NgoSchema

Release 1.0.10

Cédric ROMAN

Dec 29, 2022

CONTENTS

1 Overview **1**
- 1.1 Description . . . 1
- 1.2 Installation . . . 1
- 1.3 Documentation . . . 2
- 1.4 Development . . . 2

2 Installation **3**

3 Overview **5**
- 3.1 Description . . . 5
- 3.2 Installation . . . 5
- 3.3 Documentation . . . 6
- 3.4 Development . . . 6

4 Schemas **7**

5 Usage **9**
- 5.1 Generated Classes . . . 9

6 Contributing **11**
- 6.1 Bug reports . . . 11
- 6.2 Documentation improvements . . . 11
- 6.3 Feature requests and feedback . . . 11
- 6.4 Development . . . 12

7 Authors **13**

8 Changelog **15**
- 8.1 0.1.0 (2018-06-04) . . . 15

9 Indices and tables **17**

CHAPTER

ONE

OVERVIEW

docs	
tests	
package	

1.1 Description

I'm Cedric ROMAN.

`ngoschema` aims at automate the building of classes based on a JSON schema.

User can declare all class attributes in a schema (along with their type, default value) and the class will be built with accessors to check and validate data.

User can add methods and override setters/getters, but the library provides a boiler plate to automatically create the class, nicely instrumented (with loggers, exception handling, type checking, data validation, serialization, etc…).

The classbuilder allows to easily load definitions based on a canonical name and a namespace.

Instance of these classes can be iterated and behave as standard collections.

`ngoschema` aims at being a toolkit for Domain-Driven Design and Model-Driven Architecture.

The library is build on top of python-jsonchema, a python implementation for JSON schema validation.

- Free software: GNU General Public License v3

1.2 Installation

To install, with the command line:

```
pip install ngoschema
```

1.3 Documentation

https://python-ngoschema.readthedocs.io/

Settings are managed using simple-settings and can be overriden with configuration files (cfg, yaml, json) or with environment variables prefixed with **NGOSCHEMA_**.

1.4 Development

To run the all tests run:

```
tox
```

Note, to combine the coverage data from all the tox environments run:

Windows	`set PYTEST_ADDOPTS=--cov-append` `tox`
Other	`PYTEST_ADDOPTS=--cov-append tox`

CHAPTER

TWO

INSTALLATION

To install, with the command line:

```
pip install ngoschema
```

Settings are managed using simple-settings and can be overriden with configuration files (cfg, yaml, json) or with environment variables prefixed with `NGOSCHEMA_`.

CHAPTER

THREE

OVERVIEW

docs	
tests	
package	

3.1 Description

I'm Cedric ROMAN.

`ngoschema` aims at automate the building of classes based on a JSON schema.

User can declare all class attributes in a schema (along with their type, default value) and the class will be built with accessors to check and validate data.

User can add methods and override setters/getters, but the library provides a boiler plate to automatically create the class, nicely instrumented (with loggers, exception handling, type checking, data validation, serialization, etc…).

The classbuilder allows to easily load definitions based on a canonical name and a namespace.

Instance of these classes can be iterated and behave as standard collections.

`ngoschema` aims at being a toolkit for Domain-Driven Design and Model-Driven Architecture.

The library is build on top of python-jsonchema, a python implementation for JSON schema validation.

- Free software: GNU General Public License v3

3.2 Installation

To install, with the command line:

```
pip install ngoschema
```

3.3 Documentation

https://python-ngoschema.readthedocs.io/

Settings are managed using simple-settings and can be overriden with configuration files (cfg, yaml, json) or with environment variables prefixed with **NGOSCHEMA_**.

3.4 Development

To run the all tests run:

```
tox
```

Note, to combine the coverage data from all the tox environments run:

Windows	`set PYTEST_ADDOPTS=--cov-append` `tox`
Other	`PYTEST_ADDOPTS=--cov-append tox`

CHAPTER

FOUR

SCHEMAS

The library intends to deal with complex schemas, possibly using inheritance which is not yet supported in JSON Schema, as well as data types which can be useful in generated class.

For this purpose, a meta-schema is built on top of the standard ones, adding specific features, but which won't be recognized by standard validators. Though, the schema valid against this meta-schema should usually be processed without problem by standard validation libraries (with warnings for the unknown field) with the exception.

