
settei Documentation

Release 0.4.0

Spoqa, Inc

May 14, 2017

Contents

1	Loading a configuration is easy	3
1.1	settei — App object holding configuration	3
1.1.1	settei.base — Basic app object	4
1.1.2	settei.presets — Richer presets for several frameworks	6
1.1.3	settei.version — Version data	7
1.2	Changelog	8
1.2.1	Version 0.4.0	8
1.2.2	Version 0.3.0	8
1.2.3	Version 0.2.2	8
1.2.4	Version 0.2.0	8
1.2.5	Version 0.1.1	9
1.2.6	Version 0.1.0	9
2	Indices and tables	11
	Python Module Index	13

Configuration utility for common Python applications and services. FYI, “settei” () means settings in Japanese. :)

CHAPTER 1

Loading a configuration is easy

Suppose you use `Flask` with `Settei`.

```
from flask import Flask
from settei import Configuration, config_property

class WebConfiguration(Configuration):
    """Load Configuration::

        [web]
        debug = true

    """

    #: debug option
    debug = config_property('web.debug', bool, default=False)

conf = WebConfiguration.from_path(pathlib.Path('.') / 'dev.toml')
app = Flask(__name__)

if __name__ == '__main__':
    app.run(debug=conf.debug)
```

settei — App object holding configuration

copyright

3. 2016—2017 Spoqa, Inc.

license Apache License 2.0, see LICENSE for more details.

settei.base — Basic app object

New in version 0.2.0.

exception `settei.base.ConfigError`

The base exception class for errors related to `Configuration` and `config_property()`.

New in version 0.4.0.

exception `settei.base.ConfigKeyError`

An exception class rises when there's no a configuration key. A subtype of `ConfigError` and `KeyError`.

New in version 0.4.0.

exception `settei.base.ConfigTypeError`

An exception class rises when the configured value is not of a type the field expects.

New in version 0.4.0.

class `settei.base.Configuration` (*config: typing.Mapping[str, object] = {}, **kwargs*)

Application instance with its settings e.g. database. It implements read-only `Mapping` protocol as well, so you can treat it as a dictionary of string keys.

Changed in version 0.4.0: Prior to 0.4.0, it had raised Python's built-in `KeyError` on missing keys, but since 0.4.0 it became to raise `ConfigKeyError`, a subtype of `KeyError`, instead.

classmethod `from_file` (*file*) → `settei.base.Configuration`

Load settings from the given *file* and instantiate an `Configuration` instance from that.

Parameters *file* – the file object that contains TOML settings

Returns an instantiated configuration

Return type `Configuration`

classmethod `from_path` ()

Load settings from the given path and instantiate an `Configuration` instance from that.

Parameters *path* (`pathlib.Path`) – the file path that contains TOML settings

Returns an instantiated configuration

Return type `Configuration`

exception `settei.base.ConfigValueError`

An exception class rises when the configured value is somewhat invalid.

New in version 0.4.0.

exception `settei.base.ConfigWarning`

Warning category which raised when a default configuration is used instead due to missing required configuration.

class `settei.base.config_object_property` (*key: str, cls, docstring: str = None, **kwargs*) → `None`

Similar to `config_property` except it purposes to represent more complex objects than simple values. It can be utilized as dependency injector.

Suppose a field declared as:

```
from werkzeug.contrib.cache import BaseCache

class App(Configuration):
    cache = config_object_property('cache', BaseCache)
```


Also a configuration:

```
[cache]
class = "werkzeug.contrib.cache:RedisCache"
host = "a.nodes.redis-cluster.local"
port = 6379
db = 0
```

The above instantiates the following object:

```
from werkzeug.contrib.cache import RedisCache
RedisCache(host='a.nodes.redis-cluster.local', port=6380, db=0)
```

There's a special field named `*` which is for positional arguments as well:

```
[cache]
class = "werkzeug.contrib.cache:RedisCache"
"*" = [
    "a.nodes.redis-cluster.local",
    6379,
]
db = 0
```

The above configuration is equivalent to the following Python code:

```
from werkzeug.contrib.cache import RedisCache
RedisCache('a.nodes.redis-cluster.local', 6380, db=0)
```

New in version 0.4.0.

class `settei.base.config_property` (*key: str, cls, docstring: str = None, **kwargs*) → None
 Declare configuration key with type hints, default value, and docstring.

