invenio-config Documentation

Release 1.0.3

CERN

May 06, 2020

Contents

Invenio configuration loader.

Invenio-Config is a base package of the Invenio digital library framework. It is usually installed automatically as a dependency. It facilitates configuration loading from various sources such as a Python module, an instance folder or environment variables.

Further documentation is available on https://invenio-config.readthedocs.io/.

CHAPTER 1

User's Guide

This part of the documentation will show you how to get started in using Invenio-Config.

1.1 Installation

Invenio-Config is on PyPI so all you need is:

```
$ pip install invenio-config
```

1.2 Usage

Invenio configuration loader.

Invenio-Config is a *base package* of the Invenio digital library framework. It is usually installed automatically as a dependency. It should facilitate configuration loading from various sources to an application instance.

The following configuration loaders exists:

- *invenio_config.default.InvenioConfigDefault* - ensure required configuration values are set.
- *invenio_config.module.InvenioConfigModule* - for loading configuration from a Python module.
- *invenio_config.entrypoint.InvenioConfigEntryPointModule* - for loading configuration from a Python module specified by an entry point (by default `invenio_config.module`).
- *invenio_config.folder.InvenioConfigInstanceFolder* - for loading configuration from `cfg` file in an instance folder.
- *invenio_config.env.InvenioConfigEnvironment* - for loading configuration from environment variables with defined prefix (e.g. `INVENIO_SECRET_KEY`).

It also includes configuration loader factory that it is used to merge these sources in predefined order ensuring correct behavior in common scenarios.

1.2.1 Initialization

Following example needs a writable instance folder, hence we start by creating a temporary directory.

```
>>> import tempfile
>>> tmppath = tempfile.mkdtemp()
```

Now we can create a Flask application:

```
>>> from flask import Flask
>>> app = Flask('myapp', instance_path=tmppath, instance_relative_config=True)
```

1.2.2 Loaders

You can check default configuration values in newly created `app`.

```
>>> 'DEBUG' in app.config
True
>>> app.config.get('SECRET_KEY') is None
True
```

Default

The default configuration loader makes sure that the required configuration values are always loaded. You should call it **after** all configuration loaders have been already called.

The following default configuration values exist:

- `SECRET_KEY` - A secret key that will be used for securely signing the session cookie and can be used for any other security related needs.
- *`ALLOWED_HTML_TAGS`* - allowed tags used for html sanitizing by bleach.
- *`ALLOWED_HTML_ATTRS`* - allowed attributes used for html sanitizing by bleach.

The default configuration loader will warn if the `SECRET_KEY` is not defined:

```
>>> import warnings
>>> from invenio_config import InvenioConfigDefault
>>> with warnings.catch_warnings(record=True) as w:
...     config_default = InvenioConfigDefault(app=app)
...     assert len(w) == 1
>>> app.config['SECRET_KEY']
'CHANGE_ME'
```

Module

The module loader accepts an object and proxies the call to `flask.Config.from_object()`.

Here is an example of a configuration object:

```
>>> class Config:
...     EXAMPLE = 'module'
>>> from invenio_config import InvenioConfigModule
>>> config_module = InvenioConfigModule(app=app, module=Config)
```

(continues on next page)

(continued from previous page)

```
>>> app.config['EXAMPLE']
'module'
```

Entry point

The entry point loader works similar to the module loader, it just loads the config module from the entry point `invenio_config.module`:

```
>>> from invenio_config import InvenioConfigEntryPointModule
>>> config_ep = InvenioConfigEntryPointModule(app=app)
```

Instance Folder

The runtime configuration should be stored in a separate file, ideally located outiside the actual application package. The configuration files are handled as Python files where only variables in uppercase are stored in the application config.

```
>>> import os
>>> from invenio_config import InvenioConfigInstanceFolder
>>> with open(os.path.join(tmppath, 'myapp.cfg'), 'w') as f:
...     result = f.write("EXAMPLE = 'instance folder'")
>>> config_instance_folder = InvenioConfigInstanceFolder(app=app)
>>> app.config['EXAMPLE']
'instance folder'
```

Environment

Using environment variables is very handy when it comes to configuring connections to services like database, Redis server, RabbitMQ, etc. used via containers (e.g. Docker). In order to protect your application from reading environment variables set by the system or other applications, you should define a variable prefix used by the loader.

```
>>> os.environ['MYAPP_EXAMPLE'] = 'environment'
>>> from invenio_config import InvenioConfigEnvironment
>>> config_environment = InvenioConfigEnvironment(app=app, prefix='MYAPP_')
>>> app.config['EXAMPLE']
'environment'
```

You can also set more complex Python literal variables (e.g. dictionaries or lists):

```
>>> os.environ['MYAPP_COMPLEX'] = "{'items': [{'num': 42}, {'foo': 'bar'}]}"
>>> # ...or export MYAPP_COMPLEX="{'items': [{'num': 42}, {'foo': 'bar'}]}"
>>> config_environment = InvenioConfigEnvironment(app=app, prefix='MYAPP_')
>>> app.config['COMPLEX']
{'items': [{'num': 42}, {'foo': 'bar'}]}
```

1.2.3 Factory Pattern

The Invenio-Config comes with an opinionated way of loading configuration, that combines loaders in predictable way. You can use *`invenio_config.utils.create_config_loader()`* if you would like to:

1. Load configuration from `invenio_config.module` entry point group.

2. Load configuration from `config` module if provided as argument.
3. Load configuration from the instance folder: `<app.instance_path>/<app.name>.cfg`.
4. Load configuration keyword arguments provided.
5. Load configuration from environment variables with the prefix `env_prefix`.

```
>>> from invenio_config import create_config_loader
>>> app = Flask('myapp', instance_path=tmppath, instance_relative_config=True)
>>> config_loader = create_config_loader(config=Config, env_prefix='MYAPP')
>>> config_loader(app=app, MYARG='config loader')
>>> app.config['EXAMPLE']
'environment'
>>> app.config['MYARG']
'config loader'
```

CHAPTER 2

API Reference

If you are looking for information on a specific function, class or method, this part of the documentation is for you.