The additional grammar adds:

- **extra object attributes:**
  - `isAbtract` boolean to indicate an abstract class.
  - `extends` allowing to specify the `id` of parent classes.
  - `readOnly` and `notSerialized` to specify properties which cannot beset or are not serialized.
- extra literal types (`date`, `time`, `datetime`, `path`, `importable`).
- **extra property attributes for specific types:**
  - `isPathDir` boolean to indicate the path of a directory
  - `isPathFile` boolean to indicate the path of a file
  - `isPathExisting` boolean to indicate an existing path
  - `foreignKey` dictionary of options to define a foreign key to another object

It also comes with a few definitions that can be useful in a Domain-Driven Design implementation.

This meta-schema is available as `https://numengo.org/ngoschema <https://numengo.org/ngoschema#` and can be optionally referred as $schema. in the definitions (instead of the standard draft)

Additional types are available for literals, and can then be used already properly casted in further code. Those types are mapped as follows:

- `date`: `datetime.date`
- `datetime`: `datetime.datetime`
- `time`: `datetime.time`
- `time`: `datetime.time`
- `path`: `pathlib.Path`

CHAPTER

FIVE

USAGE

User can register json files as schemas in his module using `load_module_schemas("{module_folder}")` in the module `__init__.py`.

A proper JSON-schema document should have a property `$id` set to an absolute URI (it s domain/namespace).

To add a schema to a class, user needs to have the class use the `SchemaMetaclass` and can build a class refering to a domain/namespace which will be looked first in the available modules schemas, and eventually on-line. Some schemas from json-schema.org are included in the schemas directory of the module.

The library adds some meta-programing to create instrumented classes following a `ProtocolBase` One could create a class extending the Card class from json-schema.org as follows:

```
from future.utils import with_metaclass
from ngoschema.protocols import SchemaMetaclass, ProtocolBase

class MyCardClass(with_metaclass(SchemaMetaclass, ProtocolBase)):
    __schema__ = "http://json-schema.org/card"
```

The schema can be indicated using different fields: * `__schema__` indicates a URI that the resolver will look for in the schema store. The library comes with a derived resolver which automatically looks for some schemas to load. see `ngoschema.resolver` * `__schema_path__` indicates a path to a file containing the schema

The class should always inherit from `with_metaclass(SchemaMetaclass, Parent1, Parent2)`

If user redefines the `__init__` method, it should always call the ProtocolBase initialization method.

User can't define additional public properties, but is free to do anything with protected or private properties.

`SchemaMetaclass` will build the class doing a lot of magic: * it adds a logger that can be accessed with self.logger * it adds proper logging and exception handling to all methods * it add type conversion/checking and data validation to methods according to their documentation

5.1 Generated Classes

Classes generated using `ngoschema` expose all defined properties as both attributes and through dictionary access.

In addition, classes contain a number of utility methods for serialization, deserialization, and validation.

CHAPTER

SIX

CONTRIBUTING

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

6.1 Bug reports

When 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.

6.2 Documentation improvements

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

6.3 Feature requests and feedback

The best way to send feedback is to file an issue at https://github.com/numengo/python-ngoschema/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 code contributions are welcome :)

6.4 Development

To set up *python-ngoschema* for local development:

1. Fork python-ngoschema (look for the "Fork" button).
2. Clone your fork locally:

```
git clone git@github.com:your_name_here/python-ngoschema.git
```

3. Create a branch for local development:

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

   Now you can make your changes locally.

4. When you're done making changes, run all the checks, doc builder and spell checker with tox one command:

```
tox
```

5. 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
```

6. Submit a pull request through the GitHub website.

6.4.1 Pull Request Guidelines

If you need some code review or feedback while you're developing the code just make the pull request.

For merging, you should:

1. Include passing tests (run `tox`)[1].
2. Update documentation when there's new API, functionality etc.
3. Add a note to `CHANGELOG.rst` about the changes.
4. Add yourself to `AUTHORS.rst`.

6.4.2 Tips

To run a subset of tests:

```
tox -e envname -- pytest -k test_myfeature
```

To run all the test environments in *parallel* (you need to `pip install detox`):

```
detox
```

[1] If you don't have all the necessary python versions available locally you can rely on Travis - it will run the tests for each change you add in the pull request.
It will be slower though ...

CHAPTER

SEVEN

AUTHORS

- Cédric ROMAN - http://www.numengo.com

CHAPTER

EIGHT

CHANGELOG

8.1 0.1.0 (2018-06-04)

- First release on PyPI.

CHAPTER

NINE

INDICES AND TABLES

- genindex
- modindex
- search