Parameters

- **key** (`str`) – the dotted string of key path. for example `abc.def` looks up `config['abc']['def']`
- **cls** (`type`) – the allowed type of the configuration
- **docstring** (`str`) – optional documentation about the configuration. it will be set to `__doc__` attribute
- **default** – keyword only argument. optional default value used for missing case. cannot be used with `default_func` at a time
- **default_func** (`collections.abc.Callable`) – keyword only argument. optional callable which returns a default value for missing case. it has to take an App mapping, and return a default value. cannot be used with `default` at a time
- **default_warning** (`bool`) – keyword only argument. whether to warn when default value is used. does not warn by default. this option is only available when `default` value is provided

Changed in version 0.4.0: Prior to 0.4.0, it had raised Python's built-in `KeyError` on missing keys, but since 0.4.0 it became to raise `ConfigKeyError`, a subtype of `KeyError`, instead.

In the same manner, while prior to 0.4.0, it had raised Python's built-in `TypeError` when a configured value is not of a type it expects, but since 0.4.0 it became to raise `ConfigTypeError` instead. `ConfigTypeError` is also a subtype of `TypeError`.

docstring

(*str*) The properly indented `__doc__` string.

`settei.base.get_union_types(type_)` → bool

Return a *tuple* of the given Union *type_*'s parameters.

```
>>> get_union_types(typing.Union[int, str, bool])
(int, str, bool)
```

If it's not an Union type or even not a type it returns None.

New in version 0.3.0.

settei.presets — Richer presets for several frameworks

New in version 0.2.0.

settei.presets.celery — Preset for Celery

`class settei.presets.celery.WorkerConfiguration` (*config: typing.Mapping[str, object] = {},*
***kwargs*)

The application object mixin which holds configuration for Celery.

on_worker_loaded (*app*)

Be invoked when a Celery app is ready.

Parameters *app* (`celery.Celery`) – a ready celery app

worker_broker_url

The url of the broker used by Celery. See also Celery's and Kombu's docs about broker urls:

<http://docs.celeryproject.org/en/latest/configuration.html#broker-url> <http://kombu.readthedocs.org/en/latest/userguide/connections.html#connection-urls>

worker_config

(`typing.Mapping[str, object]`) The configuration mapping for worker that will go to Celery.conf.

worker_result_backend

The backend used by Celery to store task results. See also Celery's docs about result backends:

<http://docs.celeryproject.org/en/latest/configuration.html#celery-result-backend>

worker_schedule

(`typing.Mapping[str, typing.Mapping[str, object]]`) The schedule table for Celery Beat, scheduler for periodic tasks.

There's some preprocessing before reading configuration. Since TOML doesn't have custom types, you can't represent `timedelta` or `crontab` values from the configuration file. To workaround the problem, it evaluates strings like `'f()'` pattern if they are appeared in a `schedule` field.

For example, if the following configuration is present:

```
[worker.celerybeat_schedule.add-every-30-seconds]
task = "tasks.add"
schedule = "timedelta(seconds=30)" # string to be evaluated
args = [16, 16]
```

it becomes translated to:

```

CELERYBEAT_SCHEDULE = {
    'add-every-30-seconds': {
        'task': 'tasks.add',
        'schedule': datetime.timedelta(seconds=30), # evaluated!
        'args': (16, 16),
    },
}

```

Note that although `timedelta` and `crontab` is already present in the context, you need to import things if other types. It can also parse and evaluate the patterns like `'module.path:func()'`.

Also `args` fields are translated from array to tuple.

See also Celery's docs about periodic tasks:

<http://docs.celeryproject.org/en/latest/userguide/periodic-tasks.html>

New in version 0.2.2.

`settei.presets.flask` — Preset for Flask apps

New in version 0.2.0.

`settei.presets.logging` — Preset for logging configuration

New in version 0.2.0.

Preset for apps holding `logging` configuration. Logging can be configured through TOML file e.g.:

```

[logging]
version = 1

[logging.loggers.flask]
handlers = ["stderr"]

[logging.loggers."urllib.request"]
handlers = ["stderr"]

[logging.loggers.werkzeug]
handlers = ["stderr"]

[logging.handlers.stderr]
class = "logging.StreamHandler"
level = "INFO"
stream = "ext://sys.stderr"

```

```

class settei.presets.logging.LoggingConfiguration (config: typing.Mapping[str, object] =
                                                    {}, **kwargs)

```

Hold configuration for `logging`.

```

configure_logging() → None
    Configure logging.

