Yet Another Adapter Base Documentation

Release 0.2.2

Kolokotronis Panagiotis

May 03, 2020

Contents:

CHAPTER 1

Yet Another Adapter Base

YAAB aims to provide a flexible base for Adapter Design Pattern implementations based on dataclasses.

- Free software: MIT license
- Documentation: https://yaab.readthedocs.io.

1.1 Example Usage

Let's assume that you are interfacing an API that returns a *JSON* object with the following structure:

```
{
    "weird_name": "My name",
    "oid": "3"
}
```

And you would like to transform it into a schema that fits the rest of your API, let's assume:

```
{
    "name": "My name",
    "id": 3
}
```

Then you would define and use your model in the following way:

```
>>> from dataclasses import asdict, dataclass, field
>>>
>>> from yaab.adapter import BaseAdapter
>>>
```

(continues on next page)

(continued from previous page)

```
>>> @dataclass
... class MyModel(BaseAdapter):
...     id: int = field(metadata={"transformations": ("oid", int)})
...     name: str = field(metadata={"transformations": ("weird_name", )})
...
>>> m = MyModel.from_dict({"weird_name": "My name", "oid": "3"})
>>> print(m)
MyModel(id=3, name='My name')
>>> asdict(m)
{'id': 3, 'name': 'My name'}
```

1.2 Features

- TODO

1.3 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 2

Installation

2.1 Stable release

To install Yet Another Adapter Base, run this command in your terminal:

```
$ pip install yaab
```

This is the preferred method to install Yet Another Adapter Base, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for Yet Another Adapter Base can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/panagiks/yaab
```

Or download the tarball:

```
$ curl -OJL https://github.com/panagiks/yaab/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

To use Yet Another Adapter Base in a project, the base requirement is to create a dataclass that inherits from *BaseAdapter*. In order for *yaab* to know how to obtain the data for each of the dataclass' fields you need to provide a chain of *transformations*. *transformations* are by default provided in *field*'s *metadata* kwarg as a *tuple* under the key *transformations*.

If a *transformation* is another *dataclass* then its *from_any* classmethod is invoked with the element as it has been transformed up to now as its starting element. This allows for nesting but most importantly allows the innermost class to be agnostic as to how the encapsulating class will use it:

```
from dataclasses import asdict, dataclass, field

from yaab.adapter import BaseAdapter


@dataclass
class DbModel(BaseAdapter):
    host: str = field(metadata={"transformations": ("db_host", )})
    port: int = field(metadata={"transformations": ("db_port", int)})

@dataclass
class ConfigModel(BaseAdapter):
    name: str = field(metadata={"transformations": ("app_name", )})
    db: DbModel = field(metadata={"transformations": ("db_conf", DbModel)})

conf = ConfigModel.from_dict({"app_name": "My App", "db_conf": {"db_host": "my.db",
↪"db_port": 123}})
print(conf)
print(asdict(conf))
```

The above will print:

```
ConfigModel(name='My App', db=DbModel(host='my.db', port=123))
{'name': 'My App', 'db': {'host': 'my.db', 'port': 123}}
```

CHAPTER 4

yaab

4.1 yaab package

4.1.1 Submodules

4.1.2 yaab.adapter module

Main module.

class yaab.adapter.**BaseAdapter**
Bases: object

Base Class for Adapter pattern Models.

classmethod from_any(*element: Any, *, accessor: Optional[Callable[[Any, Any], Any]] = None, **kwargs*) → yaab.adapter.BaseAdapter

classmethod from_dict(*element: Dict[KT, VT], **kwargs*) → yaab.adapter.BaseAdapter

classmethod from_env(***kwargs*) → yaab.adapter.BaseAdapter

4.1.3 Module contents

Top-level package for Yet Another Adapter Base.

CHAPTER 5

Contributing

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

You can contribute in many ways:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/panagiks/yaab/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.

5.1.2 Fix Bugs

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

5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

5.1.4 Write Documentation

Yet Another Adapter Base could always use more documentation, whether as part of the official Yet Another Adapter Base docs, in docstrings, or even on the web in blog posts, articles, and such.

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/panagiks/yaab/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 :)

5.2 Get Started!

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

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

```
$ git clone git@github.com:your_name_here/yaab.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 yaab
$ cd yaab/
$ python setup.py develop
```

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 flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 yaab tests
$ python setup.py test or pytest
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

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

```
$ git add .
$ git commit -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.

5.3 Pull Request Guidelines

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

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.6, 3.7 and 3.8, and for PyPy. Check https://travis-ci.org/panagiks/yaab/pull_requests and make sure that the tests pass for all supported Python versions.

5.4 Tips

To run a subset of tests:

```
$ pytest tests.test_yaab
```

5.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bump2version patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER 6

Credits

6.1 Development Lead

- Kolokotronis Panagiotis <panagiks@gmail.com>

6.2 Contributors

None yet. Why not be the first?

CHAPTER 7

History

7.1 0.2.2 (2020-05-03)

- Introduce type conversion & convenience Mixins

7.2 0.2.1 (2020-02-18)

- Bugfix: from_env now utilizes default values.

7.3 0.2.0 (2020-02-18)

- Add dataclass nesting support.

7.4 0.1.0 (2020-01-21)

- First release on PyPI.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

y

Index

B

F

Y