poast Documentation

Release 0.0.1

Andrew T. Canaday

Jun 01, 2021

Contents

Poast[1] is an OpenAPI 3.0 specification parser and client library for Python.

> **Danger:** This project is still in **alpha**.

[1] Python OpenAPI Specification Toolkit

CHAPTER 1

Quickstart

1.1 Specs

1.1.1 Loading

OpenAPI specifications are loaded using `OpenApiObject`.

Specs can be loaded from an `io` object, a file path, or a URI. Both `json` and `yaml` files are supported.

First, get the thing imported:

```
>>> from poast.openapi3.spec import OpenApiObject
```

Then, load a spec (all of the following work):

```
>>> doc_from_url = OpenApiObject(
...     'https://petstore3.swagger.io/api/v3/openapi.json')

>>> doc_from_filepath = OpenApiObject('./my/openapi.yml')

>>> with open('./my/other/openapi.json', 'rb') as f:
>>>   from_io = OpenApiObject(f)
```

1.1.2 Validation

Validation is performed using the `validate()` method on the returned `OpenApiObject`, e.g.:

```
>>> my_doc = OpenApiObject('./path/to/my/openapi.yml')
>>> my_doc.validate()
```

- *Loading or validation* errors will raise `DocumentParsingException`.
- Errors with the *document* itself, will raise `MalformedDocumentException`

See also:

`poast.openapi3.spec.model.exceptions`

1.2 Clients

1.2.1 Generating

Clients are created from a `OpenApiObject` using `get_client_cls()` function, e.g.:

```
>>> from poast.openapi3.spec import OpenApiObject
>>> from poast.openapi3.client import gen_client_cls
>>>
>>> # api_spec = OpenApiObject(...).validate()
>>>
>>> # Generate a client class from the spec:
>>> MyApiClient = gen_client_cls('MyApiClient', api_spec)
```

The return value is a subclass of `OpenApiClient`, objects of which can be instantiated in the usual fashion:

```
>>> client = MyApiClient('https://myservice.com/api/root')
```

1.2.2 API Operations

The client classes created using `get_client_cls()` have a special attribute, `op` (an API-specific subclass of `OpenApiOperations`), with one method for each API operation defined in the spec. For example, if an api describes two operations, `someAction` and `anotherAction`, client instances will have the following two methods in their `op` object:

```
>>> client.op.someAction()
>>> client.op.anotherAction()
```

Invoking Operations

Methods defined on the `OpenApiOperations` subclass generally mimic the call signature of `requests.Request()`, with *path parameters passed as keyword arguments*.

Request Parameters

Spec `"in"`:	Passed as:
`path`	`**kwargs`
`query`	`params (dict)`
`header`	`headers (dict)`
`cookie`	`cookies (dict)`

Request Body

The the request body can passed - `requests`-style as either `json`, `data`, `stream`, or `files`.

Note: In the event that a path parameter collides with a Python keyword, builtin, or existing named parameter to the underlying `requests.Request` object method, both the URI template and the parameter name are adjusted to include the suffix `'_'`.

1.2.3 Example

Consider, for instance, a utility provider which provides a customer API. Let's suppose that the API provides a `getUsage` operation which requires us to make a `GET` request with:

- a `customerId` parameter specified in the URI path
- a `days` parameter, specified as a query arg
- an auth token, in the `X-API-Key` HTTP header

Finding the resource usage for user #123 over the last 30 days might look like:

```
>>> resp = my_client.getUsage(
...   customerId=123,
...   params={'days': 30},
...   headers{'X-API-KEY': MY_SECRET_API_KEY}
... ).execute()
```

The `resp` object here is a standard `requests.Response` object. We can get the body as text like so:

```
>>> print(resp.text)
```

Or (if it's JSON), like so:

```
>>> print(resp.json())
```

1.2.4 API Help

Operations for generated clients include docstrings for each method that indicates the type and location of all required parameters, e.g.:

```
>>> help(my_client.op.getPetById)
Help on method getPetById in module poast.openapi3.client.genop:

getPetById(headers=None, params=None, cookies=None, data=None, json=None, files=None,␣
↪hooks=None, **path_params) method of poast.openapi3.client.genops.
↪PoastExampleClientOperations instance
    http: GET /pet/{petId}
    summary: Find pet by ID
    description: Returns a single pet

    path parameters (keyword args):
      petId:
        description: ID of pet to return
        required: True
        deprecated: False
        allowEmptyValue: False

    Security Requirements:
```

(continues on next page)

(continued from previous page)

```
    api_key: []
    petstore_auth: ['write:pets', 'read:pets']
```

Resources

For other examples, see:

- example: PetStoreClient
- example: PetStoreClient Operations

CHAPTER 2

Advanced Usage

2.1 Prepared Requests

API operations on `OpenApiOperations` objects return standard `requests.PreparedRequest` object, patched to include an `execute()` method (an instance of `RequestExecutor`).

