# Pydantic Settings

*Release 0.2.0*

Daniel Daniels <danields761@gmail.com>

Nov 22, 2020

# CONTENTS

A set of tools helping to manage and work with application settings

# CHAPTER ONE

# GETTING STARTED

*Pydantic Settings* package available on PyPI

```
pip install pydantic-settings
```

CHAPTER

TWO

# MANUAL BY EXAMPLES

## 2.1 Environment variables

Override settings values by env variables even for nested fields

```
from pydantic import BaseModel
from pydantic_settings import BaseSettingsModel, load_settings


class ComponentOptions(BaseModel):
    val: str


class AppSettings(BaseSettingsModel):
    class Config:
        env_prefix = 'FOO'

    component: ComponentOptions


assert (
    load_settings(
        AppSettings,
        '{}',
        load_env=True,
        type_hint='json',
        environ={'FOO_COMPONENT_VAL': 'SOME VALUE'},
    ).component.val
    == 'SOME VALUE'
)
```

It's not necessary to override `BaseSettingsModel` in order to use `load_settings()` functionality. It also works with plain `pydantic.BaseModels` subclasses. Specify `env_prefix` in order to override default `"APP"` prefix.

```
from pydantic import BaseModel
from pydantic_settings import load_settings


class Foo(BaseModel):
    val: int
```

(continues on next page)

(continued from previous page)

```
assert (
    load_settings(
        Foo, load_env=True, env_prefix='EX', environ={'EX_VAL': '10'}
    ).val
    == 10
)
```

## 2.2 Rich location specifiers

Also `load_settings()` provides rich information about location of a wrong value inside the source.

### 2.2.1 Location inside text content

```
from pydantic import ValidationError, IntegerError
from pydantic_settings import BaseSettingsModel, load_settings, TextLocation
from pydantic_settings.errors import ExtendedErrorWrapper


class Foo(BaseSettingsModel):
    val: int


try:
    load_settings(Foo, '{"val": "NOT AN INT"}', type_hint='json')
except ValidationError as e:
    err_wrapper, *_ = e.raw_errors
    assert isinstance(err_wrapper, ExtendedErrorWrapper)
    assert isinstance(err_wrapper.exc, IntegerError)
    assert err_wrapper.source_loc == TextLocation(
        line=1, col=9, end_line=1, end_col=21, pos=9, end_pos=20
    )
else:
    raise Exception('must rise error')
```

### 2.2.2 Location among environment variables

Also saves exact env variable name

```
from pydantic import ValidationError, IntegerError
from pydantic_settings import BaseSettingsModel, load_settings
from pydantic_settings.errors import ExtendedErrorWrapper


class Foo(BaseSettingsModel):
    val: int


try:
    load_settings(Foo, load_env=True, environ={'APP_val': 'NOT AN INT'})
except ValidationError as e:
    err_wrapper, *_ = e.raw_errors
```

(continues on next page)

(continued from previous page)

```
    assert isinstance(err_wrapper, ExtendedErrorWrapper)
    assert isinstance(err_wrapper.exc, IntegerError)
    assert err_wrapper.source_loc == ('APP_val', None)
else:
    raise Exception('must rise error')
```

## 2.3 Extract attributes docstrings

By default, *pydantic* offers very verbose way of documenting fields, e.g.

```
class Foo(BaseModel):
    val: int = Schema(0, description='some valuable field description')
```

That verbosity may be avoided by extracting documentation from so called *attribute docstring*, which is, for reference, also supported by *sphinx-autodoc* (also there is very old rejected **PEP 224**, which proposes it), example:

```
from pydantic import BaseModel
from pydantic_settings import with_attrs_docs


@with_attrs_docs
class Foo(BaseModel):
    bar: str
    """here is docs"""

    #: docs for baz
    baz: int

    #: yes
    #: of course
    is_there_multiline: bool = True


assert Foo.__fields__['bar'].field_info.description == 'here is docs'
assert Foo.__fields__['baz'].field_info.description == 'docs for baz'
assert Foo.__fields__['is_there_multiline'].field_info.description == (
    'yes\nof course'
)
```

`BaseSettingsModel` does it automatically.

**Note:** Documented multiple-definitions inside class isn't supported because it is really unclear to which of definitions the docstring should belongs

## 2.4 API Reference

Also there is API Reference. It's a bit dirty, but still may provide helpful info.

CHAPTER

THREE

# INDICES AND TABLES

- genindex
- modindex
- search

# INDEX