envbox Documentation

Release 1.3.0

Igor ‘idle sign‘ Starikov

Jun 08, 2022

Contents

https://github.com/idlesign/envbox

CHAPTER 1

Description

Detect environment type and work within.

1.1 Features

- Environment type detection (extendable system);
- Support for `.env` files;
- Convenient `os.environ` proxying (with optional values casting into Python natives);
- Automatic submodule-for-environment import tool;
- Cosy per-thread settings container with environment var support;
- CLI for environment probing.

CHAPTER 2

Requirements

1. Python 3.7+
2. `click` package (optional, for CLI)

CHAPTER 3

Table of Contents

3.1 Quickstart

3.1.1 Basic usage

```
from envbox import get_environment

# Let's detect current environment type and get its object.
# * See and use `get_environment` function arguments to impose restrictions upon␣
↪detection system.
#
# Default detection sources:
# 1. ``PYTHON_ENV`` env variable
# 2. ``environment`` file contents
#
# By default this function will also try to read env variables from .env files.
env = get_environment()

env.name
# >> development

env.is_production
# >> False

env.get('HOME')
# The same as env['HOME'] and env.HOME
# >> /home/idle/

env.getmany('PYTHON')
# {'UNBUFFERED': '1', 'IOENCODING': 'UTF-8', 'PATH': ...}

# We can also try to cast env values into Python natives.
env.getmany_casted('PYTHON')
```

(continues on next page)

(continued from previous page)

```
# Note that `UNBUFFERED` is int now.
# {'UNBUFFERED': 1, 'IOENCODING': 'UTF-8', 'PATH': ...}
```

3.1.2 .env files as a source

You may want to put your environment vars into `.env` files (e.g.: `.env`, `.env.development` `.env.production`) to be read by `envbox`:

```
MY_VAR_1 = value1
HOME = /home/other/

# comments are ignored, just as lines without definitions

# mathing quotes (" and ') are stripped
MY_QUOTED = "some quoted "

# ${VARNAME} will be replaced by value from env (if available)
MY_VAR_2 = ${MY_QUOTED}

# multiline with dangling quotes
MULTI_1 = "
line1
line2
"

# multiline classic
MULTI_2 = "line1
line2
line3"

# multiline as one line
MULTI_3 = "one\ntwo"
```

`envbox` will try to load such files from the current working directory for the current environment type automatically.

3.1.3 Settings container

If you need a per-thread settings storage you can do the following:

```
# Somewhere in your setting module declare settings:
class _Settings(SettingsBase):

    ONE = 1
    SOME = 'two'
    ANOTHER = True

Settings = _Settings()


# Now access those settings from other modules(s).
if Settings.ANOTHER:
    Settings.SOME = 'three'
```

Accessing any setting which was not set in the session, will lead to appropriate environment variable probing.

3.1.4 Environment type aliases

```
from envbox import get_environment, PRODUCTION

# Let's make `prod` string identify production environment.
register_type(PRODUCTION, alias='prod')

# Now if someone has used `prod`
# we correctly identify it as production environment.
get_environment().is_production  # True
```

3.1.5 Automatic submodule import

envbox features `import_by_environment()` function which automatically imports symbols of a submodule of a package for the given (or detected) environment into globals of an entry-point submodule.

Note: This could be useful not only for Django-projects where submodule-based settings definition is rather usual but also for various other cases.

Example:

```
- project
--- __init__.py
--- settings.py
--- settings_development.py
```

1. Here `project` is a package available for import (note `__init__.py`).
2. `settings.py` is an entry point module for settings using `import_by_environment()`.

    ```
    from envbox import import_by_environment


    current_env = import_by_environment()

    print(f'Environment type: {current_env}')
    ```

3. `settings_development.py` is one of module files for certain environment (development).
4. `import_by_environment()` call in `settings.py` makes symbols from `settings_development.py` available from `settings.py`.

