OpenAPI for Python

Release 0.0.1

Chris Modzelewski

Apr 13, 2019

CONTENTS:

Serialization/De-serialization for OpenAPI Documents

Version Compatability

OpenAPI for Python is designed to be compatible with:

- Python 2.7 and Python 3.4 or higher, and
- OpenAPI Specification 3.0 or higher

Branch	Unit Tests
latest	
v.0.1	
develop	

CHAPTER

ONE

CONTRIBUTING TO OPENAPI FOR PYTHON

Note: As a general rule of thumb, **OpenAPI for Python** applies **PEP 8** styling, with some important differences.

Branch	Unit Tests
latest	
v.0.1	
develop	

What makes an API idiomatic?

One of my favorite ways of thinking about idiomatic design comes from a talk given by Luciano Ramalho at Pycon 2016[5] where he listed traits of a Pythonic API as being:

- don't force [the user] to write boilerplate code
- provide ready to use functions and objects
- don't force [the user] to subclass unless there's a *very good* reason
- include the batteries: make easy tasks easy
- are simple to use but not simplistic: make hard tasks possible
- leverage the Python data model to:
  - provide objects that behave as you expect
  - avoid boilerplate through introspection (reflection) and metaprogramming.

Contents:

[5] https://www.youtube.com/watch?v=k55d3ZUF3ZQ

1.1 Design Philosophy

OpenAPI for Python is meant to be a "beautiful" and "usable" library. That means that it should offer an idiomatic API that:

- works out of the box as intended,
- minimizes "bootstrapping" to produce meaningful output, and
- does not force users to understand how it does what it does.

In other words:

> Users should simply be able to drive the car without looking at the engine.

1.2 Style Guide

1.2.1 Basic Conventions

- Do not terminate lines with semicolons.
- Line length should have a maximum of *approximately* 90 characters. If in doubt, make a longer line or break the line between clear concepts.
- Each class should be contained in its own file.
- If a file runs longer than 2,000 lines… it should probably be refactored and split.
- All imports should occur at the top of the file.
- Do not use single-line conditions:

```
# GOOD
if x:
  do_something()

# BAD
if x: do_something()
```

- When testing if an object has a value, be sure to use `if x is None:` or `if x is not None`. Do **not** confuse this with `if x:` and `if not x:`.
- Use the `if x:` construction for testing truthiness, and `if not x:` for testing falsiness. This is **different** from testing:
  - `if x is True:`
  - `if x is False:`
  - `if x is None:`
- As of right now, because we feel that it negatively impacts readability and is less-widely used in the community, we are **not** using type annotations.

1.2.2 Naming Conventions

- `variable_name` and not `variableName` or `VariableName`. Should be a noun that describes what information is contained in the variable. If a `bool`, preface with `is_` or `has_` or similar question-word that can be answered with a yes-or-no.
- `function_name` and not `function_name` or `functionName`. Should be an imperative that describes what the function does (e.g. `get_next_page`).
- `CONSTANT_NAME` and not `constant_name` or `ConstantName`.
- `ClassName` and not `class_name` or `Class_Name`.

1.2.3 Design Conventions

- Functions at the module level can only be aware of objects either at a higher scope or singletons (which effectively have a higher scope).
- Functions and methods can use **one** positional argument (other than `self` or `cls`) without a default value. Any other arguments must be keyword arguments with default value given.

```
def do_some_function(argument):
  # rest of function...

def do_some_function(first_arg,
                     second_arg = None,
                     third_arg = True):
  # rest of function ...
```

- Functions and methods that accept values should start by validating their input, throwing exceptions as appropriate.
- When defining a class, define all attributes in `__init__`.
- When defining a class, start by defining its attributes and methods as private using a single-underscore prefix. Then, only once they're implemented, decide if they should be public.

- Don't be afraid of the private attribute/public property/public setter pattern:

```
class SomeClass(object):
  def __init__(*args, **kwargs):
    self._private_attribute = None

  @property
  def private_attribute(self):
    # custom logic which  may override the default return

    return self._private_attribute

  @setter.private_attribute
  def private_attribute(self, value):
    # custom logic that creates modified_value

    self._private_attribute = modified_value
```

- Separate a function or method's final (or default) `return` from the rest of the code with a blank line (except for single-line functions/methods).

1.2.4 Documentation Conventions

We are very big believers in documentation (maybe you can tell). To document **OpenAPI for Python** we rely on several tools:

Sphinx[1]

Sphinx[1] is used to organize the library's documentation into this lovely readable format (which is also published to ReadTheDocs[2]). This documentation is written in reStructuredText[3] files which are stored in `<project>/docs`.

Tip: As a general rule of thumb, we try to apply the ReadTheDocs[2] own Documentation Style Guide[4] to our RST documentation.