This allows the client opportunity to make last minute modifications to a request before it is sent, e.g.:

```
>>> req = my_client.someAction()
>>> # Make some adjustment to the body:
>>> req.body += r'This has to get appended'

>>> # Fire it off and get a standard requests.Response:
>>> resp = req.execute()
```

2.2 Client Configuration

See also:

poast.openapi3.client.config.ClientConfig.

2.3 Custom Sessions

2.3.1 Using a specific session

A particular `requests.Session` (or compatible) session instance can be passed in at the time of client instantiation, using the `session` parameter:

```
>>> my_session = requests.Session()
>>> my_client = MyClientClass(root_url='http://something.com',
...   session=my_session)
```

2.3.2 Using a custom class

To use a specific *class* of session object for a client:

```
>>> my_client = MyClientClass(root_url='http://something.com',
...   config=ClientConfig(session_cls=SpecialSessionClass))
```

2.4 Custom Requests

To use a specific *class* of request object for a client:

```
>>> my_client = MyClientClass(root_url='http://something.com',
...   config=ClientConfig(request_cls=SpecialRequestClass))
```

CHAPTER 3

Client API

3.1 OpenApiClient

3.2 ClientConfig

class poast.openapi3.client.config.**ClientConfig**(*logger=None*, *session_cls=None*, *request_cls=None*, *headers: dict = None*, *cookies: dict = None*)

Bases: object

Configuration object used to customize OpenApiClient creation.

logger
optional logger client created using this config

Type logging.Logger

session_cls
a requests.Session-like class used to create HTTP sessions

Type type

request_cls
a requests.Request-like class used to create HTTP requests

Type type

headers
a list of headers common to all requests for client created from this config

Type dict

cookies
a list of cookies common to all requests for client created from this config

Type dict

Notes

- headers and cookies are copied via the *copy* module!

__init__(*logger=None*, *session_cls=None*, *request_cls=None*, *headers: dict = None*, *cookies: dict = None*)
Utility class to package up client configuration for re-use across multiple clients.

NOTE: headers and cookies are copied via the *copy* module!

__setattr__(*name: str*, *value*)
Prevent anything but logger from being set to None.

3.3 OpenApiOperations

3.4 RequestExecutor

CHAPTER 4

Spec API

4.1 Object Interface

All fields in OAS 3.0 objects are accessed using python's `[]` operator. This is done to avoid any confusion between which fields are in the spec vs attributes of the implementation class and to avoid workarounds wherein spec fields which conflict with keywords or builtins - e.g. `in` - can retain their original names.

For example, accessing the `paths` field in a document is done like so:

```
my_doc = OpenApiObject(...)
paths = my_doc['paths']
```

4.2 OpenApiBaseObject

class poast.openapi3.spec.model.baseobj.**OpenApiBaseObject**(*data*, *doc_path='#'*)

Bases: poast.openapi3.spec.model.entity.OpenApiEntity, dict

Base class for OpenAPI specification objects.

__init__(*data*, *doc_path='#'*)

Invoke child class initialization.

__missing__(*key*)

If a value is missing from the spec, return the default, as defined by the standard, if possible.

accept(*visitor*)

Depth first traversal, via visitor.

Parameters **visitor** (*func*) – a function that takes a single OpenApiEntity as an argument.

validate()

Validate the data in this object against the spec.

value(*show_unset=False*)
Gnarly (in the bad way) convenience/debug function used to return the document as a python dictionary for pprinting and dev validation.

__weakref__
list of weak references to the object (if defined)

4.3 Primitive Types

OpenApi 3.0 Primitive datatype definitions.

See also: - https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.2.md#data-types

4.4 Container Types

This module defines the OpenApi 3.0 container types

For more info, see:

- https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.2.md#format

4.5 Document Objects

CHAPTER 5

Planned Features

Note: All of the following are either planned or current WIP.

5.1 *Coming Soon!*

- Automatic parameter completion for linked operations
- Support for callbacks
- Pluggable auth handlers
- Pluggable extension handlers
- Response parsing

5.2 On the Horizon

- Support for common pagination styles
- Integrated support for `requests_oauthlib` to automatically handle OpenAPI OAuth Flows
- Extensions for (*some*) common vendor extensions/auth schemes

5.3 Maybe

- Request validation
- Response validation

CHAPTER 6

History

6.1 0.0.1 (2020-12-01)

- Initial public push
- **Still TODO:**
  - Travis setup
  - Readthedocs
  - Unit tests