3.2 API

3.2.1 Basic

`envbox.base.`**`get_environment`**(*default: Union[str, envbox.envs.Environment, None] = 'development', detectors: List[envbox.detectors.Detector] = None, detectors_opts: dict = None, use_envfiles: bool = True*) → Optional[envbox.envs.Environment]

Returns current environment type object.

Parameters

- **default** – Default environment type or alias.
- **detectors** – List of environment detectors to be used in chain. If not set, default builtin chain is used.
- **detectors_opts** – Detectors options dictionary. Where keys are detector names and values are keyword arguments dicts.
- **use_envfiles** – Whether to set environment variables (if not already set) using data from .env files.

`envbox.base.`**`import_by_environment`**(*environment: envbox.envs.Environment = None*, *module_name_pattern: str = 'settings_%s'*, *silent: bool = False*, *package_name: str = None*) → Optional[envbox.envs.Environment]

Automatically imports symbols of a submodule of a package for given (or detected) environment into globals of an entry-point submodule.

Returns``Environment`` object if module is imported or `None`.

Example:

```
- project
--- __init__.py
--- settings.py
--- settings_development.py
```

- Here `project` is a package available for import (note `__init__.py`).
- `settings.py` is an entry point module for settings using `import_by_environment()`.
- `settings_development.py` is one of module files for certain environment (development).
- `import_by_environment()` call in `settings.py` makes symbols from `settings_development.py` available from `settings.py`.

Parameters

- **environment** –
- **module_name_pattern** – Environment submodule name pattern. `%s` will be replaced with environment name.
- **silent** – If `True` no import error (if any) will be raised.
- **package_name** – Name of the package holding settings file. We'll try to guess it if not provided.

  E.g.:

  – someproject.settings

  – someproject.inner.settings

3.2.2 Environments

class `envbox.envs.`**`Development`**(*name: str = None*, *type_cast: bool = None*)

Development (local) environment.

Parameters

- **`name`** – Environment name.

  Note: This will prevail over class attribute.

- **`type_cast`** – Whether to cast values into Python natives in .get() and .getmany() by default.

  Note: This will prevail over class attribute.

`drop` (*key: str*)
Removes key from environment.

`dropmany` (*keys: Sequence[str] = None*, *prefix: str = ”*)
Drops keys in batch mode.

Parameters

- **`keys`** – Keys to drop. If not set current env keys will be used.
- **`prefix`** –

`get` (*key: str*, *default: Any = None*, *type_cast: bool = None*) → Any
Get environment variable value.

Parameters

- **`key`** –
- **`default`** – Default value to return if no value found.
- **`type_cast`** – Try to cast value into Python native type.

`get_casted` (*key: str*, *default: Any = None*) → Any
The same as *get* but tries to cast values into Python natives.

`getmany` (*prefix: str = ”*, *type_cast: bool = None*) → dict
Returns a dictionary of values for keys the given prefix.

Parameters

- **`prefix`** –
- **`type_cast`** – Try to cast value into Python native type.

`getmany_casted` (*prefix: str = ”*) → dict
The same as *getnamy* but tries to cast values into Python natives.

`set` (*key: str*, *value: Any*, *overwrite: bool = True*)
Set environment variable.

Parameters

- **`key`** –
- **`value`** –
- **`overwrite`** – Whether to overwrite value if it’s already set.

`setmany` (*key_val: dict*, *prefix: str = ”*, *overwrite: bool = True*)
Sets values in batch mode.

Parameters

- **`key_val`** –
- **`prefix`** –
- **`overwrite`** – Whether to overwrite value if it's already set.

`update_from_envfiles`()

Updates environment variables (if not already set) using data from .env files.

Files used (as they read; values read later override previous values):

- .env
- .env.<env_name>
- .env.local
- .env.<env_name>.local

<env_name> - Environment name (e.g. `production`, `development` etc.)

class `envbox.envs.`**`Production`**(*name: str = None*, *type_cast: bool = None*)

Production (stable) environment.