Hint: To build the HTML documentation locally:

1. In a terminal, navigate to `<project>/docs`.
2. Execute `make html`.

When built locally, the HTML output of the documentation will be available at `./docs/_build/index.html`.

Docstrings

- Docstrings are used to document the actual source code itself. When writing docstrings we adhere to the conventions outlined in **PEP 257**.

[1] http://sphinx-doc.org
[2] https://readthedocs.org
[3] http://www.sphinx-doc.org/en/stable/rest.html
[4] http://documentation-style-guide-sphinx.readthedocs.io/en/latest/style-guide.html

1.3 Dependencies

- PyYAML v3.10 or higher
- simplejson v3.0 or higher
- Validator-Collection v1.3.0 or higher

1.4 Preparing Your Development Environment

In order to prepare your local development environment, you should:

1. Fork the Git repository.
2. Clone your forked repository.
3. Set up a virtual environment (optional).
4. Install dependencies:

```
open-api/ $ pip install -r requirements.txt
```

And you should be good to go!

1.5 Ideas and Feature Requests

Check for open issues or create a new issue to start a discussion around a bug or feature idea.

1.6 Testing

If you've added a new feature, we recommend you:

- create local unit tests to verify that your feature works as expected, and
- run local unit tests before you submit the pull request to make sure nothing else got broken by accident.

See also:

For more information about the **OpenAPI for Python** testing approach please see: *Testing OpenAPI for Python*

1.7 Submitting Pull Requests

After you have made changes that you think are ready to be included in the main library, submit a pull request on Github and one of our developers will review your changes. If they're ready (meaning they're well documented, pass unit tests, etc.) then they'll be merged back into the main repository and slated for inclusion in the next release.

1.8 Building Documentation

In order to build documentation locally, you can do so from the command line using:

```
open-api/ $ cd docs
open-api/docs $ make html
```

When the build process has finished, the HTML documentation will be locally available at:

```
open-api/docs/_build/html/index.html
```

Note: Built documentation (the HTML) is **not** included in the project's Git repository. If you need local documentation, you'll need to build it.

1.9 References

CHAPTER

TWO

TESTING OPENAPI FOR PYTHON

Contents

2.1 Testing Philosophy

Note: Unit tests for **OpenAPI for Python** are written using pytest[1] and a comprehensive set of test automation are provided by tox[2].

There are many schools of thought when it comes to test design. When building **OpenAPI for Python**, we decided to focus on practicality. That means:

- **DRY is good, KISS is better.** To avoid repetition, our test suite makes extensive use of fixtures, parametrization, and decorator-driven behavior. This minimizes the number of test functions that are nearly-identical. However, there are certain elements of code that are repeated in almost all test functions, as doing so will make future readability and maintenance of the test suite easier.
- **Coverage matters…kind of.** We have documented the primary intended behavior of every function in **OpenAPI for Python**, and the most-likely failure modes that can be expected. At the time of writing, we have about 85% code coverage. Yes, yes: We know that is less than 100%. But there are edge cases which are

[1] https://docs.pytest.org/en/latest/
[2] https://tox.readthedocs.io

almost impossible to bring about, based on confluences of factors in the wide world. Our goal is to test the key functionality, and as bugs are uncovered to add to the test functions as necessary.

- **Sample Documents Matter.** The test suite relies on an extended set of sample OpenAPI documents, some of which have been collected from real (open-source) API resources. The objective is to ensure that we have solid and demonstrable support for the entire OpenAPI Specification.

2.2 Test Organization

Each individual test module (e.g. `test_OpenAPI.py`) corresponds to a conceptual grouping of functionality. For example:

- `test_OpenAPI.py` tests the `OpenAPI` class and its methods

Certain test modules are tightly coupled, as the behavior in one test module may have implications on the execution of tests in another. These test modules use a numbering convention to ensure that they are executed in their required order, so that `test_1_NAME.py` is always executed before `test_2_NAME.py`.

2.3 Configuring & Running Tests

2.3.1 Installing with the Test Suite

2.3.2 Command-line Options

OpenAPI for Python does not use any custom command-line options in its test suite.

Tip: For a full list of the CLI options, including the defaults available, try:

```
open-api $ cd tests/
open-api/tests/ $ pytest --help
```

2.3.3 Configuration File

Because **OpenAPI for Python** has a very simple test suite, we have not prepared a `pytest.ini` configuration file.

2.3.4 Running Tests

2.4 Skipping Tests

Note: Because of the simplicity of **OpenAPI for Python**, the test suite does not currently support any test skipping.

2.5 Incremental Tests

Note: The **OpenAPI for Python** test suite does support incremental testing using, however at the moment none of the tests designed rely on this functionality.

