
settei Documentation

Release 0.3.0

Spoqa, Inc

Jan 26, 2017

Contents

1	Loading a configuration is easy	3
2	References	5
2.1	settei — App object holding configuration	5
2.1.1	settei.base — Basic app object	5
2.1.2	settei.presets — Richer presets for several frameworks	6
2.1.3	settei.version — Version data	8
3	Indices and tables	9
	Python Module Index	11

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)
```


2.1 settei — App object holding configuration

copyright

3. 2016 Spoqa, Inc.

license Apache License 2.0, see LICENSE for more details.

2.1.1 settei.base — Basic app object

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.

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.ConfigWarning`

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

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

docstring

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

2.1.2 settei.presets — Richer presets for several frameworks

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 0.2.1.

settei.presets.flask — Preset for Flask apps

settei.presets.logging — Preset for logging configuration

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`.

2.1.3 settei.version — Version data

`settei.version.VERSION = '0.3.0'`

(`str`) The version string e.g. '1.2.3'.

`settei.version.VERSION_INFO = (0, 3, 0)`

(`typing.Tuple[int, int, int]`) The triple of version numbers e.g. (1, 2, 3).

CHAPTER 3

Indices and tables

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

S

- `settei`, 5
- `settei.base`, 5
- `settei.presets`, 6
 - `settei.presets.celery`, 6
 - `settei.presets.flask`, 7
 - `settei.presets.logging`, 7
- `settei.version`, 7

C

`config_property` (class in `settei.base`), 5

`Configuration` (class in `settei.base`), 5

`configure_logging()` (`settei.presets.logging.LoggingConfiguration` method), 7

`ConfigWarning`, 5

D

`docstring` (`settei.base.config_property` attribute), 6

F

`from_file()` (`settei.base.Configuration` class method), 5

`from_path()` (`settei.base.Configuration` class method), 5

L

`LoggingConfiguration` (class in `settei.presets.logging`), 7

O

`on_worker_loaded()` (`settei.presets.celery.WorkerConfiguration` method), 6

S

`settei` (module), 5

`settei.base` (module), 5

`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`), 8

`VERSION_INFO` (in module `settei.version`), 8

W

`worker_broker_url` (`settei.presets.celery.WorkerConfiguration` attribute), 6

`worker_config` (`settei.presets.celery.WorkerConfiguration` attribute), 6

`worker_result_backend` (`settei.presets.celery.WorkerConfiguration` attribute), 6

`worker_schedule` (`settei.presets.celery.WorkerConfiguration` attribute), 6

`WorkerConfiguration` (class in `settei.presets.celery`), 6