Parameters

- **`name`** – Environment name.

  Note: This will prevail over class attribute.

- **`type_cast`** – Whether to cast values into Python natives in .get() and .getmany() by default.

  Note: This will prevail over class attribute.

`drop`(*key: str*)

Removes key from environment.

`dropmany`(*keys: Sequence[str] = None*, *prefix: str = ''*)

Drops keys in batch mode.

Parameters

- **`keys`** – Keys to drop. If not set current env keys will be used.
- **`prefix`** –

`get`(*key: str*, *default: Any = None*, *type_cast: bool = None*) → Any

Get environment variable value.

Parameters

- **`key`** –
- **`default`** – Default value to return if no value found.
- **`type_cast`** – Try to cast value into Python native type.

`get_casted`(*key: str*, *default: Any = None*) → Any

The same as *get* but tries to cast values into Python natives.

`getmany`(*prefix: str = ''*, *type_cast: bool = None*) → dict

Returns a dictionary of values for keys the given prefix.

Parameters

- **`prefix`** –
- **`type_cast`** – Try to cast value into Python native type.

`getmany_casted`(*prefix: str = ''*) → dict
The same as *getnamy* but tries to cast values into Python natives.

`set`(*key: str*, *value: Any*, *overwrite: bool = True*)
Set environment variable.

Parameters

- **`key`** –
- **`value`** –
- **`overwrite`** – Whether to overwrite value if it's already set.

`setmany`(*key_val: dict*, *prefix: str = ''*, *overwrite: bool = True*)
Sets values in batch mode.

Parameters

- **`key_val`** –
- **`prefix`** –
- **`overwrite`** – Whether to overwrite value if it's already set.

`update_from_envfiles`()
Updates environment variables (if not already set) using data from .env files.

Files used (as they read; values read later override previous values):

- .env
- .env.<env_name>
- .env.local
- .env.<env_name>.local

<env_name> - Environment name (e.g. `production`, `development` etc.)

class `envbox.envs.`**`Staging`**(*name: str = None*, *type_cast: bool = None*)
Staging (prestable) environment.

Parameters

- **`name`** – Environment name.

  Note: This will prevail over class attribute.

- **`type_cast`** – Whether to cast values into Python natives in .get() and .getmany() by default.

  Note: This will prevail over class attribute.

`drop`(*key: str*)
Removes key from environment.

dropmany (*keys: Sequence[str] = None*, *prefix: str = ''*)
Drops keys in batch mode.

Parameters

- **keys** – Keys to drop. If not set current env keys will be used.
- **prefix** –

get (*key: str*, *default: Any = None*, *type_cast: bool = None*) → Any
Get environment variable value.

Parameters

- **key** –
- **default** – Default value to return if no value found.
- **type_cast** – Try to cast value into Python native type.

get_casted (*key: str*, *default: Any = None*) → Any
The same as *get* but tries to cast values into Python natives.

getmany (*prefix: str = ''*, *type_cast: bool = None*) → dict
Returns a dictionary of values for keys the given prefix.

Parameters

- **prefix** –
- **type_cast** – Try to cast value into Python native type.

getmany_casted (*prefix: str = ''*) → dict
The same as *getnamy* but tries to cast values into Python natives.

set (*key: str*, *value: Any*, *overwrite: bool = True*)
Set environment variable.

Parameters

- **key** –
- **value** –
- **overwrite** – Whether to overwrite value if it's already set.

setmany (*key_val: dict*, *prefix: str = ''*, *overwrite: bool = True*)
Sets values in batch mode.

Parameters

- **key_val** –
- **prefix** –
- **overwrite** – Whether to overwrite value if it's already set.

update_from_envfiles ()
Updates environment variables (if not already set) using data from .env files.

Files used (as they read; values read later override previous values):

- .env
- .env.<env_name>
- .env.local
- .env.<env_name>.local

<env_name> - Environment name (e.g. `production`, `development` etc.)

class `envbox.envs.`**`Testing`**(*name: str = None*, *type_cast: bool = None*)

Testing environment.