A variety of test functions are designed to test related functionality. As a result, they are designed to execute incrementally. In order to execute tests incrementally, they need to be defined as methods within a class that you decorate with the `@pytest.mark.incremental` decorator as shown below:

```
@pytest.mark.incremental
class TestIncremental(object):
    def test_function1(self):
        pass
    def test_modification(self):
        assert 0
    def test_modification2(self):
        pass
```

This class will execute the `TestIncremental.test_function1()` test, execute and fail on the `TestIncremental.test_modification()` test, and automatically fail `TestIncremental.test_modification2()` because of the `.test_modification()` failure.

To pass state between incremental tests, add a `state` argument to their method definitions. For example:

```
@pytest.mark.incremental
class TestIncremental(object):
    def test_function(self, state):
        state.is_logged_in = True
        assert state.is_logged_in = True
    def test_modification1(self, state):
        assert state.is_logged_in is True
        state.is_logged_in = False
        assert state.is_logged_in is False
    def test_modification2(self, state):
        assert state.is_logged_in is True
```

Given the example above, the third test (`test_modification2`) will fail because `test_modification` updated the value of `state.is_logged_in`.

Note: `state` is instantiated at the level of the entire test session (one run of the test suite). As a result, it can be affected by tests in other test modules.

CHAPTER

THREE

RELEASE HISTORY

Contents

3.1 Release 0.1.0

- First public release

CHAPTER

FOUR

GLOSSARY

JavaScript Object Notation (JSON) A lightweight data-interchange format that has become the *de facto* standard for communication across internet-enabled APIs.

For a formal definition, please see the ECMA-404 Standard: JSON Data Interchange Syntax

De-serialization De-Serialization - as you can probably guess - is the reverse of *serialization*. It's the process whereby data is received in one format (say a JSON string) and is converted into a Python object that you can more easily work with in your Python code.

Think of it this way: A web app written in JavaScript needs to ask your Python code to register a user. Your Python code will need to know that user's details to register the user. So how does the web app deliver that information to your Python code? It'll most typically send JSON - but your Python code will need to then de-serialize (translate) it from JSON into an object representation (your `User` object) that it can work with.

OpenAPI An OpenAPI document (which may be a single file or a collection of files) is a human and machine-readable formal description of a REST API that conforms to the OpenAPI Specification v.3.0 or later.

Per the OpenAPI Initiative:

> The OpenAPI Specification (OAS) defines a standard, programming language-agnostic interface description for REST APIs, which allows both humans and computers to discover and understand the capabilities of a service without requiring access to source code, additional documentation, or inspection of network traffic.
>
> When properly defined via OpenAPI, a consumer can understand and interact with the remote service with a minimal amount of implementation logic. Similar to what interface descriptions have done for lower-level programming, the OpenAPI Specification removes guesswork in calling a service.

The specification is a community-driven open source collaboration within the OpenAPI Inititative, a Linux Foundation Collaborative Project.

Serialization Serialization is a process where a Python object is converted into a different format, typically more suited to transmission to or interpretation by some other program.

Think of it this way: You've got a virtual representation of some information in your Python code. It's an object that you can work with in your Python code. But how do you give that information to some other application (like a web app) written in JavaScript? You serialize (translate) it into a format that other language can understand.

Swagger Swagger was an earlier form of the OpenAPI Specification that was donated by SmartBear to the OpenAPI Initiative in 2015. The Swagger v.2.0 format was formally re-named the OpenAPI Specification v.2.0 and formed the basis for the development of the current OpenAPI Specification v.3.0.

YAML Ain't a Markup Language (YAML) YAML is a text-based data serialization format similar in some respects to *JSON*. For more information, please see the YAML 1.2 (3rd Edition) Specification.

Note: If we're being absolutely formal, JSON is actually a subset of YAML's syntax. But that's being needlessly formal.

CHAPTER

FIVE

OPENAPI FOR PYTHON LICENSE

OpenAPI for Python is a Python library that provides Python support for documents written in the OpenAPI Specification. Use the library to programmatically read, construct, and validate descriptions of RESTful APIs compliant with version 3 of the *OpenAPI Standard*. Specific features include:

- *de-serializing* existing OpenAPI documents from file or URL
- traversing the entire document structure, whether self-contained or using `$ref`
- programmatically modifying the specification of API endpoints, requests, and responses
- *serializing* your API description to OpenAPI v.3.0
- providing validation and error-checking for requests against a RESTful API and responses generated by a RESTful API

The library has been extensively tested on Python 2.7, 3.4, 3.5, 3.6, and 3.7.

CHAPTER

SIX

INSTALLATION

To install **OpenAPI for Python**, just execute:

```
$ pip install open-api
```

6.1 Dependencies

- PyYAML v3.10 or higher
- simplejson v3.0 or higher
- Validator-Collection v1.3.0 or higher

CHAPTER

SEVEN

WHY OPENAPI FOR PYTHON

If you've been involved in the development or documentation of RESTful APIs, odds are you have come across either *Swagger* (a.k.a. OpenAPI v.2.0) or the OpenAPI Initiative. While the standard is an excellent format for describing RESTful APIs in a comprehensive fashion that is both machine and human-readable, the utility of the standard for developers working in Python is only as good as the Python tooling that exists for that standard.