```

`settei.version` — Version data

New in version 0.2.0.

```
settei.version.VERSION = '0.4.0'
    (str) The version string e.g. '1.2.3'.
settei.version.VERSION_INFO = (0, 4, 0)
    (typing.Tuple[int, int, int]) The triple of version numbers e.g. (1, 2, 3).
```

Changelog

Version 0.4.0

Released on May 14, 2017.

- `config_object_property` was added. It's a kind of dependency injection, but very limited version.
- `ConfigError`, `ConfigKeyError`, `ConfigTypeError`, and `ConfigValueError`.

Prior to 0.4.0, `Configuration` had raised Python's built-in `KeyError` on missing keys, but since 0.4.0 it became to raise `ConfigKeyError`, a subtype of `KeyError`, instead.

In the same manner, while prior to 0.4.0, it had raised Python's built-in `TypeError` when a configured value is not of a type it expects, but since 0.4.0 it became to raise `ConfigTypeError` instead. `ConfigTypeError` is also a subtype of `TypeError`.

Version 0.3.0

Released on January 22, 2017.

- As `tsukkomi` is now abandoned, it's replaced by `typeguard`.

Version 0.2.2

Released on November 18, 2016. Note that the version 0.2.1 has never been released due to our mistake on versioning.

- `WorkerConfiguration` became to have `worker_schedule` config property to configure Celery beat — Celery's periodic tasks.

Version 0.2.0

Released on July 13, 2016.

- `settei` became a package (had been a module), which contains `settei.base` module.
- `settei.Configuration`, `settei.ConfigWarning`, and `settei.config_property` were moved to `settei.base` module. Although aliases for these previous import paths will be there for a while, we recommend to import them from `settei.base` module since they are deprecated.
- Presets were introduced: `settei.presets`.
 - `settei.presets.celery` is for configuring Celery apps.
 - `settei.presets.flask` is for configuring Flask web apps.
 - `settei.presets.logging` is for configuring Python standard logging system.
- `settei.version` module was added.
- `typeannotations` was replaced by `tsukkomi`.

- Settei now requires `pytoml` 0.1.10 or higher. (It had required 0.1.7 or higher.)

Version 0.1.1

Released on April 15, 2016.

- `settei.base.config_property` became to support `typing.Union` type.

Version 0.1.0

Released on April 1, 2016. Initial release.

CHAPTER 2

Indices and tables

- `genindex`
- `modindex`
- `search`

S

- `settei`, 3
- `settei.base`, 3
- `settei.presets`, 6
 - `settei.presets.celery`, 6
 - `settei.presets.flask`, 7
 - `settei.presets.logging`, 7
- `settei.version`, 7

C

[config_object_property](#) (class in [settei.base](#)), 4
[config_property](#) (class in [settei.base](#)), 5
[ConfigError](#), 4
[ConfigKeyError](#), 4
[ConfigTypeError](#), 4
[Configuration](#) (class in [settei.base](#)), 4
[configure_logging\(\)](#)
 [tei.presets.logging.LoggingConfiguration](#)
 method), 7
[ConfigValueError](#), 4
[ConfigWarning](#), 4

D

[docstring](#) ([settei.base.config_property](#) attribute), 5

F

[from_file\(\)](#) ([settei.base.Configuration](#) class method), 4
[from_path\(\)](#) ([settei.base.Configuration](#) class method), 4

G

[get_union_types\(\)](#) (in module [settei.base](#)), 6

L

[LoggingConfiguration](#) (class in [settei.presets.logging](#)), 7

O

[on_worker_loaded\(\)](#) (set-
 [tei.presets.celery.WorkerConfiguration](#)
 method), 6

S

[settei](#) (module), 3
[settei.base](#) (module), 3
[settei.presets](#) (module), 6
[settei.presets.celery](#) (module), 6
[settei.presets.flask](#) (module), 7
[settei.presets.logging](#) (module), 7

[settei.version](#) (module), 7

V

[VERSION](#) (in module [settei.version](#)), 7
[VERSION_INFO](#) (in module [settei.version](#)), 8

W

[worker_broker_url](#) (set-
 [tei.presets.celery.WorkerConfiguration](#) at-
 tribute), 6
[worker_config](#) ([settei.presets.celery.WorkerConfiguration](#)
 attribute), 6
[worker_result_backend](#) (set-
 [tei.presets.celery.WorkerConfiguration](#) at-
 tribute), 6
[worker_schedule](#) ([settei.presets.celery.WorkerConfiguration](#)
 attribute), 6
[WorkerConfiguration](#) (class in [settei.presets.celery](#)), 6