Parameters

- **`name`** – Environment name.

  Note: This will prevail over class attribute.

- **`type_cast`** – Whether to cast values into Python natives in .get() and .getmany() by default.

  Note: This will prevail over class attribute.

`drop`(*key: str*)

Removes key from environment.

`dropmany`(*keys: Sequence[str] = None*, *prefix: str = ''*)

Drops keys in batch mode.

Parameters

- **`keys`** – Keys to drop. If not set current env keys will be used.
- **`prefix`** –

`get`(*key: str*, *default: Any = None*, *type_cast: bool = None*) → Any

Get environment variable value.

Parameters

- **`key`** –
- **`default`** – Default value to return if no value found.
- **`type_cast`** – Try to cast value into Python native type.

`get_casted`(*key: str*, *default: Any = None*) → Any

The same as *get* but tries to cast values into Python natives.

`getmany`(*prefix: str = ''*, *type_cast: bool = None*) → dict

Returns a dictionary of values for keys the given prefix.

Parameters

- **`prefix`** –
- **`type_cast`** – Try to cast value into Python native type.

`getmany_casted`(*prefix: str = ''*) → dict

The same as *getnamy* but tries to cast values into Python natives.

`set`(*key: str*, *value: Any*, *overwrite: bool = True*)

Set environment variable.

Parameters

- **`key`** –
- **`value`** –

- **`overwrite`** – Whether to overwrite value if it's already set.

`setmany`(*key_val: dict*, *prefix: str = ''*, *overwrite: bool = True*)
Sets values in batch mode.

Parameters

- **`key_val`** –
- **`prefix`** –
- **`overwrite`** – Whether to overwrite value if it's already set.

`update_from_envfiles`()
Updates environment variables (if not already set) using data from .env files.

Files used (as they read; values read later override previous values):

- .env
- .env.<env_name>
- .env.local
- .env.<env_name>.local

<env_name> - Environment name (e.g. `production`, `development` etc.)

`envbox.envs.`**`get_type`**(*cls_or_alias: Union[Type[Environment], str]*) → Type[envbox.envs.Environment]
Returns environment type by alias (or class itself)

Parameters **`cls_or_alias`** –

`envbox.envs.`**`register_type`**(*env_type: Union[Type[Environment], str]*, *alias: str = None*) → Type[envbox.envs.Environment]
Registers environment type.

Parameters

- **`env_type`** – Environment type or its alias (for already registered types).
- **`alias`** – Alias to register type under. If not set type name is used.

3.2.3 Environment detection

class `envbox.detectors.`**`Environ`**(***kwargs*)
Gets environment from OS environment variable.

class `envbox.detectors.`**`File`**(***kwargs*)
Gets environment from file.

`envbox.detectors.`**`get_detector`**(*cls_or_name: Union[Type[envbox.detectors.Detector], str]*) → Type[envbox.detectors.Detector]
Returns detector by alias (or class itself)

Parameters **`cls_or_name`** –

`envbox.detectors.`**`register_detector`**(*detector: Type[envbox.detectors.Detector]*)
Registers an environment detector.

Parameters **`detector`** –

3.2.4 Settings container

class envbox.settings.**SettingsBase**

Use this class as base for your classes containing settings.

Note: Settings are per-thread.

Every uppercase attribute of of a heir class will be treated as a setting.

Accessing any setting which was not set in the session, will lead to appropriate environment variable probing, thus:

1. current session value
2. environment value
3. default value

```
class _Settings(SettingsBase):

    ONE = 1
    SOME = 'two'
    ANOTHER = True

Settings = _Settings()

if Settings.ANOTHER:
    Settings.SOME = 'three'
```

get_environment() → Optional[Environment]

Return current environment.

This could be customized by a child if required.

Python Module Index

Index