# EasyCo Documentation

***Release beta***

**spacemanspiff2007**

**Nov 03, 2020**

# USER DOCUMENTATION


**1 Goal** **3**
- 1.1 Usage . . . 3
- 1.2 Examples . . . 3

**2 Configuration** **5**
- 2.1 Example lowercase keys . . . 5
- 2.2 Example lowercase keys . . . 5

**3 Class reference** **7**
- 3.1 ConfigFile . . . 7
- 3.2 ConfigContainer . . . 7
- 3.3 PathContainer . . . 8
- 3.4 ConfigEntry . . . 8

**4 Index** **9**

**Python Module Index** **11**

**Index** **13**

Easy configuration for Python programs with yaml files!

EasyCo can automatically create a default configuration from provided default values and will validate the provided data. It also provides auto complete in the IDE!

CHAPTER

# ONE

# GOAL

The goal of **EasyCo** is to provide an **easy** way of **Co**nfiguration using yaml files for Python programs. It can automatically create a default configuration from provided default values and will validate the provided data.

## 1.1 Usage

Just derive a class from *ConfigContainer* or *ConfigFile*. On Instantiation all class variables with type-hints are used. If you want more control over validators, default values or even add a description as a comment to the generated file use *ConfigEntry*.

## 1.2 Examples

### 1.2.1 Simple Example

Program code

```
from EasyCo import ConfigFile, ConfigContainer

class MyContainer(ConfigContainer):
    SubValueA: int
    SubValueB: int = 7

class MyConfigFile(ConfigFile):
    ConfValueA: int = 5
    ConfValueB: float = 5.5

    sub_values = MyContainer()

cfg = MyConfigFile('test')
cfg.load()
```

Created .yml file:

```
confvaluea: 5
confvalueb: 5.5
mycontainer:
  subvalueb: 7
```

AutoComplete in the IDE does work, too!

Accessing the loaded values is very easy:

```
if cfg.ConfValueA == 'ValueA':
    print('Test')
```

### 1.2.2 Example ConfigEntry

Program code

```
from EasyCo import ConfigFile, ConfigContainer, ConfigEntry

class MyContainer(ConfigContainer):
    SubValueA: int = ConfigEntry(required=True, default=5, description='This is␣
↪SubValueA')
    SubValueB: int = 7

class MyConfigFile(ConfigFile):
    ConfValueA: int = 5
    ConfValueB: float = 5.5

    sub_values = MyContainer()

cfg = MyConfigFile('test')
cfg.load()
```

Created .yml file:

```
confvaluea: 5
confvalueb: 5.5
mycontainer:
  subvaluea: 5  # This is SubValueA
  subvalueb: 7
```

### 1.2.3 Skipping Values

Use the `SKIP` value to define variables but prevent them from being processed. You can then set them in *`on_all_values_set`* or whenever you like

Example:

```
from EasyCo import ConfigContainer, ConfigEntry, SKIP

class MyContainer(ConfigContainer):
    ValueA: int = 5      # this one will be loaded from the file
    ValueB: int = SKIP   # this one will be ignored

    # override this function, it will be called when all values have been set
    def on_all_values_set(self):
        self.ValueB = self.ValueA + 5
```

CHAPTER

TWO

# CONFIGURATION

**class** EasyCo.**EasyCoConfig**
Control behaviour of EasyCo

**Variables**

- **lower_case_keys** (*bool*) – this will create and enforce lowercase keys
- **create_optional_keys** (*bool*) – when creating a config file this will create optional keys, too

The configuration can be set by modifying `EasyCo.DEFAULT_CONFIGURATION` or by creating an instance of *EasyCoConfig* in a container

## 2.1 Example lowercase keys

```
import io
from EasyCo import ConfigFile, DEFAULT_CONFIGURATION
DEFAULT_CONFIGURATION.lower_case_keys = True

class MyConfigFile(ConfigFile):
    ConfValueA: int = 5
    ConfValueB: float = 5.5

cfg = MyConfigFile()
```

```
confvaluea: 5
confvalueb: 5.5
```

