config()
# Raises: pipcs.pipcs.RequiredError: variable is required!
```

get_value(*key*, *check=False*)

Return value of the variable.

Parameters **check** (*bool*) – If true, the variable will be checked if it is valid or not.

```
from pipcs import Config, Required, required

config = Config()

@config('example')
class Example():
    variable: Required[int] = required
```

(continues on next page)

(continued from previous page)

```
print(config.example.get_value('variable'))
# <class 'pipcs.pipcs.required'>

print(config.example.get_value('variable', check=True))
# pipcs.pipcs.RequiredError: variable is required!
```

to_dict(*check=False*)

Convert *pipcs.Config* to dict. If the *pipcs.Condition* holds for a variable it will be included in the dictionary. *pipcs.Choices* variables will be converted in to their default values.

Parameters **check** (*bool*) – If true, the variables will be checked if they are valid or not.

class pipcs.**Choices**(*choices: List[pipcs.pipcs.T], default=<class 'pipcs.pipcs.required'>*)

Specify valid choices for a variable.

pipcs.InvalidChoiceError **error will be raised when the user** tries to set the variable to a non-valid choice in the inherited configuration.

Parameters

- **choices** (*List[T]*) – Valid choices for the configuration variable.
- **default** (*Required[T]*) – If the variable is not set by user the default value will be returned.

```
from pipcs import Config, Choices

config = Config()

@config('example')
class Example():
    variable: Choices[int] = Choices([1, 2, 3])

user_config = Config(config)

@user_config('example')
class UserExample():
    variable = 1

print(user_config.example.variable)
# 1

user_config = Config(config)

@user_config('example')
class UserExample():
    variable = 4
# Raises: pipcs.pipcs.InvalidChoiceError: 4 is not valid for variable, valid␣
↪choices: [1, 2, 3]
```

class pipcs.**Condition**(*data: pipcs.pipcs.T, comp: pipcs.pipcs.Comparison*)

Mark a variable as valid, only if the condition is hold. It is used combined with *pipcs.Choices*.

Parameters

- **data** (*T*) – Value of the variable.
- **comp** – Comparison function.

```python
from pipcs import Config, Choices, Condition

config = Config()

@config('example')
class Example():
    variable: Choices[int] = Choices([1, 2, 3])
    conditional_variable: Condition[int] = Condition(5, variable==2)

# Example 1
user_config = Config(config)

@user_config('example')
class UserExample():
    variable = 2

print(user_config.example.to_dict())
# {'variable': 2, 'conditional_variable': 5}

# Example 2
user_config = Config(config)

@user_config('example')
class UserExample():
    variable = 2
    conditional_variable = 1

print(user_config.example.to_dict())
# {'variable': 2, 'conditional_variable': 1}

# Example 3
user_config = Config(config)

@user_config('example')
class UserExample():
    variable = 1
    conditional_variable = 2

print(user_config.example.to_dict())
# {'variable': 1}
```

class pipcs.**required**
Mark a variable as required.

class pipcs.**InvalidChoiceError**
Raised when the user tries to assign a non-valid variable to *pipcs.Choices* variable.

class pipcs.**RequiredError**
Raised if a user doesn't set *pipcs.required* variable in the inherited config. It is also raised if a *pipcs.required* variable is not set during *pipcs.Config.check_config()*.

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

INDEX