2.1 API Docs

2.1.1 Loaders

Invenio default configuration.

invenio_config.default.**ALLOWED_HTML_ATTRS = {'*': ['class'], 'a': ['href', 'title', 'name**
Allowed attributes used for html sanitizing by bleach.

invenio_config.default.**ALLOWED_HTML_TAGS = ['a', 'abbr', 'acronym', 'b', 'blockquote', 'br**
Allowed tags used for html sanitizing by bleach.

class invenio_config.default.**InvenioConfigDefault**(*app=None*)
Load configuration from module.

New in version 1.0.0.

Initialize extension.

init_app(*app*)
Initialize Flask application.

Invenio environment configuration.

class invenio_config.env.**InvenioConfigEnvironment**(*app=None*, *prefix='INVENIO_'*)
Load configuration from environment variables.

New in version 1.0.0.

Initialize extension.

init_app(*app*)
Initialize Flask application.

Invenio entry point module configuration.

class invenio_config.entrypoint.**InvenioConfigEntryPointModule**(*app=None*, *entry_point_group='invenio_config.module'*)

Load configuration from module defined by entry point.

Configurations are loaded in alphabetical ascending order, meaning that an entry point named `00_name` will be loaded first and an entry point named `10_name` will be loaded as last. This ensures that configurations defined in `10_name` app override configurations defined in `00_name` app.

New in version 1.0.0.

Initialize extension.

init_app(*app*)

Initialize Flask application.

Invenio instance folder configuration.

class invenio_config.folder.**InvenioConfigInstanceFolder**(*app=None*)

Load configuration from py file in folder.

If the application have instance relative config then the file will be read from the instance folder, otherwise it will be read from the application root path.

More about instance folders.

New in version 1.0.0.

Initialize extension.

init_app(*app*)

Initialize Flask application.

Invenio module configuration.

class invenio_config.module.**InvenioConfigModule**(*app=None*, *module=None*)

Load configuration from module.

New in version 1.0.0.

Initialize extension.

init_app(*app*)

Initialize Flask application.

2.1.2 Utilities

Default configuration loader usable by e.g. Invenio-Base.

invenio_config.utils.**create_conf_loader**(**args*, ***kwargs*)

Create a default configuration loader.

Deprecated since version 1.0.0b1: Use *create_config_loader()* instead. This function will be removed in version 1.0.1.

invenio_config.utils.**create_config_loader**(*config=None*, *env_prefix='APP'*)

Create a default configuration loader.

A configuration loader takes a Flask application and keyword arguments and updates the Flask application's configuration as it sees fit.

This default configuration loader will load configuration in the following order:

1. Load configuration from `invenio_config.module` entry points group, following the alphabetical ascending order in case of multiple entry points defined. For example, the config of an app with entry point name `10_app` will be loaded after the config of an app with entry point name `00_app`.
2. Load configuration from `config` module if provided as argument.
3. Load configuration from the instance folder: `<app.instance_path>/<app.name>.cfg`.
4. Load configuration keyword arguments provided.
5. Load configuration from environment variables with the prefix `env_prefix`.

If no secret key has been set a warning will be issued.

Parameters

- **config** – Either an import string to a module with configuration or alternatively the module itself.
- **env_prefix** – Environment variable prefix to import configuration from.

Returns A callable with the method signature `config_loader(app, **kwargs)`.

New in version 1.0.0.

CHAPTER 3

Additional Notes

Notes on how to contribute, legal information and changes are here for the interested.

3.1 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

3.1.1 Types of Contributions

Report Bugs

Report bugs at https://github.com/inveniosoftware/invenio-config/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

Write Documentation

Invenio-Config could always use more documentation, whether as part of the official Invenio-Config docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/inveniosoftware/invenio-config/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

3.1.2 Get Started!

Ready to contribute? Here's how to set up *invenio* for local development.

1. Fork the *invenio* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/invenio-config.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv invenio-config
$ cd invenio-config/
$ pip install -e .[all]
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass tests:

```
$ ./run-tests.sh
```

   The tests will provide you with test coverage and also check PEP8 (code style), PEP257 (documentation), flake8 as well as build the Sphinx documentation and run doctests.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -s -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

3.1.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests and must not decrease test coverage.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring.
3. The pull request should work for Python 2.7, 3.3, 3.4 and 3.5. Check https://travis-ci.com/inveniosoftware/invenio-config/pull_requests and make sure that the tests pass for all supported Python versions.

3.2 Changes

Version 1.0.3 (released 2020-05-06)

- Deprecated Python versions lower than 3.6.0. Now supporting 3.6.0 and 3.7.0.

Version 1.0.2 (released 2019-07-29)

- Added *ALLOWED_HTML_TAGS* and *ALLOWED_HTML_ATTRS* config keys.

Version 1.0.1 (released 2018-10-02)

- Application configurations are now sorted and loaded in alphabetical order.

Version 1.0.0 (released 2018-03-23)

- Initial public release.

3.3 License

3.4 Contributors

- Alexander Ioannidis
- Alizee Pace
- Jiri Kuncar
- Lars Holm Nielsen
- Paulina Lach
- Sami Hiltunen
- Tibor Simko

Python Module Index

i

Index