OpenAPI for Python is meant to be a foundational library for OpenAPI tooling in Python. It is to some extent inspired by (but does not directly leverage) the excellent work in:

- openapi-core
- pyswagger
- openapi3

Where **OpenAPI for Python** differs from these other libraries is in three key areas:

- Full Standard Support. Where each of the projects listed above fall down is in their support of the entire OpenAPI standard. By design, **OpenAPI for Python** supports the entire OpenAPI v.3 standard, when programmatically editing an OpenAPI specification in Python, when reading an OpenAPI specification from JSON or YAML, or when writing an OpenAPI specification to JSON or YAML.
- Comprehensive Object Model and Consistent API. As mentioned, the **OpenAPI for Python** object model is intentionally designed to support the full OpenAPI v.3 standard, and that object model is inherently designed to provide a consistent, practical, and inherently Pythonic API for use by Python developers.
- Microframework Architecture. The **OpenAPI for Python** library is meant to be a foundational utility, and not a full-featured application. If you are looking for code generators that can implement a RESTful API based on an OpenAPI specification, then you should look elsewhere. **OpenAPI for Python** cannot do that kind of thing - but it can *enable* that kind of thing and make the development of robust API test suites, API code-generators, API documentation platforms, etc. in Python easier. If those kinds of libraries and applications are a house, **OpenAPI for Python** is meant to provide the bricks.

7.1 Key OpenAPI for Python Features

- **Easy to adopt**: Minimal dependencies, with native support for Python 2.7 and Python 3.4 and higher, and entirely `pip` installable.
- With one method call, generate a Python representation of an OpenAPI specification from a URL, locally on-disk, or from a string object.
- With one method call, output an OpenAPI specification in whole or in part from its Python object representation.
- Validate inbound HTTP requests against their corresponding API endpoints.
- Validate your API's output.

7.2 OpenAPI for Python vs Alternatives

CHAPTER

EIGHT

HELLO, WORLD AND BASIC USAGE

8.1 1. Import OpenAPI for Python

```
from open_api import OpenAPI
```

8.2 2. Load an Existing OpenAPI Specification

```
## From URL
specification = OpenAPI.from_url('http://testing.dev/openapi.yaml')
specification = OpenAPI.from_url('http://testing.dev/openapi.json')

## From File
specification = OpenAPI.from_file('../openapi.yaml')
specification = OpenAPI.from_file('../openapi.json')
```

8.3 3. Modify the OpenAPI Specification

```
specification.title = 'My Updated Title'

my_new_path = specification.add_path(id = 'my_new_path',
                                     path = 'some/new/path/\{id\}',
                                     method = 'GET')
my_new_path_again = specification.get_path(id = 'my_new_path',
                                           method = 'GET')

## The entire OpenAPI Specification is available and supported. See API Reference
## for more details.
```

8.4 4. Validate Complete Specification for Error Checking

```
specification.validate(target_file = './error.log')
```

8.5 5. Validate an HTTP Request

```
## By Path
request_body = request.json
request_headers = request.headers
is_valid = specification.is_valid_request(request_body,
                                          url = 'http://testing.dev/some/new/path',
                                          method = 'POST',
                                          headers = request_headers)
```

8.6 6. Validate an API Response

```
## By Path
is_valid = specification.is_valid_response(response_body,
                                           url = 'http://testing.dev/some/new/path/',
                                           method = 'POST',
                                           headers = response_headers)

## By Schema Object
schema_object = specification.get_schema(object_id = 'my_new_path')
is_valid = schema_object.is_valid(response_body)
```

8.7 7. Output an OpenAPI Specification

```
## In-Memory / In Object Form
json_string = specification.to_json()
yaml_string = specification.to_yaml()
python_dict = specification.to_dict()

## To File
specification.to_json(target_file = './openapi.json')
specification.to_yaml(target_file = './openapi.yaml')
```

CHAPTER

NINE

QUESTIONS AND ISSUES

You can ask questions and report issues on the project's Github Issues Page

CHAPTER

TEN

CONTRIBUTING

We welcome contributions and pull requests! For more information, please see the *Contributor Guide*

CHAPTER

ELEVEN

TESTING

We use TravisCI for our build automation and ReadTheDocs for our documentation.

Detailed information about our test suite and how to run tests locally can be found in our *Testing Reference*.

CHAPTER

TWELVE

LICENSE

OpenAPI for Python is made available under an *MIT License*.

12.1 Indices and tables

- genindex
- modindex
- search

PYTHON MODULE INDEX

t

INDEX