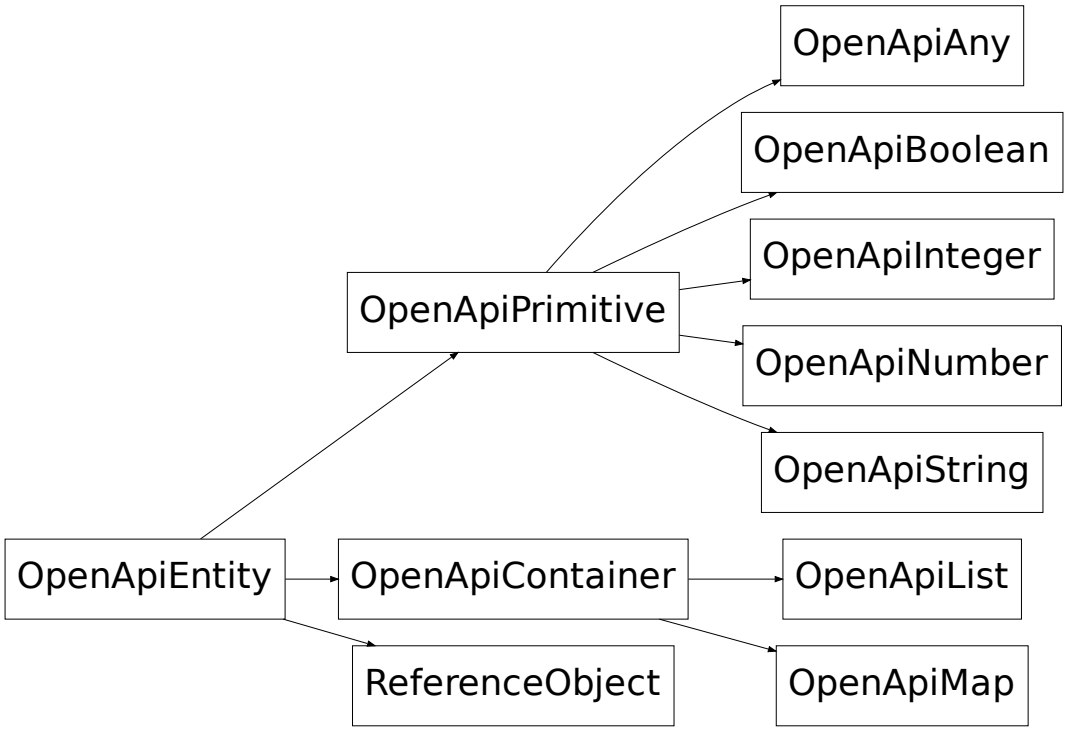
CHAPTER 7

Maintainer Docs

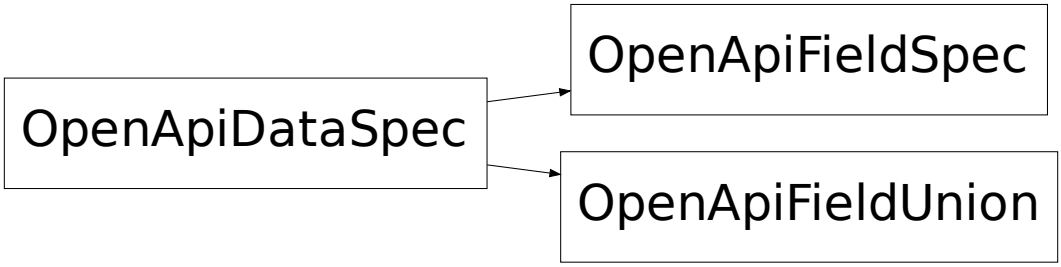
Poast maintainer documentation can be found below. The canonical source can be found at poast on github.

7.1 Parser Object Model

7.1.1 Data Objects



7.1.2 Field Specs



CHAPTER 8

Features

- OpenAPI 3.0.3 compliant parsing and validation
- Runtime client generation
- Utilizes `requests` for underlying HTTP request/response
- Compatible with `requests_futures` and `requests_oauthlib` sessions

CHAPTER 9

Demo

Creating a client from a spec::

```
>>> from poast.openapi3.spec import OpenApiObject
>>> from poast.openapi3.client import gen_client_cls

>>> # Load the spec (path, io stream, url, or string!)
>>> api_spec = OpenApiObject(
...     'https://petstore3.swagger.io/api/v3/openapi.json')

>>> # [Optionally, perform validation]
>>> api_spec.validate()

>>> # Generate a client class from the spec:
>>> PetStoreClient = gen_client_cls('PetStoreClient', api_spec)

>>> # Instantiate a client and start using the API!
>>> client = PetStoreClient('https://petstore3.swagger.io/api/v3')

>>> print(client.op.getInventory().execute().json())
{'approved': 57, 'placed': 100, 'delivered': 50}

>>> print(client.op.getPetById(petId=2).execute().json())
{'id': 2, 'category': {'id': 2, 'name': 'Cats'}, 'name': 'Cat 2', 'photoUrls': ['url1
↪', 'url2'], 'tags': [{'id': 1, 'name': 'tag2'}, {'id': 2, 'name': 'tag3'}], 'status
↪': 'sold'}
```

Python Module Index

p

Index

Symbols

A

C

H

L

O

P

R

S

V