## 2.2 Example lowercase keys

This example also shows a per container configuration

```
import io
from EasyCo import ConfigFile, EasyCoConfig, DEFAULT_CONFIGURATION
DEFAULT_CONFIGURATION.lower_case_keys = True

class MyConfigFile(ConfigFile):
    ConfValueA: int = 5
    ConfValueB: float = 5.5
```

(continues on next page)

(continued from previous page)

```
    # name of config doesn't matter and can be private so it doesn't show up in auto-
↪complete
    __cfg = EasyCoConfig()
    __cfg.lower_case_keys = False

cfg = MyConfigFile()
```

```
ConfValueA: 5
ConfValueB: 5.5
```

CHAPTER

# THREE

# CLASS REFERENCE

## 3.1 ConfigFile

**class** EasyCo.**ConfigFile**(*path=None*)

**set_path**(*path*)
Set the path to the configuration file. If no file extension is specified `.yml` will be automatically appended.

**Parameters** **path** (`Union[Path, str]`) – Path obj or str

**load**(*path=None*)
Load values from the configuration file. If the file doesn't exist it will be created. Missing required config entries will also be created.

**Parameters** **path** (`Optional[Path]`) – if not already set a path instance to the config file

## 3.2 ConfigContainer

**class** EasyCo.**ConfigContainer**(*key_name=None*)

**on_set_value**(*var_name*, *new_value*)
Override this function to perform datatype conversions when values get loaded from the file

**Parameters**

- **var_name** (`str`) – variable name
- **new_value** – new value which was loaded from file

**Returns** Value which will be set

**on_all_values_set**()
Override this function. It'll be called when all values from the file have been correctly set. Use it e.g. to calculate and set additional variables.

**subscribe_for_changes**(*func*)
When a value in this container changes the passed function will be called.

**Parameters** **func** – function which will be called

## 3.3 PathContainer

**class** EasyCo.**PathContainer**

Container which converts all values with type `str` to `Path` objects. Relative paths will be resolved in relation to the folder where the config file is.

## 3.4 ConfigEntry

**class** EasyCo.**ConfigEntry**(*default=MISSING*, *default_factory=MISSING*, *validator=MISSING*, *required=True*, *description=''*, *key_name=None*)

**__init__**(*default=MISSING*, *default_factory=MISSING*, *validator=MISSING*, *required=True*, *description=''*, *key_name=None*)

ConfigEntry

**Parameters**

- **default** – Default value for this entry
- **default_factory** (`Callable[[], Any]`) – Factory function which creates the default value, use for list, dict, etc.
- **validator** – validator which validates the loaded values
- **required** (`bool`) – is this entry required
- **description** (`str`) – description of this entry which will also be added to the config file as a comment
- **key_name** (`Optional[str]`) – key name in the config file

CHAPTER

FOUR

# INDEX

genindex

# PYTHON MODULE INDEX

## e

EasyCo, 6

# INDEX

## Symbols

__init__() (*EasyCo.ConfigEntry method*), 8

## C

ConfigContainer (*class in EasyCo*), 7
ConfigEntry (*class in EasyCo*), 8
ConfigFile (*class in EasyCo*), 7

## E

EasyCo
    module, 6
EasyCoConfig (*class in EasyCo*), 5

## L

load() (*EasyCo.ConfigFile method*), 7

## M

module
    EasyCo, 6

## O

on_all_values_set() (*EasyCo.ConfigContainer method*), 7
on_set_value() (*EasyCo.ConfigContainer method*), 7

## P

PathContainer (*class in EasyCo*), 8

## S

set_path() (*EasyCo.ConfigFile method*), 7
subscribe_for_changes() (*EasyCo.ConfigContainer method*), 7